Finding Matrix Multiplication Algorithms with Classical Planning

David Speck, Paul Höft, Daniel Gnad, Jendrik Seipp

Linköping University, Linköping, Sweden
⟨david.speck, paul.hoft, daniel.gnad, jendrik.seipp⟩@liu.se

Abstract

Matrix multiplication is a fundamental operation of linear algebra, with applications ranging from quantum physics to artificial intelligence. Given its importance, enormous resources have been invested in the search for faster matrix multiplication algorithms. Recently, this search has been cast as a single-player game. By learning how to play this game efficiently, the newly-introduced AlphaTensor reinforcement learning agent discovers many new faster algorithms. In this paper, we show that finding matrix multiplication algorithms can also be cast as a classical planning problem. Based on this observation, we introduce a challenging benchmark suite for classical planning and evaluate state-of-the-art planning techniques on it. We analyze the strengths and limitations of different planning approaches in this domain and show that we can use classical planning to find lower bounds and concrete algorithms for matrix multiplication.

Introduction

In the age of big data and deep learning the demand for efficient computation is higher than ever. One particularly important operation that is crucial for many applications is the multiplication of matrices. Many fields in industry and research depend on matrix multiplication, ranging from weather simulations, via quantum physics, to computer graphics and machine learning.

Given the ubiquity of matrix multiplication (MM), great effort has been spent on deriving more efficient algorithms (Strassen 1973; Laderman 1976; Bläser 2003; Heule, Kauers, and Seidl 2019; Fawzi et al. 2022). In this context, *more efficient* means that an algorithm uses fewer multiplications, which are the critical operations. Even a minor reduction of these operations for the multiplication of small matrices will result in huge savings of compute time and energy. This is because (1) the multiplication of large matrices can be composed of algorithms for smaller ones, and (2) given the sheer number of multiplications being executed for example to train a neural network, boosting this basic operation can tremendously speed up the overall process.

Finding more efficient MM algorithms is extremely challenging, since minimizing the number of multiplications is an NP-complete problem (Håstad 1989). Recently, Fawzi et al. (2022) presented a novel approach to MM algorithm discovery based on casting the problem as a single-player game. The action space of this game is vast, with over 2^{27} actions for two 3×3 matrices, which is orders of magnitude larger than that of the game Go with hundreds of actions. By training a reinforcement learning agent, AlphaTensor, they established faster algorithms for several matrix sizes.

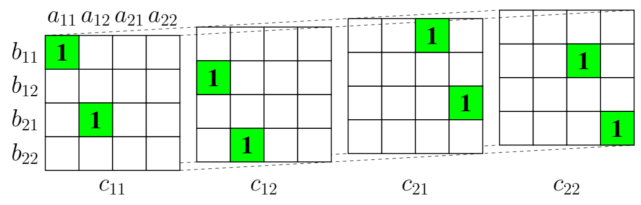

Figure 1: Initial tensor $\mathcal{T}_{2,2,2}$ of a matrix multiplication task $\mathbf{C} = \mathbf{AB}$ of two 2×2 matrices. The highlighted cells indicate which entries to read from the input matrices $\mathbf{A}$ and $\mathbf{B}$ and where to write the result into the $\mathbf{C}$ matrix. For example, $c_{11} = a_{11}b_{11} + a_{12}b_{21}$.

In this work, we propose to model the search for MM algorithms as a classical planning problem. Similarly to previous work on MM discovery (e.g., Heule, Kauers, and Seidl 2021; Fawzi et al. 2022), we restrict the solution space to MM over modular arithmetic in $\mathbb{Z}_2$, so shorter algorithms could possibly exist for $\mathbb{Z}$ that cannot be discovered using our encoding. We show the correctness of our model and evaluate a variety of planning techniques on the MM discovery problem. In contrast to Fawzi et al. (2022), today's off-the-shelve planners fail to uncover *novel* algorithms (with a tiny fraction of Fawzi et al.'s compute power). However, already with the current (non-tailored) state of the art in planning, we can derive non-trivial lower bounds and even find MM algorithms that are structurally different from the textbook algorithm.

A key contribution of this work is a novel benchmark suite that is highly relevant for practical applications. We believe that it is an exciting avenue for future planning research to use these benchmarks and derive stronger techniques for removing planner bottlenecks that only appear when faced with the vast size of MM problems.

