Local Search for Fast Matrix Multiplication

Marijn J.H. Heule[1]⋆, Manuel Kauers[2]⋆⋆, and Martina Seidl[3]⋆⋆⋆

[1] Department of Computer Science, The University of Texas at Austin, United States
[2] Institute for Algebra, J. Kepler University Linz, Austria
[3] Institute for Formal Models and Verification, J. Kepler University Linz, Austria

Abstract. Laderman discovered a scheme for computing the product of two 3×3 matrices using only 23 multiplications in 1976. Since then, some more such schemes were proposed, but nobody knows how many such schemes there are and whether there exist schemes with fewer than 23 multiplications. In this paper we present two independent SAT-based methods for finding new schemes using 23 multiplications. Both methods allow computing a few hundred new schemes individually, and many thousands when combined. Local search SAT solvers outperform CDCL solvers consistently in this application.

1 Introduction

Matrix multiplication is a fundamental operation with applications in nearly any area of science and engineering. However, after more than 50 years of work on matrix multiplication techniques (see, e.g., [5, 13, 3, 11]), the complexity of matrix multiplication is still a mystery. Even for small matrices, the problem is not completely understood, and understanding these cases better can provide valuable hints towards more efficient algorithms for large matrices.

The naive way for computing the product C of two 2×2 matrices A, B requires 8 multiplications:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{21}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Strassen observed 50 years ago that C can also be computed with only 7 multiplications [15]. His scheme proceeds in two steps. In the first step he introduces auxiliary variables $M_1, \ldots, M_7$ which are defined as the product of certain linear combinations of the entries of A and B. In the second step the entries of C are

⋆ Supported by NSF grant CCF-1526760 and AFRL Award FA8750-15-2-0096.
⋆⋆ Supported by the Austrian FWF grants P31571-N32 and F5004.
⋆⋆⋆ Supported by the Austrian FWF grant NFN S11408-N23 and the LIT AI Lab funded by the State of Upper Austria.

obtained as certain linear combinations of the M_i:

$$\begin{aligned}
M_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) & c_{11} &= M_1 + M_4 - M_5 + M_7 \\
M_2 &= (a_{21} + a_{22})(b_{11}) & c_{12} &= M_3 + M_5 \\
M_3 &= (a_{11})(b_{12} - b_{22}) & c_{21} &= M_2 + M_4 \\
M_4 &= (a_{22})(b_{21} - b_{11}) & c_{22} &= M_1 - M_2 + M_3 + M_6 \\
M_5 &= (a_{11} + a_{12})(b_{22}) \\
M_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\
M_7 &= (a_{12} - a_{22})(b_{21} + b_{22})
\end{aligned}$$

Recursive application of this scheme gave rise to the first algorithm for multiplying arbitrary $n \times n$ matrices in subcubic complexity. Winograd [16] showed that Strassen's scheme is optimal in the sense that there does not exist a similar scheme with fewer than 7 multiplications, and de Groote [7] showed that Strassen's scheme is essentially unique.

Less is known about 3×3 matrices. The naive scheme requires 27 multiplications, and in 1976 Laderman [10] found one with 23. Similar as Strassen, he defines $M_1, \ldots, M_{23}$ as products of certain linear combinations of the entries of A and B. The entries of $C = AB$ are then obtained as linear combinations of $M_1, \ldots, M_{23}$. It is not known whether 23 is optimal (the best lower bound is 19 [2]). It is known however that Laderman's scheme is *not* unique. A small number of intrinsically different schemes have been found over the years. Of particular interest are schemes in which all coefficients in the linear combinations are $+1$, -1, or 0. The only four such schemes (up to equivalence) we are aware of are due to Laderman, Smirnov [14], Oh et al. [12], and Courtois et al. [6].

While Smirnov and Oh et al. found their multiplicatoin schemes with computer-based search using non-linear numerical optimization methods, Courtois found his multiplication scheme using a SAT solver. This is also what we do here. We present two approaches which allowed us to generate more than 13,000 mutually inequivalent new matrix multiplication schemes for 3×3 matrices, using altogether about 35 years of CPU years. We believe that the new schemes are of interest to the matrix multiplication community. We therefore make them publicly available in various formats and grouped by invariants at

`http://www.algebra.uni-linz.ac.at/research/matrix-multiplication/`.

