Fast Matrix Multiplication Without Tears: A Constraint Programming Approach

Arnaud Deza[1]
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Chang Liu[1]
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Pashootan Vaezipoor
Department of Computer Science, University of Toronto, Canada

Elias B. Khalil
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Abstract

It is known that the multiplication of an $N \times M$ matrix with an $M \times P$ matrix can be performed using fewer multiplications than what the naive NMP approach suggests. The most famous instance of this is Strassen's algorithm for multiplying 2×2 matrices in 7 instead of 8 multiplications. This gives rise to the constraint satisfaction problem of fast matrix multiplication, where a set of $R < NMP$ multiplication terms must be chosen and combined such that they satisfy correctness constraints on the output matrix. Despite its highly combinatorial nature, this problem has not been exhaustively examined from that perspective, as evidenced for example by the recent deep reinforcement learning approach of AlphaTensor. In this work, we propose a simple yet novel Constraint Programming approach to find algorithms for fast matrix multiplication or provide proof of infeasibility otherwise. We propose a set of symmetry-breaking constraints and valid inequalities that are particularly helpful in proving infeasibility. On the feasible side, we find that exploiting solver performance variability in conjunction with a sparsity-based problem decomposition enables finding solutions for larger (feasible) instances of fast matrix multiplication. Our experimental results using CP Optimizer demonstrate that we can find fast matrix multiplication algorithms for matrices up to 3×3 with $R = 23$ in a short amount of time.

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases fast matrix multiplication, computer-assisted proofs, constraint programming, constraint satisfaction problem

Digital Object Identifier 10.4230/LIPIcs.CP.2023.14

Supplementary Material
Software (Source Code): `https://github.com/khalil-research/Matrix-Mult-CP/tree/main`

1 Introduction

Matrix multiplication is a fundamental operation in linear algebra with applications in virtually every computational domain. As a result, extensive research has been dedicated to the development of faster matrix multiplication algorithms.

The elementary way of multiplying two $N \times N$ matrices requires N^3 multiplications. For example, multiplying two 2×2 matrices naively requires a total of $2^3 = 8$ multiplications. In 1969, Strassen [16] constructed an algorithm that finds the product of two 2×2 matrices in only 7 multiplications. This discovery has had significant implications as it opened up

[1] These authors contributed equally.

the door for potentially faster algorithms for large-scale matrix or tensor computations. Strassen's algorithm has later been proved to be both canonical [4] (no smaller rank exists) and essentially unique [6] (all other solutions of the same rank are equivalent up to symmetry).

Currently, the best-known algorithm for multiplying 3×3 matrices requires $R = 23$ multiplications, compared to the naive elementary method that requires 27 multiplications. A known theoretical lower bound of $R = 19$ exists [2], however, it remains unclear whether $19 \leq R \leq 22$ is truly attainable. This is a testament to the difficulty of the *fast matrix multiplication* (FMM) problem, which has been intractable for existing methods even for tiny matrices.

In the literature, the general approach to finding FMM algorithms starts by representing matrix multiplication as a tensor operation using the multiplication tensor T_N followed by finding exact or approximate low-rank decompositions that represent T_N. The factor matrices that are used in the low-rank decomposition encode FMM algorithms. A rank-7 decomposition (i.e., a multiplication algorithm that uses 7 multiplication operations) of a 2×2 matrix multiplication using Strassen's algorithm is shown in Figure 1. Existing methods for finding such factor matrices have several limitations. The most successful and common methods include local search [13] techniques for low-rank approximation, which cannot guarantee optimality. A more recent successful approach [7] searches for low-rank decomposition using *reinforcement learning* (RL) and was successful in finding faster algorithms for $N = 4$. However, this method is not exhaustive and hence cannot prove the infeasibility of a given rank.

In this work, we propose a novel approach to finding FMM algorithms by formulating the tensor decomposition problem, for the first time, as a *constraint satisfaction problem* (CSP) that is solved using *Constraint Programming* (CP). We believe that this is a very natural formulation of this highly combinatorial problem. CP is advantageous for FMM in that it is a flexible framework that can bring to bear a wide range of search and logical inference techniques that have been developed over the last few decades. It provides the ability to prove infeasibility when it is not possible to multiply two matrices using a given number of multiplications.

Besides a base CP formulation for FMM, we propose a set of symmetry-breaking constraints and valid inequalities that are useful for infeasibility proofs. On the feasible side, we show that "performance variability" w.r.t. solver random seeds can be exploited in conjunction with a sparsity-based decomposition of FMM for faster solving. Our experimental results, while limited to matrices of size up to 3×3, demonstrate the effectiveness of the aforementioned constraints and techniques. The CP approach to FMM is uniquely positioned to close open questions such as whether it is possible to multiply two 3×3 matrices in 19 to 22 multiplications. While we do not yet resolve this or other open questions, our work opens up the potential for further enhancements to the CP formulation and search such as customized branching strategies and CP-based heuristics.

2 Fast Matrix Multiplication: Problem Statement

The multiplication of two matrices A and B of sizes $N \times M$ and $M \times P$, respectively, results in a product matrix C of size $N \times P$. This operation can be represented by a binary third-order tensor T_{NMP} (T_N for square matrices A and B of size $N \times N$). An entry $T_{i,j,k}$ of this tensor is equal to 1 if and only if the k^{th} entry in the output matrix C uses the scalar product of the i^{th} entry of A and the j^{th} entry of B. Here, i, j, and k are indices of a matrix entry starting with 1 in the first row and column; and proceeding entry by entry, left to right, top to bottom. For example, for $N = M = P = 2$, it must be that $T_{2,3,1} = 1$ since the first entry of C, c_1, is equal to $a_1b_1 + \mathbf{a_2b_3}$. Similarly, $T_{1,2,1} = 0$ must hold since a_1b_2 is not part of c_1. Figures 1a and 1b show a complete example of the indexing and tensor representation.

