London Mathematical Society ISSN 1461–1570

ON TENSOR-FACTORISATION PROBLEMS, I: THE COMBINATORIAL PROBLEM

PETER M. NEUMANN AND CHERYL E. PRAEGER

Abstract

A k-multiset is an unordered k-tuple, perhaps with repetitions. If x is an r-multiset $\{x_1, \ldots, x_r\}$ and y is an s-multiset $\{y_1, \ldots, y_s\}$ with elements from an abelian group A the tensor product $x \otimes y$ is defined as the rs-multiset $\{x_i y_j \mid 1 \leqslant i \leqslant r,\ 1 \leqslant j \leqslant s\}$. The main focus of this paper is a polynomial-time algorithm to discover whether a given rs-multiset from A can be factorised. The algorithm is not guaranteed to succeed, but there is an acceptably small upper bound for the probability of failure. The paper also contains a description of the context of this factorisation problem, and the beginnings of an attack on the division-problem: is a given rs-multiset divisible by a given r-multiset, and if so, how can division be achieved in polynomially bounded time?

1. *Introduction*

There is a wide range of factorisation problems that arise from constructions related to the tensor product. Although our interest in questions of this type originated in computational group theory, it quickly spread to other contexts. In this paper we formulate four such problems, and treat the most general of them in some detail.

Combinatorial tensor factorisation. The most comprehensive context for tensor-factorisation problems is combinatorial. Let k be a natural number. By a *k-multiset* from a set A, or a *multiset of size k*, we mean an unordered k-tuple of members of A. Thus the set $A^{[k]}$ of all k-multisets is the quotient set $A^k / \mathrm{Sym}(k)$, where A^k is the cartesian product of k copies of A and the symmetric group $\mathrm{Sym}(k)$ of degree k acts in the natural way, permuting the factors. It is natural to write $a = \{a_1, \ldots, a_k\}$ for the multiset whose members, in some order and perhaps with repetitions, are $a_1, \ldots, a_k$, and to define $|a| := k$. Now suppose that A is an abelian group (written multiplicatively because in the most important applications, it will be the multiplicative group of a field). Let n, r and s be natural numbers such that $n = rs$. If $x \in A^{[r]}$ and $y \in A^{[s]}$, then we define

$$x \otimes y := \{x_i y_j \mid 1 \leqslant i \leqslant r,\ 1 \leqslant j \leqslant s\} \in A^{[n]}.$$

It is convenient (and not very misleading) to refer to $\otimes$ as a *tensor product* of multisets. Associated with it, there is an algorithmic factorisation problem. We refer to an expression $a = b \otimes c$, where $b \in A^{[r]}$ and $c \in A^{[s]}$, as a *tensor (r, s)-factorisation*, or simply an *(r, s)-factorisation* of a.

COMBINATORIAL TENSOR-FACTORISATION PROBLEM. Given an abelian group A and positive integers n, r and s such that $n = rs$, design and analyse efficient algorithms that will accept

Received 24 November 2003, revised 27 January 2004; *published* 26 April 2004.
2000 Mathematics Subject Classification 05-04, 20-04

as input a multiset $a \in A^{[n]}$ and give as output either an (r, s)-factorisation $a = b \otimes c$ or the information that no (r, s)-factorisation exists.

Our formulation of the problem allows the design of the algorithm to depend on the abelian group A, and on the size of the input. There is good reason for this. Although we hope and expect to have algorithms that work in much the same way for all abelian groups and all values of n, r and s, we must be alive to the possibility that significantly better-performing procedures might be available in certain important special cases. Some cases that we have in mind are that A is cyclic, or that r is small.

The combinatorial tensor-factorisation problem makes sense if A is replaced by a commutative semigroup. In full generality, there is little that we wish to say about this problem. But the case where A is replaced by (the multiplicative semigroup of) a field K is applicable to other factorisation problems, and this special case can be reduced to the one that we have already formulated. All that is needed is, in fact, an integral domain.

OBSERVATION 1.1. *Let R be an integral domain, let n, r and s be natural numbers such that $n = rs$, and let a be an n-multiset from R. Define a_0 to be a with all elements equal to 0 removed, and let $n_0 := |a_0|$. Then there is a tensor factorisation $a = b \otimes c$ with $b \in R^{[r]}$ and $c \in R^{[s]}$ if and only if there exist $r_0 \leqslant r$, $s_0 \leqslant s$, $b_0 \in R^{[r_0]}$ and $c_0 \in R^{[s_0]}$ such that $n_0 = r_0 s_0$ and $a_0 = b_0 \otimes c_0$.*

Proof. Given a tensor factorisation $a = b \otimes c$ with $b \in R^{[r]}$ and $c \in R^{[s]}$, we define b_0 and c_0 to be what results from removing all zeros from b and c, respectively. If $r_0 := |b_0|$ and $s_0 := |c_0|$, then we obtain a tensor factorisation $a_0 = b_0 \otimes c_0$ with $b_0 \in R^{[r_0]}$ and $c_0 \in R^{[s_0]}$. Moreover, this multiset a_0 is what is obtained by removing all zeros from a. The converse follows similarly from the fact that if $a = b \otimes c$, then any zero in a will have arisen from a zero either in b or in c. □

Polynomial tensor factorisation. The second context for tensor-factorisation problems is that of polynomials. Let F be a field, let x and y be (monic) polynomials of degrees r and s respectively in $F[t]$, and let $\xi_1, \ldots, \xi_r$ and $\eta_1, \ldots, \eta_s$ be the roots of x and y respectively in some algebraic closure of F. Using a harmless but suggestive extension of the notation $\otimes$, we write

$$(x \otimes y)(t) := \prod_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant s}} (t - \xi_i \eta_j) \in F[t],$$

so that $(x \otimes y)$ is a monic polynomial over F of degree rs.

POLYNOMIAL TENSOR-FACTORISATION PROBLEM. Given a field F and positive integers n, r and s such that $n = rs$, design and analyse efficient algorithms that will accept as input a monic polynomial $f(t)$ of degree n in $F[t]$ and give as output either a factorisation $f = x \otimes y$, where x and y are monic polynomials of degrees r and s respectively over F, or the information that no such factorisation exists.

Matrix tensor factorisation. Let X and Y be $r \times r$ and $s \times s$ matrices $(x_{i,j})$ and $(y_{k,l})$ respectively over a field F. The Kronecker product $X \otimes Y$ is defined to be that $rs \times rs$ matrix that, when it is partitioned as an $r \times r$ array of $s \times s$ blocks, has $x_{i,j}Y$ as its (i, j)th block. Thus its $((i-1)s + k, (j-1)s + l)$ entry is $x_{i,j}y_{k,l}$ (see, for example, [7, vol. II, Chapter VII, Section 5]).

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

MATRIX TENSOR-FACTORISATION PROBLEM. Given a field F and positive integers n, r and s such that $n = rs$, design and analyse efficient algorithms that will accept as input an $n \times n$ matrix A over F, and give as output either an invertible $n \times n$ matrix P over F and a factorisation $P^{-1}AP = X \otimes Y$, where X and Y are $r \times r$ and $s \times s$ matrices respectively over F, or the information that no such factorisation exists.

In more abstract terms, this problem may be re-cast as the following. Given a linear transformation $\alpha : V \longrightarrow V$, where V is an n-dimensional vector space over F, discover whether there exist tensor factorisations $V \cong U \otimes W$ and $\alpha = \xi \otimes \eta$, where U and W are r- and s-dimensional vector spaces over F, and ξ and η are linear transformations $U \longrightarrow U$ and $W \longrightarrow W$, respectively.

Group-theoretic problems. As we indicated in the opening paragraph above, the original context of our questions was computational group theory. We wanted usable algorithms to decide whether or not a given matrix representation could be expressed as a tensor product of representations of smaller degree.

MODULE TENSOR-FACTORISATION PROBLEM. Given a field F and positive integers n, r and s such that $n = rs$, design and analyse efficient algorithms that will accept as input a group G and an FG-module M of dimension n over F, and will give as output either a factorisation $M \cong X \otimes Y$, where X and Y are FG-modules of dimensions r and s respectively, or the information that no such tensor factorisation exists.

We defer a detailed explanation and discussion of this representation-theoretic problem to a later paper. Here it must suffice to repeat that it provided our starting point. The matrix, polynomial, and combinatorial tensor-factorisation problems emerged, in the reverse order to that listed here, not merely as natural generalisations, but also as usable components in an analysis of the group-theoretic problem.

Analysis and efficiency. Our statements of the problems contain the words 'design and analyse efficient algorithms'. In the first place, we want practical methods that can be implemented and used for a fair range of values of n, say $n \leqslant 1000$. In the second place, we want algorithms whose theoretical efficiency is measured asymptotically by complexity estimates that are polynomial in n, and, if possible, independent of the abelian group A or the field F. In these estimates, it will be assumed that comparison of two elements of A or of F takes one unit of time; likewise, that performing one algebraic operation xy or xy^{-1} in A, or one algebraic operation $x + y$, xy, $x - y$ or x/y in F, takes one unit of time. Depending on the group A or the field F, each of these time units will cost $c(k)$ bit operations, where k is some measure of the number of bits required to describe the input multiset a, and c is an appropriate cost function that depends on A or F and the way that their elements are described.

For comparison with a more familiar problem, recall that an n-bit integer has size less than 2^n. Factorisation by testing all numbers up to its square root, using long division, has time complexity of the order of $2^{n/2} \log^2 n$, and although somewhat more efficient methods are known, no polynomial-time algorithm is known (or expected to be discovered in the foreseeable future). Our combinatorial tensor-factorisation problem has a similarly inefficient answer. Consider the element a_1 of a. If a is factorisable as $b \otimes c$, then a_1 will be of the form $b_i c_j$ for some i and j, and without loss of generality we may suppose that $a_1 = b_1 c_1$. Let $b' := bc_1 = \{b_1c_1, b_2c_1, \ldots, b_rc_1\}$ and $c' := b_1c = \{b_1c_1, b_1c_2, \ldots, b_1c_s\}$. If $a = b \otimes c$, then b' and c' are sub-multisets of a containing a_1, and $a = (a_1^{-1})b' \otimes c'$. Therefore, the following simple procedure will find an (r, s)-factorisation if there is one.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

ALGORITHM 1.2. *Input*: a multiset $a \in A^{[n]}$ and a factorisation $n = rs$.

Output: a tensor factorisation $a = b \otimes c$ with $b \in A^{[r]}$, $c \in A^{[s]}$, or FALSE if no such factorisation exists.

Cost: $O(n^{r+s})$ time units.

Begin:
  for each pair (I, J) of disjoint subsets of $\{2, \ldots, n\}$ with $|I| = r - 1$, $|J| = s - 1$ do
    define $b' := \{a_\nu \mid \nu = 1 \text{ or } \nu \in I\}$; $c' := \{a_\nu \mid \nu = 1 \text{ or } \nu \in J\}$;
    define $b := a_1^{-1} b'$ and $c := c'$;
    if $a = b \otimes c$ then exit and return (b, c);
  endfor;
  return FALSE;
end.

The for-loop is traversed at most $\binom{n-1}{r-1}\binom{n-1}{s-1}$ times, which is smaller than n^{r+s-2}. Testing whether $a = b \otimes c$ can obviously be done at a cost of $n^2 + n$ multiplications and n^2 comparisons in A, so the cost of this procedure is at most $O(n^{r+s})$ time units. Note, however, that $n^{r+s} \geqslant n^{2\sqrt{n}}$ and, although decisive, this strategy is too slow to be acceptable.

The main theorem. We do not know whether there is a polynomial-time deterministic solution to the combinatorial tensor-factorisation problem. It seems unlikely, but we have no real evidence. Therefore we curb our ambitions and seek a method that will succeed almost always in polynomial time. Our strategy will be to consider pairs $(b, c) \in A^{[r]} \times A^{[s]}$ that have a certain useful property that we call *recognisability* (see Definition 5.1 below). That property is chosen to meet two conditions: first, it should be possible to design an efficient algorithm to detect factorisations $a = b \otimes c$ where the pair (b, c) is recognisable; secondly, it should be possible to prove that almost all pairs (b, c) have the property. The main results of this paper are to be found in Sections 5 and 6, and may be summarised as the following theorems.

THEOREM A. *There is an algorithm which will accept as input an n-multiset a from an abelian group A, where $n = rs$ and $2 \leqslant r \leqslant s$, and which yields as output either a pair $(b, c) \in A^{[r]} \times A^{[s]}$ such that $a = b \otimes c$, or the information that no such tensor factorisation, in which the pair (b, c) is recognisable, exists. The computation costs $O(rn^3 \log n)$ time units.*

The algorithm referred to here is Algorithm 5.8. A much simplified version, which is appropriate only if A has no involutions, but which should work approximately eight times faster, is described in Section 4. In his 2003 Oxford MSc dissertation [**6**], Jia Lun Huang has produced an implementation of this algorithm in the GAP language. His tests, which involved factorising multisets of size n for various values of n up to 600 in various cyclic groups and elementary abelian 2-groups, confirm that it is practicable. Details will be published elsewhere.