2 Encoding and Workflow

To search for multiplication schemes of 3×3 matrices having the above form, we define the M_i as product of linear combination of all entries of A and B with undetermined coefficients $\alpha_{ij}^{(\ell)}, \beta_{ij}^{(\ell)}$:

$$\begin{aligned}
M_1 &= (\alpha_{11}^{(1)} a_{11} + \cdots + \alpha_{33}^{(1)} a_{33})(\beta_{11}^{(1)} b_{11} + \cdots + \beta_{33}^{(1)} b_{33}) \\
&\vdots \\
M_{23} &= (\alpha_{11}^{(23)} a_{11} + \cdots + \alpha_{33}^{(23)} a_{33})(\beta_{11}^{(23)} b_{11} + \cdots + \beta_{33}^{(23)} b_{33})
\end{aligned}$$

Similarly, we define the c_{ij} as linear combinations of the M_i with undetermined coefficients $\gamma_{i,j}^{(\ell)}$:

$$c_{11} = \gamma_{11}^{(1)} M_1 + \cdots + \gamma_{11}^{(23)} M_{23}, \quad \ldots, \quad c_{33} = \gamma_{33}^{(1)} M_1 + \cdots + \gamma_{33}^{(23)} M_{23}$$

Comparing the coefficients of all terms $a_{i_1 i_2} b_{j_1 j_2} c_{k_1 k_2}$ in the equations $c_{ij} = \sum_k a_{ik} b_{kj}$ leads to the polynomial equations

$$\sum_{\ell=1}^{23} \alpha_{i_1 i_2}^{(\ell)} \beta_{j_1 j_2}^{(\ell)} \gamma_{k_1 k_2}^{(\ell)} = \delta_{i_2 j_1} \delta_{i_1 k_1} \delta_{j_2 k_2}$$

for $i_1, i_2, j_1, j_2, k_1, k_2 \in \{1, 2, 3\}$. These 729 cubic equations with 621 variables are also known as *Brent equations* [4]. The δ_{uv} on the right are Kronecker-deltas, i.e., $\delta_{uv} = 1$ if $u = v$ and $\delta_{uv} = 0$ otherwise. Each solution of the system of these equations corresponds to a matrix multiplication scheme. The equations become slightly more symmetric if we flip the indices of the γ_{ij}, and since this is the variant mostly used in the literature, we will also adopt it from now on.

Another view on the Brent equations is as follows. View the $\alpha_{i_1 i_2}^{(\ell)}, \beta_{j_1 j_2}^{(\ell)}, \gamma_{k_1 k_2}^{(\ell)}$ as variables, as before, and regard $a_{i_1 i_2}, b_{j_1 j_2}, c_{k_1 k_2}$ as polynomial indeterminants. Then the task consists of instantiating the variables in such a way that

$$\sum_{\ell=1}^{23} (\alpha_{11}^{(\ell)} a_{11} + \cdots)(\beta_{11}^{(\ell)} b_{11} + \cdots)(\gamma_{11}^{(\ell)} c_{11} + \cdots) = \sum_{i=1}^{3} \sum_{j=1}^{3} \sum_{k=1}^{3} a_{ij} b_{jk} c_{ki}$$

holds as equation of polynomials in the variables $a_{i_1 i_2}, b_{j_1 j_2}, c_{k_1 k_2}$. Expanding the left hand side and equating coefficients leads to the Brent equations as stated before (but with indices of γ flipped, as agreed). In other words, expanding the left hand side, all terms have to cancel except for the terms on the right. We found it convenient to say that a term $a_{i_1 i_2} b_{j_1 j_2} c_{k_1 k_2}$ has "type m" if $m = \delta_{i_2 j_1} + \delta_{j_2 k_1} + \delta_{k_2 i_1}$. With this terminology, all terms of types 0, 1, 2 have to cancel each other, and all terms of type 3 have to survive. Note that since all 27 type 3 terms must be produced by the 23 summands on the left, some summands must produce more than one type 3 term.