Background

We consider the important special case of Boolean matrix multiplication (MM), i.e., over modular arithmetic in $\mathbb{Z}_2 = \{0, 1\}$, which forms a quotient ring. Boolean MM is a fundamental operation in many areas such as graph algorithms, databases, data mining, and context-free grammar parsing (Lingas 2009; Yu 2018). In addition, Heule, Kauers, and Seidl (2021) describe an algorithm for converting an MM algorithm over $\mathbb{Z}_2$ into an algorithm over $\mathbb{Z}$ that requires the same number of multiplications. This algorithm is not guaranteed to find a generalization, but the authors report that it almost always succeeds in practice. Therefore, using $\mathbb{Z}_2$ is common in the literature on finding algorithms for MM. Note that in the $\mathbb{Z}_2$ ring, multiplication translates into a logical $\wedge$ (and), while subtraction and addition are equivalent and translate into a logical $\oplus$ (xor).

An $m \times n$ *matrix* $\mathbf{A}$ is a two-dimensional array with m rows, n columns, and $m \cdot n$ elements $a_{ij} \in \mathbb{Z}_2^{m \times n}$, where a_{ij} denotes the entry in row i and column j. Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{B}$ be an $n \times p$ matrix, then the matrix product $\mathbf{C} = \mathbf{AB}$ is an $m \times p$ matrix with $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. By the *textbook algorithm*, we refer to the most widely known algorithm for MM, which uses a nested loop over all three matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, requiring $m \cdot n \cdot p$ multiplications.

Matrix Multiplication as Tensor Decomposition

An *x-D tensor* $\mathcal{T}$ is an x-dimensional array of entries (from $\mathbb{Z}_2$), a *scalar* $z \in \mathbb{Z}_2$ is a 0-D tensor, a *vector* $\mathbf{v}$ is a 1-D tensor, and a matrix $\mathbf{A}$ is a 2-D tensor. Throughout the paper, we use boldface and uppercase to distinguish between the different types, i.e., a bold uppercase $\mathbf{A}$ refers to a matrix, a bold lowercase $\mathbf{v}$ refers to a vector, a non-bold lowercase z refers to a scalar. Since MM is bilinear, it is possible to represent the operation as a 3-D tensor (Strassen 1973; Lim 2021; Fawzi et al. 2022). More precisely, a MM of two matrices with sizes $m \times n$ and $n \times p$, respectively, can be represented by a 3-D tensor $\mathcal{T}_{m,n,p}$ of size $m^2 \times n^2 \times p^2$.

Example 1 *Consider a matrix multiplication $\mathbf{C} = \mathbf{AB}$ of two 2×2 matrices. Figure 1 shows the tensor $\mathcal{T}_{2,2,2}$, where the entries with 1 describe which elements from $\mathbf{A}$ and $\mathbf{B}$ have to be considered and where the result has to be inserted into $\mathbf{C}$.*

A tensor $\mathcal{T}_{m,n,p}$ describing a MM can be written as a column-wise outer product of tensors of rank one, $\mathcal{T}_{m,n,p} = \sum_{q=1}^{R} \mathbf{u}_q \otimes \mathbf{v}_q \otimes \mathbf{w}_q$, where R is the number of multiplications of the underlying algorithm (Fawzi et al. 2022).

Example 2 *Figure 2 shows a decomposition of the tensor $\mathcal{T}_{2,2,2}$ from Figure 1 known as Strassen algorithm (Strassen 1969). When $\mathcal{T}_{m,n,p}$ is decomposed into $\mathbf{U}, \mathbf{V}$, and $\mathbf{W}$, each entry in $\mathbf{u_q}$ and $\mathbf{v_q}$ is a selection of entries from $\mathbf{A}$ and $\mathbf{B}$, that are individually summed and then multiplied. The entries of $\mathbf{w_q}$ are the places in $\mathbf{C}$ where the result is written.*