THEOREM B. *Amongst all pairs $(b, c) \in A^{[r]} \times A^{[s]}$, the proportion that are not recognisable is less than $2n^2|A|^{-1}$, where $n = rs$.*

If the abelian group A is small, then Theorem B is weak. In practice, however, A will be very large compared with n^2, so that the algorithm should succeed in factorising almost

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

all factorisable multisets. What we mean is this: in practice, we have in mind to use our algorithms for finding tensor factorisations of polynomials f of degree n over the finite field $\mathbb{F}_q$ that do not have 0 as a root (often, but not exclusively, the characteristic polynomials of invertible $n \times n$ matrices over $\mathbb{F}_q$). Then the abelian group A is naturally taken to be the multiplicative group $K^\times$ of a splitting field of f. The expected degree of K is approximately $n^{(1/2)\log n}$ (see, [**9**, **2**]); indeed, one can prove quite easily that, for large enough n, with probability greater than $1 - \mathrm{e}^{-\sqrt{n}/2}$ the splitting field degree is greater than $\sqrt{n}$. In this situation, therefore, $|A| > q^{\sqrt{n}}$ with very high probability. And of course $q^{\sqrt{n}}$ is very much larger than $2n^2$.

The comparison with factorisation of integers can be extended: the integer-factorisation algorithm which accepts as input an n-bit integer x, and which tests for divisibility by integers less than n costs $O(n^2 \log n)$ bit-operations; but it fails with probability $O(1/\log n)$ in the sense that the proportion of natural numbers $x < 2^n$ that have no factor less than n is $O(1/\log n)$.

As has been mentioned above, our combinatorial tensor-factorisation algorithm is ultimately intended to be used as a component in an algorithm for tensor factorisation of modules. In this context, it will be used on the multisets of eigenvalues of matrices X. Using it repeatedly with different input matrices X then gives a Monte Carlo algorithm for the module tensor-factorisation problem.

Other results. It is natural to ask for a method to divide one multiset by another. Given $a \in A^{[n]}$ and $b \in A^{[r]}$ where, as usual, $n = rs$, how can we discover whether b tensor divides a, and if so, find c such that $a = b \otimes c$? In general, this seems a difficult question, and we can say little about it. Recently, however, Jia Lun Huang has devised a very interesting algorithm, which succeeds with high probability (see [**6**]). For the special case where $r = 2$, we have an efficient deterministic method.

THEOREM C. *There is an algorithm which will accept as input an n-multiset a and a 2-multiset b from an abelian group A, where $n = 2s$, and which yields as output either a multiset $c \in A^{[s]}$ such that $a = b \otimes c$, or the information that no such tensor factorisation exists. The computation costs $O(n \log n)$ time units.*

This will be proved in Section 4. In Section 7 we turn to the question of uniqueness of tensor factorisations and show that if the pair (b, c) satisfies a strengthened version of the recognisability condition, then $b \otimes c$ will have no tensor factorisations other than those obtained by reversing the factors or multiplying them suitably by 'scalars'.

Organisation of this paper. As has already been indicated, we concentrate here on the combinatorial tensor-factorisation problem. We plan a sequel devoted to the polynomial, matrix and module problems. Section 2 is devoted to notation and terminology, Section 3 to elementary considerations (namely to testing for factorisability, and to the special case of tensor factorisations in an abelian group of odd order), Section 4 to an algorithm for the $(2, s)$-tensor-factorisation problem based on an algorithm to compute whether a given $2s$-multiset a can be divided by a given 2-multiset b, Section 5 to our tensor-factorisation algorithm, Section 6 to complexity analysis, and Section 7 to probability estimates.

We hope that by giving a full explanation of the combinatorial tensor-factorisation problem, we may have anticipated many of the difficulties that might arise in other, similar problems. That this hope is not unrealistic is illustrated by the work of Catherine Greenhill

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

who, in [3] (now published in [4] and [5]), deals with analogous 'exterior square root' problems. She uses adaptations, albeit original and non-trivial ones, of the methods developed in the first draft (1995) of the present paper.

2. *Notation, terminology, and other preliminaries*

Throughout the paper, A will denote an abelian group, n, r and s will denote positive integers such that $n = rs$, and it will be assumed, obviously without loss of generality, that $2 \leqslant r \leqslant s$. The symbols a, b and c will be reserved for multisets of sizes n, r and s respectively. Recall that $A^{[N]}$ denotes the set of all N-multisets from A.

- For $g \in A$ and $x \in A^{[N]}$, the multiplicity of g as an element of x is denoted by $\mathrm{mult}(g;\ x)$.
- For $x \in A^{[N]}$, define $|x| := N$. Thus $|x| = \sum_{g \in x} \mathrm{mult}(g;\ x)$.
- A multiset will be said to be *multiplicity-free* if its elements are distinct or, equivalently, if all its multiplicities are 1; that is, if it is a true set.
- For multisets x and y, the multiset $x + y$ is defined to be the union of x and y, counting multiplicities; that is, $\mathrm{mult}(g;\ x + y) = \mathrm{mult}(g;\ x) + \mathrm{mult}(g;\ y)$ for all $g \in A$.
- For multisets x and y, the multiset $x \cap y$ is defined to be the intersection of x and y, counting multiplicities; that is, $\mathrm{mult}(g;\ x \cap y) = \min\{\mathrm{mult}(g;\ x), \mathrm{mult}(g;\ y)\}$ for all $g \in A$.
- Let x be a multiset of members of A, and let $g \in A$. Define $gx := \{gx_i \mid x_i \in x\}$ and $xg := \{x_i g \mid x_i \in x\}$. Note that since A is abelian, $gx = xg$, and that gx is in fact $\{g\} \otimes x$.
- Clearly, if $a = b \otimes c$, then also $a = bg \otimes g^{-1}c$ for any $g \in A$. Tensor factorisations related in this way will be said to be *equivalent*.
- If $x = \{x_1, \ldots, x_N\} \in A^{[N]}$, define

$$xx^{-1} := \{x_\alpha x_\beta^{-1} \mid 1 \leqslant \alpha \leqslant N, 1 \leqslant \beta \leqslant N, \alpha \neq \beta\},$$

so that xx^{-1} is a multiset of size $N(N-1)$. We refer to x_α as the *numerator* and x_β as the *denominator* of the element $x_\alpha x_\beta^{-1}$ of xx^{-1}.

- Note that if $a = b \otimes c$, then, with a natural convention about multiplication of multisets by positive integers,

$$aa^{-1} = s.bb^{-1} + r.cc^{-1} + bb^{-1}cc^{-1},$$

where $bb^{-1}cc^{-1} := bb^{-1} \otimes cc^{-1}$.

- The symbols ν and ν' will be reserved for indices running from 1 to n, the symbols i, i', k and k' will be reserved for indices running from 1 to r, the symbols j, j', l, and l' will be reserved for indices running from 1 to s, and we maintain the conventions that $\nu \neq \nu'$, that $i \neq k$, and that $j \neq l$.

The following simple lemma limits the search for tensor factorisations.

LEMMA 2.1. *Let a be an n-multiset from the abelian group A, and let $A_0 := \langle a \rangle$, the subgroup of A generated by a. Any (r, s)-tensor factorisation of a in A is equivalent to an (r, s)-tensor factorisation in A_0.*

Proof. Certainly, any tensor factorisation is equivalent to a factorisation $a = b \otimes c$ in which $b = \{b_1, \ldots, b_r\}$, where $b_1 = 1$. Then $c \subseteq a$, and since for any i there exist j and ν such that $b_i = a_\nu c_j^{-1}$, we have $b, c \subseteq A_0$. This proves the lemma. □

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Since the combinatorial tensor-factorisation problem is a computational question, it is to be understood in the following way. The abelian group A must be given in computable form. We do not propose to be explicit about this because different practical contexts require differing data structures, and our intention is to focus on the underlying algorithmic problems. But because of Lemma 2.1 above, A can be assumed to be finitely generated (in practice, it will usually be finite). Then it may be taken to be presented as a direct product of cyclic groups, since there are good algorithms to achieve this. Using a natural linear ordering of each factor, and the lexicographic ordering of the product, it can then be assumed that there is an easily computable linear ordering of A, which, even if not compatible with the multiplication on A, can be used to ensure that lists of length N can be sorted at a cost of $O(N \log N)$ comparisons of members of A (see, for example, [**1**] or [**8**]). This is important because, although a multiset is an unordered sequence, in practice it will be represented by an ordered sequence (a list of its members), and efficient comparison of multisets therefore requires efficient sorting. To compare two N-multisets x and y, one may sort x, sort y, and then compare elements of the sorted lists one by one, in order. Accordingly, we have the following theorem.

THEOREM 2.2. *Let $f(N)$ be the number of element-comparisons required to sort an N-multiset of elements of a set A, given a computable ordering of A. Then the cost of multiset-comparison is at most $2f(N) + N$ element-comparisons. Thus this cost may be taken to be $O(N \log N)$ element-comparisons.*

Suppose that the N-multiset x has already been sorted. If $g \in A$, then using list-bisection to find the first and last occurrences of g in x costs $O(\log N)$ element-comparisons. This leads to our next theorem.

THEOREM 2.3. *If x is a sorted N-multiset from A and $g \in A$, then* $\mathrm{mult}(g; x)$ *can be computed at a cost of $O(\log N)$ element-comparisons.*

3. *Elementary considerations*

Most n-multisets do not arise as tensor products, and therefore it is valuable to have a quick test to discard candidates for (r, s)-factorisation. Such a test can be based on the fact that $(b \otimes c)(b \otimes c)^{-1} = sbb^{-1} + rcc^{-1} + bb^{-1}cc^{-1}$. The multiset bb^{-1} is *self-inverse* in the sense that for all $g \in A$, $\mathrm{mult}(g; bb^{-1}) = \mathrm{mult}(g^{-1}; bb^{-1})$, and if $g = g^{-1}$, then $\mathrm{mult}(g; bb^{-1})$ is even. The same is of course true of cc^{-1}. Therefore we can exploit the following simple fact.

LEMMA 3.1. *Suppose that $r \leqslant s$. Let x, y and z be multisets such that $sy + rz \subseteq x$. If y' is a multiset such that $|y'| = |y|$ and $sy' \subseteq x$, then there exists a multiset z' such that $|z'| = |z|$ and $sy' + rz' \subseteq x$.*

The same assertion holds with the extra assumption that y, y' and z, z' are self-inverse.

Proof. It is sufficient to prove the first assertion in the case where y' differs from y in just one element. Thus we may suppose that $y = \{y_1, y_2, \ldots, y_R\}$ and $y' = \{y'_1, y_2, \ldots, y_R\}$, where $R = |y| = |y'|$. If $y'_1 \notin z$, then we take $z' := z$. Otherwise, take $z' := (z \setminus \{y'_1\}) + \{y_1\}$ and, since $r \leqslant s$, we have $sy' + rz' \subseteq x$, as required. The case of self-inverse multisets y, y' and z, z' is similar, and is omitted here. □

As a consequence of this lemma, and given that $r \leqslant s$, to test whether a given $n(n-1)$ multiset a contains a submultiset of the form $sb^* + rc^*$, where b^* is a self-inverse

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

$r(r-1)$ multiset and c^* is a self-inverse $s(s-1)$ multiset, we can pick out any candidate for b^* first. Note that the size of a self-inverse multiset is even. The following is the main component in our elementary test.

ALGORITHM 3.2. *Input*: an N-multiset x from A, a positive integer m and an even positive integer R.

Output: a self-inverse R-multiset y such that $my \subseteq x$, or FALSE if no such multiset y exists.

Cost: $O(N \log N)$ time units.

Begin:

sort x into the form $m[1]g[1] + \cdots + m[t]g[t]$,
  where $g[1] < \cdots < g[t]$ in A and $m[\sigma] = \text{mult}(g[\sigma];\ x)$; (1)
re-sort x so that $g[2i-1]^{-1} = g[2i]$ for $1 \leqslant i \leqslant t_1$,
$g[i]^2 = 1$ for $2t_1 + 1 \leqslant i \leqslant t_2$, and $g[i]^{-1} \notin x$ for $t_2 + 1 \leqslant i \leqslant t$;
re-number the $m[i]$ accordingly; (2)
initialise y as $\emptyset$; (3)
for $i = 1$ to t_1 do (4)
  while $m[2i-1] \geqslant m$ and $m[2i] \geqslant m$ do (5)
    adjoin $g[2i-1]$ and $g[2i]$ to y; (6)
    if $|y| = R$ then exit and return y; (7)
    reduce $m[2i-1]$ and $m[2i]$ by m; (8)
  endwhile; (5′)
endfor; (4′)
for $i = 2t_1 + 1$ to t_2 do (9)
  while $m[i] \geqslant 2m$ do (10)
    adjoin $2g[i]$ to y; (11)
    if $|y| = R$ then exit and return y; (12)
    reduce $m[i]$ by $2m$; (13)
  endwhile; (10′)
endfor; (9′)
return FALSE; (14)

end.

