Sparsifying the Operators of Fast Matrix Multiplication Algorithms

Gal Beniamini
The Hebrew University of Jerusalem
gal.beniamini@mail.huji.ac.il

Nathan Cheng
University of California at Berkeley
ncheng@berkeley.edu

Oded Schwartz
The Hebrew University of Jerusalem
odedsc@cs.huji.ac.il

Elaye Karstadt
The Hebrew University of Jerusalem
elaye.karstadt@mail.huji.ac.il

1 INTRODUCTION
Matrix multiplication is a fundamental computation kernel, used in many fields ranging from imaging to signal processing and artificial neural networks. The need to improve performance has attracted much attention from the science and engineering communities. Strassen’s discovery of the first sub-cubic algorithm [38] sparked intensive research into the complexity of matrix multiplication algorithms (cf. [1–3, 9–13, 16, 18, 20–23, 25–27, 30, 31, 33–37, 39, 42, 43]).

The research efforts can be divided into two main branches. The first revolves around the search for asymptotic upper bounds on the arithmetic complexity of matrix multiplication (cf. [9–11, 16, 27, 37, 39, 42]). This approach focuses on asymptotics, typically disregarding the hidden constants of the algorithms and other aspects of practical importance. Many of these algorithms remain highly theoretical due to their large hidden constants, and furthermore, they apply only to matrices of very high dimensions.

In contrast, the second branch focuses on obtaining matrix multiplication algorithms that are both asymptotically fast and practical. This requires the algorithms to have reasonable hidden constants that are applicable even to small instances (cf. [1–3, 18, 20–23, 25, 26, 30, 31, 33–36, 43]).

1.1 Previous work
Reducing the leading coefficients. Winograd [43] reduced the leading coefficient of Strassen’s algorithm’s arithmetic complexity from 7 to 6 by decreasing the number of additions and subtractions in the 2 × 2 base case from 18 to 15. Later, Bodrato [4] introduced the intermediate representation method, that successfully reduces the leading coefficient to 5, for repeated squaring and chain matrix multiplication. Cenk and Hasan [7] presented a non-uniform implementation of Strassen-Winograd’s algorithm [43], which also reduces the leading coefficient from 6 to 5, but incurs additional penalties such as a larger memory footprint and higher communication costs. Independently, Karstadt and Schwartz [22, 23] used a technique similar to Bodrato’s, and obtained a matrix multiplication algorithm with a 2 × 2 base case, using 7 multiplications, and a leading coefficient of 5. Their method also applies to other base cases, improving the leading coefficients of multiple algorithms. Beniamini and Schwartz [1] introduced the decomposed recursive bilinear framework, which generalizes [22, 23]. Their technique allows a further reduction of the leading coefficient, yielding several fast matrix multiplication algorithms with a leading coefficient of 2, matching that of the classical algorithm.

Lower bounds on leading coefficients. Probert [32] proved that 15 additions are necessary for any recursive-bilinear matrix multiplication algorithm with a 2 × 2 base case using 7 multiplications over \( \mathbb{F}_2 \), which corresponds to a leading coefficient of 6. This was later matched by Bshouty [6], who used a different technique to obtain the same lower bound over an arbitrary ring. Both cases have been interpreted as a proof of optimality for the leading coefficient of Winograd’s algorithm [43].

Karstadt and Schwartz’s (2, 2, 2, 7)-algorithm[2] [22, 23] requires 12 additions (thus having a leading coefficient of 5) and seemingly contradicts these lower bounds. Indeed, they showed that these lower bounds [6, 32] do not hold under alternative basis multiplication. In addition, they extended the lower bounds to apply to algorithms that utilize basis transformations, and showed that 12 additions are necessary for any recursive-bilinear matrix multiplication algorithm with a 2 × 2 base case using 7 multiplications, regardless of basis. Thus proving a lower bound of 5 on the leading coefficient of such algorithms.

1See Section 2.1 for the connection between the number of additions and the leading coefficient.
2See Section 2.1 for definition.
Table 1: Examples of improved leading coefficients

| Algorithm | Leading Monomial | Arithmetic Operations | Leading Coefficient | Improvement |
|-----------|-----------------|-----------------------|---------------------|-------------|
| (2, 2, 2, 7) [38] | \(n^{3/2} \approx n^{2.80735}\) | Original [22, 23] | Here | Original [22, 23] | Here | [22, 23] | Here |
| (3, 2, 3, 15) [2] | \(n\log_2 7 \approx n^{2.81076}\) | 18 | 12 | 12 | 7 | 5 | 5 | 28.57% | 28.57% |
| (4, 2, 3, 20) [35] | \(n\log_2 20.5 \approx n^{2.82789}\) | 64 | 52 | 39 | 9.61 | 7.94 | 6.17 | 17.37% | 35.84% |
| (3, 3, 3, 23) [2] | \(n\log_3 23 \approx n^{2.85404}\) | 78 | 58 | 51 | 8.9 | 7.46 | 5.88 | 16.17% | 33.96% |
| (6, 3, 3, 40) [35] | \(n\log_4 40.5 \approx n^{2.77429}\) | 87 | 75 | 66 | 7.21 | 6.57 | 5.71 | 8.87% | 20.79% |
| (2, 2, 2, 7) [38] | \(n^{3/2} \approx n^{2.80735}\) | 1246 | 202 | 190 | 55.63 | 9.36 | 8.9 | 83.17% | 84.01% |

The leading monomial of rectangular \((n, m, k; t)\)-algorithms refers to their composition [19] into square \((nmk, nmk, nmk; t^2)\)-algorithms. The improvement column is the ratio between the new and the original leading coefficients of the arithmetic complexity. See Table 2 for a full list of results.

Beniamini and Schwartz [1] extended the lower bound to the generalized setting, in which the input and output can be transformed to a basis of larger dimension. They also found that the leading coefficient of any such algorithm with a \(2 \times 2\) base case using 7 multiplications is at least 5.

**Obtaining alternative basis algorithms.** Recursive-bilinear algorithms can be described by a triplet of matrices, dubbed the encoding and decoding matrices (see Section 2.1). The alternative basis technique [22, 23] utilizes a decomposition of each of these matrices into a pair of matrices – a basis transformation, and a sparse encoding or decoding matrix. Once a decomposition is found, applying the algorithm is straightforward (see Section 2.1).

The leading coefficient of the arithmetic complexity is determined by the number of non-zero (and non-singleton) entries in each of the encoding/decoding matrices, while the basis transformations only affect the low order terms of the arithmetic complexity (see Section 2.1). Thus, reducing the leading coefficient of fast matrix multiplication algorithms translates to the matrix sparsiﬁcation (MS) problem.

**Matrix sparsiﬁcation.** Unfortunately, matrix sparsiﬁcation is NP-Hard to solve [28] and NP-Hard to approximate to within a factor of \(2^{\log_{n^{1.8}} n}\) [15] (Over \(\mathcal{Q}\), assuming NP does not admit quasi-polynomial time deterministic algorithms). Despite the problem being NP-hard, search heuristics can be leveraged to obtain bases which signiﬁcantly sparsiﬁes the encoding/decoding matrices of fast matrix multiplication algorithms with small base cases.

There are a few heuristics that can solve the problem, under severe assumptions, such as the full rank of any square submatrix, and requiring that the rank of each submatrix be equal to the size of the largest matching in the induced bipartite graph (cf., [8, 17, 28, 29]). These assumptions rarely hold in practice, and speciﬁcally, do not apply to any matrix multiplication algorithm we know.

Gottlieb and Neylon’s algorithm [15] sparsiﬁes an \(n \times m\) matrix with no assumptions about the input. It does so by using calls to an oracle for the sparsest independent vector problem.

**1.2 Our contribution.**

We obtain three new methods for matrix sparsiﬁcation, based on Gottlieb and Neylon’s [15] matrix sparsiﬁcation algorithm. We apply these methods to multiple matrix multiplication algorithms and obtain novel alternative-basis algorithms, often resulting in arithmetic complexity with leading coefﬁcients superior to those known previously (See Table 1, Table 2, and Appendix A).

