A Normal Form for Matrix Multiplication Schemes

Manuel Kauers*[0000−0001−8641−6661] and Jakob Moosbauer**[0000−0002−0634−4854]

Institute for Algebra, Johannes Kepler University, Linz, Austria
{manuel.kauers,jakob.moosbauer}@jku.at

Abstract. Schemes for exact multiplication of small matrices have a large symmetry group. This group defines an equivalence relation on the set of multiplication schemes. There are algorithms to decide whether two schemes are equivalent. However, for a large number of schemes a pairwise equivalence check becomes cumbersome. In this paper we propose an algorithm to compute a normal form of matrix multiplication schemes. This allows us to decide pairwise equivalence of a larger number of schemes efficiently.

1 Introduction

Computing the product of two $n \times n$ matrices using the straightforward algorithm costs $O(n^3)$ operations. Strassen found a multiplication scheme that allows to multiply two 2×2 matrices using only 7 multiplications instead of 8 [13]. This scheme can be applied recursively to compute the product of $n \times n$ matrices in $O(n^{\log_2 7})$ operations. This discovery lead to a large amount of research on finding the smallest ω such that two $n \times n$ matrices can be multiplied using at most $O(n^\omega)$ operations. The currently best known bound is $\omega < 2.37286$ and is due to Alman and Williams [1].

Another interesting question is to find the exact number of multiplications needed to multiply two $n \times n$ matrices for small numbers n. For $n = 2$ Strassen provided the upper bound of 7. Winograd showed that we also need at least 7 multiplications [14]. De Groote proved that Strassen's algorithm is unique [6] modulo a group of equivalence transformations.

For the case $n = 3$ Laderman was the first to present a scheme that uses 23 multiplications [9], which remains the best known upper bound, unless the coefficient domain is commutative [11]. The currently best lower bound is 19 and was proved by Bläser [3]. There are many ways to multiply two 3×3 matrices using 23 multiplications [8,5,10,12,7,2].

For every newly found algorithm the question arises whether it is really new or it can be mapped to a known solution by one of the transformations described by de Groote. These transformations define an equivalence relation on the set of

* M.K. was supported by the Austrian Science Fund (FWF) grant P31571-N32.
** J.M. was supported by the Land Oberösterreich through the LIT-AI Lab.

matrix multiplication algorithms. Some authors used invariants of the action of the transformation group to prove that their newly found schemes are inequivalent to the known algorithms. The works of Berger et al. [2] and Heule et al. [7] provide algorithms to check if two given schemes are equivalent. Berger et al. give an algorithm that can check equivalence over the ground field $\mathbb{R}$ if the schemes fulfill a certain assumption. Heule et al. provide an algorithm to check equivalence over finite fields.

Heule et al. presented more than 17,000 schemes for multiplying 3×3 matrices and showed that they are pairwise nonequivalent, at least when viewed over the ground field $\mathbb{Z}_2$. Their collection has since been extended to more than 64,000 pairwise inequivalent schemes. For testing whether a newly found scheme is really new, we would need to do an equivalence test for each of these schemes. Due to the large number of schemes this becomes expensive.

In this paper we propose an algorithm that computes a normal form for the equivalence class of a given scheme over a finite field. If all known schemes already are in normal form, then deciding whether a newly found scheme is equivalent to any of them is reduced to a normal form computation for the new scheme and a cheap syntactic comparison to every old scheme. Although the transformation group over a finite field is finite, it is so large that checking equivalence by computing every transformation is not feasible. Thus, Heule et al. use a strategy that iteratively maps one scheme to another part by part. We use a similar strategy to find a minimal element of an equivalence class.

2 Matrix Multiplication Schemes

Let K be a field and let $\mathbf{A}, \mathbf{B} \in K^{n\times n}$. The computation of the matrix product $\mathbf{C} = \mathbf{AB}$ by a Strassen-like algorithm proceeds in two stages. In the first stage we compute some intermediate products $M_1, \ldots, M_r$ of linear combinations of entries of $\mathbf{A}$ and linear combinations of entries of $\mathbf{B}$. In the second stage we compute the entries of $\mathbf{C}$ as linear combinations of the M_i.