It has been shown that the rank decomposition for MM is NP-complete (Håstad 1989). In fact, the problem is so difficult that, while for $\mathcal{T}_{2,2,2}$ the optimal decomposition with $R = 7$ (Strassen algorithm) is known (Winograd 1971), already for $\mathcal{T}_{3,3,3}$ only a lower bound of 19 and an upper bound

$$\mathbf{U} = \begin{bmatrix} 1&0&1&0&1&1&0 \\ 0&0&0&0&1&0&1 \\ 0&1&0&0&0&1&0 \\ 1&1&0&1&0&0&1 \end{bmatrix} \qquad \begin{aligned} m_1 &= \boxed{(a_{11} + a_{22})}\boxed{(b_{11} + b_{22})} \\ m_2 &= (a_{21} + a_{22}) b_{11} \\ m_3 &= a_{11}(b_{12} + b_{22}) \\ m_4 &= a_{22}(b_{21} + b_{11}) \end{aligned}$$

$$\mathbf{V} = \begin{bmatrix} 1&1&0&1&0&1&0 \\ 0&0&1&0&0&1&0 \\ 0&0&0&1&0&0&1 \\ 1&0&1&0&1&0&1 \end{bmatrix} \qquad \begin{aligned} m_5 &= (a_{11} + a_{12}) b_{22} \\ m_6 &= (a_{21} + a_{11})(b_{11} + b_{12}) \\ m_7 &= (a_{12} + a_{22})(b_{21} + b_{22}) \end{aligned}$$

$$\mathbf{W} = \begin{bmatrix} 1&0&0&1&1&0&1 \\ 0&0&1&0&1&0&0 \\ 0&1&0&1&0&0&0 \\ 1&1&1&0&0&1&0 \end{bmatrix} \qquad \begin{aligned} c_{11} &= \boxed{m_1} + m_4 + m_5 + m_7 \\ c_{12} &= m_3 + m_5 \\ c_{21} &= m_2 + m_4 \\ c_{22} &= \boxed{m_1} + m_2 + m_3 + m_6 \end{aligned}$$

Figure 2: A decomposition (left) for a matrix multiplication problem $\mathcal{T}_{2,2,2}$ of two matrices of size 2×2 and the Strassen algorithm described by it (right) in $\mathbb{Z}_2$.

of 23 have been proven (Laderman 1976; de Groote 1978; Bläser 2003). This is remarkable because finding faster algorithms for small matrices, i.e., algorithms that require fewer multiplications, can be of significant practical importance, since algorithms such as Strassen can be used recursively to multiply larger matrices.

Classical Planning

A *planning problem* is a tuple $\Pi = (\mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{O})$, where $\mathcal{V}$ is a set of *state variables*. Each variable $v \in \mathcal{V}$ can be false (denoted by $v = 0$) or true (denoted by $v = 1$). A *partial state* is a truth assignment over a subset of variables $V \subseteq \mathcal{V}$ and a *state* is a truth assignment over all variables in $\mathcal{V}$. $\mathcal{I}$ is the *initial state*, and $\mathcal{G}$ is the partial state that describes the *goal*. $\mathcal{O}$ is a set of *operators* where each $o \in \mathcal{O}$ is a pair $\langle pre_o, \mathit{eff}_o \rangle$. The *precondition* $pre_o \subseteq \mathcal{V}$ is a partial state and eff_o is a set of conditional effects $(cond \rhd v = val)$, where $cond \subseteq \mathcal{V}$ is a partial state, $v \in \mathcal{V}$, and $val \in \{0, 1\}$. An operator is *applicable* in a state s if $pre_o \subseteq s$. Applying an operator o in a state s yields a state $s' = s[\![o]\!]$, where for all $v \in \mathcal{V}$, $s'(v) = val$ if there is a $(cond \rhd v = val) \in \mathit{eff}_o$ and $cond \subseteq s$, and $s'(v) = s(v)$ otherwise. The objective of classical planning is to determine a sequence of operators (a *plan*) π whose execution leads from the initial state to a goal state. By $|\pi|$ we denote the number of operators in a plan π. A plan is considered *optimal* if there is no plan with fewer operators.

Matrix Multiplication as Classical Planning

In the following, we show how the search for MM algorithms can be modeled as a classical planning problem and prove that our encoding is sound and complete.