The FMM problem for a given tensor T_{NMP}, rank $R \in \mathbb{Z}^+$, and field $\mathbb{F}$ (e.g., $\mathbb{F} = \{-1, 0, 1\}$) asks: can each entry $T_{i,j,k}$ of T_{NMP} be expressed as the sum of exactly R trilinear terms involving the *factor matrices* $U \in \mathbb{F}^{N\cdot M\times R}$, $V \in \mathbb{F}^{M\cdot P\times R}$, and $W \in \mathbb{F}^{N\cdot P\times R}$, as follows:

$$T_{i,j,k} = \sum_{r=1}^{R} U_{i,r} \cdot V_{j,r} \cdot W_{k,r} \qquad \forall i \in \{1, \dots, N \cdot M\}, j \in \{1, \dots, M \cdot P\}, k \in \{1, \dots, N \cdot P\}$$

Note that we use the notation $\mathbb{F}^{L\times Q}$ to refer to the set of matrices of dimension $L \times Q$ and entries in $\mathbb{F}$. The CSP is to find factor matrices with entries in $\mathbb{F}$ that produce the tensor T_{NMP} for a given rank R.

This decomposition is also referred to as the polyadic decomposition and its associated rank is the minimal R needed. The rank can be interpreted as the number of multiplications required to compute the product. For example, for 2×2 matrices, the rank of the decomposition using Strassen's algorithm is 7. Figure 1 walks through an example of the low-rank decomposition of a 2×2 matrix multiplication using Strassen's algorithm. The matrix multiplication of the two 2×2 matrices can be seen in Figure 1a, its associated tensor representation T_N in Figure 1b, the low-rank decomposition in Figure 1c, and the factor matrices U, V, and W in Figure 1d.

3 Related Work

Since Strassen's discovery [16], there has been substantial research on finding faster algorithms for matrix multiplication. Mathematicians have discovered such algorithms manually over the years for a variety of matrix dimensions and ranks. In this section, however, we will focus on automated methods for discovering such algorithms and briefly discuss some of the existing methods. A recent survey on the topic can be found in [3].

3.1 Continuous Local Search Methods

The most common approach in the literature to compute the factor matrices U, V, and W is to use (heuristic, continuous) local search methods for low-rank tensor decomposition. The state-of-the-art local method [13] uses alternating least squares with regularization. This method has been the most successful in finding fast algorithms whilst remaining computationally tractable and has been scaled up to $N = M = P = 4$, $R = 49$[2]. However, this approach has limitations which include getting stuck at local minima, facing ill-conditioned linear least-squares problems, and solutions being only adequate up to machine precision. Additionally, these methods are not exhaustive and hence cannot be used to provide a proof of infeasibility for a given rank R.

3.2 AlphaTensor

More recently, DeepMind released AlphaTensor [7], a deep RL method that searches this large combinatorial space by playing a single-player game, the TensorGame, formulated as a *Markov decision process* (MDP). At every step t of this MDP, the state is characterized

[2] Note that this particular result is not very useful as an $R = 49$ solution can be obtained by applying Strassen's $R = 7$ algorithm for 2×2 matrices on the four 2×2 blocks of the 4×4 matrices.

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

(a) Multiplication of two 2×2 matrices. We highlight the term $c_1 = a_1 b_1 + a_2 b_3$.

$$T_{:,:,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad T_{:,:,2} = \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad T_{:,:,3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \end{pmatrix} \quad T_{:,:,4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix}$$

(b) Tensor representation of the 2×2 matrix multiplication operation. $T_{:,:,1}$ represents c_1, the entry $T_{2,3,1}$ (in yellow) is set to 1 because the product $a_2 b_3$ is required to compute c_1 (similarly for $T_{1,1,1}$ in red).

$$\begin{aligned} m_1 &= (a_1 + a_4)(b_1 + b_4) & m_5 &= (a_1 + a_2)(b_4) \\ m_2 &= (a_3 + a_4)(b_1) & m_6 &= (a_3 - a_1)(b_1 + b_2) \\ m_3 &= (a_1)(b_2 - b_4) & m_7 &= (a_2 - a_4)(b_3 + b_4) \\ m_4 &= (a_4)(b_3 - b_1) \end{aligned}$$

$$\begin{aligned} c_1 &= m_1 + m_4 - m_5 + m_7 \\ &= (a_1 + a_4)(b_1 + b_4) + (a_4)(b_3 - b_1) - (a_1 + a_2)(b_4) + (a_2 - a_4)(b_3 + b_4) \\ &= a_1 b_1 + \cancel{a_1 b_4} + \cancel{a_4 b_1} + \cancel{a_4 b_4} + \cancel{a_4 b_3} - \cancel{a_4 b_1} - \cancel{a_1 b_4} - \cancel{a_2 b_4} + a_2 b_3 + \cancel{a_2 b_4} - \cancel{a_4 b_3} - \cancel{a_4 b_4} \\ &= a_1 b_1 + a_2 b_3 \\ c_2 &= m_3 + m_5 \\ c_3 &= m_2 + m_4 \\ c_4 &= m_1 - m_2 + m_3 + m_6 \end{aligned}$$

(c) A low-rank decomposition of the 2×2 matrix multiplication using Strassen's algorithm. The m terms are the multiplication terms and the c terms represent the entries in the product matrix. Here $c_1 = m_1 + m_4 - m_5 + m_7$ gives $c_1 = a_1 b_1 + a_2 b_3$ after expansion.

$$\begin{array}{cc} & \begin{array}{ccccccc} m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 \end{array} \\ U = & \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \end{array} \\ V = & \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \end{array} \\ W = & \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \end{array}$$

(d) The factor matrices U, V, and W for Strassen's algorithm. The columns in U and V represent the coefficient of the a and b terms in each m. Each row in W represents the coefficient of the m terms in one c term.