The ﬁrst two methods were obtained by the introduction of new solutions to the Sparsest Independent Vector problem, which were then used as oracles for Gottlieb and Neylon’s algorithm. As matrix sparsiﬁcation is known to be NP-Hard, it is no surprise that these methods exhibit exponential worst case complexity. Nevertheless, they perform well in practice on the encoding/decoding matrices of fast matrix multiplication algorithms.

Our third method for matrix sparsiﬁcation simultaneously mini- mizes the number of non-singular values in the matrix. This method does not guarantee an optimal solution for matrix sparsiﬁcation. Nonetheless, it obtains solutions with the same (and, in some cases, better) leading coefﬁcients than the former two methods when applied to many of the fast matrix multiplication algorithms in our corpus, and runs signiﬁcantly faster than the ﬁrst two when implemented using Z3 [14]. For completeness, we also present the sparsiﬁcation heuristic used in [22, 23].

**1.3 Paper Organization.**

In Section 2, we recall preliminaries regarding fast matrix multiplication and recursive-bilinear algorithms, followed by a summary of the Alternative Basis technique [22, 23]. We then present Matrix Sparsiﬁcation (MS, Problem 2.13), alongside Gottlieb and Neylon’s [15] algorithm for solving MS by relying on an oracle for Sparsest Independent Vector (SIV, Problem 2.15). In Section 3 we present our two algorithms (Algorithms 3 and 4) for implementing SIV. In Section 4, we introduce Algorithm 5 - the sparsiﬁcation heuristic of [22, 23], and a new efﬁcient heuristic for sparsiﬁng matrices while simultaneously minimizing non-singular values (Algorithm 6). In Section 5 we present the resulting fast matrix multiplication algorithms. Section 6 contains a discussion and plans for future work.

**2 PRELIMINARIES**

**2.1 Encoding and Decoding matrices.**

Fast matrix multiplication algorithms are recursive divide-and-conquer algorithms, which utilize a small base case. We use the
notation \( (n_0, m_0, k_0; t_0) \)-algorithm to refer to an algorithm mul-
tiplying \( n_0 \times m_0 \) by \( m_0 \times k_0 \) matrices in its base case, using \( t_0 \) scalar
multiplications, where \( n_0, m_0, k_0 \) and \( t_0 \) are fixed positive integers.

When multiplying \( n \times m \) by \( m \times k \) matrix multiplication, the
algorithm splits each matrix into blocks (each of size \( \frac{n}{m_0} \times \frac{m}{m_0} \)
and \( \frac{m_0}{m} \times \frac{k_0}{k} \), respectively), and works block-wise, according to
the base algorithm. Additions and subtractions in the base-case
algorithm become block-wise additions and subtractions. Similarly,
multiplication by a scalar become multiplication of a block matrix
by a scalar. Matrix multiplications in the algorithm are performed
via recursion.

Throughout this paper, we refer to an algorithm by its base case.
Hence, an \( (n, m, k; t) \)-algorithm may refer to either the algorithm’s
base case or the corresponding block recursive algorithm, as obvi-
ous from the context.

Fact 2.1. [22, 23] Let \( R \) be a ring, and let \( f : R^n \times R^m \rightarrow R^k \)
be a bilinear function that performs \( t \) multiplications. There exist
\( U \in R^{R \times n}, V \in R^{k \times m}, W \in R^{t \times k} \) such that
\[
\forall x \in R^n, y \in R^m, f(x, y) = W^T((U \cdot x) \odot (V \cdot y))
\]
where \( \odot \) is the element-wise product (Hadamard product).

Definition 2.2. [22, 23] (Encoding/Decoding matrices). We refer
to the matrix triplet \((U, V, W) \) of a recursive-bilinear algorithm (see
Fact 2.1) as its encoding/decoding matrices \((U, V) \) and \( W \) is the decoding
circuit.

Notation 2.3. [1] Denote the number of nonzero entries in a
matrix by \( nnz(A) \), and the number of non-singleton (i.e., \( \neq 1 \))
entries in a matrix by \( nns(A) \). Let the number of rows/columns be
\( \text{nrows}(A) \) and \( \text{ncols}(A) \), respectively.

Remark 2.4. [1] The number of linear operations used by a bilin-
arithmic algorithm is determined by its encoding/decoding matrices.
The number of arithmetic operations performed by each of the encodings is:

\[
\begin{align*}
\text{Ops}_U &= nnz(U) + nns(U) - \text{nrows}(U) \\
\text{Ops}_V &= nnz(V) + nns(V) - \text{nrows}(V)
\end{align*}
\]

The number of operations performed by the decoding is:

\[
\text{Ops}_W = nnz(W) + nns(W) - \text{ncols}(W)
\]

Remark 2.5. We assume that none of the rows of the \( U \), \( V \), and \( W \)
matrices is zero. This is because any zero row in \( U \), \( V \) is equivalent
to an identically 0 multiplicant, and any zero row in \( W \) is equivalent
to a multiplication that is never used in the output. Hence, such
rows can be omitted, resulting in asymptotically faster algorithms.

Corollary 2.6. [1] Let \( \text{ALG} \) be an \( (n_0, m_0, k_0; t_0) \)-algorithm that
performs \( \text{Ops}_U, \text{Ops}_V, \text{Ops}_W \) linear operations at the base case and let
\( n = n_0^l \), \( m = m_0^l \), \( k = k_0^l \) \( (l \in \mathbb{N}) \). The arithmetic complexity of \( \text{ALG} \) is

\[
F(n, m, k) = \left[ 1 + \frac{\text{Ops}_U}{t_0 - n_0 m_0} + \frac{\text{Ops}_U}{t_0 - m_0 k_0} + \frac{\text{Ops}_W}{t_0 - n_0 k_0} \right] t_0^l
\]

Definition 2.7. Let \( P_{IJ} \) denote the permutation matrix that ex-
changes row-order for column-order of the vectorization of an \( I \times J \)
matrix.

Lemma 2.8. [19] Let \( \langle U, V, W \rangle \) be the encoding/decoding mat-
rices of an \( (m, k; t) \)-algorithm. Then \( WP_{n,n} \), \( U, V_{P_{n,k}} \) are the
encoding/decoding matrices of an \( (m, k; t) \)-algorithm.

Remark 2.9. In addition to Lemma 2.8, Hopcroft and Musin-
ski [19] proved that any \( (n, m, k; t) \)-algorithm defines algorithms
for all permutations of \( n, m, \) and \( k \). Note, however, that while the
number of non-zero and non-singular entries does not change, it fol-
ows from Remark 2.4 and Corollary 2.6 that the leading coefficient
varies according to the dimensions of the encoding matrix.

2.2 Alternative Basis Matrix Multiplication.

Definition 2.10. [22, 23] Let \( R \) be a ring and let \( \phi, \psi, \upsilon \) be au-
tomorphisms of \( R^{n \times m}, R^{m \times k}, R^{n \times k} \) (respectively). We denote
a recursive bilinear matrix multiplication algorithm which takes
\( \phi(A), \psi(B) \) as inputs and outputs \( \psi(A \cdot B) \) using \( t \) multiplications
by \langle \phi, \psi, \upsilon \rangle. \langle \phi, \psi, \upsilon \rangle \) algorithm. This notation extends the \langle \phi, \psi, \upsilon \rangle-
algorithm notation, as the latter applies when the three basis trans-
formations are the identity map.

Given a recursive bilinear, \langle \phi, \psi, \upsilon \rangle \)-algorithm \text{ALG}, an
alternative basis matrix multiplication operates as follows:

Algorithm 1 Alternative Basis Matrix Multiplication Algorithm

\begin{itemize}
\item \textbf{Input:} \( A \in R^{n \times m}, B \in R^{m \times k} \)
\item \textbf{Output:} \( n \times k \) matrix \( C = A \cdot B \)
\item 1. \textbf{function} \text{Mult}(A, B)
\item 2. \( \hat{A} = \phi(A) \) \hspace{1cm} \( \triangleright R^{n \times m} \) basis transformation
\item 3. \( \hat{B} = \psi(B) \) \hspace{1cm} \( \triangleright R^{m \times k} \) basis transformation
\item 4. \( \hat{C} = \text{ALG}(\hat{A}, \hat{B}) \) \hspace{1cm} \( \triangleright \langle \phi, \psi, \upsilon \rangle \)-algorithm
\item 5. \( \tilde{C} = \upsilon^{-1}(\hat{C}) \) \hspace{1cm} \( \triangleright R^{n \times k} \) basis transformation
\item 6. \textbf{return} \( \tilde{C} \)
\end{itemize}

Lemma 2.11. [22, 23] Let \( R \) be a ring, and let \( \phi, \psi, \upsilon \) be au-
tomorphisms of \( R^{n \times m}, R^{m \times k}, R^{n \times k} \) (respectively). Then \((U, V, W) \) are
encoding/decoding matrices of an \( (n, m, k; t) \)-algorithm if and only
if \((\hat{U}, \hat{V}, \hat{W}, \hat{W}^\top) \) are encoding/decoding matrices of an \( (n, m, k; t) \)-
algorithm.