Similar to Fawzi et al. (2022), the underlying idea is that a state represents a tensor and actions describe how to update that tensor by choosing different $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ vectors. Definition 1 specifies a planning problem $\Pi_{m,n,p}$ that describes the search for MM algorithms of two matrices with sizes $m \times n$ and $n \times p$, respectively. Since such a problem can be described by a tensor $\mathcal{T}_{m,n,p}$ with $|\mathcal{T}_{m,n,p}| = m^2 \cdot n^2 \cdot p^2$ entries, we use $|\mathcal{T}_{m,n,p}|$ propositional state variables $\mathcal{V}$ to represent every possible assignment of zeros and ones to the tensor. The initial state describes which entries to read from the input matrices and where to write the result. This is specified by true state variables. In the unique goal state, all variables are false. Finally, we define an operator o for each possible choice of the three vectors $\mathbf{u} = \mathbf{u}(o)$, $\mathbf{v} = \mathbf{v}(o)$, and $\mathbf{w} = \mathbf{w}(o)$. The sizes of $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ are $m \cdot n$, $n \cdot p$, and $m \cdot p$ respectively, which is reflected in the number of operators. The operators update the current state s, i.e., the tensor T_s, in such a way that $s' = s[\![o]\!]$ represents the tensor $T_{s'} = T_s - \mathbf{u}(o) \otimes \mathbf{v}(o) \otimes \mathbf{w}(o)$. Recall that subtraction and addition are equivalent in the quotient ring $\mathbb{Z}_2$ and correspond to a logical xor. Therefore, the effect $\mathit{eff}(o)$ of an operator $o \in \mathcal{O}$ inverts the value of a state variable iff the corresponding entry in the tensor $\mathbf{u}(o) \otimes \mathbf{v}(o) \otimes \mathbf{w}(o)$ is 1.

Definition 1 *Given a MM problem $\mathbf{C} = \mathbf{AB}$ of two matrices with sizes $m \times n$ and $n \times p$, we define the corresponding planning problem $\Pi_{m,n,p} = (\mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{O})$ as follows:*

$$\mathcal{V} = \{v_i \mid 1 \leq i \leq m^2 \cdot n^2 \cdot p^2\}$$

Each variable v_i corresponds to an entry of a 3-D tensor. By $v[a_{ij}, b_{kl}, c_{xy}]$ we denote the entry in the tensor representing the position a_{ij} in the matrix $\mathbf{A}$, the position b_{kl} in $\mathbf{B}$, and the position c_{xy} in $\mathbf{C}$ (see Figure 1). Furthermore, we define $M = \{1, \dots, m\}$, $N = \{1, \dots, n\}$, and $P = \{1, \dots, p\}$.

$\mathcal{I}(v[a_{ij}, b_{jk}, c_{ik}]) = 1$ *for all $i \in M$, $j \in N$, and $k \in P$*
$\mathcal{I}(v) = 0$ *otherwise*
$\mathcal{G}(v) = 0$ *for all $v \in \mathcal{V}$*

We have a set of operators for all possible binary vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ that update the current tensor state.

$$\mathcal{O} = \{o_i \mid 1 \leq i \leq 2^{m \cdot n + n \cdot p + m \cdot p}\}$$

Each operator o encodes three concrete binary vectors $\mathbf{u}(o), \mathbf{v}(o), \mathbf{w}(o)$ and is defined as:

$\mathit{pre}(o) = \emptyset$
$\mathit{eff}(o) = \{v_i = 0 \rhd v_i = 1, v_i = 1 \rhd v_i = 0\}$

for all i where $x_i = 1$ in the vectorized/flattened result $(x_1, \dots, x_{|\mathcal{V}|}) = vec(\mathbf{u}(o) \otimes \mathbf{v}(o) \otimes \mathbf{w}(o))$.

Example 3 *Consider the MM task $\mathcal{T}_{2,2,2}$. The planning problem $\Pi_{2,2,2}$ modeling this task has 64 binary variables, where the initial state represents the tensor shown in Figure 1. A possible plan $\pi = \langle o_1, \dots, o_7 \rangle$ for $\Pi_{2,2,2}$ is represented by the matrices of Figure 2 (left). Each operator o_i corresponds to the vectors $\mathbf{u} = \mathbf{u}(o)$, $\mathbf{v} = \mathbf{v}(o)$, and $\mathbf{w} = \mathbf{w}(o)$ of the i-th column of the matrix $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$. This plan π is optimal and describes the Strassen algorithm.*

Theoretical Properties

Similar to the TensorGame described by Fawzi et al. (2022), which was used to express a reinforcement learning problem, the planning problem specified in Definition 1 models the decomposition of a tensor in a sequential manner. Thus, the reasoning for why the solutions of such planning problems describe concrete matrix multiplication algorithms is analogous. We briefly summarize the main arguments with respect to the defined planning problems.