Figure 1 A low-rank decomposition of a 2×2 matrix multiplication using Strassen's algorithm.

by a tensor S_t which is initially set to the target multiplication tensor, i.e., $S_0 = T_N$. An action a_t at iteration t corresponds to the player selecting a triplet of vectors $(u^{(t)}, v^{(t)}, w^{(t)})$ which in turn will provide the next state $S_t = S_{t-1} - u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$ where $\otimes$ denotes the outer tensor product. The goal of the player is to reach the zero tensor $S_t = \mathbf{0}$ in the fewest number of steps possible. This is done by providing a reward of -1 to the player after every non-terminal state whereas a large negative reward $-\gamma(S_{R_{\text{limit}}})$ is given to the player if the number of steps R_{limit} is met, where $\gamma(S_{R_{\text{limit}}})$ upper bounds the rank of the tensor at iteration R_{limit}. If the agent successfully reaches the zero tensor, the sequence of actions taken constitutes a valid low-rank decomposition of T_N, and hence an FMM algorithm is found with the rank R corresponding to the number of steps taken by the agent.

This approach is the first to directly incorporate learning into the search which resulted in the discovery of new minimal ranks for certain non-trivial cases. The largest case tackled by this method is $N = M = P = 5$, $R = 98$. The sole focus of this purely heuristic method is to find lower ranks than currently best-known ranks but it cannot prove the infeasibility of a given rank. Additionally, rather complex architectures and multiple training phases were required for successful learning. It is worth noting that AlphaTensor was trained for one week on 64 Tensor Processing Units (TPUs), Google's proprietary chip. The paper [7] does not provide any estimates of the amount of computation required to produce the reported results, namely how long the trained "agent" must be run to discover FMM algorithms. Our CP runs use much fewer resources while leveraging thread parallelism in the CP solver on readily-available CPU machines.

3.3 Integer Programming

The work that is the most related to our approach tackles this problem through a *mixed-integer linear program* (MILP) formulation in an unpublished technical report [14]. The goal of this methodology is to linearize the trilinear products in the low-rank decomposition of T_N to a MILP that aims to 1) maximize the sparsity of the integer decision variables representing factor matrices U, V, and W and 2) minimize the reconstruction loss (L1 norm) from the input T_N and the multiplication tensor attained by the decision variables representing factor matrices. The report [14] focuses solely on presenting the MILP formulation for square matrices but does not include any computational experiments. However, the MILP formulations for $N \in \{2, 3\}$ are benchmark problems in MIPLIB 2017 [9][3]. The linearization of the trilinear products likely leads to a weak linear programming relaxation as well as an explosion in the number of integer variables and constraints, which might explain why the MILP approach to FMM has not picked up significant interest. A CP formulation is more natural and compact, as we will show in this paper.

3.4 Classical AI Planning

Very recently, AI planning techniques were used for FMM [15]. They use a similar state space as AlphaTensor but use various planning tools (with and without exhaustive search) to solve this problem. They compared a number of heuristic and exact planning methods from the literature on matrices of size up to 3×3. However, the experiments show that planning approaches are severely limited, even failing to find Strassen's algorithm for the 2×2 case (see Table 1 in [15]). We will show that our CP approach is significantly more effective as we are able to attack the 3×3 case with $R = 23$, matching the known upper bound from the literature.

[3] See `https://miplib.zib.de/instance_details_fastxgemm-n3r21s3t6.html` for example.

3.5 Boolean Satisfiability-based Approaches

Boolean Satisfiability (SAT) formulations to solve the FMM problem were first proposed in 2011 [5]. More specifically, the authors studied the multiplication of 3×3 matrices and were able to find a new general algorithm with rank 23 with help from SAT solver tweaks and improvements in a few days with one CPU. More recently, Heule et al. improved on the SAT-based methods for the multiplication of 3×3 matrices and have successfully found many thousands new general algorithms with rank 23 [10, 11]. In their SAT-formulation, the authors transformed the multiplication to 'and' clauses and the addition to 'xor' clauses, then a Tseistin transformation is performed to formulate the algebraic FMM problem into a SAT problem.

The authors propose a random pairing method where value assignments are initialized based on observed results from previously known solutions with streamlining constraints to find new general solutions; then local search is used to find additional solutions. The SAT-based formulations of the FMM problem demonstrates to be very difficult for complete SAT solvers, thus making it difficult to provide proof of infeasibility.

4 Constraint Programming for Fast Matrix Multiplication

In the FMM problem, all variables have the same domain $\mathbb{F} = \{-1, 0, 1\}$[4]. Since the variable domains are small and this problem is highly structured, CP is a promising solution paradigm.

The base CP model for FMM is given in Equation (1). Let $\mathcal{U}$ denote the set $\{1, \ldots, N \cdot M\}$, $\mathcal{V}$ denote the set $\{1, \ldots, M \cdot P\}$, $\mathcal{W}$ denote the set $\{1, \ldots, N \cdot P\}$, and $\mathcal{R}$ denote the set $\{1, \ldots, R\}$. The CP model uses three sets of variables: $u_{i,r}$ where $i \in \mathcal{U}$, $v_{j,r}$ where $j \in \mathcal{V}$, and $w_{k,r}$ where $k \in \mathcal{W}$; $r \in \mathcal{R}$ in all three cases. Each variable $u_{i,r}$, $v_{j,r}$ and $w_{k,r}$ represents the value of the $i/j/k^{\text{th}}$ row and r^{th} column of the matrices U, V, and W. The domain of all variables is $\{-1, 0, 1\}$. The set of constraints presented here requires that the decomposition algorithm's output matches the original tensor multiplication T_{NMP}. Therefore the input to the CSP model is 4 integers: (N, M, P) and R. The model then reads as:

$$\begin{aligned}
&\sum_{r \in \mathcal{R}} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) = T_{i,j,k}, && \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W} \\
&u_{i,r}, v_{j,r}, w_{k,r} \in \{-1, 0, 1\}, && \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W}, r \in \mathcal{R}
\end{aligned} \tag{1}$$

The search space for this (NP-complete) problem grows very quickly with increasing matrix sizes N, M, P and rank R. With only one set of equality constraints, a CP solver may struggle with constraint propagation, thus failing to scale with increasing N, M, P. To that end, we will introduce additional valid constraints to help CP prune and propagate more efficiently.