For example if $n = 2$, we can write

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} \quad \text{and} \quad \mathbf{C} = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}.$$

Strassen's algorithm computes $\mathbf{C}$ in the following way:

$$\begin{aligned} M_1 &= (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2}) \\ M_2 &= (a_{2,1} + a_{2,2})(b_{1,1}) \\ M_3 &= (a_{1,1})(b_{1,2} - b_{2,2}) \\ M_4 &= (a_{2,2})(b_{2,1} - b_{1,1}) \\ M_5 &= (a_{1,1} + a_{1,2})(b_{2,2}) \\ M_6 &= (a_{2,1} - a_{1,1})(b_{1,1} + b_{1,2}) \\ M_7 &= (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2}) \end{aligned}$$

$$
\begin{aligned}
c_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
c_{1,2} &= M_3 + M_5 \\
c_{2,1} &= M_2 + M_4 \\
c_{2,2} &= M_1 - M_2 + M_3 + M_6.
\end{aligned}
$$

A Strassen-like multiplication algorithm that computes the product of two $n \times n$ matrices using r multiplications has the form

$$
\begin{aligned}
M_1 &= (\alpha_{1,1}^{(1)} a_{1,1} + \alpha_{1,2}^{(1)} a_{1,2} + \cdots)(\beta_{1,1}^{(1)} b_{1,1} + \beta_{1,2}^{(1)} b_{1,2} + \cdots) \\
&\vdots \\
M_r &= (\alpha_{1,1}^{(r)} a_{1,1} + \alpha_{1,2}^{(r)} a_{1,2} + \cdots)(\beta_{1,1}^{(r)} b_{1,1} + \beta_{1,2}^{(r)} b_{1,2} + \cdots) \\
c_{1,1} &= \gamma_{1,1}^{(1)} M_1 + \gamma_{1,1}^{(2)} M_2 + \cdots + \gamma_{1,1}^{(r)} M_r \\
&\vdots \\
c_{n,n} &= \gamma_{n,n}^{(1)} M_1 + \gamma_{n,n}^{(2)} M_2 + \cdots + \gamma_{n,n}^{(r)} M_r.
\end{aligned}
$$

All the information about such a multiplication scheme is contained in the coefficients $\alpha_{i,j}, \beta_{i,j}$ and $\gamma_{i,j}$. We can write these coefficients as a tensor in $K^{n\times n} \otimes K^{n\times n} \otimes K^{n\times n}$:

$$
\sum_{l=1}^{r} ((\alpha_{i,j}^{(l)}))_{i=1,j=1}^{n,n} \otimes ((\beta_{i,j}^{(l)}))_{i=1,j=1}^{n,n} \otimes ((\gamma_{i,j}^{(l)}))_{i=1,j=1}^{n,n}. \tag{1}
$$

A multiplication scheme, seen as an element of $K^{n\times n} \otimes K^{n\times n} \otimes K^{n\times n}$ is equal to the matrix multiplication tensor defined by $\sum_{i,j,k=1}^{n} E_{i,k} \otimes E_{k,j} \otimes E_{i,j}$ where $E_{u,v}$ is the matrix with 1 at position (u, v) and zeros everywhere else [4]. Formulas become a bit more symmetric if we look at the tensor $\sum_{i,j,k=1}^{n} E_{i,k} \otimes E_{k,j} \otimes E_{j,i}$ corresponding to the product $\mathbf{C}^T = \mathbf{AB}$, so we will consider this tensor instead.

We represent a scheme as a table containing the matrices in this tensor. We will refer to the rows and columns of this table as the rows and columns of a scheme. For example Strassen's algorithm is represented as shown in Table 1.

3 The Symmetry Group

There are several transformations that map one matrix multiplication scheme to another one. We call two schemes equivalent if they can be mapped to each other by one of these transformations. De Groote [6] first described the transformations and showed that Strassen's algorithm is unique modulo this equivalence.