The initial state represents a tensor $\mathcal{T}_{m,n,p}$ describing a MM, the unique goal state describes the zero tensor $\mathbf{0} = \mathcal{T}_0$, and an operator o yields a successor state $s[\![o]\!] = s'$ such that $T_{s'} = T_s - \mathbf{u}(o) \otimes \mathbf{v}(o) \otimes \mathbf{w}(o)$. Thus, since a solution $\pi = \langle o_1, \dots, o_R \rangle$ to a planning problem $\Pi_{n,m,p}$ yields a sequence of operators from the initial state to the goal, it also holds that $\mathcal{T}_{m,n,p} = \sum_{t=1}^{R} \mathbf{u}(o_t) \otimes \mathbf{v}(o_t) \otimes \mathbf{w}(o_t)$ (Fawzi et al. 2022) which is why π describes a concrete algorithm with $R = |\pi|$ multiplications.

Proposition 1 *The planning problem $\Pi_{n,m,p}$ is* sound *and* complete*, i.e., the solutions of $\Pi_{n,m,p}$ exactly capture all possible matrix multiplication algorithms for a matrix multiplication task of two matrices of sizes $m \times n$ and $n \times p$. Further, any solution π to a planning problem $\Pi_{n,m,p}$ describes an algorithm with $|\pi|$ multiplications.*

The MM Benchmark Suite

To support solving MM planning tasks with different classical planners, we wrote automatic problem generators that output different versions of $\Pi_{m,n,p}$. The model described in Definition 1 can be directly transformed into the ground SAS$^+$-based (Bäckström and Nebel 1995) task representation supported by all planners that build on the Fast Downward planning system (Helmert 2006). Additionally, we generated lifted versions using the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). In total, we generated the planning problems of MM algorithm discovery for matrices with sizes between 1×1 and 3×3, resulting in 18 instances ranging from the trivial $\Pi_{1,1,1}$ problem up to the $\Pi_{3,3,3}$ task. The $\Pi_{3,3,3}$ problem has over 134 million non-trivial operators, making it an extremely challenging task for modern planners.[1]

Experiments

We evaluated a variety of planning approaches on the MM benchmark suite. The objective of this experiment is to investigate which approaches and planners perform well on the challenging problem of finding algorithms for MM. In the following, we discuss the experimental setup, the most interesting results, and analyze the strengths of the different planning approaches, as well as the limiting factors.

Setup

In total, we used four different planning systems to investigate different optimal and satisfying planning techniques. This includes the dual best-first width search planner (DualBFWS) from the International Planning Competition 2018 (Francès et al. 2018) and the lifted SAT-based planner LiSAT

[1]If at least one of the vectors $\mathbf{u}(o)$, $\mathbf{v}(o)$, or $\mathbf{w}(o)$ represented by an operator o is the zero vector $\mathbf{0}$, the operator has no effect and can be omitted.

(Höller and Behnke 2022). For both planners, the input language was PDDL, although for LiSAT we used a model without conditional effects, as this planner does not support this language feature. We also ran experiments with the symbolic search planner Symk (Speck, Mattmüller, and Nebel 2020) and a slightly adapted version of the heuristic search planner Scorpion (Seipp, Keller, and Helmert 2020).[2] Both of these planners are based on Fast Downward (Helmert 2006), so we directly used the ground SAS$^+$ models for these experiments. For each MM planning task, we ran the considered planning algorithms on a single CPU core with a time limit of 10 hours and a memory limit of 80 GB. All of our benchmark generators, source code, and experimental data are available online (Speck et al. 2023).

Satisficing Planning

In satisficing planning, one is interested in quickly finding a short solution, but not necessarily the shortest one.

Heuristic Search Dual-BFWS is a state-of-the-art heuristic search approach that performs a best-first width search with novelty pruning, followed by subsequent heuristic searches (Lipovetzky and Geffner 2017; Francès et al. 2018). Table 1 shows that Dual-BFWS finds several solutions, i.e., concrete algorithms, up to a matrix size of $\mathcal{T}_{2,3,2}$. Comparing the results with the best known bounds, we find that Dual-BFWS is not able to find better algorithms than the textbook algorithm. For the largest tasks, Dual-BFWS cannot ground the task due to the memory constraints.