Alternative basis multiplication is fast since the basis transfor-
manations are fast and incur an asymptotically negligible overhead:

Claim 2.12. [22, 23] Let \( R \) be a ring, let \( \psi : R_{n \times m} \rightarrow R_{n \times m} \)
be a linear map, and let \( A \in R^{n \times m} \) where \( n = k_0^l, m = m_0^l \). The
complexity of \( \psi(A) \) is

\[
F(n, m) = \frac{q}{n_0 m_0} \cdot \text{nn}(n, m) \cdot \log_{n_0 m_0}(nm)
\]

where \( q \) is the number of linear operations performed.

2.3 Matrix Sparsification.

Finding a basis that minimizes the number of additions and sub-
tractions performed by a fast matrix multiplication algorithm is
equivalent, by Remark 2.4, to the Matrix Sparsification problem:
Problem 2.13. Matrix Sparsification Problem (MS): Let \( U \) be an \( n \times m \) matrix. The objective is to find an invertible matrix \( A \) such that \( A = \arg\max_{A \in GL_n} (\text{nnz}(AU)) \).

Remark 2.14. It is traditional to think of the matrices \( U, V, \) and \( W \) as “tall and skinny”, i.e., with \( n \geq m \). However, in the area of matrix sparsification, it is traditional to deal with matrices satisfying \( n \leq m \) and transformations applied from the left. However, since \( \text{nnz}(AU) = \text{nnz}(U^T A^T) \), we can simply apply MS to \( U^T \) and use \( A^T \) as our basis transformation. From now on, we will therefore switch to the convention \( n \leq m \) used in matrix sparsification.

To solve MS, we make use of Gottlieb and Neylon’s algorithm [15], which solves the matrix sparsification problem for \( n \times m \) matrices, by repeatedly invoking an oracle for the Sparsest Independent Vector problem (Problem 2.15).

Problem 2.15. Sparsest Independent Vector Problem (SIV): Let \( U \in \mathbb{R}^{n \times m} (n \leq m) \) and let \( \Omega = \{ \omega_1, \ldots, \omega_k \} \subset [m] \). Find a vector \( v \in \mathbb{R}^m \) s.t. \( v \) is in the row space of \( U \), \( v \) is not in the span of \( [U_{\omega_1}, \ldots, U_{\omega_k}] \), and \( v \) has a minimal number of nonzero entries.

Given a subroutine \( \text{SIV}(U, \Omega) \) which returns a pair \((v, i)\), where \( v \) is the sparse vector as required by SIV, and \( i \in [n] \setminus \Omega \) is an integer such that the \( i \)th row of \( U \) can be replaced by \( v \) without changing the span of \( U \). Then Algorithm 2 returns an exact solution for MS [15].

Algorithm 2 MS via SIV [15]

1. \( \text{procedure } \text{MS}(U) \)
2. \( \Omega \leftarrow \emptyset \)
3. \( \text{for } j = 1, \ldots, n \)
4. \( (v_j, i) \leftarrow \text{SIV}(U, \Omega) \)
5. \( \text{Replace } i \text{th row of } U \text{ with } v_j \)
6. \( \Omega \leftarrow \Omega \cup \{i\} \)
7. \text{return } U \)

3 OPTIMAL SPARSIFICATION METHODS

In this section, we reframe SIV as a problem of finding a maximal subset of columns of the input matrix \( U \) according to constraints given by \( \Omega \) (see Definition 3.2). We refer to such sets as \( \Omega \)-valid sets and show that \( \Omega \)-valid sets are tied to sparse independent vectors (Section 3.1) and that any algorithm which finds an \( \Omega \)-valid set of maximal cardinality can be used as an oracle in Algorithm 2. Finally, we show how to find maximal \( \Omega \)-valid sets (Section 3.2), and obtain two algorithms that solve SIV.

Recall that we use the convention that \( U \in \mathbb{F}^{n \times m} \) where \( n \leq m \) (see Remark 2.14). Throughout this section, we also assume that \( U \) is of full rank \( n \) and \( \Omega \subset [n] \).

Notation 3.1. For a set \( S \) and an integer \( k \), let \( C_k(S) \) denote the set of all subsets of \( S \) with \( k \) elements.

Definition 3.2. \( S \subset [m] \) is \( \Omega \)-valid if there exists \( i \notin \Omega \) such that \( U_{\omega_i} \) is in the span of rows \( \{U_{[m]\setminus\{i\}}, S\} \).

Formally, a set \( S \subset [m] \) is \( \Omega \)-valid if \( \exists \lambda \in \mathbb{F}^m \) with \( \text{supp}(\lambda) \subset \Omega \) s.t. \( \lambda^T U_{\omega_i} = 0 \) (where \( \text{supp}(\lambda) = \{i : \lambda_i \neq 0\} \)).

Notation 3.3. Given an \( \Omega \)-valid set \( S \), we will refer to a vector \( \lambda \in \mathbb{F}^m \) with \( \text{supp}(\lambda) \subset \Omega \) s.t. \( \lambda^T U_{\omega_i} = 0 \) as an \( \Omega \)-validator of \( S \).

Next, we provide a definition for vectors which are candidates for a solution of SIV:

Definition 3.4. A vector \( v \) in the row space of \( U \) is called \( \Omega \)-independent if \( v \) is not in the row space of \( U_{\Omega} \).

Note that any solution to SIV (Problem 2.15) is, by definition, an optimally sparse \( \Omega \)-independent vector.

Remark 3.5. Note that given a set \( S \subset [m] \), it is possible to verify whether \( S \) is \( \Omega \)-valid and find an appropriate \( \Omega \)-validator for it in cubic time (e.g., via Gaussian elimination).

3.1 Sparse Independent Vectors and maximal \( \Omega \)-valid sets.

The crux of our algorithms lies in the idea of finding an \( \Omega \)-valid set of maximal cardinality and using it to compute a solution for SIV, which can then be used by Algorithm 2. The connection between \( \Omega \)-valid sets and \( \Omega \)-independent vectors is given by the following lemmas:

Lemma 3.6. Let \( v \in \mathbb{F}^m \) be an \( \Omega \)-independent vector. Then the set \( S = \{j : v_j = 0\} \) is an \( \Omega \)-valid set of size zeros \( v \).

Proof. By Definition 3.4, there exists a vector \( \lambda \in \mathbb{F}^m \) s.t. \( v = \sum_{i=1}^k \lambda_i U_i \) (i.e., \( v = \lambda^T U \)) and \( \lambda_i \neq 0 \) for some \( i_0 \notin \Omega \) (hence \( \text{supp}(\lambda) \not\subset \Omega \)). Thus, \( \lambda \) is an \( \Omega \)-validator of \( S \), and therefore, \( S \) is \( \Omega \)-valid.

Lemma 3.7. Let \( S \subset [m] \) be an \( \Omega \)-valid set and let \( \lambda \in \mathbb{F}^m \) be an \( \Omega \)-validator of \( S \). Then \( v = \lambda^T U \) is an \( \Omega \)-independent vector with at least \( |S| \) zero entries.