The first transformation is permuting the rows of a scheme. This corresponds to just changing the order of the M_i's in the algorithm. Another transformation comes from the fact that $\mathbf{AB} = \mathbf{C}^T \Leftrightarrow \mathbf{B}^T\mathbf{A}^T = \mathbf{C}$. It acts on a tensor by transforming a summand $A \otimes B \otimes C$ to $B^T \otimes A^T \otimes C^T$. Moreover, it follows from

	α	β	γ
1	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2	$\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 1 & -1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$
4	$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$
5	$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 \\ 0 & 0 \end{pmatrix}$
6	$\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$
7	$\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

Table 1: Strassen's Algorithm

the condition that the sum (1) is equal to the matrix multiplication tensor, that also a cyclic permutation of the coefficients α, β and γ is a symmetry transformation. Taking those together we get an action that is composed by an arbitrary permutation of the columns of a scheme and transposing all the matrices if the permutation is odd.

Finally, we can use that for any invertible matrix V we have $\mathbf{AB} = \mathbf{A}VV^{-1}\mathbf{B}$. The corresponding action on a tensor $A \otimes B \otimes C$ maps it to $AV \otimes V^{-1}B \otimes C$. Since we can permute A, B and C we also can insert invertible matrices U and W which results in the action

$$(U, V, W) * A \otimes B \otimes C = UAV^{-1} \otimes VBW^{-1} \otimes WCU^{-1}. \tag{2}$$

This transformation is called the sandwiching action.

If we combine all these transformations we get the group $G = S_r \times S_3 \ltimes \mathrm{GL}(K, n)^3$ of symmetries of $n \times n$ matrix multiplication schemes with r rows. By $\mathrm{Aut}(G)$ we denote the group of automorphisms of a group G.

Definition 1. *Let* $\varphi\colon S_3 \to \mathrm{Aut}(\mathrm{GL}(K, n)^3)$ *be defined by*

$$\varphi(\pi) = \begin{cases} (U, V, W) \mapsto \pi((U, V, W)) & \text{if } \mathrm{sgn}(\pi) = 1 \\ (U, V, W) \mapsto \pi((V^{-T}, W^{-T}, U^{-T})) & \text{if } \mathrm{sgn}(\pi) = -1 \end{cases}$$

The symmetry group of $n \times n$ *matrix multiplication schemes with* r *rows is defined over the set* $G = S_r \times S_3 \times \mathrm{GL}(K, n)^3$ *with the multiplication given by*

$$(\sigma_1, \pi_1, (U_1, V_1, W_1)) \cdot (\sigma_2, \pi_2, (U_2, V_2, W_2)) = \\ (\sigma_1\sigma_2, \pi_1\pi_2, (U_1, V_1, W_1)\varphi(\pi_1)((U_2, V_2, W_2))).$$

The action $g * s$ of a group element $g = (\sigma, \pi, (U, V, W)) \in G$ on a multiplication scheme $s \in (K^{n\times n})^{r\times 3}$ is defined by first letting σ permute the rows of s then letting π permute the columns of s and transposing every matrix if $\operatorname{sgn}(\pi) = -1$ and finally letting U, V and W act on every row as defined in equation (2).

One can show that this action fulfills the criteria of a group action.

4 Minimal Orbit Elements

Two schemes are equivalent if they belong to the same orbit under the action of the group G. Our goal in this section is to define a normal form for every orbit. The particular choice of the normal form is partly motivated by implementation convenience and not by any special properties. From now on we assume that K is a finite field. Since over a finite field the symmetry group is finite we could decide equivalence or compute a normal form by exhaustive search. However, already for $n = 3$ the symmetry group over $\mathbb{Z}_2$ has a size of $23! \cdot 6 \cdot 4741632 \approx 7 \cdot 10^{29}$.

Definition 2. *Let $s \in (K^{n\times n})^{r\times 3}$ be a matrix multiplication scheme. The rank pattern of the scheme is defined as the table*

$$((\operatorname{rank} s_{i,1}, \operatorname{rank} s_{i,2}, \operatorname{rank} s_{i,3}))_{i=1}^{r}.$$

The rank vector of a row (A, B, C) is $(\operatorname{rank}(A), \operatorname{rank}(B), \operatorname{rank}(C))$.

Since the matrices U, V and W are invertible, the sandwiching action leaves the rank pattern invariant. Transposing the matrices does not change their rank either. Therefore the only way a group element changes the rank pattern of a scheme is by permuting it accordingly. So for two equivalent schemes their rank patterns only differ by a permutation of rows and columns. This allows us to permute the rows and columns of the scheme such that the rank pattern becomes maximal under lexicographic order.