We leave the reader to check that this is correct. Note that t_1 could be 0, in which case the for-loop starting at line (4) is empty; similarly, t_2 could be $2t_1$, in which case the for-loop starting at line (9) would be empty. The cost of sorting x is $O(N \log N)$. The cost of re-sorting into inverse pairs costs at most $O(\log t)$ for each element (and hence at most $O(t \log t)$ overall), and so is at most $O(N \log N)$. Within each for-loop and while-loop, each step has small bounded cost. The total number of these steps is at most $\frac{1}{2}R$. Clearly, if $R > N$, then FALSE will be returned after at most N steps. Thus the cost of lines (3)–(14) is bounded above by $O(N)$, and hence the total cost is $O(N \log N)$.

Now the test we promised is as follows.

ALGORITHM 3.3. *Input*: a multiset a of size n from A, where $n = rs$ and $2 \leqslant r \leqslant s$.

Output: a self-inverse $r(r-1)$-multiset b^* and a self-inverse $s(s-1)$-multiset c^* such that $sb^* + rc^* \subseteq aa^{-1}$, or FALSE if no such multisets exist.

Cost: $O(n^2 \log n)$ time units.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Begin:
apply Algorithm 3.2 with $x := aa^{-1}$, $m := s$, $R := r(r-1)$; (1)
if the output is FALSE then (2)
exit and return FALSE; (3)
else (2′)
define $b^* := y$; (4)
apply Algorithm 3.2 with $x := aa^{-1} \setminus sb^*$, $m := r$, $R := s(s-1)$; (5)
if the output is FALSE then (6)
exit and return FALSE; (7)
else define $c^* := y$; (8)
endif; (6′)
endif; (2″)
return b^* and c^*; (9)
end.

Because of Lemma 3.1, it should be clear that this algorithm is correct. The cost is effectively that of using Algorithm 3.2 twice, with $N := n(n-1)$, and is therefore $O(n^2 \log n)$.

Although the above procedure will frequently be effective as a means of telling that a is *not* decomposable, it is far from capable of giving a positive answer. To do that, we seek to improve on Algorithm 1.2, which was discarded in Section 1 as being too slow. The idea is to examine aa^{-1} with some care in order to obtain information that can limit the search for the submultisets b' and c' of a that figured there. In $(b \otimes c)(b \otimes c)^{-1}$, the elements that obviously have multiplicity at least s are those of the form $b_i b_{i'}^{-1}$, occurring as $(b_i c_j)(b_{i'} c_j)^{-1}$ for $1 \leqslant j \leqslant s$. The numerators of these particular occurrences form the multiset $b_i c$. Similarly, the elements $c_j c_{j'}^{-1}$ occur with multiplicity at least r, and in suitable occurrences as quotients of members of $b \otimes c$ the numerators form the multiset bc_j. This leads to a strategy that is illustrated in the following algorithm, which works well if A has odd order or, more generally, if A has no involutions (elements of order 2), but which has limited value otherwise. It is based on the following definition—which, however, will need to be modified in Section 5.

DEFINITION 3.4. Provisionally (for this section only), the pair (b, c) in $A^{[r]} \times A^{[s]}$ will be said to be *recognisable* if the following conditions hold:

(1) there exists $g \in bb^{-1}$ such that $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$;

(2) there exists $h \in cc^{-1}$ such that $\mathrm{mult}(h;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$.

LEMMA 3.5. *Suppose that $a = b \otimes c$, that $2 \leqslant r \leqslant s$, and that the pair (b, c) is recognisable. Let g and h be elements of bb^{-1} and cc^{-1} that witness the recognisability of (b, c). Then:*

(1) $\mathrm{mult}(h;\ aa^{-1}) = r$ *and* $\mathrm{mult}(g;\ aa^{-1}) = s$;

(2) *if b' is the r-multiset of numerators of quotients in aa^{-1} equal to h, then* $\mathrm{mult}(g;\ b'b'^{-1}) = 1$;

(3) *if c' is the s-multiset of numerators of quotients in aa^{-1} equal to g, then $|b' \cap c'| = 1$, and if d is the unique element of $b' \cap c'$ then $a = (d^{-1})b' \otimes c'$.*

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Proof. The equation $aa^{-1} = s.bb^{-1} + r.cc^{-1} + bb^{-1}cc^{-1}$ yields that $\text{mult}(h;\ aa^{-1}) = r$ and $\text{mult}(g;\ aa^{-1}) = s$, as stated in (1).

Without loss of generality, $h = c_1c_2^{-1}$ and b', the r-multiset of numerators of quotients in aa^{-1} equal to h, is $\{a_1, \ldots, a_r\}$. After suitable adjustment of the indexing, $a_i = b_ic_1$ (and the denominators of the occurrences of h in aa^{-1} are the elements b_ic_2). Thus $b' = bc_1$. Now $b'b'^{-1} = bb^{-1}$ and so, since the pair (b, c) is recognisable, $\text{mult}(g;\ b'b'^{-1}) = 1$, as stated in (2).

We may suppose that $g = b_1b_2^{-1}$. The indexing of the elements a'_j of c' (which, recall, is the s-multiset $\{a'_1, \ldots, a'_s\}$ of numerators of the occurrences of g in aa^{-1}) may be adjusted so that $a'_j = b_1c_j$; that is, $c' = b_1c$. Clearly, $b_1c_1 \in b' \cap c'$, since $a_1 = a'_1 = b_1c_1$. If $|b' \cap c'| > 1$, then we would have $b_ic_1 = b_1c_j$ for some pair $(i, j) \neq (1, 1)$. Then we would have $g = b_1b_2^{-1} = b_ib_2^{-1}c_1c_j^{-1}$, contradicting the fact that g occurs only as $b_1b_2^{-1}$ in $bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}$. Therefore $b' \cap c' = \{b_1c_1\}$. Now $b_ic_j = (b_1c_1)^{-1}(b_ic_1)(b_1c_j)$, and so $a = (d^{-1})b' \otimes c'$, where d is the unique element of $b' \cap c'$. □

This lemma tells us that, by finding first b', then c', and then the unique element a_1 of $b' \cap c'$, we can compute a tensor factorisation $a = (a_1^{-1})b' \otimes c'$ equivalent to the original one. It leads to the following simple algorithm to discover a recognisable tensor factorisation if there is one.

ALGORITHM 3.6. *Input*: a multiset a of size n from A, where $n = rs$ and $2 \leqslant r \leqslant s$.

Output: either a tensor factorisation $a = b \otimes c$, in which $b \in A^{[r]}$ and $c \in A^{[s]}$, or FALSE if no such factorisation, in which the pair (b, c) is recognisable, exists.

Cost: $O(rn^3 \log n)$ time units.

Begin:

```
LIST-C := {h ∈ aa^{-1} | mult(h; aa^{-1}) = r};                          (1)
for h ∈ LIST-C do                                                       (2)
    b' := r-multiset of numerators of h as members of aa^{-1};          (3)
    LIST-B := {g ∈ b'b'^{-1} | mult(g; b'b'^{-1}) = 1, mult(g; aa^{-1}) = s};   (4)
    for g ∈ LIST-B do                                                   (5)
        c' := s-multiset of numerators of g as members of aa^{-1};      (6)
        if b' ∩ c' is a singleton {d} then                              (7)
            b := d^{-1}b';   c := c';                                   (8)
            if a = b ⊗ c, then exit and return (b, c);                  (9)
        endif [line (7)];
    endfor [line (5)];
endfor [line (2)];
return FALSE;                                                           (10)
end.
```

The cost of this algorithm may be estimated as follows. We may suppose that aa^{-1} has been sorted in a pre-processing phase of the computation (at a cost of $O(n^2 \log n)$). Since at most $n(n-1)/r$ elements of A can have multiplicity r in aa^{-1}, the for-loop at line (2), processing elements h of LIST-C, is traversed at most $n(n-1)/r$ times. For each element g of $b'b'^{-1}$, the cost of finding $\text{mult}(g;\ aa^{-1})$ is, by Theorem 2.3, at most $O(\log n(n-1))$, which is $O(\log n)$. Therefore line (4) costs at most $r(r-1)O(\log n)$ element-comparisons, and the algorithm could be implemented so that line (6) is done at the same time.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

By Theorem 2.2, line (9) costs $O(n \log n)$ comparisons. Other costs in the inner loop starting at line (5) are relatively small, and so the cost of one pass of the for-loop starting at line (2) is at most $O(r^2 n \log n)$ time units. Consequently, the overall cost is at most $O(rn^3 \log n)$ units. Since $r \leqslant n^{1/2}$, this is $o(n^4)$.

Our second criterion for acceptability was that the proportion of pairs $(b, c) \in A^{[r]} \times A^{[s]}$ for which the algorithm fails to give a factorisation (that is, the proportion of pairs that are not recognisable) should be small. For this, we assume that A is finite of odd order, and that $|A| \gg n^2$. A careful analysis will be deferred to Section 6; for the present, we offer a heuristic argument to illustrate the ideas. Consider for the moment pairs (b^*, c^*) of sequences (not multisets) in $A^r \times A^s$, in which the quotient $b_1 b_2^{-1}$ cannot serve as the element g of Definition 3.4(1). This will be because $b_1 b_2^{-1} = b_i b_{i'}^{-1}$ for some pair (i, i') other than $(1, 2)$, or because $b_1 b_2^{-1} = c_j c_{j'}^{-1}$ for some pair (j, j'), or because $b_1 b_2^{-1} = (b_i b_{i'}^{-1})(c_j c_{j'}^{-1})$ for some quadruple (i, i', j, j'). One of these conditions, namely that $b_1 b_2^{-1} = b_2 b_1^{-1}$, is anomalous in that it is of the form $b_1^2 = b_2^2$; but since A is assumed not to contain any involutions, this is the condition that $b_1 = b_2$. Each of the other possibilities can be written in the form $u = v$, where u is one of the elements b_k or c_l, and v is a product of others and their inverses. Therefore each condition is satisfied by precisely $|A|^{r+s-1}$ of the $|A|^{r+s}$ pairs of sequences. The number of possible conditions is $r(r-1)-2+s(s-1)+r(r-1)s(s-1)$, which is less than n^2. Thus the proportion of pairs (b^*, c^*) of sequences in $A^r \times A^s$ in which the quotient $b_1 b_2^{-1}$ cannot serve as the element g of Definition 3.4(1) is less than $n^2|A|^{-1}$. Similarly, the proportion of pairs (b^*, c^*) of sequences in which the quotient $c_1 c_2^{-1}$ cannot serve as the element h of Definition 3.4(2) is less than $n^2|A|^{-1}$. Consequently, the proportion of pairs of sequences that fail Definition 3.4 is certainly less than $2n^2|A|^{-1}$. Assuming that probabilities for multisets are the same as probabilities for sequences (which is not true, but is a good approximation), we would expect the probability that a randomly chosen pair (b, c) of multisets is not recognisable to be smaller than $2n^2|A|^{-1}$, which is small if $|A|$ is much larger than n^2.

There are two points to be made about this argument. One is that it deals with sequences rather than multisets: in Section 6, this point will be treated with proper care. Another is that it over-estimates the probability of non-recognisability. It estimates only the probability that $b_1 b_2^{-1}$ or $c_1 c_2^{-1}$ cannot serve for g and h respectively. If the conditions that $b_i b_{i'}^{-1}$ cannot serve for g were independent, then the probability that condition (1) of Definition 3.4 fails would be $(n^2|A|^{-1})^{r(r-1)}$. Those conditions are far from independent. Nevertheless, although this suggests that it is reasonable to expect the probability of non-recognisability to be considerably smaller than $2(n^2|A|^{-1})$, we do not propose to pursue this idea further.

4. *Tensor division*

In Section 1 we mentioned the tensor division problem: given multisets $a \in A^{[rs]}$ (our standing assumption that $r \leqslant s$ is temporarily suspended) and $b \in A^{[r]}$, how can we discover whether or not b tensor divides a and, if so, find $c \in A^{[s]}$ such that $a = b \otimes c$? In certain special circumstances, the matter may be decided by methods similar to those to be used in Section 5, as follows.