Proof. Since \( S \) is valid, there exists \( \lambda \in \mathbb{F}^m \) s.t. \( \lambda^T U_{\omega_i} = 0 \) and \( \text{supp}(\lambda) \not\subset \Omega \). Denote \( v = \lambda^T U \). By definition, \( v \) has at least \( |S| \) zero entries since \( \forall i \in S, v_i = \left(\lambda^T U_{\omega_i}\right)_i = 0 \). Next we show that \( v \) is \( \Omega \)-independent. Note that, \( v = \lambda^T U = \sum_{i=1}^k \lambda_i U_i \), is in the row space of \( U \) since it is a linear combination of the rows of \( U \). Furthermore, since \( \text{supp}(\lambda) \not\subset \Omega \), there exists \( i_0 \notin \Omega \) s.t. \( \lambda_{i_0} \neq 0 \). Therefore, \( v \) is not in the row span of \( U_{\Omega} \), which is to say, \( v \) is not in the span of \( U_{\Omega} \).

Corollary 3.8. Let \( M \subset [m] \) be a maximal \( \Omega \)-valid set (i.e., \( M \) is not a subset of any other \( \Omega \)-valid set), and let \( v \in \mathbb{F}^m \) be an \( \Omega \)-independent vector s.t. \( \forall i \in M, v_i \neq 0 \). Then \( \forall j \notin M, v_j = 0 \).

Proof. Denote the set of indices of zero entries of \( v \) by \( M' = \{j : v_j = 0\} \). Since \( v \) is \( \Omega \)-independent, Lemma 3.6 yields that \( M' \) is valid. Hence, by maximality of \( M, M = M' \) and \( |M| = |M'| \). Therefore, \( \forall i \in [m], v_i = 0 \) if, and only if, \( i \in M \).

Corollary 3.9. Let \( M \subset [m] \) be a maximal \( \Omega \)-valid set and let \( \lambda \in \mathbb{F}^m \) be an \( \Omega \)-validator of \( M \). Then \( v = \lambda^T U \) is an \( \Omega \)-independent vector with exactly \( |M| \) zero entries.

Proof. Follows directly from Lemma 3.7 and Corollary 3.8.

□
The final two claims will show how \( \Omega \)-validity can serve as an oracle for Algorithm 2. Recall that Algorithm 2 uses an oracle which returns a pair \((\nu, i)\), where \( \nu \) is an optimally sparse \( \Omega \)-independent vector, and replacing the \( i \)th row of \( U \) with \( \nu \) does not change the row span of \( U \). The next claim shows that a maximally sparse \( \Omega \)-independent vector is equivalent to an \( \Omega \)-valid set of maximal cardinality.

**Claim 3.10.** An \( \Omega \)-independent vector \( \nu \in \mathbb{F}^m \) is optimally sparse if, and only if, \( M = \{ i : \nu_i = 0 \} \) is an \( \Omega \)-valid set of maximal cardinality.

**Proof.** First, assume that \( \nu \in \mathbb{F}^m \) is a maximally sparse \( \Omega \)-independent vector (i.e., for any \( \Omega \)-independent vector \( u \), \( \text{zeros}(u) \subseteq \text{zeros}(\nu) \)). From Lemma 3.6, we know that \( M \) is \( \Omega \)-valid. Lemma 3.7 shows that if there exists an \( \Omega \)-valid set \( S \) s.t. \( |M| < |S| \), then there also exists an \( \Omega \)-independent vector \( u \in \mathbb{F}^m \) s.t. \( \text{zeros}(u) \geq |S| \cap \text{zeros}(\nu) \). This contradicts \( \nu \) being a maximally sparse \( \Omega \)-independent vector.

Now, assume that \( M \) is an \( \Omega \)-valid set of maximal cardinality (i.e., for any \( \Omega \)-valid set \( S \), \( |S| \leq |M| \)) and let \( \lambda_M \) be an \( \Omega \)-validator of \( M \). By Corollary 3.9, \( \nu \in \lambda_M^U \) is an \( \Omega \)-independent vector with exactly \( |M| \) zero entries. Assume by contradiction that exists \( u \in \mathbb{F}^m \) with \( \nu \) > \( |M| \) zero entries, then by Lemma 3.6, there is an \( \Omega \)-valid set \( S \subset |M| < |S| \), in contradiction to \( M \) being an \( \Omega \)-valid set of maximal cardinality. Therefore, \( \nu_M = \lambda_M^U \) is a maximally sparse \( \Omega \)-independent vector.

The following claim shows that given an \( \Omega \)-valid set, \( S \), and its corresponding \( \Omega \)-independent vector \( \nu \) (as in Lemma 3.7), the support of the \( \Omega \)-validator of \( S \) can be used to find an index \( i \) s.t. the \( i \)th row of \( U \) can be replaced with \( \nu \) without changing the row span of \( U \).

**Claim 3.11.** Let \( S \) be an \( \Omega \)-valid set, let \( \lambda \) be an \( \Omega \)-validator of \( S \), and let \( \nu = \lambda^TU \). Then for any \( i \in \text{supp}(\lambda) \setminus \Omega \), replacing row \( i \) of \( U \) with \( \nu \) does not change the row span of \( U \). That is:

\[
\text{span}(\text{rows}(U)) = \text{span}(\text{rows}(U_{[n] \setminus \{i\},:}) \cup \{\nu\})
\]

**Proof.** Fix \( \nu \in \text{supp}(\lambda) \setminus \Omega \). Since \( \nu \) is a linear combination of rows of \( U \) and \( \lambda_i \neq 0 \), \( u \in \text{span}(\text{rows}(U_{[n] \setminus \{i\},:}) \cup \{\nu\}) \), for any \( u \in \text{span}(\text{rows}(U)) \). Now, let \( \alpha \in \mathbb{F}^m \) be the vector \( \alpha_i = -\lambda_i \) (for \( j \neq i \)) and \( \alpha_i = 0 \). Then \( w = \alpha^TU \in \text{span}(\text{rows}(U_{[n] \setminus \{i\},:}) \cup \{\nu\}) \), therefore \( w + \nu = \lambda_i U_{i,:} + \nu \in \text{span}(\text{rows}(U_{[n] \setminus \{i\},:}) \cup \{\nu\}) \). Hence, \( \text{span}(\text{rows}(U)) = \text{span}(\text{rows}(U_{[n] \setminus \{i\},:}) \cup \{\nu\}) \).

Therefore, any algorithm which finds an \( \Omega \)-valid set of maximal cardinality is an oracle for Algorithm 2.

### 3.2 Computing maximal \( \Omega \)-valid sets.

Given a maximal \( \Omega \)-valid set, we now have the tools to compute optimally sparse \( \Omega \)-independent vectors. As the next stage, we show how to compute a maximal \( \Omega \)-valid set \( M \) using a small subset of columns \( S \subseteq M \). The key intuition here is that if \( \lambda \in \mathbb{F}^m \) is an \( \Omega \)-validator of \( S \), then \( \lambda \) is orthogonal to all columns indexed by \( S \) (since \( \lambda^TU(:,S) = 0 \)), and any linear combinations of columns of \( S \). This leads to the following extension of sets:

**Definition 3.12.** Let \( S \subset [m] \). We define the extension of \( S, E(S) \), to be the largest set \( E \subset [m] \) s.t. \( \text{span}(\text{col}(U_{:,S})) = \text{span}(\text{col}(U_{:,E})) \).

**Lemma 3.13.** Let \( S \subset [m] \). Then \( S \) is \( \Omega \)-valid if, and only if, \( E(S) \) is \( \Omega \)-valid.

**Proof.** Assume \( E(S) \) is \( \Omega \)-valid. By definition of \( \Omega \)-validity, exists a vector \( \lambda \in \mathbb{F}^m \) s.t. \( \text{supp}(\lambda) \not\subset \Omega \) and \( \lambda^TU_{:,E(S)} = 0 \). Since \( S \subseteq E(S) \), \( \lambda^TU_{:,S} = 0 \), therefore, \( S \) is valid.

Let \( S \subset [m] \) be an \( \Omega \)-valid set, and let \( \lambda \in \mathbb{F}^m \) with \( \text{supp}(\lambda) \not\subset \Omega \) s.t. \( \lambda^TU_{:,S} = 0 \). Since \( \text{col}(U(:,E(S))) = \text{col}(U(:,S)) \), all columns indexed by \( E(S) \) are linear combinations of the columns indexed by \( S \). Since \( \lambda \) is orthogonal to all columns of \( U \) indexed by \( S \), it is also orthogonal to all their linear combinations. Therefore, \( \lambda^TU_{:,E(S)} = 0 \). Hence \( E(S) \) is valid.