This maximal rank pattern is a well-known invariant of the symmetry group that has been used to show that two schemes are not equivalent. For example Courtois et al. [5] and Oh et al. [10] used this test to prove that their schemes were indeed new. However, this method only provides a sufficient condition for the inequivalence of schemes and can not decide equivalence of schemes. In Heule et al.'s data for certain rank patterns there are almost 1000 inequivalent schemes having this rank pattern.

We choose the normal form to be an orbit element which has a maximal rank pattern and is minimal under a certain lexicographic order. For doing so fix a total order on K such that $0 < 1 < x$ for all $x \in K \setminus \{0, 1\}$. The order need not be compatible with $+$ or $\cdot$ in any sense. For the matrices in the schemes we use colexicoraphic order by columns, with columns compared by lexicographic order. This means for two column vectors $v = (x_1, \ldots, x_n)^T$ and $v' = (x'_1, \ldots, x'_n)^T$ we define recursively

$$v < v' :\Leftrightarrow x_1 < x'_1 \vee (x_1 = x'_1 \wedge (x_2, \ldots x_n) < (x'_2, \ldots, x'_n))$$

For two matrices $M = (v_1 \mid \cdots \mid v_n)$ and $M' = (v'_1 \mid \cdots \mid v'_n)$ we define

$$M < M' :\Leftrightarrow v_n < v'_n \vee (v_n = v'_n \wedge (v_1 \mid \cdots \mid v_{n-1}) < (v'_1 \mid \cdots \mid v'_{n-1}))$$

For ordering the schemes we use the common lexicographic order. So we compare two schemes row by row from top to bottom and in each row we compare the matrices from left to right using the order defined above.

Definition 3. *Let $s \in (K^{n\times n})^{r\times 3}$ be a matrix multiplication scheme. We say s is in normal form if $s = \min\{s' \in G * s \mid$ the rank pattern of s' is sorted$\}$, where the minimum is taken with respect to the order defined above.*

Such a normal form clearly exists and it is unique since the group G is finite and the lexicographic order is a total order.

The strategy to compute the normal form is as follows:

Let s be a multiplication scheme and let N be its normal form.

We start by going over all column permutations of s and sort their rows by rank pattern to find a scheme s' with maximal rank pattern. If there are several column permutations that lead to the same maximal rank pattern, we consider each of them separately, since there are at most six.

Then we proceed row by row. For all rows of s' that have maximal rank pattern, we determine the minimal element of their orbit under the action of $\mathrm{GL}(K, n)^3$. From the definition of the normal form, it follows that the smallest row we can produce this way has to be the first row (A, B, C) of N. However, we might be able to reach the first row of N from several different rows and also the choice of U, V and W is in general not unique.

Apart from the first row of N we also compute the stabilizer of the first row, which is the set of all triples $(U, V, W) \in \mathrm{GL}(K, n)^3$ such that $(A, B, C) = (UAV^{-1}, VBW^{-1}, WCU^{-1})$. For each possible row that can be mapped to the first row we compute the *tail*, by which we mean the list of all remaining rows after applying a suitable triple (U, V, W).

We then continue this process iteratively. We go over each tail and determine a row that has maximal rank vector and becomes minimal under the action of the stabilizer. To do this we apply every element of the stabilizer to all possible candidates for the next row. This uniquely determines the next row of the normal form and we get again a list of tails and the stabilizer of the already determined rows.

The full process is listed in Algorithm 1.

Proposition 1. *Algorithm 1 terminates and is correct.*

Proof. The termination of the algorithm is guaranteed, since in line 19 the new tails contain one row less than in the previous step, so eventually the list of tails only contains empty elements.

To prove correctness we first note that the choice of P ensures that it contains a scheme that can be mapped to its normal form without applying further column permutations. From now on we only consider the iteration of the loop in line 3 where s' is this scheme.