We also ran LAMA (Richter and Westphal 2010), which uses greedy-best first search and multiple heuristics. In addition, we tested hill climbing (HC) with the goal count heuristic h^{GC}. Both algorithms are part of Scorpion (Seipp, Keller, and Helmert 2020) and we passed them the ground SAS$^+$ models as input. LAMA solves most of the tasks suboptimally and is not able to optimize the first solution found, which is the textbook algorithm. These results show that finding shorter solutions than the textbook algorithm is a challenge for modern satisficing planners. For the largest task $\Pi_{3,3,3}$, with more than 130 million actions, LAMA starts its first search, but it immediately runs out of memory. Hill climbing with the goal count heuristic h^{GC} overcomes this issue as it requires very little memory and is capable of finding a concrete algorithm for $\mathcal{T}_{3,3,3}$. This is an interesting observation that opens the door to further research on how such a memory-efficient but greedy approach can be used to find better MM algorithms.

Lifted SAT-based Planning Since grounding the larger tasks is challenging, we considered planners that operate on a lifted representation. In Table 1 we show the results of the LiSAT planner (Höller and Behnke 2022) using its default satisficing configuration as a representative of lifted SAT-based planners. It finds solutions for the smaller tasks, but since SAT-based planners model the planning task with a fixed upper plan length bound, the solutions are very long.

	Satisficing				Optimal				Rank	
Matrix Sizes	D–BFWS	LAMA	HC + h^{GC}	Lifted SAT	A* + h^{blind}	A* + h^{PDB}	Lifted SAT	Symbolic	Bounds	Textbook
1 1 1	*1	*1	*1	*101	*1	*1	*1	*1	1	1
1 1 2	*2	*2	*2	*21	*2	*2	*2	*2	2	2
1 2 1	*2	*2	*2	*21	*2	*2	*2	*2	2	2
1 1 3	*3	*3	*3	*12	*3	*3	*3	*3	3	3
1 3 1	*3	*3	*3	*12	*3	*3	*3	*3	3	3
1 2 2	*4	*4	*4	*12	*4	*4	*4	*4	4	4
2 1 2	*4	*4	*4	*9	*4	*4	*4	*4	4	4
1 2 3	*6	*6	*6	*6	4	*6	1	*6	6	6
1 3 2	*6	*6	*6	*8	4	*6	1	*6	6	6
2 1 3	*6	*6	*6	*6	4	*6	1	*6	6	6
2 2 2	*8	*8	*8	–	3	5	1	7	7	8
1 3 3	*9	*9	*9	–	3	4	1	7	7/9	9
3 1 3	*9	*9	*9	–	3	4	1	7	7/9	9
2 2 3	*12	*12	*12	–	3	3	1	3	7/11	12
2 3 2	*12	*12	*12	–	3	4	1	3	7/11	12
2 3 3	–	*18	*18	–	2	2	–	3	7/15	18
3 2 3	–	*18	*18	–	2	2	–	3	7/15	18
3 3 3	–	–	*27	–	2	–	–	–	19/23	27

Table 1: Solution lengths of matrix multiplication tasks found by various satisficing and optimal planning algorithms. The entries with asterisks indicate the length of a found plan describing an algorithm, all other entries show proved lower bounds. With ”–” we indicate that no solution or lower bound could be found due to running out of memory or time. The two right columns serve as a reference for the rank of the decomposition problem with the lower and upper bounds we are aware of and the textbook algorithm.

We observed similar results with the SAT-based Madagascar planner that considers the ground task (Rintanen, Heljanko, and Niemelä 2006). Manually specifying informed bounds can lead to better solutions, although we found no improvement over the textbook algorithm. Other lifted planners, such as the search-based Powerlifted planner (Corrêa et al. 2020; Corrêa and Seipp 2022), had difficulty finding a solution in most instances, mainly due to memory.

Optimal Planning

In optimal planning, the objective is to provably find the shortest plan. Unlike other approaches such as AlphaTensor (Fawzi et al. 2022), optimal planners can not only *find* algorithms, but also *prove* that no algorithm with a lower number of multiplications in $\mathbb{Z}_2$ exists.

