

The Time Complexity of Fully Sparse Matrix Multiplication

Amir Abboud*
Weizmann Institute of Science

Karl Bringmann†
Saarland University
Max Planck Institute for Informatics

Nick Fischer‡
Weizmann Institute of Science

Marvin Künnemann§
RPTU Kaiserslautern-Landau

Abstract

What is the time complexity of matrix multiplication of sparse integer matrices with m_{in} nonzeros in the input and m_{out} nonzeros in the output? This paper provides improved upper bounds for this question for almost any choice of m_{in} vs. m_{out} , and provides evidence that these new bounds might be optimal up to further progress on fast matrix multiplication.

Our main contribution is a new algorithm that reduces sparse matrix multiplication to dense (but smaller) rectangular matrix multiplication. Our running time thus depends on the optimal exponent $\omega(a, b, c)$ of multiplying *dense* $n^a \times n^b$ by $n^b \times n^c$ matrices. We discover that when $m_{\text{out}} = \Theta(m_{\text{in}}^r)$ the time complexity of sparse matrix multiplication is $O(m_{\text{in}}^{\sigma+\epsilon})$, for all $\epsilon > 0$, where σ is the solution to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$. No matter what $\omega(\cdot, \cdot, \cdot)$ turns out to be, and for all $r \in (0, 2)$, the new bound beats the state of the art, and we provide evidence that it is optimal based on the complexity of the all-edge triangle problem.

In particular, in terms of the input plus output size $m = m_{\text{in}} + m_{\text{out}}$ our algorithm runs in time $O(m^{1.3459})$. Even for Boolean matrices, this improves over the previous $m^{\frac{2\omega}{\omega+1}+\epsilon} = O(m^{1.4071})$ bound [Amossen, Pagh; 2009], which was a natural barrier since it coincides with the longstanding bound of all-edge triangle in sparse graphs [Alon, Yuster, Zwick; 1994]. We find it interesting that matrix multiplication can be solved faster than triangle detection in this natural setting. In fact, we establish an equivalence to a *special case* of the all-edge triangle problem.

*This work is part of the project CONJEXITY that has received funding from the European Research Council (ERC) under the European Union’s Horizon Europe research and innovation programme (grant agreement No. 101078482). Supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.

†This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

‡Supported by the project CONJEXITY as above.

§Research partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 462679611.

1 Introduction

Matrix multiplication is one of the most fundamental and important computational problems. Countless papers are devoted to it (e.g., the surveys [Blä13, GJC⁺23]), including celebrated algorithms that bound the famous exponent $2 \leq \omega < 2.3719$ of its time complexity on worst-case $n \times n$ matrices [Str69, Pan78, BCRL79, Pan80, Sch81, Rom82, CW82, Str86, CW90, CU03, CKSU05, Sto10, Wil12, CU13, AW21, DWZ23] as well as some lower bounds [Blä99, Raz03, Shp03, Lan14, LM18]. Like any other problem, in most applications (theoretical or practical) the input matrices of interest could be *sparse* (or could be sparsified without much harm) in the sense that the number of nonzero entries is $m_{\text{in}} = o(n^2)$. Consequently, the following question has been repeatedly raised by researchers in various domains, seeking a quantitative understanding of the gains from sparsity:

What is the time complexity of matrix multiplication if the matrices are sparse?

It is natural to expect a fine-grained answer in the form of a bound that depends on m_{in} and ω . However, the answer may not be satisfying if we do not consider the additional parameter m_{out} – the number of nonzeros in the *output* matrix.¹ This is because even two very sparse matrices with $m_{\text{in}} = O(n)$ could, in pathological cases, produce a very dense output matrix with $m_{\text{out}} = \Omega(n^2)$, giving a trivial lower bound of $\Omega(m_{\text{in}}^2)$ just to write the output. But this lower bound is too simplistic: in almost all interesting cases m_{out} is much closer to m_{in} . For example, two matrices with $O(n)$ nonzeros in *random* locations can be multiplied in $O(n)$ expected time.² There is a rich landscape between these two extremes that we cannot capture if we ignore m_{out} . The dream goal is an “instance optimal” algorithm that achieves the best possible time complexity for any input matrices, based on their specific properties. Towards that goal, we are seeking a bound that includes *both* m_{in} and m_{out} .