Input : A matrix multiplication scheme s
Output: An equivalent scheme in normal form

1 $P := \{s' \in (S_3 \times S_r) * s \mid s' \text{ has maximal rank pattern in } (S_3 \times S_r) * s\}$
2 $o := s$
3 **for** $s' \in P$ **do**
4 $\quad candidate := ()$
5 $\quad tails = \{s'\}$
6 $\quad stab = \mathrm{GL}(K, n)^3$
7 $\quad$ **while** $tails \neq \{()\}$ **do**
8 $\quad\quad min := (1, \ldots, 1)^T$
9 $\quad\quad newtails := \{\}$
10 $\quad\quad$ **for** $t \in tails$ **do**
11 $\quad\quad\quad$ **for** $r \in t$ *with maximal rank vector* **do**
12 $\quad\quad\quad\quad g := \mathrm{argmin}_{g \in stab}\, g * r$
13 $\quad\quad\quad\quad$ **if** $g * r < min$ **then**
14 $\quad\quad\quad\quad\quad min := g * r$
15 $\quad\quad\quad\quad\quad newtails := \{\}$
16 $\quad\quad\quad\quad$ **if** $g * r = min$ **then**
17 $\quad\quad\quad\quad\quad newtails := newtails \cup (g * t \setminus \{g * r\})$
18 $\quad\quad stab := \{g \in stab \mid g * min = min\}$
19 $\quad\quad tails := newtails$
20 $\quad\quad$ append min to $candidate$
21 $\quad$ **if** $candidate < o$ **then**
22 $\quad\quad o := candidate$
23 **return** o

Algorithm 1: Normal Form Computation

It remains to show that after lines 4 to 20 the candidate is in normal form. To this end we prove the following loop invariant for the while loop: *candidate* is an initial segment of the normal form and there is a $g \in stab$ and a $t \in tails$ such that $g * t$ is a permutation of the remaining rows of the normal form.

The lines 4, 5 and 6 ensure that the loop invariant is true at the start of the loop. We now assume that the loop invariant holds at the beginning of an iteration and prove that it is still true after the iteration. Since we know that there are $g \in stab$ and $t \in tails$ such that $g * t$ is a permutation of the remaining part of the normal form and the rank vector is invariant under the group action the lines 10 and 11 will at some point select an r that can be mapped to the next row of the normal form.

Since the normal form is the lexicographically smallest scheme in its equivalence class, the next row must always be the smallest row that has not been added to *candidate* yet. Therefore by choosing g such that $g * r$ is minimal in line 12 we ensure that *min* is the next row of the normal form.

In line 17 a transformed version $t' = g * t \setminus \{g * r\}$ of the element t with r removed is added to *newtails*. Therefore, *newtails* still contains an element t' that can be mapped to the remaining rows of the normal form.

Finally, we have to show that *stab* still contains a suitable element. Let $g' \in stab$ be such that g' maps t to a permutation of the remaining rows of the normal form. Let g be the element chosen to minimize r in line 11. Since *stab* is a group it must contain $g' \cdot g^{-1}$. Moreover, *newtails* contains $g * t \setminus \{g * r\}$ which is mapped to $g' * t \setminus \{g' * r\}$. Therefore, $g' \cdot g^{-1}$ has the desired property. □

5 Minimizing the First Row

Algorithm 1 is more efficient than a naive walk through the whole symmetry group G because we can expect the stabilizer to quickly become small during the computation. However, in the first iteration we still go over the full group $\mathrm{GL}(K, n)^3$. In this section we describe how this can be avoided.

The order we have chosen ensures that the first row has a particular form.

Proposition 2. *Let $G = \mathrm{GL}(K, n)^3$ and let $(A, B, C) \in (K^{n \times n})^3$ be such that (A, B, C) is the minimal element of $G * (A, B, C)$. Then the following hold:*

1. *A has the form*
$$\begin{pmatrix} 0 & 0 \\ I_r & 0 \end{pmatrix}$$
where $r = \operatorname{rank} A$.
2. *B is in column echelon form.*
3. *If $\operatorname{rank} A = n$, then $A = I_n$ and B has the form*
$$\begin{pmatrix} 0 & 0 \\ I_r & 0 \end{pmatrix} \tag{3}$$
where $r = \operatorname{rank} B$.

Proof. Using Gaussian elimination we can find

$$(A', B', C') = (U, V, W) * (A, B, C)$$

where A' and B' are in the described form. Note that for part 3 we can first determine V and W and then choose $U = VA^{-1}$. To show that (A, B, C) already is in this form we proceed by induction on n. If $n = 1$, then the claims are true. For the induction step assume that the claims are true for $n - 1$.