Next we show that the search for a maximal \( \Omega \)-valid set can be reduced to the search over maximal extensions of sets of size \( n - 1 \).

**Remark 3.14.** Note that \( \text{rank}(U_{:,S}) \leq n - 1 \) for any \( \Omega \)-valid set \( S \). This is due to the fact that if \( \text{rank}(U_{:,S}) = n \) then \( \lambda^TU_{:,S} = 0 \) implies that \( \lambda = 0 \) since the rows of \( U \) are linearly independent.

**Lemma 3.15.** Let \( S \) be an \( \Omega \)-valid set and let \( \lambda \in \mathbb{F}^m \) be an \( \Omega \)-validator of \( S \). Then

\[
E(S) \subset \{ i : \left( \lambda^TU \right)_i = 0 \}
\]

**Proof.** Let \( D = \{ i : \left( \lambda^TU \right)_i = 0 \} \). By Definition 3.12, columns indexed by \( E(S) \) are linear combinations of the columns indexed by \( S \) and \( \lambda \) is orthogonal to all columns of \( U_{:,S} \) (and their linear combinations). Hence, \( \lambda^TU_{:,E(S)} = 0 \) and \( E(S) \subset D \).

**Lemma 3.16.** Let \( S \) be an \( \Omega \)-valid set s.t. \( \text{rank}(U_{:,S}) = n - 1 \), and let \( D \) be an \( \Omega \)-valid set s.t. \( S \subset D \). Then \( D \subset E(S) \).

**Proof.** Since \( S \subset D, n - 1 = \text{rank}(U_{:,S}) \leq \text{rank}(U_{:,D}) \).

However, from Remark 3.14, we know that \( \text{rank}(U_{:,D}) \leq n - 1 \), therefore, \( \text{span}(\text{col}(U_{:,S})) = \text{span}(\text{col}(U_{:,D})) \). Hence, by definition, \( D \subset E(S) \).

**Corollary 3.17.** Let \( S \) be an \( \Omega \)-valid set s.t. \( \text{rank}(U_{:,S}) = n - 1 \), and let \( \lambda \in \mathbb{F}^m \) be an \( \Omega \)-validator of \( S \). Then

\[
E(S) = \{ i : \left( \lambda^TU \right)_i = 0 \}
\]

**Proof.** This is a direct result of Lemma 3.15 and Lemma 3.16.

Note that Corollary 3.17 gives us the tools to quickly compute the extension of any \( \Omega \)-valid set \( S \) such that \( \text{rank}(U_{:,S}) = n - 1 \). Next we prove that any maximal \( \Omega \)-valid set is an extension of an \( \Omega \)-valid set of \( n - 1 \) linearly independent columns of \( U \).

**Claim 3.18.** Let \( S \subset [m] \) be a maximal \( \Omega \)-valid set, then

\[
\text{rank}(U_{:,S}) = n - 1
\]
Proof. Let $S \subseteq \{m\}$ be a maximal $\Omega$-valid set, and let $i_0 \notin \Omega$ such that $U_{i_0, s} \in \text{span}\{U_{i_0, s_0} : s_0 \in S, \text{rank}(U_{i_0, s}) = \text{rank}(U_{i_0, s_0}) = n - r\}$ (such $i_0$ exists by definition of an $\Omega$-valid set). Suppose, by contradiction, that rank$(U_{i_0, s}) = n - r$ for some $r > 1$.

Note that since $U_{i_0, s}$ is in the row span $U_{[n], i_0, s}$, $\text{rank}(U_{i_0, s}) = \text{rank}(U_{[n], i_0, s}) = n - r$. Therefore, exists $s_0 \subset S$ s.t. $|S| = n - r$ and rank$(U_{i_0, s_0}) = n - r$.

Let $Q \subseteq \{m\}\setminus \{s_t\}$ s.t. $|Q| = r - 1$, rank$(U_{[n], i_0, Q}) = r - 1$, and each column indexed by $Q$ is not in the column span of $U_{[n], i_0, s}$. Such $Q$ exists because the matrix $U_{[n], (i_0)}$ has full rank $n - 1$ (since $U$ is of full row rank $n$).

Since the matrix $U_{[n], (i_0)} : S_{Q} \cup Q$ is a square $n - 1 \times n - 1$ matrix of full rank, $U_{i_0, S_{Q}\cup Q}$ is in the span of $\text{row}(U_{[n], (i_0)} : S_{Q}\cup Q)$. Therefore, $S_0 \subset Q$ is an $\Omega$-valid set.

By Lemma 3.13, the extension of $S_0 \cup Q$ is also valid. Furthermore, $S \cup Q \subseteq E(S_0 \cup Q)$ because we have chosen $S_0$ s.t. it spans the same column space as $S$. However, by construction of $Q$, we know that $S \cap Q = \emptyset$, meaning that $|E(S_0 \cup Q)| \geq |S \cup Q| > |S|$. This in contradiction to maximality of $S$.

**Corollary 3.19.** Let $S \subseteq \{m\}$ be a maximal $\Omega$-valid set and let $C \subset S$ s.t. rank$(U_{i, C}) = n - 1$. Then $S = E(C)$.

**Proof.** This is a direct result of Corollary 3.19.

### 3.3 First algorithm for SIV.

Our first algorithm performs an exhaustive search over all maximal $\Omega$-valid sets in order find one with maximal cardinality. This is a result of the observation given by Claim 3.10, which states that any solution to SIV is tied to an $\Omega$-valid set of maximal cardinality (and vice versa). The search is done using by combining Corollary 3.20, which states that any maximal $\Omega$-valid set is the extension of an $\Omega$-valid set of $n - 1$ independent columns, and Corollary 3.17, which provides a method to compute such extension.

**Lemma 3.21.** Algorithm 3 iterates over all maximal $\Omega$-valid sets.

**Proof.** By Corollary 3.20, for any maximal $\Omega$-valid set $E$, there exist an $\Omega$-valid set $C \in C_{n-1}(\{m\})$ s.t. rank$(U_{i, C}) = n - 1$ and $E$ is the extension of $C$. Therefore, the algorithm iterates over all $\Omega$-valid sets $C \in C_{n-1}(\{m\})$ s.t. rank$(U_{i, C}) = n - 1$. Furthermore, by Corollary 3.17, if rank$(U_{i, C}) = n - 1$ and $\lambda$ is an $\Omega$-validator of $C$ then $E(C) = \{i : \lambda^TU = 0\}$. The algorithm performs this computation at lines 11-13. Hence, the algorithm iterates over all $\Omega$-valid sets.

**Algorithm 3 Sparsest Independent Vector (1)**

1. procedure SIV($U$, $\Omega$)
2. sparsity $\leftarrow 0$
3. sparsest $\leftarrow$ null
4. $i \leftarrow$ null
5. for $C \in C_{n-1}(\{1, ..., m\})$
6. if rank$(U_{i, C}) < n - 1$ or $C$ is not $\Omega$-valid
7. continue
8. $\lambda \leftarrow \Omega$-validator of $C$
9. $v \leftarrow \lambda^TU$
10. $E \leftarrow \{i : v_i = 0\}$
11. if $|E| >$ sparsity
12. sparsity $\leftarrow |E|
13. sparsest $\leftarrow v$
14. return $(\lambda, i)$

**Theorem 3.22.** Algorithm 3 produces an optimal solution to SIV, and is an oracle for Algorithm 2.

**Proof.** By Lemma 3.21, Algorithm 3 iterates over all maximal $\Omega$-valid sets. Lines 14-17 check whether a given $\Omega$-valid set has greater cardinality than any previously found maximal $\Omega$-valid set and if it does, the algorithm choose this set as a working solution. Hence, at the end of the algorithm, the chosen vector $v$ correlates to a maximal cardinality $\Omega$-valid set. By Claim 3.10, $v$ is an optimal solution to SIV (a maximally sparse $\Omega$-independent vector) if, and only if, the set $E = \{i : v_i = 0\}$ is an $\Omega$-valid set of maximal cardinality. Therefore, the vector chosen at the end of the algorithm is a maximally sparse $\Omega$-independent vector. Finally, by Claim 3.11, the pair $(\lambda, i)$ serves as the oracle for SIV required by Algorithm 2.