For solving the Brent equations with a SAT solver, we use $\mathbb{Z}_2$ as coefficient domain, so that multiplication translates into 'and' and addition translates into 'xor'. When, for example, the variable $\alpha_{i_1 i_2}^{(\ell)}$ is true in a solution of the corresponding SAT instance, this indicates that the term $a_{i_1 i_2}$ occurs in M_ℓ, and likewise for the b-variables. If $\gamma_{k_1 k_2}^{(\ell)}$ is true, this means that M_ℓ appears in the linear combination for $c_{k_1 k_2}$. We call $\alpha_{i_1 i_2}^{(\ell)}$, $\beta_{j_1 j_2}^{(\ell)}$, and $\gamma_{k_1 k_2}^{(\ell)}$ the *base variables*.

In order to bring the Brent equations into CNF, we use Tseitin transformation, i.e., we introduce definitions for subformulas to avoid exponential blow-up. To keep the number of fresh variables low, we do not introduce one new variable for every cube but only for pairs of literals, i.e., we encode a cube $(\alpha \land \beta \land \gamma)$ as $u \leftrightarrow (\alpha \land \beta)$ and $v \leftrightarrow (u \land \gamma)$. In this way, we can reuse u. Furthermore, a sum $v_1 \oplus \cdots \oplus v_m$ with $m \geq 4$ is encoded by $w \leftrightarrow (v_1 \oplus v_2 \oplus v_3)$ and $v_4 \oplus \cdots \oplus v_m \oplus w$, with the latter sum being encoded recursively. This encoding seems to require the smallest sum of the number of variables and the number of clauses—a commonly used optimality heuristic. The used scripts are available at

`https://github.com/marijnheule/matrix-challenges/tree/master/src`.

The generation of new schemes proceeds in several steps, with SAT solving being the first and main step. If the SAT solver finds a solution, we next check whether it is equivalent to any known or previously found solution modulo de Groote's symmetry group [7]. If so, we discard it. Otherwise, we next try to simplify the new scheme by searching for an element in its orbit which has a smaller number of terms. The scheme can then be used to initiate a new search. In the fourth step, we use Gröbner bases to lift the scheme from the coefficient domain $\mathbb{Z}_2$ to $\mathbb{Z}$. Finally, we cluster large sets of similar solutions into parameterized families.

$$\begin{tikzcd}
\text{known schemes} \;\bullet \arrow[r, "\text{solve}"] & \bullet \arrow[r, "\text{filter}"] & \bullet \arrow[r, "\text{simplify}"] & \bullet \arrow[r, "\text{lift}"] \arrow[lll, bend left] & \bullet \arrow[r, "\text{cluster}"] & \bullet\; \text{new schemes}
\end{tikzcd}$$

In the present paper, we give a detailed description of the first step in this workflow. The subsequent steps use algebraic techniques unrelated to SAT and will be described in [9].

3 Solving Methods

The *core* of a scheme is the pairing of the type 3 terms. Our first method focuses on finding schemes with new cores, while our second method searches for schemes that are similar to an existing one and generally has the same core. For all experiments we used the local search SAT solver `yalsat` [1] as this solver performed best on instances from this application. We also tried solving these instances using CDCL solvers, but the performance was disappointing. We observed a possible explanation: The runtime of CDCL solvers tends to be exponential in the average backtrack level (ABL) on unsatisfiable instances. For most formulas arising from other applications, ABL is small (< 50), while on the matrix multiplication instances ABL is large (> 100).

3.1 Random Pairings of Type 3 Terms and Streamlining

Two of the known schemes, those of Smirnov [14] and Courtois et al. [6], have the property that each type 3 term occurs exactly once and at most two type 3 terms occur in the same summand. We decided to use this pattern to search for new schemes: randomly pair four type 3 terms and assign the remaining type 3 terms to the other 19 summands. Only in very rare cases, random pairing could be extended to a valid scheme in reasonable time, say a few minutes. In the other cases it is not known whether the pairing cannot be extended to a valid scheme or whether finding such a scheme is very hard.

Since the number of random pairings that could be extended to a valid scheme was very low, we tried adding *streamlining constraints* [8] to formulas. A streamlining constraint is a set of clauses that guides the solver to a solution, but these clauses may not (and generally are not) implied by the formula. Streamlining constraints are usually patterns observed in solutions of a given problem, potentially of smaller sizes. We experimented with various streamlining constraints, such as enforcing that each type 0, type 1, and type 2 term occurs either zero times or twice in a scheme (instead of an even number of times). The most effective streamlining constraint that we came up with was observed in the Smirnov scheme: for each summand that is assigned a single type 3 term, enforce that (i) one matrix has either two rows, two columns or a row and a column fully assigned to zero and (ii) another matrix has two rows and two columns assigned to zero, i.e., the matrix has a single nonzero entry. This streamlining constraint reduced the runtime from minutes to seconds. Yet some random pairings may only be extended to a valid scheme that does not satisfy the streamlining constraint.