4.1 Symmetry Breaking

There are many symmetric solutions to the FMM problem. We can reduce the search space of our problem significantly by prohibiting symmetries.

[4] One can consider bigger fields such as $\{-2, -1, 0, 1, 2\}$ but the bulk of the work in the literature has been with $\{-1, 0, 1\}$.

4.1.1 Permutation Symmetry

Since addition is commutative, i.e., $(a_1 + a_2) = (a_2 + a_1)$, there are many equivalent solutions to the tensor decomposition problem. Therefore, any permutation of the columns of matrices U, V, and W produces an equivalent solution. If we consider Strassen's solution for the 2×2 case, Figure 2 provides an example of two equivalent solutions.

$$\text{sol1:}\quad U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \quad V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\text{sol2:}\quad U = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 1 & 0 \end{pmatrix} \quad V = \begin{pmatrix} 1 & -1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 1 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2 Two equivalent solutions for Strassen's solution of 2×2 matrix multiplication. sol2 is the `lexicographic-strict` presentation of this solution.

In order to break this symmetry, we introduce a `lexicographic-strict`[5] constraint on the $u_{i,r}$ and $v_{j,r}$ variables. When applied to two variable arrays x and y, the lexicographic ordering constraint enforces that x is strictly less than y in the defined lexicographic order. Because of the strictness, this also enforces that the two variable arrays must be different. This set of symmetry-breaking constraints is modelled as follows:

$$\texttt{lexicographic-strict}([u_{:,r}; v_{:,r}], [u_{:,r+1}; v_{:,r+1}]), \qquad \forall r \in \mathcal{R}$$

where $[u_{:,r}; v_{:,r}]$ represents the vector concatenating the r^{th} column of the matrix U and V. In Figure 2, sol2 satisfies the `lexicographic-strict` constraint.

4.1.2 Sign Symmetry

For the multiplicative m_i terms, one can easily see that multiplying both sets of terms from A and B by -1 will result in the same solution. For example, $(a_1 + a_4)(b_1 + b_4) = (-a_1 - a_4)(-b_1 - b_4)$, where we could multiply any subset of columns of U and V by -1 to achieve the same solution. We call this symmetry the sign symmetry. In order to break it, we introduce the following constraints:

$$u_{1,r} \leq 0$$
$$u_{i,r} \leq \sum_{i'=1}^{i-1} |u_{i',r}| \qquad \forall r \in \mathcal{R}, i > 1, i \in \mathcal{U}$$

The main idea of these constraints is to enforce that the first non-zero entry in a column of U can only take on the value of -1, enforcing that the first entry of the columns is either 0 or -1. The subsequent constraints ensure that for any column r, an entry in row $i > 1$ can only be 1 if there has been an entry in the same column in an earlier row with value -1. This set of constraints applies to the concatenation of the columns in U and V, however, in modelling, it only needs to be applied to the columns of the U matrix as none of the columns can be zero, so the leading -1 must appear in the U matrix. By applying these constraints, we make sure that $\{-u_{i,r}\}$ is infeasible for any feasible $\{u_{i,r}\}$. Employing this sign symmetry breaking constraint to sol2 from Figure 2, we arrive at sol3 shown in Figure 3.

[5] `https://www.ibm.com/docs/en/icos/22.1.0?topic=variables-lexicographic-constraint`

$$\text{sol3:}\quad U=\begin{pmatrix} -1 & 0 & 0 & 0 & -1 & -1 & -1\\ 0 & 0 & 0 & -1 & 0 & 0 & -1\\ 1 & 0 & -1 & 0 & 0 & 0 & 0\\ 0 & -1 & -1 & 1 & 0 & -1 & 0 \end{pmatrix}\quad V=\begin{pmatrix} 1 & 1 & -1 & 0 & 0 & -1 & 0\\ 1 & 0 & 0 & 0 & -1 & 0 & 0\\ 0 & -1 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 1 & -1 & -1 \end{pmatrix}\quad W=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & -1\\ 0 & 0 & 0 & 0 & 1 & 1 & 1\\ 0 & 1 & 1 & 0 & 0 & 0 & 0\\ 1 & 0 & -1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 3 sol3 is the solution derived from enforcing the sign symmetry constraints on sol2.

Similarly, the same type of sign symmetry-breaking constraints can be applied to the W factor matrix as follows:

$$w_{1,r} \leq 0$$
$$w_{k,r} \leq \sum_{k'=1}^{k-1} |w_{k',r}| \qquad \forall r \in \mathcal{R}, k > 1, k \in \mathcal{W}.$$

The interpretation is as follows. In Figure 1c, consider $c_4 = m_1 - m_2 + m_3 + m_6$, and notice that one can redefine $m_6 = (a_3 - a_1)(b_1 + b_2)$ to become $m_6 = (a_3 - a_1)(-b_1 - b_2)$ and then rewrite c_4 as $c_4 = m_1 - m_2 + m_3 - m_6$. Recall that the coefficients of the m terms in an output entry c_k are the entries of row k of factor matrix W. The transformation we just performed produces two equivalent solutions and is an instance of "value symmetry" that is broken by the above constraint set as it forces the first non-zero entry of a column of W to be -1.