### 3.4 Implementation of our first optimal algorithm.

In order for Algorithm 3 to perform well, we have added a blacklist to the algorithm’s operation. Since the maximal $\Omega$-valid sets are generated by computing the extension (Definition 3.12) of $n - 1$ independent columns, once a given $\Omega$-valid set is found, we wish to blacklist all of its subsets of size $n - 1$ since we need not revisit that extension. However, in addition to memory costs, looking up an element in the blacklist incurs a significant overhead as the blacklist grows. To address this problem, rather than storing all subsets $C_{n-1}$ of a given set $S$, we store $S$ itself in the blacklist, in which case $C$ is not blacklisted if $V B \in \text{blacklist} \subseteq B$. Despite this measure, in some cases the blacklist still grew too large, so we and imposed a limit on the maximum size of the blacklist, storing only the $M$ largest sets found so far.

### 3.5 Second algorithm for SIV.

While our first algorithm performs well in many cases, we have found that it performs poorly when the largest $\Omega$-valid set is very large. In such cases the algorithm quickly finds the correct solution, but then continues its exhaustive search for a very long time. Our second algorithm is slightly simpler and avoids this inefficiency by using a top-down approach, searching for $\Omega$-valid sets in descending order of cardinality to find an $\Omega$-valid set of maximal
cardinality. Just like our first algorithm, it relies on the observation of Claim 3.10, which ties any solution of SIV (maximally sparse, \(\Omega\)-independent vector) to an \(\Omega\)-valid set of maximal cardinality.

**Algorithm 4** Sparsest Independent Vector (2)

1. procedure SIV \((U, \Omega)\)
2. \(\textbf{for } z = n - 1, \ldots, 1\)
3. \(\textbf{for } i \in C \subset \{1, \ldots, m\}\)
4. \(\textbf{if } \text{rank} (U, C) = n - 1 \text{ and } C \text{ is } \Omega\)-valid
5. \(\lambda \leftarrow \Omega\)-validator of \(C\)
6. \(\nu \leftarrow \lambda^T U\)
7. \(i \leftarrow \text{any element of } \text{supp}(\lambda) \setminus \Omega\)
8. \(\text{return } (\nu, i)\)

To prove the correctness of our Algorithm 4, we use the following lemma, which provides bounds on the size of a maximal \(\Omega\)-valid set.

**Lemma 3.23.** Let \(S \subset [m] \text{ be a maximal } \Omega\)-valid set, then \(n - 1 \leq |S| \leq m - 1\).

**Proof.** First, we show that \(|S| < m\). Assume, by contradiction, that \(|S| = m\) and let \(\lambda \in \mathbb{F}^m\) be an \(\Omega\)-validator of \(S\). Then \(\lambda^T U = 0\), which means that \(\sum_{i \in [n]} \lambda_i U_{i,:} = 0\), in contradiction to \(U\) having full row rank \(n\). Hence, \(|S| \leq m - 1\).

Next, by Claim 3.18, since \(S\) is a maximal \(\Omega\)-valid set, its rank is \(n - 1\), therefore, \(n - 1 \leq |S| \leq m - 1\). \(\square\)

**Theorem 3.24.** Algorithm 3 produces an optimal solution to SIV, and is an oracle for Algorithm 2.

**Proof.** Claim 3.10 states that \(v \in \mathbb{F}^m\) is a solution to SIV (an optimally sparse, \(\Omega\)-independent vector) if and only if \(S = \{i : v_i = 0\}\) is \(\Omega\)-valid. The algorithm iterates all subsets of \([m]\) in descending order of cardinality. Therefore, the first \(\Omega\)-valid set found is an \(\Omega\)-valid set of maximal cardinality. Furthermore, Lemma 3.23 states that any maximal \(\Omega\)-valid set is of size \(n - 1 \leq z \leq m - 1\), hence, the algorithm iterates all candidates \(S \subset [m]\) that could be \(\Omega\)-valid sets of maximal cardinality. Therefore, Algorithm 4 returns a sparsest \(\Omega\)-independent vector. Finally, by Claim 3.11, the pair \((\nu, i)\) serves as the oracle for SIV required by Algorithm 2. \(\square\)

4 ADDITIONAL SPARSIFICATION METHODS

4.1 Sparsification via subset of rows.

The alternative bases presented in Karstadt and Schwartz’s [22, 23] paper were found using a straightforward heuristic of iterating over all sets of \(n\) linearly independent rows of an \(n \times m\) matrix of full rank (where \(n \geq m\)). This heuristic was based on the observation that using the columns of the original matrix for sparsification ensures that the sparsified matrix contains \(n\) rows, each with only a single non-zero entry.

While this method is inefficient, requiring \(\binom{n}{m}\) passes, it finds sparsifications which significantly improve the leading coefficients of multiple algorithms. The refinement of this method led to the development of Algorithm 3. It is therefore presented here for completeness.

**Algorithm 5** Row basis sparsification [22, 23]

1. procedure KS-Sparsification(\(U\))
2. \(\text{sparsity } \leftarrow \text{nnz}(U)\)
3. \(\text{basis } \leftarrow I_n\)
4. \(\textbf{for } C \in \mathcal{C}(m)\)
5. \(\textbf{if } U_C \text{ is of full rank}\)
6. \(\text{sparsifier } \leftarrow U_C^{-1}\)
7. \(\textbf{if } \text{nnz}(\text{sparsifier} \cdot U) < \text{sparsity}\)
8. \(\text{sparsity } \leftarrow \text{nnz}(\text{sparsifier} \cdot U)\)
9. \(\text{basis } \leftarrow \text{sparsifier}\)
10. \(\text{return } \text{basis}\)

4.2 Greedy sparsification.

A second heuristic for matrix sparsification, inspired by Gottlieb and Neylon’s algorithm (Algorithm 2), employs an even simpler greedy approach.

Recall that for a given \(n \times m\) matrix \(U\) (\(n \leq m\)), we seek an \(n \times n\) matrix \(A\) which minimizes \(\text{nnz}(AU) + \text{nns}(AU)\). For this purpose, rather than searching for the entire invertible matrix \(A\) achieving this objective, we could instead search for each row of \(A\) individually. Concretely, we iteratively compose the matrix \(A\) row-wise; where at each step \(i\), we obtain the sparsest row vector \(v_i\) such that \(v_i\) is independent of \(\{v_1, \ldots, v_{i-1}\}\) and minimizes \(\text{nnz}(v_i U) + \text{nns}(v_i U)\). This yields the following algorithm:

**Algorithm 6** Greedy Sparsification

1. procedure Greedy − Sparsification(\(U\))
2. \(A \leftarrow \emptyset\)
3. \(\textbf{for } i = 1, \ldots, n\)
4. \(v \leftarrow \arg \min_{v \in \mathbb{F}^m} (\text{nnz}(v_i U) + \text{nns}(v_i U))\)
5. \(A_{i,:} \leftarrow v_i\)
6. \(\text{return } A\)

In order to implement the subroutine for finding each row vector \(v_i\), we encoded the objective as a MaxSAT instance and used Z3 [14], an SMT Theorem Prover, to find the optimal solution. Our MaxSAT instance employs two types of "soft" constraints: one which penalizes non-zero entries, and another which penalizes non-singleton entries. Therefore, optimal solutions will minimize the sum of non-zero and non-singleton entries, thereby minimizing the associated arithmetic complexity (Remark 2.4).

This algorithm, while not proven to be optimal, has the advantage of considering both non-zeros and non-singletons, and can therefore produce decompositions resulting in a lower arithmetic complexity than the optimal algorithms (Algorithms 3, 4). For a summary of these results, see Table 2.