3.2 Neighborhood Search

The second method is based on neighborhood search: we select a scheme, randomly fix some the corresponding base variables, and search for an assignment for the remaining base variables. This simple method turned out to be remarkably effective to find new schemes. The only parameter for this method is the number of base variables that will be fixed. The lower the number of fixed base variables, the higher the probability to find a different scheme and the higher the costs to find an assignment for the remaining base variables. We experimented with various values and it turned out that fixing 2/3 of the 621 base variables (414) is effective to find many new schemes in a reasonable time.

The neighborhood search is able to find many new schemes, but in almost all cases they have the same pairing of type 3 terms. Only in some rare cases the pairing of type 3 terms is different. Figure 1 shows such an example: scheme `A` has term $a_{13}b_{31}c_{11}$ in summand 22 and term $a_{23}b_{33}c_{32}$ in summand 23, while the neighboring scheme `B` has term $a_{13}b_{33}c_{31}$ in summand 22 and terms $a_{13}b_{31}c_{11}$, $a_{23}b_{33}c_{32}$, and $a_{13}b_{33}c_{31}$ in summand 23.

4 Evaluation and Analysis

The methods presented in Section 3 enabled us to find several hundreds of solutions individually, but they were particularly effective when combined. The first method allows finding schemes that can be quite different compared to the known schemes. However, finding a scheme using that method may require a few CPU hours as most pairings of type 3 terms cannot be extended to a valid scheme that satisfies the streamlining constraints. The second method can find schemes that are very similar to known ones with a second. The neighborhood of known solutions turned out to be limited to a few hundred of new schemes.

1	$(a_{11} + a_{13} + a_{21} + a_{22} + a_{23})(b_{13})(c_{22} + c_{32})$
2	$(a_{11} + a_{13} + a_{23})(b_{13} + b_{32})(c_{11} + c_{22} + c_{31} + c_{32})$
3	$(a_{11} + a_{13})(b_{32})(c_{21} + c_{22} + c_{31} + c_{32})$
4	$(a_{11} + a_{31})(b_{11} + b_{12} + b_{13})(c_{23})$
5	$(a_{11} + a_{33})(b_{11} + b_{13} + b_{32})(c_{11} + c_{23})$
6	$(a_{12} + a_{13} + a_{23})(b_{13} + b_{33})(c_{11} + c_{31})$
7	$(a_{12} + a_{22} + a_{32})(b_{21} + b_{22} + b_{23})(c_{33})$
8	$(a_{12} + a_{31} + a_{32} + a_{33})(b_{22})(c_{23} + c_{33})$
9	$(a_{12} + a_{33})(b_{13} + b_{21} + b_{33})(c_{11} + c_{33})$
10	$(a_{12})(b_{13} + b_{23} + b_{33})(c_{31} + c_{33})$
11	$(a_{21} + a_{31} + a_{33})(b_{11})(c_{12} + c_{22})$
12	$(a_{21})(b_{11} + b_{12} + b_{13})(c_{22})$
13	$(a_{22} + a_{31} + a_{33})(b_{13} + b_{22})(c_{12} + c_{13} + c_{22} + c_{33})$
14	$(a_{22} + a_{32} + a_{33})(b_{21})(c_{13} + c_{33})$
15	$(a_{22})(b_{13} + b_{21} + b_{22})(c_{12} + c_{13})$
16	$(a_{22})(b_{13} + b_{23})(c_{32} + c_{33})$
17	$(a_{23})(b_{31})(c_{11} + c_{12} + c_{31} + c_{32})$
18	$(a_{31} + a_{33})(b_{11} + b_{13} + b_{22})(c_{12} + c_{13} + c_{22} + c_{23})$
19	$(a_{33})(b_{11} + b_{21} + b_{31})(c_{11} + c_{13})$
20A	$(a_{12})(b_{22})(c_{21} + c_{23})$
21A	$(a_{11})(b_{12} + b_{32})(c_{21} + c_{23})$
22A	$(a_{13} + a_{33})(b_{31} + b_{32} + b_{33})(c_{11})$
23A	$(a_{23})(b_{31} + b_{32} + b_{33})(c_{11} + c_{31} + c_{32})$
20B	$(a_{11} + a_{12})(b_{22})(c_{21} + c_{23})$
21B	$(a_{11})(b_{12} + b_{22} + b_{32})(c_{21} + c_{23})$
22B	$(a_{13} + a_{33})(b_{31} + b_{32} + b_{33})(c_{31} + c_{32})$
23B	$(a_{13} + a_{23} + a_{33})(b_{31} + b_{32} + b_{33})(c_{11} + c_{31} + c_{32})$