4.2 Valid Inequalities

Based on the structure of this problem, we can also introduce a series of valid inequalities that could potentially help a CP solver with propagation.

First, for the W matrix, we know that each multiplicative term m_r must be used at least once for sufficiently small R (i.e., for non-trivial cases of the FMM problem where $R \leq NMP$). This means that the sum of each column in W must be at least one:

$$\sum_{k \in \mathcal{W}} |w_{k,r}| \geq 1, \qquad \forall r \in \mathcal{R}.$$

Each result term c_l must use at least M terms. This is due to a basic fact in algebraic complexity theory which states that the dot-product of two vectors of size M requires at least M multiplications [17]. This means that the sum of each row of W must be greater or equal to M:

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \geq M, \qquad \forall k \in \mathcal{W}.$$

Each result term c_l must differ in at least two m_r terms; a simple proof by contradiction is omitted for brevity. This can be modelled as follows:

$$\sum_{r \in \mathcal{R}} |w_{k,r} - w_{k',r}| \geq 2, \qquad \forall k \neq k' \in \mathcal{W}.$$

Each term in the A and B matrices must appear in at least one of the multiplicative terms m_r. This translates to each row of U and V having at least one non-zero term as shown in the constraints below:

$$\sum_{r \in \mathcal{R}} |u_{i,r}| \geq 1, \qquad \forall i \in \mathcal{U}$$
$$\sum_{r \in \mathcal{R}} |v_{j,r}| \geq 1, \qquad \forall j \in \mathcal{V}.$$

Furthermore, each valid product of two terms from the A and B matrices, e.g., a_2b_3 for 2×2 matrices, must appear in at least one of the R multiplication terms. For a_2b_3 appears in c_1 and c_2, see Figure 1a. This can be modelled as follows:

$$\sum_{r \in \mathcal{R}} |u_{i,r} \cdot v_{j,r}| \geq 1, \qquad \forall \text{ valid } i, j.$$

4.3 Full CP Model

Finally, the full CP model is presented in Figure 4. The constraints in Equation (2) ensure that the output matches the original multiplication tensor and thus the validity of an assignment as a matrix multiplication algorithm. We enforce permutation symmetry-breaking with Equation (3) and sign symmetry-breaking with Equations (4)–(7). The valid inequalities are modelled through Equations (8)–(13).

$$\begin{aligned}
&\sum_{r \in \mathcal{R}} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) = T_{i,j,k}, && \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W} && (2)\\
&\texttt{lexicographic-strict}([u_{:,r}; v_{:,r}], [u_{:,r+1}; v_{:,r+1}]), && \forall r \in \mathcal{R} && (3)\\
&u_{1,r} \leq 0 && \forall r \in \mathcal{R} && (4)\\
&u_{i,r} \leq \sum_{i'=1}^{i-1} |u_{i',r}|, && \forall r \in \mathcal{R}, i > 1, i \in \mathcal{U} && (5)\\
&w_{1,r} \leq 0 && \forall r \in \mathcal{R} && (6)\\
&w_{k,r} \leq \sum_{k'=1}^{k-1} |w_{k',r}| && \forall r \in \mathcal{R}, k > 1, k \in \mathcal{W} && (7)\\
&\sum_{k \in \mathcal{W}} |w_{k,r}| \geq 1, && \forall r \in \mathcal{R} && (8)\\
&\sum_{r \in \mathcal{R}} |w_{k,r}| \geq M, && \forall k \in \mathcal{W} && (9)\\
&\sum_{r \in \mathcal{R}} |w_{k,r} - w_{k',r}| \geq 2, && \forall k \neq k' \in \mathcal{W} && (10)\\
&\sum_{r \in \mathcal{R}} |u_{i,r}| \geq 1, && \forall i \in \mathcal{U} && (11)\\
&\sum_{r \in \mathcal{R}} |v_{j,r}| \geq 1, && \forall j \in \mathcal{V} && (12)\\
&\sum_{r \in \mathcal{R}} |u_{i,r} \cdot v_{j,r}| \geq 1, && \forall \text{ valid } i, j && (13)
\end{aligned}$$

Figure 4 Full CP Model with symmetry-breaking constraints and valid inequalities.

4.4 Sparsity-based Problem Decomposition

Given that the factor matrices that have been found for known decompositions tend to be sparse, we introduce some inexact inequalities to induce sparsity and trim candidate assignments that have a high likelihood to be infeasible or that are unnecessarily dense. For example, observe that Strassen's solution in Figure 1c leads to many zeros in the factor matrices; no m term uses more than 2 out of 4 of the a or b terms, no c term uses more than 4 out of the 7 m terms. It has been observed that as the matrix sizes grow, the best solutions become even sparser.

We first introduce a constraint limiting the number of active (i.e., nonzero) terms in each column r (i.e., multiplication term) of U and V. This constraint is written as:

$$\sum_{i \in \mathcal{U}} |u_{i,r}| + \sum_{j \in \mathcal{V}} |v_{j,r}| \leq K_1, \qquad \forall r \in \mathcal{R}.$$

A similar constraint can be imposed on W, by restricting that each output must use at most K_2 multiplication terms. This constraint is written as:

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \leq K_2, \qquad \forall k \in \mathcal{W}.$$

Based on these constraints, K_1 has an upper bound of $(NM + MP)$ and K_2 is upper bounded by R. By observing decompositions for small to medium-scale matrices, we can estimate K_1 and K_2. For example, for 3×3 matrices with $R = 23$, we observe that $K_1 = 9$ and $K_2 = 10$ is the safest estimate possible compared to the upper bounds of 18 and 23, respectively, which could restrict the CP search dramatically. Note that one could start with any such estimates of the decomposition parameters K_1 and K_2, iteratively increasing them if the restricted instances are found to be infeasible by the CP solver, eventually resulting in a complete resolution of the original problem.