5 APPLICATION AND RESULTING ALGORITHMS

Table 2 contains a list of alternative basis algorithms found using our new methods. All of the algorithms used were taken from the repository of Ballard and Benson [2]³. The alternative basis

³The algorithms can be found at github.com/arbenson/fast-matmul
Table 2: Alternative Basis Algorithms

| Algorithm | Leading Monomial | Arithmetic Operations | Leading Coefficient |
|-----------|------------------|-----------------------|---------------------|
| (2, 2; 7) [38] | $n^8 8/11 = n^2 6.80735$ | 18 | 12 | 7 | 5 | 28.57% |
| (3, 2; 11) [2] | $n^6 11/3 = n^2 8.89495$ | 22 | 18 | 5.06 | 4.26 | 15.82% |
| (2, 3; 14) [40] | $n^6 14/2 = n^2 8.89495$ | 22 | 18 | 4.71 | 3.91 | 16.97% |
| (4, 2; 15) [18] | $n^9 15/3 = n^2 8.85515$ | 48 | 28 | 8.33 | 5.27 | 36.8% |
| (3, 2; 15) [2] | $n^9 15/3 = n^2 8.81076$ | 55 | 39 | 8.28 | 6.17 | 25.5% |
| (5, 2; 16) [2] | $n^9 16/2 = n^2 8.89495$ | 64 | 39 | 9.61 | 6.17 | 35.84% |
| (4, 3; 20) [35] | $n^6 20/3 = n^2 8.28789$ | 78 | 51 | 8.9 | 5.88 | 33.96% |
| (4, 2; 20) [2] | $n^6 20/3 = n^2 8.28789$ | 82 | 51 | 9.19 | 5.88 | 36.01% |
| (4, 3; 20) [2] | $n^6 20/3 = n^2 8.28789$ | 86 | 54 | 9.38 | 6.12 | 34.77% |
| (4, 3; 20) [2] | $n^6 20/3 = n^2 8.28789$ | 104 | 56 | 11.38 | 6.38 | 43.9% |
| (2, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 96 | 58 | 9.96 | 6.12 | 38.59% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 87 | 66 | 7.21 | 5.71 | 20.79% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 88 | 65 | 7.29 | 5.64 | 22.55% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 89 | 65 | 7.36 | 5.64 | 23.3% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 97 | 61 | 7.93 | 5.36 | 32.43% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 166 | 73 | 12.86 | 6.21 | 51.67% |
| (3, 3; 23) [2] | $n^6 23/3 = n^2 8.54044$ | 98 | 74 | 8 | 6.29 | 21.43% |
| (3, 3; 23) [35] | $n^6 23/3 = n^2 8.54044$ | 84 | 68 | 7 | 5.86 | 16.33% |
| (4, 4; 2; 26) [2] | $n^6 26/3 = n^2 8.28026$ | 235 | 105 ($\star$) | 18.1 | 7.81 | 56.84% |
| (4, 3; 29) [36] | $n^6 29/3 = n^2 8.18998$ | 164 | 102 | 10.27 | 6.73 | 34.49% |
| (3, 3; 29) [36] | $n^6 29/3 = n^2 8.19889$ | 137 | 109 | 8.54 | 6.96 | 18.46% |
| (3, 4; 29) [36] | $n^6 29/3 = n^2 8.18998$ | 167 | 105 | 10.27 | 6.73 | 34.49% |
| (3, 3; 36) [36] | $n^6 36/3 = n^2 8.28414$ | 199 | 139 | 9.62 | 6.87 | 28.6% |
| (6, 3; 40) [35] | $n^6 40/3 = n^2 7.77429$ | 1246 | 190 ($\star$) | 55.63 | 8.9 | 84.01% |
| (3, 3; 40) [41] | $n^6 40/3 = n^2 7.77429$ | 1822 | 190 ($\star$) | 79.28 | 8.9 | 88.78% |

($\star$) Denotes algorithms with non-singular values, where the result of Algorithm 6 was better than those of the exhaustive algorithms.

algorithms obtained represent a significant improvement over the original versions, with the reduction in the leading coefficient ranging between 15% and 88%. Almost all of the results were found using our exhaustive methods (Algorithms 3 and 4). In certain cases (marked ($\star$)), where the $U$, $V$, $W$ matrices contain non-singular values, our search heuristic’s (Algorithm 6) result exceeded those of our exhaustive algorithms. For example, bases obtained for the (4, 4, 2; 26)-algorithm by Algorithms 3 and 4 reduced the number of arithmetic operations from 235 to 110, while Algorithm 6 reduced the number of arithmetic operations even further, to 105.

Comparison of different search methods. The exhaustive algorithms (Algorithms 3, 4) solve the SIV problem. Their proof of correctness, coupled with that of Gottlieb and Neylon’s algorithm, guarantee that they obtain decompositions minimizing the number of non-zero entries. As MS and SIV are both NP-Hard problems, these algorithms exhibit an exponential worst-case complexity. For this reason, the decomposition of some of the larger instances required the use of Mira supercomputer. However after some tuning of Algorithms 3 and 4 (see Section 3.4) and the implementation of Algorithm 6 using Z3, all decompositions completed on a PC within a reasonable time. Specifically, all runs of Algorithms 3 and 4 completed within 40 minutes, while Algorithm 6 took less than one minute, on a PC.

6 DISCUSSION AND FUTURE WORK

We have improved the leading coefficient of several fast matrix multiplication algorithms by introducing new methods to solve to

---

8 Matebook X (i7-7500U CPU and 8GB RAM)
sparsify the encoding/decoding matrices of fast matrix multiplication algorithms. The number of arithmetic operations depends on both both non-zero and non-singular entries. This means that in order to minimize the arithmetic complexity, the sum of both non-zero and non-singular entries should be minimized, otherwise an optimal sparsification may result in a 2-approximation of the minimal number of arithmetic operations when matrix entries are not limited to 0, ±1. Further work is required in order to find a provably optimal algorithm which minimizes both non-zero and non-singleton values.

We attempted sparsification of additional algorithms for larger dimensions (e.g., Pan’s (44, 44, 44; 36133)-algorithm [31], which is asymptotically faster than those presented here). However, the size of the base case of these algorithms led to prohibitively long runtimes.

The methods presented in this paper apply to finding square invertible matrices solving the $M_3$ problem. Other classes of sparse decompositions exist which do not fall within this category. For example, Beniamini and Schwartz’s [1] decomposed recursive-bilinear framework relies upon decompositions in which the sparsifying matrix may be rectangular, rather than square. Some of the leading coefficients in [1] are better than those presented here. For example, they obtained a leading coefficient of 2 for a (3, 3, 3, 23)-algorithm of [2] a (4, 3, 3, 29)-algorithm of [3], compared to our values 5.36 and 6.96 respectively. However, the arithmetic overhead of basis transformation in Karastad and Schwartz [22, 23] (and therefore here as well) is $O(n^3 \log n)$, whereas in [1] it may be larger. Note also that the decomposition heuristic of [1] does not always guarantee optimality. Further work is required to find new decomposition methods for such settings.

7 ACKNOWLEDGEMENTS

We thank Austin R. Benson for providing details regarding the (2, 3, 2, 11)-algorithm. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was supported by the PetaCloud industry-academia consortium. This research was supported by a grant from the United States-Israel Bi-national Science Foundation, Jerusalem, Israel. This work was supported by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 818252).

REFERENCES

[1] Gil Beniamini and Oded Schwartz. 2019. Faster Matrix Multiplication via Sparse Decomposition. In Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures. ACM, 11–22.

[2] Austin R Benson and Gery Ballard. 2015. A framework for practical parallel fast matrix multiplication. ACM SIGPLAN Notices 50, 8 (2015), 42–53.

[3] Dario Bini, Mihailo Capovani, Francesco Romani, and Grazia Lotti. 1979. $O(n^2 \cdot 779)$ complexity for $n \times n$ approximate matrix multiplication. Information processing letters 8, 5 (1979), 254–255.

[4] Marco Bodrato. 2010. A Strassen-like matrix multiplication suited for squaring and higher power computation. In Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation. ACM, 257–260.

[5] Richard B Brent. 1970. Algorithms for matrix multiplication. Technical Report. Stanford university CA department of computer science.

[6] Nader H Bshouty. 1995. On the additive complexity of $2 \times 2$ matrix multiplication. Information processing letters 56, 6 (1995), 329–335.

[7] Michael Cenk and M Anwar Hasan. 2017. On the arithmetic complexity of Strassen-like matrix multiplications. Journal of Symbolic Computation 80 (2017), 484–501.

[8] S Frank Chang and S Thomas McCormick. 1992. A hierarchical algorithm for making sparse matrices sparser. Mathematical Programming 56, 1 (1992), 1–30.

[9] Henry Cohn and Christopher Umans. 2003. A group-theoretic approach to fast matrix multiplication. In Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on. IEEE, 438–449.