Observation 4.1. (1) *If $sbb^{-1} \not\subseteq aa^{-1}$, then b does not tensor divide a.*

(2) *If there exists $g \in bb^{-1}$ such that* $\mathrm{mult}(g; aa^{-1}) = s$, *then it can easily be decided whether or not b tensor divides a.*

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Proof. Clause (1) follows directly from the equation $(b \otimes c)(b \otimes c)^{-1} = sbb^{-1} + rcc^{-1} + bb^{-1}cc^{-1}$. For clause (2), suppose that $g \in bb^{-1}$ and $\mathrm{mult}(g;\ aa^{-1}) = s$. If $a = b \otimes c$, the only way this could happen is for g to be $b_i b_{i'}^{-1}$ and to occur as $(b_i c_j)(b_{i'} c_j)^{-1}$ (for $1 \leqslant j \leqslant s$) in aa^{-1}. Therefore, the only possible candidate for c is $b_i^{-1}c'$, where c' is the multiset of numerators of members of aa^{-1} equal to g. If, with this value of c, we find that $a = b \otimes c$, then we have the factorisation sought; otherwise b does not tensor divide a. □

We do not propose to pursue the general question further (see [6] for significant progress). There is, however, an efficient elementary approach for the special case where $r = 2$, and, since it may be used to solve the $(2, s)$-tensor factorisation problem, which escapes part of our analysis in Section 5 below, we treat it here.

Suppose then that we have $n = 2s$, $a \in A^{[n]}$, and $b = \{b_1,\ b_2\} \in A^{[2]}$. Define $g := b_2 b_1^{-1}$, so that $a = b \otimes c$ if and only if $a = b_1 c + g\,(b_1 c)$. Thus b tensor divides a if and only if there is an s-submultiset c' of a such that $a = c' + gc'$.To discover such a multiset c', if it exists, one may proceed as follows.

We assume (as always) that there is a computable linear ordering on A, and that the n-multiset a may be sorted at a cost of $O(n \log n)$ units and searched at a cost of $O(\log n)$ units. Suppose that a has been sorted into the form $m[1]a[1] + \cdots + m[t]a[t]$, where $a[1] < \cdots < a[t]$ in A and $m[\sigma] = \mathrm{mult}(a[\sigma];\ a)$.

The special case in which $g = 1$ (that is, $b_1 = b_2$) may be handled very quickly, since then b tensor divides a if and only if all multiplicities in a are even; moreover, if this condition is satisfied, then $a = b \otimes c$, where $c := \frac{1}{2}(b_1^{-1}a)$ in the sense that c has the same members as $b_1^{-1}a$ but with half their multiplicities. A very quick routine will handle this situation, and this is implemented in lines (3)–(6) of the algorithm below.

For the case where $g \neq 1$, we define a directed graph Γ on vertices $1, 2, \ldots, t$ by specifying that there is an edge $\sigma \longrightarrow \tau$ if and only if $ga[\sigma] = a[\tau]$. Thus Γ is a sort of Cayley graph describing the action (or partial action) of g on the set of distinct elements of a. It should be clear that every in-degree and every out-degree is 0 or 1. Therefore Γ is a disjoint union of directed paths and directed cycles, the components of the underlying undirected graph. The paths correspond to sequences of the form $(a[\sigma],\ ga[\sigma],\ g^2a[\sigma],\ \ldots,\ g^{l-1}a[\sigma])$, where $g^{-1}a[\sigma] \notin a$ and $g^l a[\sigma] \notin a$ (note that l will be 1 if σ is an isolated vertex), and the cycles correspond to sequences of the form $(a[\sigma],\ ga[\sigma],\ g^2a[\sigma],\ \ldots,\ g^{k-1}a[\sigma])$, where k is the order of g (note that $k = 1$ is now excluded).

Consider any vertex σ. Suppose for the moment that it is the initial vertex of a component which is a directed path. If c' exists, then it must contain $m[\sigma]a[\sigma]$; moreover, there must be a vertex τ with an edge $\sigma \longrightarrow \tau$, and we must have $m[\sigma] \leqslant m[\tau]$. Suppose next that σ is not the initial vertex of a component which is a directed path. Then there is an edge $\rho \longrightarrow \sigma$. If $m[\rho] < m[\sigma]$, and if c' exists, then it will have to contain $a[\sigma]$ with multiplicity at least $m[\sigma] - m[\rho]$.

These observations suggest a simple recursive procedure to discover whether a usable multiset c' exists, which works well unless all components are directed cycles and the multiplicities are constant over components. In this case, every component of Γ is of the form $\sigma_0 \longrightarrow \sigma_1 \longrightarrow \cdots \longrightarrow \sigma_{k-1} \longrightarrow \sigma_0$ (where, recall, $k = \mathrm{ord}(g)$ and $k > 1$). If k is even, then, taking Γ' to consist of alternate vertices from each component and $c' := \sum_{\sigma \in \Gamma'} m[\sigma]a[\sigma]$, we get a usable sub-multiset c' of a. If in each component the multiplicities $m[\sigma]$ are even, then, taking $c' := \sum_{\sigma \in \Gamma} \frac{1}{2}m[\sigma]a[\sigma]$, we get a usable sub-multiset c' of a. If, lastly, k is odd and there is a component in which the multiplicities are odd, then there cannot exist a usable multiset c', and so b does not tensor divide a.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

This analysis justifies the following algorithm. The graph Γ (which we shall identify with its vertex set) initially has vertex set $\{1, \dots, t\}$ and is described by arrays $\mathrm{in}[1\,..\,t]$ and $\mathrm{out}[1\,..\,t]$. These are arrays of length t. The σ-entries are $\mathrm{in}[\sigma]$ and $\mathrm{out}[\sigma]$ respectively, and the equations $\mathrm{in}[\sigma] = \rho$ and $\mathrm{out}[\rho] = \sigma$ hold if and only if there is an edge $\rho \longrightarrow \sigma$. Subsets Γ_0 and Γ_1 of Γ are maintained as part of the data structure, to record the following information, which would be costly to recalculate as the recursion progresses.

- Γ_0 is the set of initial vertices of components of Γ that are directed paths (possibly isolated vertices).
- Γ_1 is the set of vertices σ for which there is an edge $\rho \longrightarrow \sigma$ with $m[\rho] < m[\sigma]$ (where, as above, $m[\sigma]$ is the multiplicity associated with the vertex σ).

For the reader's convenience, we make the following points about our presentation of the algorithm.

- Lines (1), (2), and (7)–(12) initialise the data structure.
- Lines (17)–(24), (27), (28), (30), (31), (34)–(36), (38) and (39) update it as the calculation proceeds.
- Lines (3)–(6) deal with the case where $g = 1$.
- Each pass of the while-loop at lines (13)–(39) focuses on one vertex σ, as follows.
  - If $\Gamma_0 \neq \emptyset$, then σ is chosen from Γ_0, lines (15)–(24) operate, and the next pass of the while-loop starts after line (20″).
  - If $\Gamma_0 = \emptyset$ (so that all components of Γ are directed k-cycles where $k = \mathrm{ord}(g)$) but $\Gamma_1 \neq \emptyset$, then σ is chosen from Γ_1, lines (26)–(31) operate, and the next pass of the while-loop starts after line (31).
  - If $\Gamma_0 = \Gamma_1 = \emptyset$, then lines (32)–(39) operate.
- Note that at line (26) we have $\tau \in \Gamma$ and $\upsilon \in \Gamma$ because the component of Γ containing σ is a cycle.
- Note that at line (36) the cycle containing σ is broken and replaced by a directed path of length $k - 2$ with initial vertex τ; thus for the next $\frac{1}{2}k - 1$ passes of the while-loop, Γ_0 will be non-empty and the remainder of the cycle, which originally contained σ, will be processed to extinction (while c' duly grows, of course).
- Similarly, at line (39) the set Γ_1 becomes non-empty (while Γ_0 remains empty), and so at the next few ($k - 1$, in fact) passes of the while-loop, the cycle that contained σ will be processed to extinction by lines (26)–(31).

ALGORITHM 4.2. *Input*: multisets $a \in A^{[2s]}$ and $b = \{b_1, b_2\} \in A^{[2]}$.

Output: either $c \in A^{[s]}$ such that $a = b \otimes c$, or FALSE if no such factorisation exists.

Cost: $O(n \log n)$ time units.

Begin:

sort a as $m[1]a[1] + \cdots + m[t]a[t]$,
  where $a[1] < \cdots < a[t]$ in A and $m[\sigma] = \mathrm{mult}(a[\sigma];\ a)$; (1)
define $g := b_2 b_1^{-1}$ and initialise c' as $\emptyset$; (2)

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

if $g = 1$ then (3)
  for $\sigma = 1$ to t do (4)
    if $m[\sigma]$ is even then $c' := c' + \frac{1}{2}m[\sigma]a[\sigma]$ else exit and return FALSE; (5)
  endfor; (4′)
  define $c := b_1^{-1}c'$, exit and return c; (6)
endif; (3′)

[Comment: from now on, $g \neq 1$.]

initialise Γ and Γ_0 as $\{1, 2, \ldots, t\}$; initialise Γ_1 as $\emptyset$; (7)
initialise $\text{in}[\rho]$, $\text{out}[\rho]$ to be 0 for all $\rho \in \Gamma$; (8)
for $\sigma = 1$ to t do (9)
  if $(\exists \rho \in \Gamma)(g\,a[\rho] = a[\sigma])$ then (10)
    $\text{in}[\sigma] := \rho$; $\text{out}[\rho] := \sigma$; (11)
    remove σ from Γ_0; if $m[\rho] < m[\sigma]$ then adjoin σ to Γ_1; (12)
  endif; (10′)
endfor; (9′)
while $\Gamma \neq \emptyset$ do (13)
  if $\Gamma_0 \neq \emptyset$ then (14)
    select σ from Γ_0 and define $\tau := \text{out}[\sigma]$; (15)
    if $\tau = 0$ or $m[\tau] < m[\sigma]$ then exit and return FALSE; (16)
    $c' := c' + m[\sigma]a[\sigma]$; (17)
    remove σ from Γ and from Γ_0 and reset $\text{in}[\tau] := 0$; (18)
    reset $m[\tau] := m[\tau] - m[\sigma]$ and set $\upsilon := \text{out}[\tau]$; (19)
    if $m[\tau] = 0$ then (20)
      remove τ from Γ; (21)
      if $\upsilon \neq 0$ then adjoin υ to Γ_0; (22)
    else (20′)
      adjoin τ to Γ_0; (23)
      if $\upsilon \neq 0$ and $m[\tau] < m[\upsilon]$ then adjoin υ to Γ_1; (24)
    endif; (20″)
  else (14′)

  [Comment: now all components of Γ are k-cycles, where $k = \text{ord}(g) > 1$.]

    if $\Gamma_1 \neq \emptyset$ then (25)
      select σ from Γ_1 and define $\rho := \text{in}[\sigma]$, $\tau := \text{out}[\sigma]$, $\upsilon := \text{out}[\tau]$; (26)
      $c' := c' + (m[\sigma] - m[\rho])a[\sigma]$; (27)
      $m[\tau] := m[\tau] - (m[\sigma] - m[\rho])$; $m[\sigma] := m[\rho]$; (28)
      if $m[\tau] < 0$ then exit and return FALSE; (29)
      if $m[\tau] = 0$ then remove τ from Γ and adjoin υ to Γ_0; (30)
      if $m[\tau] < m[\upsilon]$ then adjoin υ to Γ_1; (31)
    else (25′)

    [Comment: now multiplicities are constant over components of Γ.]

      select σ from Γ and define $\rho := \text{in}[\sigma]$, $\tau := \text{out}[\sigma]$; (32)
      if $\text{ord}(g)$ is even then (33)
        $c' := c' + m[\rho]a[\rho]$; (34)
        remove ρ and σ from Γ; (35)
        if $\tau \neq \rho$ (that is, if $g^2 \neq 1$), then adjoin τ to Γ_0,
                and reset $\text{in}[\tau] := 0$ and $\text{out}[\text{in}[\rho]] := 0$; (36)

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

else (33′)
if $m[\sigma]$ is odd then exit and return FALSE; (37)
$c' := c' + \frac{1}{2}m[\rho]a[\rho]$; (38)
$m[\rho] := \frac{1}{2}m[\rho]$; $m[\sigma] := \frac{1}{2}m[\sigma]$; adjoin τ to Γ_1; (39)
endif; (33″)
endif; (25′)
endif; (14′)
endwhile; (13′)
set $c := b_1^{-1}c'$ and return c; (40)
end.

This algorithm may be costed as follows. Sorting a in line (1) costs $O(n \log n)$ units. Lines (3)–(6) cost $O(t)$, hence $O(n)$ units. Searching for ρ in line (10) costs $O(\log n)$ units, and so, since $t \leqslant n$, the cost of setting up the data structure describing the graph Γ is $O(n \log n)$ units. The point of using the arrays in and out, and the sets Γ_0 and Γ_1, is to ensure that each pass of the while-loop starting at line (13) costs a constant amount. At each pass of the loop, either the output FALSE is returned and the computation stops, or the value of $\sum_{\sigma \in \Gamma} m[\sigma]$ is reduced by an even positive integer. Therefore the number of passes of the loop is at most s, and so the cost from line (13) on is at most $O(n)$. Thus overall the computation costs $O(n \log n)$.