Fig. 1. Two neighboring schemes with 19 identical summands and 4 different ones.

In contrast, some of the schemes found using the first method have a large neighborhood. We approximated the size of the neighborhood of a scheme using the following experiment: Start with a given scheme S and find a neighboring scheme by randomly fixing 2/3 of the base variables. Once a neighboring scheme S' is found, find a neighboring scheme of S', etc. We ran this experiment on a machine with 48 cores of the Lonestar 5 cluster of Texas Advanced Computing Center. We started each experiment using 48 threads with each thread assigned a different seed. Figure 2 shows the number of different schemes (after sorting) found in 1000 seconds when starting with one of the four known schemes and a scheme that was computed from the streamlining method. Some of these different schemes are new, while others are equivalent to each other or known ones. We only assure here that they are not identical after sorting the summands.

Observe that the number of different schemes found in 1000 seconds depends a lot on the starting scheme. No different neighboring scheme was found for

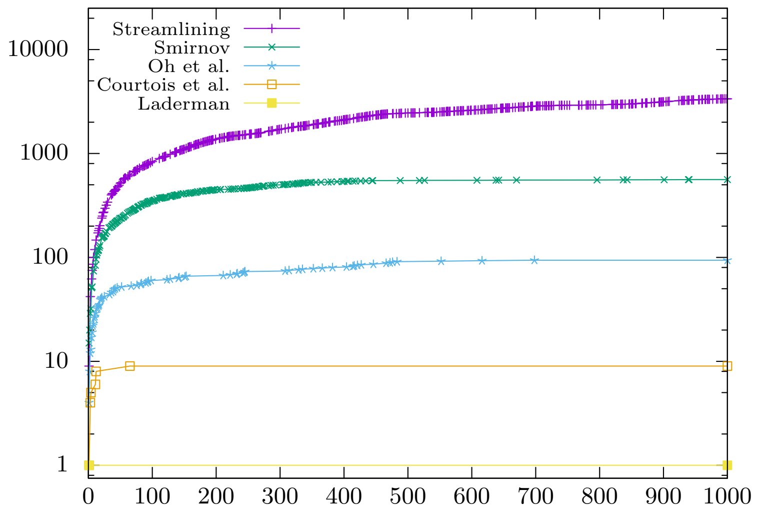

Fig. 2. The number of different schemes (vertical axis in logscale) found within a period of time (horizontal axis in seconds) during a random walk in the neighborhood of a given scheme.

Laderman's scheme, only 9 different schemes were found for the scheme of Courtois et al., 94 different schemes were found for the scheme of Oh et al., 561 new schemes were found for Smirnov's scheme, and 3359 schemes were found using a randomly selected new scheme obtained with the streamlining method.

In view of the large number of solutions we found, it is also interesting to compare them with each other. For example, if we define the *support* of a solution as the number of base variables set to 1, we observe that the support seems to follow a normal distribution with mean around 160, see Fig. 3 for a histogram. We can also see that the Laderman scheme differs in many ways from all the other solutions. It is, for example, the only scheme whose core consists of four quadruples of type 3 terms. In 89% of the solutions, the core consists of four pairs of type 3 terms, about 10% of the solution have three pairs and one quadrupel, and less than 1% of the schemes have cores of the form 2-2-2-2-3 or 2-2-2-3-4.

5 Challenges

The many thousands of new schemes that we found may still be just the tip of the iceberg. However, we also observed that the state-of-the-art SAT solving techniques are unable to answer several other questions. This section provides four challenges for SAT solvers with increasing difficulty. For each challenge we constructed one or more formulas that are available at

`https://github.com/marijnheule/matrix-challenges`.