Heuristic Search We report results for A* with the blind heuristic h^{blind} and A* with a PDB-based heuristic h^{PDB} (Edelkamp 2001), which partitions the variables into disjoint patterns and computes the maximum over the PDB estimates. We considered pattern sizes of 5, 10, 15, and 20 and report the best result from these configurations for each task. Exploratory experiments with other state-of-the-art heuristics showed that none of them is preferable to these two simple optimal approaches for MM tasks. Table 1 shows that

[2]Our version of the Scorpion planner supports only one condition per conditional effect to be more memory efficient.

both approaches, A* with h^{blind} and h^{PDB}, are able to find multiple optimal algorithms, and using h^{PDB} solves three more tasks than h^{blind}. For the larger tasks, both approaches prove non-trivial lower bounds, but run out of memory before the task can be solved. Overall, it can be seen that the PDB heuristics have a positive impact, as more tasks can be solved and the proven lower bounds are often higher.

Lifted SAT-based Planning Table 1 also shows the results of LiSAT when run in optimal mode. SAT-based planning has the advantage that it often requires less memory than traditional search-based approaches, but it is also often more time-consuming, which is the limiting factor for our MM benchmark suite. In 10 hours, LiSAT finds optimal algorithms for seven tasks and proves that there is no algorithm of length zero for the remaining tasks up to a matrix size of $\mathcal{T}_{2,3,2}$. Nevertheless, due to their memory efficiency, SAT-based planners are interesting candidates for finding optimal MM algorithms in the future (Abdulaziz 2021).

Symbolic Search We used the Symk planner (Speck, Mattmüller, and Nebel 2020; Speck 2022) to perform symbolic bidirectional blind search with and without dynamic variable reordering during search (Kissmann and Hoffmann 2014; Torralba et al. 2017). Again, we report the best result from these two configurations for each task. Overall, symbolic search solved the same number of tasks as heuristic search in the form of A* with PDB heuristics. However, since Symk is a top-k planner, we were able to generate not only a single optimal solution, i.e., MM algorithm, but several or even all structurally different optimal algorithms for the solved MM tasks, which can be of interest since some algorithms are numerically more stable than others (Ballard et al. 2016). By structurally different, we mean that the plans found use different operators and are not simple reorderings (Katz, Sohrabi, and Udrea 2020). Moreover, Symk proved some interesting non-trivial lower bounds, e.g., that every algorithm for $\mathcal{T}_{2,2,2}$ requires at least 7 multiplications, confirming the known lower bound (Winograd 1971). Memory proved to be the limiting factor, although we found that dynamic reordering had a positive effect (Kissmann and Hoffmann 2014).

Discussion and Future Work

In the following, we discuss the presented approach for finding matrix multiplication algorithms with classical planning and interesting future work.

Operator Order and Grouping In the presented planning model of an MM problem, the order in which operators are applied is irrelevant. This can be difficult for modern planners, which typically face tasks where the order of operators is important. Moreover, looking more closely at the underlying MM problem, we find that operators representing the same $\mathbf{u}$ and $\mathbf{v}$ vectors can be grouped together. Since these operators differ only in the $\mathbf{w}$ vector, at most one operator from each of these groups needs to be applied in any optimal solution. There are different ways to utilize this observation in the future. One is to encode the order of operator applications in the model and/or enforce that only one operator from each group can be applied. However, this can lead to a severe increase in the size of the model. Another way is to address the algorithmic side by not considering the order in which operators are applied in the search, but only whether an operator needs to be applied or not. Such ideas have been explored in delete-free planning (Pommerening and Helmert 2012) using methods from operations research such as branch-and-bound (Lawler and Wood 1966). Finally, we want to mention that partial order reduction (Alkhazraji et al. 2012) is not straightforward to apply to MM tasks, since there are many conflicting operator effects.

Partial Models While in the presented work we explicitly model all possible operators, operators are sampled in the TensorGame (Fawzi et al. 2022). This helps to reduce memory requirements, but at the same time the guarantee that the model contains optimal solutions is lost, as is the ability to prove lower bounds. Nevertheless, the generation of partial models of MM problems by sampling operators or by the more informed way of partially grounding the problem (Gnad et al. 2019) is a promising direction to overcome the memory bottleneck of modern planners when dealing with tasks from this domain.

Conclusions

In this paper, we showed that finding algorithms for matrix multiplication can be modeled as a classical planning problem in a sound and complete way. Our work provides a challenging benchmark suite, highly relevant to practical applications, on which we evaluated modern planners. The results show that we can use planning to find concrete algorithms and prove non-trivial bounds. This is remarkable considering the small resources of ten hours we used, compared to several years of CPU and GPU time of other works (Heule, Kauers, and Seidl 2019; Fawzi et al. 2022). However, our research also shows that memory is the most limiting factor for planning approaches in this domain. We believe that this work lays a new foundation for research in this domain and hope that it will encourage researchers from the planning community to join us in this quest.