1. We first consider the special case $\operatorname{rank} A = n$. Denote by $v_1, \ldots, v_n$ the columns of A. Since $A \leq A' = I_n$ there are two cases:
   Case 1: $v_n < e_n$. Then $v_n = 0$ contradicting the assumption that A has full rank.
   Case 2: $v_n = e_n$. Then the last row of A contains only zeros apart from the 1 in the bottom right corner. Otherwise we could use column reduction to make A smaller. Since A is minimal, also the matrix we get when we remove the last column and row from A has to be minimal. So by the induction hypothesis A has the desired form.

Now suppose $\operatorname{rank} A < n$. Since the last column of A' contains only zeros and A is minimal, the last column of A consists only of zeros. We can use row reduction to form a matrix A'' that is equivalent to A, has a zero row and all other rows equal to those of A. So $A'' < A$. We then shift the zero row of A'' to the top. Since this doesn't make A'' bigger, it is still not greater than A. Because of the minimality of A, its first row has then to be zero as well. Now we can remove the last column and first row of A and the resulting matrix must still be minimal. So by the induction hypothesis A is of the desired form.

2. Since we already showed $A = A'$ we can assume $U = V = I_n$. So B' is the column echelon form of B. We write B as $(v_1 \mid \cdots \mid v_n)$ and B' as $(v'_1 \mid \cdots \mid v'_n)$. We again have two cases:
   Case 1: $v_n < v'_n$. So $v'_n \neq 0$ and since B' is in column echelon form this implies $v'_n = e_n$. Then $v_n = 0$ which contradicts that B' is the column echelon form of B.
   Case 2: $v_n = v'_n$. Since B' is in column echelon form we either have $v_n = e_n$ or $v_n = 0$. We claim that the matrix we get by removing the last column and row from B is minimal. If not, there is a sequence of column operations that makes that matrix smaller. Let $B'' = (v''_1 \mid \cdots \mid v''_n)$ be the matrix we get by applying these operations to B and let i be the index of the right most column that was changed. So v''_i with the last element removed must be smaller than v_i with the last element removed. However, this implies that $v''_i < v_i$ and therefore $B'' < B$, which is a contradiction. So by the induction hypothesis B with the last row and column removed must be in column echelon form.
   It remains to show that the last rows of B and B' are equal. There must exist a sequence of column operations that turn B into B'. If $v_n = e_n = v'_n$, then these operations would eliminate all elements in the last row of B, except the one in the bottom right corner. This implies $B' \leq B$ and therefore $B' = B$. If $v_n = 0 = v'_n$, then this sequence cannot change the last row because any column operation not involving the last column would destroy the column echelon form in the upper left part. Therefore $B = B'$.
3. Let $\operatorname{rank} A = n$. We have already shown that $A = I_n$. For any choice of V we can choose $U = VA^{-1}$ to ensure $A' = I_n$. So B is minimal under arbitrary row and column permutations. So in this case the claim can be shown the same way as 1. □

Let $s \in (K^{n \times n})^{r \times 3}$ be a matrix multiplication scheme. Denote by (U, V, W) the element of $\mathrm{GL}(K, n)^3$ used to transform s into normal form and denote by (A_1, B_1, C_1) the first row of the normal form of s. Let (A, B, C) be the row that is mapped to (A_1, B_1, C_1) and assume that the columns of s do not need to be permuted.

Then (A, B, C) must have a maximal rank vector. Therefore, A has the maximal rank of all the matrices in the scheme. So if the scheme contains a matrix of full rank then A has full rank. Moreover, A_1 is the minimal element equivalent to A under the action of $\mathrm{GL}(K, n)^3$.