Considerable effort has gone towards this question, from multiple communities, leading to a state of affairs with several incomparable and complicated bounds. Some of the bounds are discussed below and summarized in Table 1. In this paper, we clean up the picture by giving (1) a new algorithm for sparse matrix multiplication, (2) an upper bound on its complexity for any setting of m_{in} vs. m_{out} , and (3) evidence that the achieved bound is tight no matter what the complexity of *dense* (rectangular) matrix multiplication turns out to be.

1.1 Previous Work

We briefly review relevant results for sparse matrix multiplication. Let A, B be $n \times n$ matrices, let m_{in} denote the number of nonzeros in A and B , and m_{out} the number of nonzeros in $A \cdot B$.

The goal of early works was to achieve bounds for *input-sparse* matrix multiplication that get as close as possible to the $O(n^2)$ bound, under minimal assumptions on m_{in} and ω . The starting point is a folklore approach, first described in [Gus78], of computing $C[i, j] = \sum_{k: A[i, k] \neq 0} A[i, k] \cdot B[k, j]$ over all i, j in time $O(m_{\text{in}} \cdot n)$. An analysis of this approach for uniformly distributed input is given in [Sch82]. Using fast rectangular matrix multiplication, a seminal algorithm due to Yuster and Zwick [YZ05] computes $A \cdot B$ faster than dense matrix multiplication whenever $m_{\text{in}} = O(n^{\frac{\omega+1}{2}-\epsilon})$. Their arguments have been adapted to rectangular input matrices in [KSV06]. Based on recent

¹We assume that the matrices are represented as lists of nonzero entries. Logarithmic factors will not matter in our bounds.

²For any (i, k) such that $A[i, k] = 1$, in expectation there are only $O(1)$ relevant entries (k, j) such that $B[k, j] = 1$.

assumptions from fine-grained complexity, it can be shown that the Yuster-Zwick algorithm is best-possible; see the discussion in Appendix A.

To obtain further improvements, many subsequent works also exploit sparsity of the output, i.e., take m_{out} into account, and thus could hope to achieve subquadratic running times. Randomized algorithms are given in [Lin11, Pag13, VGWWZ15, JS15, Roc18] and deterministic algorithms in [AP09, Kut13, GLL⁺17, Kün18, DHK20]. We give a summary in Table 1. Even if $m_{\text{in}}, m_{\text{out}} = \Theta(n)$, only three of the above algorithms can truly beat quadratic running time $n^{2 \pm o(1)}$ (achieved by dense matrix multiplication if $\omega = 2$): the Amossen-Pagh bound [AP09]³, and the bound achieved by van Gucht et al. [VGWWZ15] and Roche [Roc18]⁴. Since our main interest is in subquadratic running times, we will only compare our results to these two bounds.

Some works study additional parameters such as the distribution of nonzeros over the rows and columns [IS09, Roc18]. The closely related setting of error correction for matrix products has been studied in [GLL⁺17, Roc18].⁵ The communication complexity of output-sensitive matrix multiplication has been studied in [WY14]. Finally, output-sensitive quantum algorithms have been given, e.g., in [BS06, WW18, Gal12].

Combinatorial Algorithms. There has been interest in “combinatorial” matrix multiplication algorithms [ADKF70, BW12, WW18, Cha15, Yu18, DKS18] that do not exploit algebraic ideas, partly because these algorithms can be more efficient in practice. So far, such methods are only able to save polylogarithmic factors over the naive $O(n^3)$ time complexity in the dense case. In the sparse case, the $\tilde{O}(m_{\text{out}} + \sqrt{m_{\text{out}}m_{\text{in}}})$ bound of [VGWWZ15] is combinatorial, and it cannot be improved by more than $n^{o(1)}$ factors without breaking through the cubic bound in the dense case.⁶ The goal of this paper is to quantify how well non-combinatorial methods perform in the sparse case.