Now the promised $(2, s)$-factorisation algorithm concludes this section.

ALGORITHM 4.3. *Input*: a multiset $a \in A^{[n]}$, where $n = 2s$.

Output: either $b \in A^{[2]}$ and $c \in A^{[s]}$ such that $a = b \otimes c$, or FALSE if no such factorisation exists.

Cost: $O(n^2 \log n)$ time units.

Begin:
for $\lambda = 2$ to n do (1)
$b := \{a_1, a_\lambda\}$; (2)
if Algorithm 4.2 returns c, then exit and return the pair (b, c); (3)
endfor [line (1)];
return FALSE; (4)
end.

Clearly, the cost of this factorisation procedure is at most n times the cost of Algorithm 4.2, and so it is at most $O(n^2 \log n)$ time units.

5. *The main factorisation algorithm*

Algorithm 3.6 does what we want in many cases. It is useless, however, if A has many elements of order 2. If, for example, A is an elementary abelian 2-group, then—as is easy to see—there are no recognisable pairs in the sense of Definition 3.4. Therefore the notion of recognisability needs to be improved.

DEFINITION 5.1. The pair (b, c) in $A^{[r]} \times A^{[s]}$ will be said to be *recognisable* if the following conditions hold:

(1) there exists $g \in bb^{-1}$ such that either $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ or $g^2 = 1$ and $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$;

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

(2) there exists $h \in cc^{-1}$ such that either $\mathrm{mult}(h;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ or $h^2 = 1$ and $\mathrm{mult}(h;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$.

For later use, and to begin to give some substance to the concept of recognisability, we prove the following lemma.

LEMMA 5.2. *Suppose that the pair (b, c) in $A^{[r]} \times A^{[s]}$ is recognisable, as witnessed by elements $g \in bb^{-1}$ and $h \in cc^{-1}$. Then*

(1) *each of b and c is multiplicity-free; in particular, $g \neq 1$ and $h \neq 1$;*

(2) *if $g = b_i b_{i'}^{-1}$ and $h = c_j c_{j'}^{-1}$, then $b_i c \cap bc_l = \{b_i c_l\}$ for any l in the relevant range; similarly, $b_{i'} c \cap bc_l = \{b_{i'} c_l\}$, $b_k c \cap bc_j = \{b_k c_j\}$, and $b_k c \cap bc_{j'} = \{b_k c_{j'}\}$ for any k and l in the relevant range.*

Proof. Suppose that one of b and c is not multiplicity-free. Without loss of generality, we may suppose that $b_1 = b_2$. Then $c_j c_l^{-1} = (b_1 b_2^{-1})(c_j c_l^{-1}) \in bb^{-1}cc^{-1}$. This is true for all j and l, and so requirement (2) of Definition 5.1 fails. Thus the pair (b, c) is not recognisable. This proves clause (1).

Suppose that $b_i c_{l'} \in bc_l$. Then $b_i c_{l'} = b_k c_l$ for some k. This leads to the equation $g = b_i b_{i'}^{-1} = (b_k b_{i'}^{-1})(c_l c_{l'}^{-1})$, which, if $l' \neq l$, contradicts condition (1) of Definition 5.1. Thus $b_i c \cap bc_l = \{b_i c_l\}$. The other parts of clause (2) are proved in the same way. □

The definition of recognisability is, in effect, a disjunction of four different possibilities. Let $a^* := bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}$. The four cases are:

B1C1: $(\exists g \in bb^{-1})(\exists h \in cc^{-1})(\mathrm{mult}(g;\ a^*) = \mathrm{mult}(h;\ a^*) = 1)$;

B1C2: $(\exists g \in bb^{-1})(\exists h \in cc^{-1})(\mathrm{mult}(g;\ a^*) = 1,\ h^2 = 1,\ \mathrm{mult}(h;\ a^*) = 2)$;

B2C1: $(\exists g \in bb^{-1})(\exists h \in cc^{-1})(g^2 = 1,\ \mathrm{mult}(g;\ a^*) = 2,\ \mathrm{mult}(h;\ a^*) = 1)$;

B2C2: $(\exists g \in bb^{-1})(\exists h \in cc^{-1})(g^2 = 1,\ \mathrm{mult}(g;\ a^*) = 2,\ h^2 = 1,\ \mathrm{mult}(h;\ a^*) = 2)$.

Although they have points in common, these cases differ sufficiently that we find it best to treat them separately. The following simple fact is basic for most of the discussion.

LEMMA 5.3. *In case* BβCγ, $\mathrm{mult}(g;\ aa^{-1}) = \beta s$ *and* $\mathrm{mult}(h;\ aa^{-1}) = \gamma r$.

These equations follow from the fact that $aa^{-1} = sbb^{-1} + rcc^{-1} + bb^{-1}cc^{-1}$, while in the case B$\beta$C$\gamma$,

$$\mathrm{mult}(g;\ bb^{-1}) = \beta; \quad \mathrm{mult}(g;\ cc^{-1}) = 0; \quad \mathrm{mult}(g; bb^{-1}cc^{-1}) = 0;$$
$$\mathrm{mult}(h;\ bb^{-1}) = 0; \quad \mathrm{mult}(h;\ cc^{-1}) = \gamma; \quad \mathrm{mult}(h; bb^{-1}cc^{-1}) = 0.$$

LEMMA 5.4. *Suppose that $a = b \otimes c$, that $2 \leqslant r \leqslant s$, and that the pair (b, c) is* B1C1*-recognisable. Let g and h be elements of bb^{-1} and cc^{-1} that witness condition* B1C1*, let b' be the r-multiset of numerators of quotients in aa^{-1} equal to h, and let c' be the s-multiset of numerators of quotients in aa^{-1} equal to g. Then:*

(1) $\mathrm{mult}(g;\ b'b'^{-1}) = 1$;

(2) *$b' \cap c'$ is a singleton $\{d\}$ and $a = (d^{-1})b' \otimes c'$.*

The proof is exactly the same as that of Lemma 3.5, and is therefore omitted here. For the remaining cases, we require that $r \geqslant 3$, but by Algorithm 4.3, we lose nothing thereby.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

LEMMA 5.5. *Suppose that $a = b \otimes c$, that $3 \leqslant r \leqslant s$, and that the pair (b, c) is* B2C1*-recognisable. Let g and h be elements of bb^{-1} and cc^{-1} that witness condition* B2C1*, let b' be the r-multiset of numerators of quotients in aa^{-1} equal to h, and let c'' be the $2s$-multiset of numerators of quotients in aa^{-1} equal to g. Then:*

(1) $\mathrm{mult}(g;\ b'b'^{-1}) = 2$;

(2) $|b' \cap c''| = 2$;

(3) *if $y \in b' \setminus c''$, $z \in b' \cap c''$ and $c' := yz^{-1}c'' \cap a$, then c' is an s-multiset, $b' \cap c'$ is a singleton $\{d\}$ and $a = (d^{-1})b' \otimes c'$.*

Proof. Without loss of generality, $g = b_1 b_2^{-1} = b_2 b_1^{-1}$ and $h = c_1 c_2^{-1}$. Then the quotients in aa^{-1} equal to h are $(b_i c_1)(b_i c_2)^{-1}$ and $b' = bc_1$. Therefore $b'b'^{-1} = bb^{-1}$, and so $\mathrm{mult}(g;\ b'b'^{-1}) = 2$, as stated in clause (1).

The quotients in aa^{-1} equal to g are $(b_1 c_j)(b_2 c_j)^{-1}$ and $(b_2 c_j)(b_1 c_j)^{-1}$, and so $c'' = b_1 c + b_2 c$. Therefore $b' \cap c'' \subseteq (bc_1 \cap b_1 c) + (bc_1 \cap b_2 c)$. Since $bc_1 \cap b_1 c = \{b_1 c_1\}$ and $bc_1 \cap b_2 c = \{b_2 c_1\}$ by Lemma 5.2(2), we have that $b' \cap c'' = \{b_1 c_1,\ b_2 c_1\}$. In particular, $|b' \cap c''| = 2$ as stated in clause (2).

Now let $y \in b' \setminus c''$ and $z \in b' \cap c''$. Without loss of generality, $y = b_3 c_1$ and $z = b_1 c_1$. Then $(yz^{-1})c'' = (b_3 b_1^{-1})c'' = b_3 c + (b_3 g)c$. Suppose that $b_3 g c_j \in a$, say $b_3 g c_j = b_k c_l$ for some k and l. Then $g = (b_k b_3^{-1})(c_l c_j^{-1})$, in contradiction to the assumption that $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$. Therefore, if $c' := (yz^{-1})c'' \cap a$, then $c' = b_3 c$. It now follows from Lemma 5.2(2) that $b' \cap c' = \{b_3 c_1\}$, and so $a = d^{-1}b' \otimes c'$, where $d := b_3 c_1$; that is, d is the unique element of $b' \cap c'$. This completes the proof of the lemma. □

LEMMA 5.6. *Suppose that $a = b \otimes c$, that $3 \leqslant r \leqslant s$, and that the pair (b, c) is* B1C2*-recognisable. Let g and h be elements of bb^{-1} and cc^{-1} that witness condition* B1C2*, let b'' be the $2r$-multiset of numerators of quotients in aa^{-1} equal to h, and let c' be the s-multiset of numerators of quotients in aa^{-1} equal to g. Then:*

(1) $\mathrm{mult}(g;\ b''b''^{-1}) = 2$;

(2) $|b'' \cap c'| = 2$;

(3) *if $y \in c' \setminus b''$, $z \in b'' \cap c'$ and $b' := yz^{-1}b'' \cap a$, then b' is an r-multiset, $b' \cap c'$ is a singleton $\{d\}$ and $a = (d^{-1})b' \otimes c'$.*

Notice that this is not quite the same as the previous lemma with the roles of b, g and c, h reversed. The reason is that for computational purposes deriving from the fact that r is the smaller of r and s, it seems better to seek h before g. Nevertheless, the proof is similar (although here $b''b''^{-1}$ is the $2r(2r-1)$-multiset $2bb^{-1} + 2bb^{-1}h + 2rh$), and will therefore be omitted.

LEMMA 5.7. *Suppose that $a = b \otimes c$, that $3 \leqslant r \leqslant s$, and that the pair (b, c) is* B2C2*-recognisable. Let g and h be elements of bb^{-1} and cc^{-1} that witness condition* B2C2*, let b'' be the $2r$-multiset of numerators of quotients in aa^{-1} equal to h, and let c'' be the $2s$-multiset of numerators of quotients in aa^{-1} equal to g. Then:*

(1) $\mathrm{mult}(g;\ b''b''^{-1}) = 4$;

(2) $|b'' \cap c''| = 4$;

(3) *if $y_1 \in b'' \setminus c''$ and $y_2 \in c'' \setminus b''$, then $\exists z \in b'' \cap c''$ such that if $b' := (y_2 z^{-1})b'' \cap a$ and $c' := (y_1 z^{-1})c'' \cap a)$, then $b' \cap c'$ is a singleton $\{d\}$ and $a = (d^{-1}b') \otimes c'$.*

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Proof. We may suppose that $g = b_1 b_2^{-1} = b_2 b_1^{-1}$ and $h = c_1 c_2^{-1} = c_2 c_1^{-1}$, and then

$$b'' = bc_1 + bc_2 = b \otimes \{c_1,\ c_2\}, \qquad c'' = b_1 c + b_2 c = \{b_1,\ b_2\} \otimes c.$$

Now $b''b''^{-1} = 2bb^{-1} + 2bb^{-1}h$ and $\mathrm{mult}(g;\ bb^{-1}h) = 0$ since $bb^{-1}h \subseteq bb^{-1}cc^{-1}$. Therefore $\mathrm{mult}(g;\ b''b''^{-1}) = 2\ \mathrm{mult}(g;\ bb^{-1}) = 4$.

Clearly, $\{b_1c_1,\ b_2c_1,\ b_1c_2,\ b_2c_2\} \subseteq b'' \cap c''$. The fact that no elements of the form b_ic_j with $i \geqslant 3$ and $j \in \{1,\ 2\}$ lie in c'' follows easily from clause (2) of Lemma 5.2. Thus $b'' \cap c'' = \{b_1c_1,\ b_2c_1,\ b_1c_2,\ b_2c_2\}$ and, in particular, $|b'' \cap c''| = 4$.