4.5 Cyclic Invariant Formulation

In contrast to the symmetries of the factor matrices discussed in Section 4.1, there exists well-known cyclic symmetry for the multiplication tensors T_N of square matrices. More precisely, it is known that $T_{i,j,k} = T_{j,k,i} = T_{k,i,j}$. The authors in [1] proposed to leverage this cyclic symmetry property and parameterize FMM algorithms with cyclic invariant factor matrices: $U = [ABCD]$, $V = [ADBC]$, $W = [ACDB]$ with $A \in \{-1,0,1\}^{N^2 \times S}$ and $B, C, D \in \{-1,0,1\}^{N^2 \times T}$ corresponding to a rank $R = S + 3T$.

Although this parametrization reduces the number of integer variables by a factor of three, helping with the combinatorial nature of the problem, there is no guarantee that the minimal rank decomposition corresponds to solutions that exhibit cyclic symmetry. That being said, Strassen's solution of $R = 7$ for $N = 2$, which is optimal, exhibits such a symmetry ($S \in \{1, 4\}$), as does the best-known rank of 23 for $N = 3$ ($S \in \{2, 5, 11\}$). Performing two steps of Strassen's algorithm for $N = 2$ yields a rank 49 cyclic invariant solution for $N = 4$. It is currently unknown whether a solution of rank less than 49 exists for $N = 4$, let alone one exhibiting cyclic invariance.

We implement Ballard and Benson's cyclic invariant reduction [1] of the FMM problem for square matrices by reducing the decision variables of our CP formulation as required and imposing the invariant structure on the factor matrices.

5 Experiments

In this section, we present our experimental results starting by showing how our CP approach can recover the best-known upper bounds on the rank in a small amount of time on multiplication problems ranging from the trivial $(N, M, P) = (1, 1, 1)$ case all the way up to the much harder $(2, 2, 4)$ and $(3, 3, 3)$ cases. We then present results for the infeasible cases for $(2, 2, 2)$. We used IBM's CP Optimizer (CPO) 22.1.0[6] to solve our CP models. We ran our experiments on a compute cluster of AMD Ryzen Threadripper 2990WX cores with 128 GB of RAM per node.

5.1 Experimental Setup

To ensure the reproducibility and robustness of our results, all our experiments are run with multiple random seeds. This accounts for the often observed performance variability in combinatorial search; this is documented for example in MILP [12]. To that end, we ran each experiment with 10 different seeds. We assigned 8 cores (CPO's `Workers` parameter) to the solver for each run (except for more compute-intensive experiments in Section 5.4 where we assigned 20 cores) and timed out the experiments after 2 hours.

5.2 Evaluation Metrics

We will report the solver runtime and the number of branches during the solution process for completed runs (i.e., runs that returned a feasible solution or a proof of infeasibility). Given that each problem is attempted with multiple random seeds, the shifted geometric mean with a shift of 0.0001[7], median, minimum, and maximum of the time in seconds and the number of branches will be reported for a complete picture of the results. Runs that terminated due to the time or memory limits will be discussed where applicable.

Table 1 Runtime results for the base CP model on various matrix dimensions. "geo mean" refers to the shifted geometric mean as described in Section 5.2; "med" refers to the median and "min"/"max" to the minimum and maximum, respectively.

				Time (sec)	Num Branches
N	M	P	R	geo mean (min, med, max)	geo mean (min, med, max)
1	1	1	1	0.00 (0.00, 0.00, 0.01)	5.05×10^1 (4.50×10^1, 5.00×10^1, 5.80×10^1)
1	1	2	2	0.00 (0.00, 0.00, 0.01)	1.31×10^2 (1.00×10^2, 1.27×10^2, 2.13×10^2)
1	2	1	2	0.00 (0.00, 0.01, 0.01)	1.68×10^2 (1.08×10^2, 1.69×10^2, 2.31×10^2)
1	1	3	3	0.00 (0.00, 0.01, 0.01)	7.09×10^2 (3.47×10^2, 7.78×10^2, 1.38×10^3)
1	3	1	3	0.00 (0.00, 0.01, 0.01)	9.44×10^2 (3.68×10^2, 9.83×10^2, 1.82×10^3)
1	2	2	4	0.01 (0.00, 0.01, 0.01)	4.86×10^3 (1.68×10^3, 5.27×10^3, 7.68×10^3)
2	1	2	4	0.01 (0.01, 0.01, 0.02)	4.36×10^3 (2.63×10^3, 4.39×10^3, 7.14×10^3)
1	2	3	6	0.05 (0.02, 0.04, 0.10)	3.51×10^4 (1.41×10^4, 2.96×10^4, 9.83×10^4)
1	3	2	6	0.05 (0.03, 0.06, 0.12)	3.19×10^4 (1.40×10^4, 3.19×10^4, 7.18×10^4)
2	1	3	6	0.05 (0.02, 0.04, 0.11)	3.79×10^4 (1.45×10^4, 3.33×10^4, 8.93×10^4)
2	2	2	7	0.74 (0.28, 0.75, 1.84)	6.41×10^5 (2.03×10^5, 6.69×10^5, 1.70×10^6)
1	3	3	9	0.37 (0.26, 0.36, 0.60)	2.92×10^5 (1.84×10^5, 3.09×10^5, 4.16×10^5)
3	1	3	9	0.42 (0.18, 0.52, 0.61)	3.10×10^5 (1.64×10^5, 3.35×10^5, 4.36×10^5)
2	2	3	11	49.64 (0.98, 71.82, 245.06)	3.40×10^7 (6.90×10^5, 4.71×10^7, 1.74×10^8)
2	3	2	11	26.47 (6.68, 29.41, 133.29)	1.56×10^7 (3.52×10^6, 1.39×10^7, 9.00×10^7)

[6] `https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer`

[7] The shifted geometric mean of a set of n values $t_1, \ldots, t_n$ is defined as $\left(\prod_{i=1}^{n} [t_i + \text{shift}]\right)^{\frac{1}{n}} - \text{shift}$. Compared to the arithmetic mean, it is less sensitive to large variations in the values.