Rectangular Matrix Multiplication and the $\omega(a, b, c)$ Notation. Our algorithms reduce sparse matrix multiplication to dense (but smaller) *rectangular* matrix multiplication. The running times thus depend on the optimal exponent $\omega(a, b, c)$ of multiplying *dense* $n^a \times n^b$ by $n^b \times n^c$ matrices, for certain values of $a, b, c \geq 0$ depending on the setting. The square case of $\omega = \omega(1, 1, 1)$ implies certain “naive” bounds on $\omega(a, b, c)$ in a black-box way by partitioning rectangles into squares. Much better bounds, however, can be obtained in a non-black-box application of the techniques. The most important constant related to rectangular matrix multiplication is $\alpha \leq 1$ defined as the largest constant such that $\omega(1, \alpha, 1) = 2$. Note that $\alpha = 1$ if and only if $\omega = 2$. It has been known for a while that $\alpha > 0$ [Cop82, Cop97, HP98, Gal12], and the current bound is $\alpha \geq 0.3138$ [GU18]. Other interesting bounds on $\omega(a, b, c)$ where a, c are not 1 have also been

³The conference version of this paper claimed a better time bound of $O(m_{\text{in}}^{2/3} m_{\text{out}}^{2/3} + m_{\text{in}}^{0.862} m_{\text{out}}^{0.408})$ without the restriction $m_{\text{in}} \leq m_{\text{out}}$. On the authors’ website this was corrected to a time bound of $O(m_{\text{in}}^{2/3} m_{\text{out}}^{2/3} + m_{\text{in}}^{0.862} m_{\text{out}}^{0.546})$ with the restriction $m_{\text{in}} \leq m_{\text{out}}$. (This time bound is the same as what we wrote in Table 1 apart from using updated values for α, ω .) In [DHK20], the analysis of the Amossen-Pagh algorithm was extended to the case $m_{\text{out}} \leq m_{\text{in}}$ achieving a running time of $m_{\text{in}} \cdot (m_{\text{out}})^{\frac{2\omega}{\omega+1} - 1 + o(1)} = O(m_{\text{in}} \cdot m_{\text{out}}^{0.407})$.

⁴Roche gives a more refined bound, involving an additional parameter r . If we only assume that $m_{\text{in}} \geq n$ and use the worst-case choice for r , we obtain the stated bound.

⁵Over rings, this turns out to be essentially equivalent to sparse matrix multiplication, see [Kün18] and Section 1.5.

⁶The *Combinatorial Boolean matrix multiplication* conjecture implies that multiplying an $n^a \times n^b$ by an $n^b \times n^a$ matrix requires time $n^{2a+b-o(1)}$. Since for this problem the input size is trivially bounded by n^{a+b} and the output size is bounded by n^{2a} , a combinatorial algorithm for sparse matrix multiplication in time $O((\sqrt{m_{\text{out}}m_{\text{in}}})^{1-\epsilon})$ would imply a combinatorial algorithm for the aforementioned problem in time $O(n^{2a+b-\epsilon'})$; a contradiction.

Table 1. Overview over sparse matrix multiplication algorithms. We list time bounds t with the understanding that there are algorithms with running time $O(t^{1+\epsilon})$ for arbitrarily small $\epsilon > 0$. For the deterministic output-sensitive algorithms in [Kut13, K un18], we assume that a close upper bound on m_{out} is given. We highlight in *italic* the only algorithms that run in time $O(n^{2-\epsilon})$ in the natural setting $m_{\text{in}}, m_{\text{out}} = \Theta(n)$. To get the numerical bounds, we plug in the current bounds of $\omega \leq 2.3719$ [DWZ23] and $\alpha \geq 0.3138$ [GU18]. Related results are given by [IS09, WY14, GLL⁺17, BS06, WW18, Gal12].