Acknowledgments

We thank Hector Geffner for bringing this interesting problem to our attention and the anonymous reviewers for their insightful comments. This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

References

Abdulaziz, M. 2021. Cost Optimal Planning as Satisfiability. arXiv:2103.02355 [cs.AI].

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A Stubborn Set Algorithm for Optimal Planning. In *Proc. ECAI 2012*, 891–892.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Ballard, G.; Benson, A. R.; Druinsky, A.; Lipshitz, B.; and Schwartz, O. 2016. Improving the numerical stability of fast matrix multiplication. *SIAM Journal on Matrix Analysis and Applications*, 37(4): 1382–1418.

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

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proc. ICAPS 2020*, 80–89.

Corrêa, A. B.; and Seipp, J. 2022. Best-First Width Search for Lifted Classical Planning. In *Proc. ICAPS 2022*, 11–15.

de Groote, H. F. 1978. On varieties of optimal algorithms for the computation of bilinear mappings II. optimal algorithms for 2×2-matrix multiplication. *Theoretical Computer Science*, 7(2): 127–148.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proc. ECP 2001*, 84–90.

Fawzi, A.; Balog, M.; Huang, A.; Hubert, T.; Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Ruiz, F. J. R.; Schrittwieser, J.; Swirszcz, G.; Silver, D.; Hassabis, D.; and Kohli, P. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53.

Francès, G.; Geffner, H.; Lipovetzky, N.; and Ramiréz, M. 2018. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In *IPC-9 Planner Abstracts*, 23–27.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proc. AAAI 2019*, 7602–7609.

Håstad, J. 1989. Tensor rank is NP-complete. In *International Colloquium on Automata, Languages, and Programming*, 451–460. Springer.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Heule, M. J.; Kauers, M.; and Seidl, M. 2021. New ways to multiply 3×3-matrices. *Journal of Symbolic Computation*, 104: 899–916.

Heule, M. J. H.; Kauers, M.; and Seidl, M. 2019. Local Search for Fast Matrix Multiplication. In *Proc. SAT 2019*, 155–163.

Höller, D.; and Behnke, G. 2022. Encoding Lifted Classical Planning in Propositional Logic. In *Proc. ICAPS 2022*, 134–144.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-Quality Planning: Finding Practically Useful Sets of Best Plans. In *Proc. AAAI 2020*, 9900–9907.

Kissmann, P.; and Hoffmann, J. 2014. BDD Ordering Heuristics for Classical Planning. *JAIR*, 51: 779–804.

Laderman, J. D. 1976. *On Algorithms for Minimizing the Number of Multiplications in Matrix Products*. Ph.D. thesis, New York University, USA.

Lawler, E. L.; and Wood, D. E. 1966. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4): 699–719.

Lim, L. 2021. Tensors in computations. arXiv:2106.08090 [math.NA].

Lingas, A. 2009. A Fast Output-Sensitive Algorithm for Boolean Matrix Multiplication. In *Algorithms – ESA*, 408–419.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI 2017*, 3590–3596.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Pommerening, F.; and Helmert, M. 2012. Optimal Planning for Delete-free Tasks with Incremental LM-cut. In *Proc. ICAPS 2012*, 363–367.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *AIJ*, 170(12–13): 1031–1080.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *JAIR*, 67: 129–167.

Speck, D. 2022. *Symbolic Search for Optimal Planning with Expressive Extensions*. Ph.D. thesis, University of Freiburg.

Speck, D.; Höft, P.; Gnad, D.; and Seipp, J. 2023. Code and data for the ICAPS 2023 paper “Finding Matrix Multiplication Algorithms with Classical Planning”. https://doi.org/10.5281/zenodo.7731696.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. AAAI 2020*, 9967–9974.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4): 354–356.

Strassen, V. 1973. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264: 184–202.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *AIJ*, 242: 52–79.

Winograd, S. 1971. On multiplication of 2×2 matrices. *Linear Algebra and its Applications*, 4(4): 381–388.

Yu, H. 2018. An Improved Combinatorial Algorithm for Boolean Matrix Multiplication. *Information and Computation*, 261: 240–247.