[10] Don Coppersmith and Shmuel Winograd. 1982. On the asymptotic complexity of matrix multiplication. SIAM J. Comput. 11, 3 (1982), 872–942.

[11] Don Coppersmith and Shmuel Winograd. 1990. Matrix multiplication via arithmetic progressions. Journal of symbolic computation 9, 3 (1990), 251–280.

[12] Hans F de Groote. 1978. On varieties of optimal algorithms for the computation of bilinear mappings I: the isotropy group of a bilinear mapping. Theoretical Computer Science 7, 1 (1978), 1–24.

[13] Hans F de Groote. 1978. On varieties of optimal algorithms for the computation of bilinear mappings II: Optimal algorithms for $2 \times 2$-matrix multiplication. Theoretical Computer Science 7, 2 (1978), 127–140.

[14] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 337–349.

[15] Lee-Ad Gottlieb and Tyler Neylon. 2010. Matrix sparsification and the sparse null space problem. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, 205–218.

[16] Vince Grolmusz. 2008. Modular representations of polynomials: Hyperdense coding and fast matrix multiplication. IEEE Transactions on Information Theory 54, 8 (2008), 3687–3692.

[17] Alan J Hoffman and ST McCormick. 1984. A fast algorithm that makes matrices optimally sparse. Progress in Combinatorial Optimization (1984), 185–196.

[18] John E Hopcroft and Leslie R Kerr. 1971. On minimizing the number of multiplications necessary for matrix multiplication. SIAM J. Appl. Math. 20, 1 (1971), 30–36.

[19] John E Hopcroft and Jean Maxisin. 1973. Duality applied to the complexity of matrix multiplications and other bilinear forms. In Proceedings of the fifth annual ACM symposium on Theory of computing. ACM, 73–87.

[20] Rodney W Johnson and Aileen M McLoughlin. 1986. Noncommutative Bilinear Algorithms for $3 \times 3$ Matrix Multiplication. SIAM J. Comput. 15, 2 (1986), 595–603.

[21] Igor Kaporin. 1999. A practical algorithm for faster matrix multiplication. Numerical linear algebra with applications 6, 8 (1999), 687–700.

[22] Elaye Karastad and Oded Schwartz. 2017. Matrix multiplication, a little faster. In Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures. ACM, 101–110.

[23] Elaye Karastad and Oded Schwartz. 2020. Matrix multiplication, a little faster. Journal of the ACM (JACM) 67, 1 (2020), 1–31.

[24] Donald Knuth. 1981. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley. Reading, MA (1981).

[25] Julian Laderman, Victor Y Pan, and Xuan-He Sha. 1992. On practical algorithms for accelerated matrix multiplication. Linear Algebra and its Applications 162 (1992), 557–588.

[26] Julian Laderman. 1976. A noncommutative algorithm for multiplying $3 \times 3$ matrices using 25 multiplications. In Am. Math. Soc. Vol. 82. 126–128.

[27] Francois Le Gall. 2014. Powers of tensors and fast matrix multiplication. In Proceedings of the 39th international symposium on symbolic and algebraic computation. ACM, 296–303.

[28] S Thomas McCormick. 1983. A Combinatorial Approach to Some Sparse Matrix Problems. Technical Report. Stanford university CA systems optimization lab.

[29] S Thomas McCormick. 1990. Making sparse matrices sparser: Computational results. Mathematical Programming 49, 1-3 (1990), 91–111.

[30] Victor Y Pan. 1978. Strassen’s algorithm is not optimal trilinear technique of aggregating, unifying and canceling for constructing fast algorithms for matrix operations. In Foundations of Computer Science, 1978, 19th Annual Symposium on. IEEE, 166–176.

[31] Victor Y Pan. 1982. Trilinear aggregating with implicit canceling for a new acceleration of matrix multiplication. Computers & Mathematics with Applications 8, 1 (1982), 23–34.

[32] Robert L Probert. 1976. On the additive complexity of matrix multiplication. SIAM J. Comput. 5, 2 (1976), 187–203.

[33] Francesco Romani. 1982. Some properties of disjoint sums of tensors related to matrix multiplication. SIAM J. Comput. 11, 2 (1982), 263–267.

[34] Arnold Schönhage. 1981. Partial and total matrix multiplication. SIAM J. Comput. 10, 3 (1981), 434–455.

[35] Alexey V Smirnov. 2013. The bilinear complexity and practical algorithms for matrix multiplication. Computational Mathematics and Mathematical Physics 53, 12 (2013), 1781–1795.

[36] Alexey V Smirnov. 2017. Several bilinear algorithms for matrix multiplication. Technical Report.
[37] Andrew James Stothers. 2010. On the complexity of matrix multiplication. Thesis (2010).

[38] Volker Strassen. 1969. Gaussian elimination is not optimal. Numerische mathematik 13, 4 (1969), 354–356.

[39] Volker Strassen. 1986. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In Foundations of Computer Science, 1986., 27th Annual Symposium on. IEEE, 49–54.

[40] Petr Tichavský and Teodor Kováč. 2015. Private communication with Ballard and Benson, see [2] for benchmarking. (2015).

[41] Petr Tichavský, Anh-Huy Phan, and Andrzej Cichocki. 2017. Numerical CP decomposition of some difficult tensors. J. Comput. Appl. Math. 317 (2017), 362–370.

[42] Virginia V Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing. ACM, 887–898.

[43] Shmuel Winograd. 1971. On multiplication of 2×2 matrices. Linear algebra and its applications 4, 4 (1971), 381–388.
A SAMPLES OF ALTERNATIVE BASIS ALGORITHMS

In this section we present the encoding/decoding matrices of the alternative basis algorithms listed in Table 2. To verify the correctness of these algorithms, recall Corollary 2.11 and use the following fact:

**Fact A.1.** (Triple product condition) [5, 24] Let $R$ be a ring, and let $U \in R^{n \times m}$, $V \in R^{m \times k}$, $W \in R^{k \times n}$. Then $\langle U, V, W \rangle$ are encoding/decoding matrices of an $(n, m, k; t)$-algorithm if and only if:

$$\forall i_1, k_1 \in [n], j_1, j_2 \in [m], j_2, k_2 \in [k]$$

$$\sum_{t=1}^{n} U_{r, (i_1, j_1)} V_{r, (j_1, j_2)} W_{r, (k_1, k_2)} = \delta_{i_1, k_1} \delta_{j_1, j_2} \delta_{j_2, k_2}$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.

### A.1 A sample of Algorithms

**Table 3: (3, 2, 3; 15)-algorithm [2]**

| $v_1$ | $v_2$ | $v_3$ |
|-------|-------|-------|
| 0 1 0 | 0 1 0 | 0 1 0 |
| 0 0 1 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 1 0 | 0 1 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 1 | 1 0 0 | 1 0 0 |
| 0 1 0 | 0 0 1 | 0 0 1 |
| 0 1 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 1 0 | 0 1 0 |
| 1 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |

**Table 4: (4, 2, 3; 20)-algorithm [35]**

| $u_1$ | $u_2$ | $u_3$ |
|-------|-------|-------|
| 0 1 0 | 0 1 0 | 0 1 0 |
| 0 0 1 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 1 0 | 0 1 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 1 | 1 0 0 | 1 0 0 |
| 0 1 0 | 0 0 1 | 0 0 1 |
| 0 1 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 1 0 | 0 1 0 |
| 1 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |

**Table 5: (3, 3, 3; 23)-algorithm [35]**

| $v_1$ | $v_2$ | $v_3$ |
|-------|-------|-------|
| 0 0 1 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 1 0 | 0 1 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 1 | 1 0 0 | 1 0 0 |
| 0 1 0 | 0 0 1 | 0 0 1 |
| 0 1 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 1 0 | 0 1 0 |
| 1 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |

**Table 6: (6, 3, 3; 40)-algorithm [35]**

| $u_1$ | $u_2$ | $u_3$ |
|-------|-------|-------|
| 0 0 1 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 1 0 | 0 1 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 1 | 1 0 0 | 1 0 0 |
| 0 1 0 | 0 0 1 | 0 0 1 |
| 0 1 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 1 0 | 0 1 0 |
| 1 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 1 | 0 0 1 |
| 0 0 0 | 0 0 0 | 0 0 0 |