The challenges are hard, but they may be doable in the coming years.

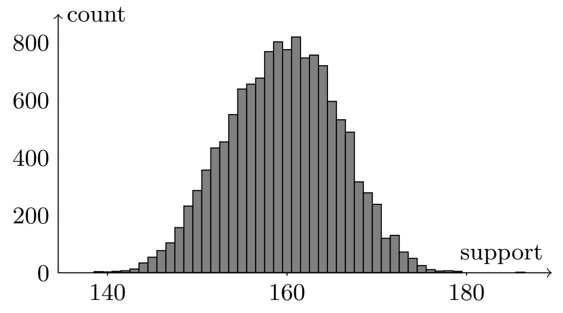

Fig. 3. Number of non-equivalent schemes found, arranged by support.

Challenge 1: Local search without streamlining. Our first method combines randomly pairing the type 3 terms with streamlining constraints. The latter was required to limit the search. We expect that local search solvers can be optimized to efficiently solve the formulas without the streamlining constraints. This may result in schemes that are significantly different compared to ones we found. We prepared ten satisfiable formulas with hardcoded pairings of type 3 terms. Five of these formulas can be solved using `yalsat` in a few minutes. All of these formulas appear hard for CDCL solvers (and many local search solvers).

Challenge 2: Prove unsatisfiability of subproblems. We observed that complete SAT solvers performed weakly on our matrix multiplication instances. It seems therefore unlikely that one could prove any optimality results for the product of two 3×3 matrices using SAT solvers in the near future. A more realistic challenge concerns proving unsatisfiability of some subproblems. We prepared ten formulas with 23 multiplications and hardcoded pairings of type 3 terms. We expect that these formulas are unsatisfiable.

Challenge 3: Avoiding a type 3 term in a summand. All known schemes have the following property: each summand has at least one type 3 term. We do not know whether there exists a scheme with 23 multiplications such that one of the summands contains no type 3 term. The challenge problem blocks the existence of a type 3 term in the last summand and does not have any additional (streamlining) constraints.

Challenge 4: Existence of a scheme with 22 multiplications. The main challenge concerns finding a scheme with only 22 multiplications. The hardness of this challenge strongly depends on whether there exists such a scheme. The repository contains a plain formula for a scheme with 22 multiplications.

Acknowledgments. The authors acknowledge the Texas Advanced Computing Center at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

References

1. Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*, pages 13–14. University of Helsinki, 2018.
2. Markus Bläser. 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.
4. Richard P. Brent. Algorithms for matrix multiplication. Technical report, Department of Computer Science, Stanford, 1970.
5. Peter Bürgisser, Michael Clausen, and Mohammad A. Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science & Business Media, 2013.
6. Nicolas Courtois, Gregory V. Bard, and Daniel Hulme. A new general-purpose method to multiply 3×3 matrices using only 23 multiplications. *CoRR*, abs/1108.2830, 2011.
7. Hans F. de Groote. On varieties of optimal algorithms for the computation of bilinear mappings i. the isotropy group of a bilinear mapping. *Theoretical Computer Science*, 7(1):1–24, 1978.
8. Carla Gomes and Meinolf Sellmann. Streamlined constraint reasoning. In *Principles and Practice of Constraint Programming (CP 2004)*, pages 274–289, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
9. Marijn J.H. Heule, Manuel Kauers, and Martina Seidl. New ways to multiply 3×3 matrices, in preparation.
10. Julian D. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bulletin of the American Mathematical Society*, 82(1):126–128, 1976.
11. Joseph M. Landsberg. *Geometry and complexity theory*, volume 169. Cambridge University Press, 2017.
12. Jinsoo Oh, Jin Kim, and Byung-Ro Moon. On the inequivalence of bilinear algorithms for 3×3 matrix multiplication. *Information Processing Letters*, 113(17):640–645, 2013.
13. Victor Y. Pan. Fast feasible and unfeasible matrix multiplication. *CoRR*, abs/1804.04102, 2018.
14. A. V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, 2013.
15. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.
16. Shmuel Winograd. On multiplication of 2×2 matrices. *Linear algebra and its applications*, 4(4):381–388, 1971.