Let $y_1 \in b'' \setminus c''$ and $y_2 \in c'' \setminus b''$. Without loss of generality, $y_1 = b_3c_1$ and $y_2 = b_1c_3$. Take $z := b_1c_1$. Then

$$\big(y_1 z^{-1}\big)c'' \cap a = \big(b_3 b_1^{-1}\big)c'' \cap a = \big(b_3 c + (b_3 g)c\big) \cap a.$$

Now $b_3c \subseteq a$, and we claim that $(b_3g)c \cap a = \emptyset$. For suppose that $b_3gc_j = b_ic_l$ for some i, j, l in the relevant range. Then $g = (b_ib_3^{-1})(c_lc_j^{-1})$, and this contradicts the fact that g is non-trivial (see Lemma 5.2) and occurs only as $b_1b_2^{-1}$ and $b_2b_1^{-1}$ in $bb^{-1}+cc^{-1}+bb^{-1}cc^{-1}$. Thus if $c' := (y_1z^{-1})c'' \cap a$, then $c' = b_3c$. Similarly, if $b' := (y_2z^{-1})b'' \cap a$, then $b' = bc_3$. Therefore $b' \cap c' = \{b_3c_3\}$ by Lemma 5.2(2), and $a = (d^{-1}b') \otimes c'$, where d is the unique element b_3c_3 of $b' \cap c'$. □

The four lemmas justify the following algorithm for tensor factorisation. To make the algorithm more readable, we have extracted four segments of it and called them procedures, one to deal with each of the forms of recognisability described above. Those procedures will be described below.

ALGORITHM 5.8. *Input*: a multiset a of size n from A, where $n = rs$, and $3 \leqslant r \leqslant s$.

Output: either a tensor factorisation $a = b \otimes c$, in which $b \in A^{[r]}$, $c \in A^{[s]}$, or FALSE if no such factorisation in which the pair (b, c) is recognisable, exists.

Cost: $O(n^3 \log n)$ time units.

Begin:

sort aa^{-1} as $m[1]g[1] + \cdots + m[t]g[t]$,
where $g[1] < \cdots < g[t]$ in A and $m[\sigma] = \mathrm{mult}(g[\sigma];\ aa^{-1})$; (1)

create an array LIST$[1\,..\,t]$ in which LIST$[\sigma]$ is a list of the pairs (v, v')
such that $g[\sigma] = a_v a_{v'}^{-1}$; (2)

LIST-C1 $:= \{h \in aa^{-1} \mid h^2 \neq 1,\ \mathrm{mult}(h;\ aa^{-1}) = r\}$; (3)

LIST-C2 $:= \{h \in aa^{-1} \mid h^2 = 1,\ \mathrm{mult}(h;\ aa^{-1}) = 2r\}$; (4)

for $h \in$ LIST-C1 do (5)

$b' :=$ r-multiset of numerators of members of aa^{-1} equal to h; (6)

sort $b'b'^{-1}$; (7)

LIST-B1 $:= \{g \in b'(b')^{-1} \mid \mathrm{mult}(g;\ b'b'^{-1}) = 1,\ \mathrm{mult}(g;\ aa^{-1}) = s\}$; (8)

LIST-B2 $:= \{g \in b'(b')^{-1} \mid g^2 = 1,$
$\mathrm{mult}(g;\ b'b'^{-1}) = 2,\ \mathrm{mult}(g;\ aa^{-1}) = 2s\}$; (9)

for $g \in$ LIST-B1 do Procedure B1C1; (10)

for $g \in$ LIST-B2 do Procedure B2C1; (11)

endfor [line (5)];

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

for $h \in$ LIST-C2 do (12)
  $b'' := 2r$-multiset of numerators of members of aa^{-1} equal to h; (13)
  sort $b''b''^{-1}$; (14)
  LIST-B1 := $\{g \in b''(b'')^{-1} \mid g^2 \neq 1,$
    $\text{mult}(g;\ b''b''^{-1}) = 2\,,\ \text{mult}(g;\ aa^{-1}) = s\}$; (15)
  LIST-B2 := $\{g \in b''(b'')^{-1} \mid g^2 = 1,$
    $\text{mult}(g;\ b''b''^{-1}) = 4\,,\ \text{mult}(g;\ aa^{-1}) = 2s\}$; (16)
  for $g \in$ LIST-B1 do Procedure B1C2; (17)
  for $g \in$ LIST-B2 do Procedure B2C2; (18)
endfor [line (12)];
return FALSE; (19)

end.

Since $|aa^{-1}| = n(n-1)$, the cost of line (1) is $O(n^2 \log n)$ (see the discussion preceding Theorem 2.2). Line (2) can be implemented by running through the collection of pairs (ν, ν') (where, recall, the convention is that $1 \leqslant \nu \leqslant n$, $1 \leqslant \nu' \leqslant n$, and $\nu \neq \nu'$), searching the sorted version of aa^{-1} for the appropriate element $g[\sigma]$ and appending (ν, ν') to LIST$[\sigma]$. Since searching by list-bisection costs $O(\log t)$ element-comparisons (where, as in line (1), t is the number of distinct elements of aa^{-1}, so that $t < n^2$), the cost of line (2) is $O(n^2 \log n)$ time units. The cost of lines (3) and (4) together is obviously bounded by t, and hence is $O(n^2)$.

Lines (6) and (7) together cost $O(r^2 \log r)$ units, and, by Theorem 2.3, lines (7) and (8) cost $O(r^2 \log n)$ units. Since $|\text{LIST-B1}| < r^2$ and $|\text{LIST-B2}| < \frac{1}{2}r^2$, lines (10) and (11) cost less than r^2cost(B1C1) and $\frac{1}{2}r^2$cost(B2C1) respectively, where cost(BβCγ) is the number of time units required for Procedure BβCγ. Therefore one pass of the for-loop starting at line (5) costs at most

$$O(r^2(\log n + \text{cost(B1C1)} + \text{cost(B2C1)})$$

time units. Similarly, one pass of the for-loop starting at line (12) costs at most

$$O(r^2(\log n + \text{cost(B1C2)} + \text{cost(B2C2)})$$

units. Since $|\text{LIST-C1}| + |\text{LIST-C2}| < n^2/r$, the total cost of the algorithm is at most $O(rn^2(k + \log n))$, where k is the greatest of the costs of running Procedures BβCγ. It will emerge below that k is $O(n \log n)$. Therefore the algorithm has a running cost at most $O(rn^3 \log n)$.

The four procedures called in the course of Algorithm 5.8 are as follows.

PROCEDURE B1C1. *Data*: the parameters n, r and s and the n-multiset a from the main algorithm; an element $h \in aa^{-1}$ such that $\text{mult}(h;\ aa^{-1}) = r$; the r-multiset b' of numerators of members of aa^{-1} equal to h; an element $g \in b'(b')^{-1}$ such that $\text{mult}(g;\ b'b'^{-1}) = 1$ and $\text{mult}(g;\ aa^{-1}) = s$.

Cost: $O(n \log n)$ time units.

Begin:
  $c' := s$-multiset of numerators of members of aa^{-1} equal to g; (1)
  if $b' \cap c'$ is a singleton $\{d\}$ then (2)
    $b := d^{-1}b'$; $\quad c := c'$; (3)
    if $a = b \otimes c$ then exit and return (b, c); (4)
  endif [line (2)];
end.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

PROCEDURE B2C1. *Data*: the parameters n, r and s and the n-multiset a from the main algorithm; an element $h \in aa^{-1}$ such that $\mathrm{mult}(h;\ aa^{-1}) = r$; the r-multiset b' of numerators of members of aa^{-1} equal to h; an element $g \in b'b'^{-1}$ such that $g \neq 1$, $g^2 = 1$, $\mathrm{mult}(g;\ b'b'^{-1}) = 1$ and $\mathrm{mult}(g;\ aa^{-1}) = 2s$.

Cost: $O(n \log n)$ time units.

Begin:
$c'' :=$ $2s$-multiset of numerators of members of aa^{-1} equal to g; (1)
if $|b' \cap c''| = 2$ then (2)
select y from $b' \setminus c''$ and z from $b' \cap c''$; (3)
$c' := (yz^{-1})c'' \cap a$; (4)
if $|c'| = s$ and $b' \cap c'$ is a singleton $\{d\}$ then (5)
$b := d^{-1}b'$; $c := c'$; (6)
if $a = b \otimes c$ then exit and return (b, c); (7)
endif [line (5)];
endif [line (2)];
end.

PROCEDURE B1C2. *Data*: the parameters n, r and s and the n-multiset a from the main algorithm; an element $h \in aa^{-1}$ such that $h^2 = 1$ and $\mathrm{mult}(h;\ aa^{-1}) = 2r$; the $2r$-multiset b'' of numerators of members of aa^{-1} equal to h; an element $g \in b''b''^{-1}$ such that $\mathrm{mult}(g;\ b''b''^{-1}) = 2$, $\mathrm{mult}(g;\ aa^{-1}) = s$, and $g \neq 1$.

Cost: $O(n \log n)$ time units.

Begin:
$c' :=$ s-multiset of numerators of members of aa^{-1} equal to g; (1)
if $|b'' \cap c'| = 2$ then (2)
select y from $c' \setminus b''$ and z from $b'' \cap c'$; (3)
$b' := (yz^{-1}b'') \cap a$; (4)
if $|b'| = r$ and $b' \cap c'$ is a singleton $\{d\}$ then (5)
$b := d^{-1}b'$; $c := c'$; (6)
if $a = b \otimes c$ then exit and return (b, c); (7)
endif [line (5)];
endif [line (2)];
end.

PROCEDURE B2C2. *Data*: the parameters n, r and s and the n-multiset a from the main algorithm; an element $h \in aa^{-1}$ such that $\mathrm{mult}(h;\ aa^{-1}) = 2r$ and $h^2 = 1$; the $2r$-multiset b'' of numerators of members of aa^{-1} equal to h; an element g from $b''b''^{-1}$ such that $\mathrm{mult}(g;\ b''b''^{-1}) = 4$, $\mathrm{mult}(g;\ aa^{-1}) = 2s$, and $g^2 = 1$.

Cost: $O(n \log n)$ time units.

Begin:
$c'' :=$ $2s$-multiset of numerators of members of aa^{-1} equal to g; (1)
if $|b'' \cap c''| = 4$ then (2)
select y_1 from $b'' \setminus c''$ and y_2 from $c'' \setminus b''$; (3)

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

```
    for z ∈ b'' ∩ c'' do                                                    (4)
      b' := (y2 z^{-1}) b'' ∩ a;   c' := (y1 z^{-1}) c'' ∩ a;               (5)
      if |b'| = r and |c'| = s and b' ∩ c' is a singleton {d} then           (6)
        b := d^{-1} b';   c := c';                                          (7)
        if a = b ⊗ c then exit and return (b, c);                           (9)
      endif [line (6)];
    endfor [line (4)];
  endif [line (2)];
end.
```

Each of the procedures is justified by the corresponding lemma. If there really is a tensor factorisation $a = b \otimes c$ in which the pair (b, c) is recognisable, then the search for possible witnesses g and h must succeed at some point, and the computation will certainly yield some factorisation; that factorisation might perhaps not be the original one, however.

The four procedures may all be costed in essentially the same way, and so we treat only Procedure B1C1. The point of line (2) of the main algorithm is to ensure that finding the numerators of members of aa^{-1} equal to g costs very little—the cost of line (1) is $O(s)$ units. Line (2) can obviously be done at a cost of at most rs (that is, n) comparisons; line (3) costs very little; and by Theorem 2.2, the cost of line (4) is $O(n \log n)$ comparisons. Therefore the procedure costs at most $O(n \log n)$ time units.

6. *The frequency of recognisable factorisations of multisets*

The algorithm that we have produced in Section 5 is a deterministic solution to a special case of the combinatorial factorisation problem. In practice (that is to say, in applications to matrix groups), we propose to use it as the basis of a Monte Carlo algorithm by interpreting the output FALSE as indicating that the input a has no tensor factorisation (with factors of sizes r and s). This interpretation will sometimes be incorrect. The algorithm would be of no interest if it gave the wrong answer too often, and so in this section we shall prove that wrong answers are rare.

This needs careful interpretation. First, there must be an acceptable concept of what it means to be rare. As far as that goes, in this section we assume that A is finite, but that $|A|$ is large compared with n^2. We use the uniform probability distribution on $A^{[n]}$ (and on $A^{[r]}$, $A^{[s]}$ and $A^{[r]} \times A^{[s]}$), and we interpret 'rare' as 'occurring with low probability'. Unfortunately, the uniform distribution is not particularly convenient for calculations. The distribution induced from the uniform distribution on A^n by the natural map $\phi : A^n \longrightarrow A^{[n]}$ would be easier to use, but does not seem as well adapted to applications. The main test of the acceptability of a definition of rarity should be that it correlates well with rarity in practice. But 'rarity in practice' is hard to define. All we can offer in defence of our choice of definition is that it seems to be appropriate for the applications that we have in mind, and that it is a reasonably natural one.