Input : A triple of $n \times n$ matrices (A, B, C)
Output: A minimal triple equivalent under the action of $\mathrm{GL}(K, n)^3$

if $\operatorname{rank} A = n$ **then**
  $A_1 := I_n$
  $C' := CA$
  if $\operatorname{rank} B = n$ **then**
    $B_1 := I_n$
    $C_1 := \min_{W \in \mathrm{GL}(K,n)} WB^{-1}C'W^{-1}$
  else
    $B_1 := \min_{V,W \in \mathrm{GL}(K,n)} VBW^{-1}$
    $S := \{(V, W) \mid V, W \in \mathrm{GL}(K, n) \wedge VB = B_1W\}$
    $C_1 := \min_{(V,W) \in S} WC'V^{-1}$
else
  $A_1 := \min_{U,V \in \mathrm{GL}(K,n)} UAV^{-1}$
  $(U, V) := \operatorname{argmin}_{U,V \in \mathrm{GL}(K,n)} UAV^{-1}$
  $B' = VB; C' = CU^{-1}$
  $S := \{(U, V) \mid U, V \in \mathrm{GL}(K, n) \wedge UA_1 = A_1V\}$
  $B_1 := \min_{(U,V) \in S, W \in \mathrm{GL}(K,n)} VB'W^{-1}$
  $C'' = WC'$, where W is chosen as in the line above
  $S' := \{(U, V, W) \in \mathrm{GL}(K, n)^3 \mid UA_1 = A_1V \wedge VB_1 = B_1W\}$
  $C_1 := \min_{(U,V,W) \in S'} WC''V^{-1}$ **return** (A_1, B_1, C_1)

Algorithm 2: Special treatment of first row

If A has full rank, then $A_1 = I_n$ by Proposition 2. So we consider the scheme $s' = (A^{-1}, I_n, I_n) * s$ instead and update A, B, C and U, V, W accordingly. Then $A = A_1 = I_n$ and therefore $U = V$. So by Proposition 2, B_1 must be of the form 3.

If B also has full rank, then we set $s'' = (B^{-1}, B^{-1}, I_n) * s'$ and adjust A, B, C and U, V, W again. So we have $B = B_1 = I_n$ and $U = V = W$. Now we can determine C_1 and the stabilizer of the first row by iterating over $\mathrm{GL}(K, n)$ and minimizing WCW^{-1}.

If B does not have full rank, we determine all invertible matrices V and W such that $VBW^{-1} = B_1$. This can be done by solving the linear system $VB = B_1W$ and discarding all solutions corresponding to singular matrices. Since $U = V$ we go through all possibilities for V and W and minimize WCV^{-1}. This allows us to determine C_1 and the stabilizer of the first row.

If A does not have full rank, we solve the linear system $UA = A_1V$ and discard all solutions corresponding to singular matrices. The remaining solutions are the possible choices for U and V such that $UAV^{-1} = A_1$. By Proposition 2, B_1 must be in column echelon form. So for all possible choices of U and V we determine W such that VBW^{-1} is in column echelon form. The smallest matrix VBW^{-1} constructed this way must be equal to B_1. Then we go over all such triples (U, V, W) that map B to B_1 and determine those that minimize WCU^{-1}. So we find C_1 and the stabilizer of the first row.

The process is summarized in Algorithm 2.

6 Timings and Analysis

All the timing and analysis is done on an extension of Heule et al.'s data set with an implementation of our algorithm for 3×3 matrices over $\mathbb{Z}_2$. This data set contains 64,150 schemes.

For a comparison we have tested the equivalence check of Heule et al. on 10,000 randomly selected pairs from the data set and computed the normal form of 10,000 randomly selected schemes. Checking equivalence of two schemes took on average 0.0092 seconds. Computing a normal form took on average 1.87 seconds. The check for syntactic equivalence of the schemes in normal form takes about 0.00002 seconds, which is negligible. Thus, in our application checking equivalence of a single new scheme against a set of known schemes in normal form is faster than directly checking equivalence as soon as we have at least 204 schemes.

To get an idea how well the algorithm scales for larger values of n we have experimentally determined the size of the stabilizers in algorithm 2. Instead of iterating over the complete group $\mathrm{GL}(K,n)^3$, Algorithm 2 only iterates over stabilizers from the beginning on. In the case that the scheme contains at least one matrix of full rank, which in the data set are slightly more than half of the schemes, we have to iterate over all elements of $\mathrm{GL}(K,n)^2$ in the worst case. However, this is a very pessimistic upper bound. The size of $\mathrm{GL}(\mathbb{Z}_2,3)^2$ is 28,224, whereas the average size of the stabilizer we actually iterate over is 460.