5.3 Feasible Cases: Searching for Solutions with the Base CP Model

Table 1 shows the time and number of branches (Num Branches) required by the base CP model (i.e., without symmetry breaking or valid inequalities) to find solution for a range of problems. Our approach was able to find Strassen's solution for the 2×2 matrix multiplication in less than a second whereas the AlphaTensor paper [7] reports a few minutes of model inference to find that solution.

Performance variability. In Table 1, we can see that for $(2, 2, 3)$ with $R = 11$, the worst seed took 245 seconds to find a feasible solution compared to 0.98 seconds for the best seed. This drastic difference in time (and ultimately the number of branches) is an indication that minute parameters such as the seed can significantly impact the CP search. For feasible instances, this phenomenon can be seen as a blessing rather than a curse if one has access to multiple cores: the randomness can be exploited by running multiple copies of the solver, terminating as soon as the first successful run is completed. This has been done in MILP [8].

5.4 Feasible Cases: Sparsity Constraints and Cyclic Invariance Help

Solving the base CP formulation, with our current time and memory budgets, does not yet yield feasible decompositions for dimensions higher than $(2, 2, 3)$ or $(2, 3, 2)$. However, after increasing both the time and memory limits, we were able to find a solution to the problem for dimension $(2, 2, 4)$ with $R = 14$ in 19.6 hours using the inexact inequalities ($K_1 = 11$ and $K_2 = 7$) developed in Section 4.4. Furthermore, our cyclic invariant formulation (with $S = 5$) with inexact inequalities ($K_1 = 9$ and $K_2 = 10$) was able to find a solution for $(3, 3, 3)$, $R = 23$. More specifically, we ran the cyclic invariant formulation for 10 hours with 5 different seeds and observed that two seeds produced a feasible solution within one hour whereas the other three seeds hit the time limit. Once again, this indicates that performance variability in the CP search is significant for our problem. Additionally, we ran the base CP formulation for $(3, 3, 3)$ without inexact inequalities for 5 seeds which all hit the time limit of 10 hours, demonstrating the benefit of the reduction of variables for the cyclic invariant formulation and sparsity constraints. We have yet to check whether using only inexact inequalities can help the base formulation for $(3, 3, 3)$. A similar result was observed for the $(2, 2, 2)$ case in which the cyclic invariant formulation ($S = 4$) with inexact inequalities ($K_1 = 6$ and $K_2 = 4$) produced an average solution time of 0.05 seconds across 10 seeds whereas the base CP model has an average of 1.46 seconds. Our current implementation is not able to find cyclic invariant solutions for $N = 4$ with $R = 48$, but we have hope that this approach is a promising tool for the search for new cyclic invariant solutions for square matrix multiplication.

5.5 Infeasible Cases: The Importance of Symmetry Breaking

Since CP performs an exhaustive search, it can provide a proof of infeasibility if a given rank R is not achievable for certain matrix dimensions. As expected, the runtime to prove infeasibility significantly increases as we approach the known minimum rank; this can be seen in Table 2 for the (2,2,2) case. It is also apparent that the addition of symmetry-breaking constraints helps tremendously when proving infeasibility given that they reduce the search space significantly. More specifically, for $R = 6$ in Table 2, it is not even currently possible to prove infeasibility without symmetry-breaking constraints in 2 hours whereas the CP model with symmetry-breaking constraints (B+S) requires around 7 minutes. These results highlight the importance of symmetry-breaking constraints when looking to prove infeasibility.

Table 2 Runtime results for the base CP model and variants to prove infeasibility of $R < 7$ for (2,2,2). "geo mean" refers to the shifted geometric mean as described in Section 5.2; "med" refers to the median and "min"/"max" to the minimum and maximum, respectively. Overall, the use of symmetry-breaking constraints (denoted by the letter "S") on top of the base CP formulation ("B") is crucial for efficient proofs of infeasibility. "V" refers to the valid inequalities of Section 4.2 which sometimes complement symmetry-breaking but are not always needed for the fastest results.