Secondly, we need to be clear about the context in which we accept that wrong answers are rare. The operation $\otimes$ is a map $A^{[r]} \times A^{[s]} \longrightarrow A^{[n]}$. Elements of $A^{[n]}$ that lie in the image will be said to be *tensor decomposable* or sometimes (r, s)*-decomposable*. Our algorithm certainly gives a correct response whenever the input a is not (r, s)-decomposable, and there is an intuitive argument that indicates that decomposable n-multisets are rare. Fix on a bijection $\{1, \ldots, r\} \times \{1, \ldots, s\} \longrightarrow \{1, \ldots, n\}$, and use this to define a tensor product map $\otimes : A^r \times A^s \longrightarrow A^n$ in the natural way. Since in this context (of sequences, rather than

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

multisets), $x \otimes y = x' \otimes y'$ if and only if $x_i y_j = x'_i y'_j$ for all i, j, one has that $x \otimes y = x' \otimes y'$ if and only if there exists $g \in A$ such that $x' = gx$ and $y' = g^{-1}y$, and so this version of $\otimes$ is a $|A|$-to-one map. On 'most' of A^n, the natural map $\phi : A^n \longrightarrow A^{[n]}$ is $(n!)$-to-one. This suggests that $|A^{[n]}|$ is approximately $|A|^n/n!$, that the number of decomposable n-multisets is approximately $|A|^{r+s-1}/n!$, and therefore that the probability that a randomly chosen n-multiset is (r, s)-decomposable should be approximately $|A|^{-(n-r-s+1)}$, that is, $|A|^{-(r-1)(s-1)}$, which is very small indeed. Since it is only for decomposable inputs that our algorithm could give a wrong answer, if input multisets a are chosen at random from $A^{[n]}$, then the probability of failure is extremely low.

In practice, however, the context is a more limited one. Often we know (or believe) that a is (r, s)-decomposable and want to use the algorithm as a method for finding a tensor factorisation. Therefore our interest becomes focused on rather different probability measures, namely versions of the probability of failure, given that a is decomposable. One natural possibility would be the conditional probability that the algorithm yields a wrong answer, given that the input n-multiset is (r, s)-decomposable. Another is the probability that a pair (x, y) chosen randomly from $A^{[r]} \times A^{[s]}$ should not be recognisable. Although these two measures of probability are likely to differ very little, that seems hard to prove, and we have had to choose between them. We shall work with the latter. There are two reasons for our choice. One is pragmatic—we have found Prob[Non-Recog] (defined explicitly below) far easier to work with than the conditional probability. The other is practical—it is Prob[Non-Recog] that turns out to be relevant in applications to matrix group algorithms. With this in mind, we define

$$\mathcal{Y} := \big\{(b, c) \in A^{[r]} \times A^{[s]} \mid (b, c) \text{ is recognisable}\big\},$$
$$\mathcal{Z} := A^{[r]} \times A^{[s]} \setminus \mathcal{Y},$$
$$\text{Prob[Non-Recog]} := |\mathcal{Z}|/|A^{[r]} \times A^{[s]}|.$$

Our intention is to prove the following theorem.

THEOREM 6.1. *Suppose that $n = rs$ and $r \leqslant s$. If $|A| \geqslant \frac{1}{2}(s-1)n$, then*

$$\text{Prob[Non-Recog]} \leqslant \frac{2n^2}{|A|}.$$

The proof will occupy the remainder of this section. We begin with a general lemma. Define

$$A^{[n]}_{\text{mult}} := \big\{a \in A^{[n]} \mid a \text{ is not multiplicity-free}\big\},$$

and

$$\text{Prob[mult]} := \frac{\big|A^{[n]}_{\text{mult}}\big|}{\big|A^{[n]}\big|},$$

so that Prob[mult] may be thought of as the probability that a randomly chosen n-multiset from A has multiplicities.

LEMMA 6.2. $\quad \text{Prob[mult]} < \dfrac{n(n-1)}{|A|}$.

Proof. An n-multiset a is determined by the family $(\text{mult}(g; a))_{g \in A}$ of multiplicities, and this family may be considered as an ordered partition of n into $|A|$ non-negative parts.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Such ordered partitions are in one-to-one correspondence with ordered partitions of $|A|+n$ into $|A|$ positive parts. Writing $|A|+n$ as $(1+1+\cdots+1)$, and replacing each of a selected set of $|A|-1$ of the $+$ symbols with the sequence of three symbols $)+($, one proves the well-known fact that the number of ordered partitions of $|A|+n$ into positive parts is the binomial coefficient

$$\binom{|A|+n-1}{|A|-1},$$

whence

$$\left|A^{[n]}\right| = \binom{|A|+n-1}{n}.$$

On the other hand, it should be clear that the number of multiplicity-free elements of $A^{[n]}$ is $\binom{|A|}{n}$. Thus

$$\left|A^{[n]}_{\text{mult}}\right| = \binom{|A|+n-1}{n} - \binom{|A|}{n}$$

and

$$\text{Prob[mult]} = \frac{\left|A^{[n]}_{\text{mult}}\right|}{\left|A^{[n]}\right|} = 1 - \prod_{\nu=1}^{n-1} \frac{|A|-\nu}{|A|+\nu}.$$

If $\varepsilon > 0$, then $(1-\varepsilon)/(1+\varepsilon) > 1-2\varepsilon$, and if $0 < \eta_\nu < 1$ for all relevant ν, then

$$\prod(1-\eta_\nu) > 1 - \sum \eta_\nu.$$

Therefore

$$\text{Prob[mult]} < \sum_{\nu=1}^{n-1} 2\,\frac{\nu}{|A|} = \frac{n(n-1)}{|A|},$$

as the lemma states. □

As a first step in the proof of Theorem 6.1, we divide $\mathcal{Z}$ into two parts. Recall from Lemma 5.2(1) that if (b,c) is a recognisable pair of multisets, then each of b and c is multiplicity-free. Define

$$\mathcal{Z}_0 := \{(b,c) \in \mathcal{Z} \mid b \text{ or } c \text{ is not multiplicity-free}\},$$
$$\mathcal{Z}_1 := \mathcal{Z} \setminus \mathcal{Z}_0.$$

An immediate corollary of Lemma 6.2 is the following lemma.

LEMMA 6.3.

$$\frac{|\mathcal{Z}_0|}{|A^{[r]} \times A^{[s]}|} < \frac{r(r-1)+s(s-1)}{|A|}.$$

Clearly, the probability we are seeking, that is, the probability that a randomly chosen pair $(b,c) \in A^{[r]} \times A^{[s]}$ is not recognisable, is at most

$$\text{Prob}[(b,c) \in \mathcal{Z}_0] + \text{Prob}[(b,c) \in \mathcal{Z}_1].$$

Lemma 6.3 gives a satisfactory estimate for $\text{Prob}[(b,c) \in \mathcal{Z}_0]$ and our problem is to find a bound for $\text{Prob}[(b,c) \in \mathcal{Z}_1]$.

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Define

$$\mathcal{Z}_b := \big\{(b,c) \in A^{[r]} \times A^{[s]} \mid b \text{ and } c \text{ are multiplicity-free and clause (1) of Definition 5.1 fails}\big\};$$

$$\mathcal{Z}_c := \big\{(b,c) \in A^{[r]} \times A^{[s]} \mid b \text{ and } c \text{ are multiplicity-free and clause (2) of Definition 5.1 fails}\big\};$$

$$P_b := \frac{|\mathcal{Z}_b|}{|A^{[r]} \times A^{[s]}|};$$

$$P_c := \frac{|\mathcal{Z}_c|}{|A^{[r]} \times A^{[s]}|}.$$

Then $\mathrm{Prob}[(b,c) \in \mathcal{Z}_1] \leqslant P_b + P_c$.

LEMMA 6.4. *Suppose that $n = rs$, as usual. If $|A| \geqslant \frac{1}{2}(r-1)n$, then*

$$P_b \leqslant \frac{n^2 - r(r-1)}{|A|},$$

and if $|A| \geqslant \frac{1}{2}(s-1)n$, then

$$P_c \leqslant \frac{n^2 - s(s-1)}{|A|}.$$

Proof. We prove only the first assertion, and ensure that we do so without using the assumption that $r \leqslant s$; the other then follows by interchange of b with c and r with s.

If $(b,c) \in \mathcal{Z}_b$, then for every relevant i,k, some equation of the form $b_i b_k^{-1} = z$ must hold, where $z \in bb^{-1} \cup cc^{-1} \cup bb^{-1}cc^{-1}$ and $z \neq (b_i b_k^{-1})^{\pm 1}$. We shall focus just on the first such equation, namely that for $b_1 b_2^{-1}$. Recall from Section 2, the convention that $1 \leqslant i \leqslant r$, $1 \leqslant k \leqslant r$, $1 \leqslant j \leqslant s$ and $1 \leqslant l \leqslant s$, and define index sets

$$I_b := \{(i,k) \mid i \neq k,\ (i,k) \neq (1,2),\ (i,k) \neq (2,1)\},$$
$$I_c := \{(j,l) \mid j \neq l\},$$
$$I_{bc} := \{(i,k;j,l) \mid (i,k) \in I_b \cup \{(2,1)\},\ (j,l) \in I_c\},$$

and $I := I_b \cup I_c \cup I_{bc}$. For $\xi \in I$, let z_ξ denote $b_i b_k^{-1}$, $c_j c_l^{-1}$, or $(b_i c_j)(b_k c_l)^{-1}$ appropriately, and define

$$\mathcal{Z}_\xi := \big\{(b,c) \in A^{[r]} \times A^{[s]} \mid b_1 b_2^{-1} = z_\xi, \text{ and } b \text{ and } c \text{ are multiplicity-free}\big\},$$

so that $\mathcal{Z}_b \subseteq \bigcup\{\mathcal{Z}_\xi \mid \xi \in I\}$. Define

$$E_\xi := \big\{(b,c) \in A^r \times A^s \mid b_1 b_2^{-1} = z_\xi, \text{ and } b \text{ and } c \text{ are multiplicity-free}\big\}.$$

If $\xi \in I$, then

$$|E_\xi| \leqslant |A|(|A|-1)\cdots(|A|-r+2) \times |A|(|A|-1)\cdots(|A|-s+1).$$

Because we are dealing here with multiplicity-free sequences, the natural projection $E_\xi \longrightarrow \mathcal{Z}_\xi$ is an $r!s!$-to-1 map, and so

$$|\mathcal{Z}_\xi| \leqslant \frac{1}{|A|-r+1}\binom{|A|}{r}\binom{|A|}{s}.$$

Now $|I| = r(r-1) - 2 + s(s-1) + (r(r-1)-1)s(s-1) = r(r-1)(s^2-s+1) - 2$, and so

$$|\mathcal{Z}_b| < \frac{r(r-1)(s^2-s+1)}{|A|-r+1}\binom{|A|}{r}\binom{|A|}{s}.$$

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

Suppose that $|A| \geqslant \frac{1}{2}(r-1)n$. Suppose also (seeking a contradiction) that

$$\frac{r(r-1)(s^2-s+1)}{|A|-r+1} \geqslant \frac{n^2-r(r-1)}{|A|}.$$

Now

$$r(r-1)(s^2-s+1) = n^2 - rn - sn + n + r(r-1),$$

and so

$$|A|\big(rn+sn-n-2r(r-1)\big) \leqslant (r-1)\big(n^2-r(r-1)\big).$$

From our assumption that $|A| \geqslant \frac{1}{2}(r-1)n$, we find that

$$n(rn+sn-n-2r(r-1)) \leqslant 2n^2 - 2r(r-1);$$

that is,

$$(r+s-3)n^2 \leqslant (2n-2)r(r-1).$$

Since $s \geqslant 2$, this yields that

$$(r-1)n^2 \leqslant (2n-2)r(r-1) < 2nr(r-1),$$

and since $n = rs$, this leads to the inequality $s < 2$, which is false. Thus if $|A| \geqslant \frac{1}{2}(r-1)n$, then

$$\frac{r(r-1)(s^2-s+1)}{|A|-r+1} < \frac{n^2-r(r-1)}{|A|},$$

and so

$$|\mathcal{Z}_b| < \frac{n^2-r(r-1)}{|A|}\binom{|A|}{r}\binom{|A|}{s}.$$

It follows immediately that

$$\frac{|\mathcal{Z}_b|}{\big|A^{[r]} \times A^{[s]}\big|} < \frac{(n^2-r(r-1))}{|A|},$$

which is what we were seeking to prove. □

Proof of Theorem 6.1. We know that

$$\text{Prob[Non-Recog]} \leqslant \text{Prob}[(b,c) \in \mathcal{Z}_0] + \text{Prob}[(b,c) \in \mathcal{Z}_1].$$

But

$$\text{Prob}[(b,c) \in \mathcal{Z}_0] \leqslant \frac{r(r-1)+s(s-1)}{|A|},$$

by Lemma 6.3, and

$$\text{Prob}[(b,c) \in \mathcal{Z}_1] \leqslant P_b + P_c \leqslant \frac{2n^2 - r(r-1) - s(s-1)}{|A|}$$

by Lemma 6.4. Therefore

$$\text{Prob[Non-Recog]} \leqslant \frac{2n^2}{|A|},$$

as the theorem states. □

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

7. *Uniqueness of factorisations of multisets*

Tensor factorisations are certainly not unique, even up to equivalence as defined in Section 2. Trivially, if $a = b \otimes c$ and $r = s$, then also $a = c \otimes b$. Even if we agree to regard such factorisations as being essentially the same, then tensor factorisations still need not be unique.