In the second case we iterate over S in line 15, which again cannot exceed the size of $\mathrm{GL}(K,n)^2$ and on the data set has on average 135 elements. We also have to iterate over S' in line 18. The size of S' is bounded by $|\mathrm{GL}(K,n)^3|$, which in our case is 4,741,632. However, for a sample of the data set the largest stabilizer that occurred contains 576 elements and on average this stabilizer has 274 elements.

In summary, naively computing a normal form by simply iterating over all elements of $\mathrm{GL}(K,n)^3$ for each row will take time $\mathrm{O}(r|\mathrm{GL}(K,n)|^3)$, where r is the length of the scheme. Assuming that after $\mathrm{O}(1)$ iterations of Alg. 1 we are left with a stabilizer of size $\mathrm{O}(1)$, the cost of Alg. 1 is only $\mathrm{O}(r+|\mathrm{GL}(K,n)|^3)$. We cannot prove that the stabilizers become so small so quickly, but the assumption is consistent with our experiments. Finally, assuming that the solution space in line 18 of Alg. 2 has at most $\mathrm{O}(|\mathrm{GL}(K,n)|^2)$ elements, Alg. 2 pushes the total cost of computing a normal form down to $\mathrm{O}(r+|\mathrm{GL}(K,n)|^2)$. Again, we cannot prove any such claim about line 18, but the assumption is consistent with our experiments.

References

1. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 522–539 (2021). https://doi.org/10.1137/1.9781611976465.32

2. Berger, G.O., Absil, P.A., De Lathauwer, L., Jungers, R.M., Van Barel, M.: Equivalent polyadic decompositions of matrix multiplication tensors. J. Comput. Appl. Math. **406**, Paper No. 113941, 17 (2022). https://doi.org/10.1016/j.cam.2021.113941
3. Bläser, M.: On the complexity of the multiplication of matrices of small formats. J. Complexity **19**(1), 43–60 (2003). https://doi.org/10.1016/S0885-064X(02)00007-9
4. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic complexity theory, vol. 315. Springer Science & Business Media (2013)
5. Courtois, N.T., Bard, G.V., Hulme, D.: A new general-purpose method to multiply 3x3 matrices using only 23 multiplications (2011). https://doi.org/10.48550/ARXIV.1108.2830, https://arxiv.org/abs/1108.2830
6. de Groote, H.F.: On varieties of optimal algorithms for the computation of bilinear mappings ii. optimal algorithms for 2×2-matrix multiplication. Theoretical Computer Science **7**(2), 127–148 (1978). https://doi.org/https://doi.org/10.1016/0304-3975(78)90045-2
7. Heule, M.J.H., Kauers, M., Seidl, M.: New ways to multiply 3×3-matrices. J. Symbolic Comput. **104**, 899–916 (2021). https://doi.org/10.1016/j.jsc.2020.10.003
8. Johnson, R.W., McLoughlin, A.M.: Noncommutative bilinear algorithms for 3×3 matrix multiplication. SIAM J. Comput. **15**(2), 595–603 (1986). https://doi.org/10.1137/0215043
9. Laderman, J.D.: A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. Bull. Amer. Math. Soc. **82**(1), 126–128 (1976). https://doi.org/10.1090/S0002-9904-1976-13988-2
10. Oh, J., Kim, J., Moon, B.R.: On the inequivalence of bilinear algorithms for 3×3 matrix multiplication. Inform. Process. Lett. **113**(17), 640–645 (2013). https://doi.org/10.1016/j.ipl.2013.05.011
11. Rosowski, A.: Fast commutative matrix algorithm (2019). https://doi.org/10.48550/ARXIV.1904.07683
12. Smirnov, A.V.: The bilinear complexity and practical algorithms for matrix multiplication. Zh. Vychisl. Mat. Mat. Fiz. **53**(12), 1970–1984 (2013). https://doi.org/10.1134/S0965542513120129
13. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354–356 (1969). https://doi.org/10.1007/BF02165411
14. Winograd, S.: On multiplication of 2×2 matrices. Linear Algebra and its Applications **4**(4), 381–388 (1971). https://doi.org/https://doi.org/10.1016/0024-3795(71)90009-7