R	Method	Time (sec) geo mean (min, med, max)	Num Branches geo mean (min, med, max)
1	B	0.01 (0.00, 0.01, 0.02)	1.08×10^3 (1.06×10^3, 1.08×10^3, 1.10×10^3)
	B+S	0.00 (0.00, 0.01, 0.01)	1.00×10^{-4} (0.00, 0.00, 0.00)
	B+V	0.00 (0.00, 0.00, 0.01)	1.00×10^{-4} (0.00, 0.00, 0.00)
	B+V+S	**0.00** (0.00, 0.00, 0.00)	1.00×10^{-4} (0.00, 0.00, 0.00)
2	B	0.01 (0.00, 0.01, 0.03)	4.38×10^3 (3.72×10^3, 4.41×10^3, 5.30×10^3)
	B+S	0.01 (0.01, 0.01, 0.02)	1.08×10^3 (1.08×10^3, 1.08×10^3, 1.08×10^3)
	B+V	**0.00** (0.00, 0.01, 0.02)	1.33×10^3 (1.31×10^3, 1.32×10^3, 1.34×10^3)
	B+V+S	0.01 (0.01, 0.01, 0.03)	1.09×10^3 (1.07×10^3, 1.08×10^3, 1.10×10^3)
3	B	0.21 (0.17, 0.21, 0.28)	1.70×10^5 (1.42×10^5, 1.70×10^5, 1.95×10^5)
	B+S	0.02 (0.01, 0.02, 0.03)	4.13×10^3 (3.06×10^3, 4.30×10^3, 5.32×10^3)
	B+V	0.20 (0.13, 0.21, 0.25)	1.38×10^5 (1.14×10^5, 1.35×10^5, 1.78×10^5)
	B+V+S	**0.02** (0.01, 0.02, 0.03)	3.61×10^3 (2.52×10^3, 3.67×10^3, 4.92×10^3)
4	B	43.79 (31.33, 41.89, 66.93)	3.85×10^7 (3.06×10^7, 3.80×10^7, 5.00×10^7)
	B+S	**0.12** (0.07, 0.13, 0.18)	8.50×10^4 (6.94×10^4, 8.54×10^4, 1.03×10^5)
	B+V	53.49 (39.10, 48.87, 69.35)	3.70×10^7 (3.18×10^7, 3.69×10^7, 4.23×10^7)
	B+V+S	0.15 (0.11, 0.15, 0.20)	8.53×10^4 (7.34×10^4, 8.10×10^4, 1.06×10^5)
5	B	T.O. (N/A, N/A, N/A)	6.03×10^9 (5.56×10^9, 5.75×10^9, 7.14×10^9)
	B+S	3.06 (2.28, 3.02, 4.15)	2.22×10^6 (1.89×10^6, 2.19×10^6, 2.67×10^6)
	B+V	T.O. (N/A, N/A, N/A)	5.57×10^9 (3.97×10^9, 5.83×10^9, 6.16×10^9)
	B+V+S	**2.98** (2.56, 2.94, 3.44)	2.14×10^6 (1.91×10^6, 2.12×10^6, 2.56×10^6)
6	B	T.O. (N/A, N/A, N/A)	5.99×10^9 (4.53×10^9, 5.79×10^9, 6.93×10^9)
	B+S	**429.26** (333.88, 441.63, 528.61)	3.28×10^8 (2.94×10^8, 3.31×10^8, 3.76×10^8)
	B+V	T.O. (N/A, N/A, N/A)	4.67×10^9 (3.82×10^9, 4.73×10^9, 5.48×10^9)
	B+V+S	517.33 (414.07, 522.81, 640.65)	3.35×10^8 (2.97×10^8, 3.28×10^8, 3.95×10^8)

6 Conclusion

We have proposed a novel CP approach to solve the fast matrix multiplication problem. We have provided a set of constraints for breaking permutation and sign symmetries as well as a set of valid inequality constraints to help CP prune and propagate more efficiently. We provide a decomposition framework that is beneficial for finding feasible solutions for the largest case we have attempted, i.e., 3×3 matrix multiplication. Based on our experimental results, we have been able to solve small instances of this problem within a reasonable amount of time. This is in contrast to some existing search-based approaches (MILP, planning) that seem to struggle. In contrast to the AlphaTensor approach [7], the CP model is far more natural for this combinatorial task and is uniquely positioned to provide proof of infeasibility for some open problems in this space.

While the results of our approach are promising given the limited amount of computing used, there are several limitations that we aim to address in future work. First, our algorithm struggles to scale for larger matrix dimensions or ranks due to the quick increase in the number of variables of the CP model. Secondly, we have found that the base CP model outperforms the addition of symmetry constraints and valid inequalities in the case of feasible solutions, likely due to the latter's tendency to prune symmetric solutions early in the tree search. However, we believe that our experiment's small matrix dimensions may have skewed these results and valid inequalities may be crucial for larger sizes. Moving forward, we propose several areas for further exploration and improvement:

- Conduct larger-scale experiments using larger compute clusters to take advantage of the parallelizability of the CP solver's search procedure.
- Analyze the highly structured nature of this problem to develop more valid inequalities that can further reduce the search space of our CP model, including inexact inequalities that may not hold for all matrix multiplication dimensions but help for some cases.
- Explore solver parameter tuning, particularly for branching strategies and other important search-related decisions.
- Further investigate the idea of sparsity-based problem decomposition as a means of improving the scalability and performance of our approach.

References

1 Austin R. Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. *ACM SIGPLAN Notices*, 50(8):42–53, January 2015. doi:10.1145/2858788.2688513.

2 Markus Blaser. On the complexity of the multiplication of matrices of small formats. *Journal of Complexity*, 19(1):43–60, 2003.

3 Markus Bläser. *Fast Matrix Multiplication.* Number 5 in Graduate Surveys. Theory of Computing Library, 2013. doi:10.4086/toc.gs.2013.005.

4 Roger W Brockett and David Dobkin. On the optimal evaluation of a set of bilinear forms. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 88–95, 1973.

5 Nicolas T Courtois, Gregory V Bard, and Daniel Hulme. A new general-purpose method to multiply 3x3 matrices using only 23 multiplications, 2011.

6 Hans F de Groote. 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):127–148, 1978.

7 Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, and Grzegorz Swirszcz. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

8 Matteo Fischetti and Michele Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014.

9 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

10 Marijn JH Heule, Manuel Kauers, and Martina Seidl. Local search for fast matrix multiplication. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 155–163. Springer, 2019.

11 Marijn JH Heule, Manuel Kauers, and Martina and Seidl. New ways to multiply 3×3-matrices. *Journal of Symbolic Computation*, 104:899–916, 2021.

12 Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013.

13 Alexey V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53:1781–1795, 2013.

14 Laurent Sorber and Marc Van Barel. A mixed-integer linear program formulation for fast matrix multiplication, 2017.

15 David Speck, Paul Höft, Daniel Gnad, and Jendrik Seipp. Finding matrix multiplication algorithms with classical planning. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press, 2023.

16 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

17 Shmuel Winograd. On the number of multiplications necessary to compute certain functions. *Communications on Pure and Applied Mathematics*, 23(2):165–179, 1970.