EXAMPLE 7.1. Suppose that $b \in A^{[r]}$, that $c \in A^{[s]}$, and that c has a tensor factorisation $b' \otimes c''$, where $b' \in A^{[r]}$. If $c' := b \otimes c''$, then $c' \in A^{[s]}$ and $b \otimes c = b' \otimes c'$.

EXAMPLE 7.2. Suppose that b is a subgroup of order r in A, that $c \in A^{[s]}$, and that $a = b \otimes c \in A^{[n]}$. Let $\phi : \{1, \ldots, s\} \longrightarrow b$ be any function, and define $c' := \{\phi(j)c_j \mid 1 \leqslant j \leqslant s\}$. Then $a = b \otimes c'$.

DEFINITION 7.3. A tensor factorisation $a = b \otimes c$, where $(b, c) \in A^{[r]} \times A^{[s]}$ will be said to be *essentially unique* if whenever $a = b' \otimes c'$, where $(b', c') \in A^{[r]} \times A^{[s]}$, either there exists $\lambda \in A$ such that $b' = \lambda b$ and $c' = \lambda^{-1}c$, or $r = s$ and there exists $\lambda \in A$ such that $b' = \lambda c$ and $c' = \lambda^{-1}b$.

It seems possible that tensor factorisations $a = b \otimes c$ where the pair (b, c) is recognisable are essentially unique, but we have not been able to prove or disprove this. The matter can be settled on stronger hypotheses, however.

DEFINITION 7.4. The pair (b, c) in $A^{[r]} \times A^{[s]}$ (where, as usual, $2 \leqslant r \leqslant s$) will be said to be *clearly recognisable* if the following conditions hold:

(1) for all $g \in bb^{-1}$, either $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ or $g^2 = 1$ and $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$;

(2) for all $g \in cc^{-1}$, either $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ or $g^2 = 1$ and $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$;

(3) for all $g \in bb^{-1}cc^{-1}$, either $\mathrm{mult}(g;\ bb^{-1}cc^{-1}) < r$ or $g^2 = 1$ and $\mathrm{mult}(g;\ bb^{-1}cc^{-1}) < 2r$.

Although this is considerably more demanding than the definition of recognisability, the proof of Theorem 6.1 shows in fact that pairs (b, c) satisfying conditions (1) and (2) occur with frequency bigger than $1 - 2n^2|A|^{-1}$, and it seems likely that clause (3) does not reduce this significantly.

THEOREM 7.5. *Suppose that $a = b \otimes c$, where (b, c) is a clearly recognisable pair from $A^{[r]} \times A^{[s]}$ and $2 \leqslant r \leqslant s$. If also $a = b' \otimes c'$, where $(b', c') \in A^{[r]} \times A^{[s]}$, then either there exists $\lambda \in A$ such that $b' = \lambda b$ and $c' = \lambda^{-1}c$, or $r = s$ and there exists $\lambda \in A$ such that $b' = \lambda c$ and $c' = \lambda^{-1}b$.*

Proof. Suppose first that $r < s$. Since $aa^{-1} = sbb^{-1} + rcc^{-1} + bb^{-1}cc^{-1}$, by (3) above, the only elements of aa^{-1} that have multiplicity greater than or equal to r are those in bb^{-1} or cc^{-1}. Then, by (2), the elements of cc^{-1} can be recognised within aa^{-1} as being those which have multiplicity r, or which have order 2 and multiplicity $2r$. And by (1) the elements of bb^{-1} can be recognised within aa^{-1} as being those which have multiplicity s, or which have order 2 and multiplicity $2s$. Thus $bb^{-1}cc^{-1}$ can be recognised as that submultiset of aa^{-1} which consists of elements with multiplicity less than r. Now $aa^{-1} = sb'b'^{-1} + rc'c'^{-1} + b'b'^{-1}c'c'^{-1}$, and it follows that $bb^{-1}cc^{-1} \subseteq b'b'^{-1}c'c'^{-1}$. Since these multisets both have size $r(r-1)s(s-1)$, in fact $b'b'^{-1}c'c'^{-1} = bb^{-1}cc^{-1}$. Then also,

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press

by comparing elements of multiplicity r or $2r$ and elements of multiplicity s or $2s$, we find that $b'b'^{-1} = bb^{-1}$ and $c'c'^{-1} = cc^{-1}$. Consequently, the pair (b', c') also is clearly recognisable.

At this point, the argument divides into four cases corresponding to B1C1, B2C1, B1C2 and B2C2, as in Section 5. In the first of these, there exist $g \in bb^{-1}$ and $h \in cc^{-1}$ such that $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ and $\mathrm{mult}(h;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$. Without loss of generality, $g = b_1b_2^{-1} = b'_1b'^{-1}_2$ and $h = c_1c_2^{-1} = c'_1c'^{-1}_2$. The multiset of numerators of quotients in aa^{-1} equal to h is on the one hand bc_1, and on the other hand $b'c'_1$. Therefore if $\lambda := c_1c'^{-1}_1$, then $b' = \lambda b$. It follows similarly that $c' = \mu c$, where $\mu := b_1b'^{-1}_1$. Now b_1c_1 is the only element of a that occurs both as a numerator of a quotient in aa^{-1} equal to g, and as a numerator of a quotient in aa^{-1} equal to h. The same is true of $b'_1c'_1$, and so $b_1c_1 = b'_1c'_1$. Consequently, $\mu = \lambda^{-1}$ and $b \otimes c$, $b' \otimes c'$ are equivalent tensor factorisations of a.

Cases B2C1 and B1C2 are left to the reader, and we turn to case B2C2. Here we have $g \in bb^{-1}$, $h \in cc^{-1}$ such that $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 2$ and $g^2 = 1$, and $\mathrm{mult}(h;\ bb^{-1}+cc^{-1}+bb^{-1}cc^{-1}) = 1$ and $h^2 = 1$. Without loss of generality, $g = b_1b_2^{-1} = b'_1b'^{-1}_2$ and $h = c_1c_2^{-1} = c'_1c'^{-1}_2$. Notice that $(gh)^2 = 1$ and $\mathrm{mult}(gh;\ bb^{-1}cc^{-1}) \geqslant 4$ since gh occurs in $bb^{-1}cc^{-1}$ as $b_1b_2^{-1}c_1c_2^{-1}$, $b_2b_1^{-1}c_1c_2^{-1}$, $b_1b_2^{-1}c_2c_1^{-1}$, $b_2b_1^{-1}c_2c_1^{-1}$. By clause (3) of Definition 7.4, we must therefore have $r \geqslant 3$. The multiset b'' of numerators of quotients in aa^{-1} equal to h is on the one hand $bc_1 + bc_2$, and on the other hand $b'c'_1 + b'c'_2$. The multiset c'' of numerators of quotients in aa^{-1} equal to g is on the one hand $b_1c + b_2c$, and on the other hand $b'_1c' + b'_2c'$. As in Lemma 5.7, $|b'' \cap c''| = 4$, and if $y_1 \in b'' \setminus c''$ and $y_2 \in c'' \setminus b''$, then $\exists z \in b'' \cap c''$ such that $(y_2z^{-1})b'' \cap a = \lambda b$ and $(y_1z^{-1})c'' \cap a = \mu c$ for some $\lambda, \mu \in A$. Equally, $(y_2z^{-1})b'' \cap a = \lambda' b'$ and $(y_1z^{-1})c'' \cap a = \mu' c'$ for some $\lambda', \mu' \in A$. It is not hard to prove that $\lambda\mu = \lambda'\mu'$, and it follows that the tensor factorisations $a = b \otimes c$ and $a = b' \otimes c'$ are equivalent.

There remains the situation where $r = s$. It follows as before that $bb^{-1}cc^{-1}$ can be recognised as the sub-multiset of aa^{-1} consisting of elements with multiplicity less than r and that $b'b'^{-1}c'c'^{-1} = bb^{-1}cc^{-1}$. Therefore also $b'b'^{-1} + c'c'^{-1} = bb^{-1} + cc^{-1}$. Consider case B1C1, in which there exist $g \in bb^{-1}$, $h \in cc^{-1}$ such that $\mathrm{mult}(g;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$ and $\mathrm{mult}(h;\ bb^{-1} + cc^{-1} + bb^{-1}cc^{-1}) = 1$. Without loss of generality, $g = b_1b_2^{-1}$ and $h = c_1c_2^{-1}$. Let b'' be the multiset of numerators of quotients in aa^{-1} equal to h, and c'' the multiset of numerators of quotients equal to g, so that $b'' = bc_1$ and $c'' = b_1c$. Considering a as $b' \otimes c'$, we will find (without loss of generality) that b'' is $b'c'_1$ or b'_1c' and c'' is $b'c'_j$ or b'_ic'. Now $b''b''^{-1} + c''c''^{-1} = bb^{-1} + cc^{-1} = b'b'^{-1} + c'c'^{-1}$, whereas $(b'c'_1)(b'c'_1)^{-1} + (b'c'_j)(b'c'_j)^{-1} = 2b'b'^{-1}$ and $(b'_1c')(b'_1c')^{-1} + (b'_ic')(b'_ic')^{-1} = 2c'c'^{-1}$. Since $\mathrm{mult}(g;\ b'b'^{-1} + c'c'^{-1}) = 1$, $b'b'^{-1} + c'c'^{-1} \neq 2b'b'^{-1}$ and $b'b'^{-1} + c'c'^{-1} \neq 2c'c'^{-1}$. Therefore either $b'' = b'c'_1$ and $c'' = b'_ic'$, or (without loss of generality) $b'' = b'_1c'$ and $c'' = b'c'_j$. Cases B2C1, B1C2 and B2C2 are similar, and lead to the same conclusion. From there on, the argument is the same as that for the situation where $r < s$, and we find that $b' \otimes c'$ is equivalent either to $b \otimes c$ or to $c \otimes b$, as required. □

Acknowledgements. We are grateful to the EPSRC for Research Grant GR/K69186, which made this collaboration possible. The research of the second author was supported also by an Australian Research Council grant. We are grateful also to Mr Jia Lun Huang for corrections to the previous draft of this work, and for confirming in [**6**] that our ideas are workable in practice.

References

1. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *Data structures and algorithms* (Addison-Wesley, Reading, MA, 1983). 79

2. Richard Arratia, A. D. Barbour and Simon Tavaré, 'On random polynomials over finite fields', *Math. Proc. Cambridge Philos. Soc.* 114 (1993) 347–368. 77

3. Catherine Greenhill, 'From multisets to matrix groups: some algorithms related to the exterior square' DPhil dissertation, Oxford, 1996. 78

4. Catherine Greenhill, 'An algorithm for recognising the exterior square of a matrix', *Linear and Multilinear Algebra* 46 (1999) 213–244. 78

5. Catherine Greenhill, 'An algorithm for recognising the exterior square of a multiset', *LMS J. Comput. Math.* 3 (2000) 96-116; http://www.lms.ac.uk/jcm/3/lms1999-021. 78

6. Jia Lun Huang, 'The implementation of a factorisation algorithm for combinatorial tensor products', MSc dissertation, Oxford, August 2003. 76, 77, 84, 99

7. Nathan Jacobson, *Lectures in abstract algebra*, vols I–III (Van Nostrand, New Jersey, 1953). 74

8. Donald E. Knuth, *The art of computer programming*, vol 3: 'Sorting and searching' (Addison-Wesley, Reading, MA, 1973). 79

9. M. Mignotte and J.-L. Nicolas, 'Statistique sur $F_q[X]$', *Ann. Inst. H. Poincaré* Sect. B (N.S.) 19 (1983) 113–121. 77

Peter M. Neumann peter.neumann@queens.ox.ac.uk

The Queen's College
Oxford OX1 4AW
United Kingdom

Cheryl E. Praeger praeger@maths.uwa.edu.au
http://www.maths.uwa.edu.au/∼praeger

School of Mathematics and Statistics
University of Western Australia
Crawley, WA 6009
Australia

https://doi.org/10.1112/S1461157000001054 Published online by Cambridge University Press