Source	Running Time	Notes
Dense matrix multiplication	n^ω	
Gustavson [Gus78] (folklore)	$n \cdot m_{\text{in}}$	
Yuster and Zwick [YZ05]	$m_{\text{in}}^{\frac{2(\omega-2)}{\omega-1-\alpha}} n^{\frac{2-\alpha\omega}{\omega-1-\alpha}} + n^2$ $\leq m_{\text{in}}^{0.703} n^{1.187} + n^2$	only relevant if $\omega > 2$
<i>Amossen and Pagh</i> ³ [AP09, DHK20]	$m_{\text{in}}^{2/3} m_{\text{out}}^{2/3} + m_{\text{in}}^{\frac{(2-\alpha)\omega-2}{(1+\omega)(1-\alpha)}} m_{\text{out}}^{\frac{2-\alpha\omega}{(1+\omega)(1-\alpha)}}$ $\leq m_{\text{in}}^{2/3} m_{\text{out}}^{2/3} + m_{\text{in}}^{0.865} m_{\text{out}}^{0.543}$	only if $m_{\text{in}} \leq m_{\text{out}}$, Boolean
Lingas [Lin11]	$n^2 m_{\text{out}}^{\omega/2-1}$ $\leq n^2 m_{\text{out}}^{0.186}$	Boolean, randomized
Pagh [Pag13]	$m_{\text{in}} + n \cdot m_{\text{out}}$	real-valued, randomized
Kutzkov [Kut13]	$n^2 + n \cdot m_{\text{out}}^2$	real-valued
Jacob and St�ockel [JS15]	$m_{\text{in}} + n^2 \left(\frac{m_{\text{out}}}{n}\right)^{\omega-2}$ $\leq m_{\text{in}} + n^2 \left(\frac{m_{\text{out}}}{n}\right)^{0.3719}$	field-valued, randomized
<i>van Gucht, Williams, Woodruff,</i> <i>Zhang</i> [VGWWZ15]	$m_{\text{out}} + \sqrt{m_{\text{out}} m_{\text{in}}}$	Boolean, randomized
K�unnemann [K�un18]	$\sqrt{m_{\text{out}}} n^2 + m_{\text{out}}^2$	integer-valued
<i>Roche</i> ⁴ [Roc18]	$m_{\text{out}} + \sqrt{m_{\text{out}} m_{\text{in}}}$	field-valued, randomized

obtained [Gal12, GU18]. In a sense, the goal of this work is to settle the sparse setting up to further progress on the rectangular setting, thus reducing the number of research topics by one.

1.2 Our Results: The Fully Sparse Setting

In Section 1.3 we present our general bound for any $m_{\text{out}} = m_{\text{in}}^r$ in the entire feasible range $r \in [0, 2]$, and evidence that it is optimal. To highlight our contributions as clearly as possible, we focus in this section on the complexity in terms of the single parameter $m := m_{\text{in}} + m_{\text{out}} \approx \max(m_{\text{in}}, m_{\text{out}})$ that represents the size of the input plus the output. We call this the *fully sparse* setting because m bounds the sparsity of both input and output. A high-degree/low-degree algorithm achieves time $O(m^{3/2})$ and the trivial lower bound is $\Omega(m)$; we would like to know the precise exponent. Note that this is a special case of our general result where $r = 1$ because in the worst-case $m_{\text{in}}, m_{\text{out}} = \Theta(m)$.

The fully sparse setting is not only elegant from a theoretical point of view (because there is only one parameter) but it also received special attention due to natural applications for joins in databases and for transitive closure computation in graphs. See Section 1.5 for more details. Let us also remark that for the extensively-studied *sparse convolution* problem [CH02, AR15, CL15,

Nak20, BFN21, BFN22], which is a one-dimensional analogue of our problem, the fully sparse setting is the predominantly studied sparse setting.

Boolean Matrix Multiplication: Beating the Triangle Bound. Previous work on sparse matrix multiplication has beaten the $m^{3/2}$ bound in the important special case of Boolean matrices over $\{0, 1\}$; the state of the art (before our paper) was time $m^{\frac{2\omega}{\omega+1}+\epsilon} \leq O(m^{1.4071})$, which becomes $m^{4/3+\epsilon}$ if $\omega = 2$ [AP09]. This is a natural barrier because it coincides with the longstanding upper bound for detecting a triangle in a graph on m edges [AYZ97].

Triangle detection and (Boolean) matrix multiplication are intimately connected by fine-grained reductions [WW18] in the dense case. Also in the sparse case, the triangle detection problem can be described as a natural *subset* matrix multiplication problem [GKLP16]: Given two matrices A, B with m nonzeros, and given a subset $S \subseteq [n] \times [n]$ of m entries, determine if any of the entries in $(AB)[i, j]$ with $(i, j) \in S$ are nonzero.

A priori, it is not clear if fully sparse matrix multiplication should be easier or harder than subset matrix multiplication. On the one hand, it feels harder because we not only need to compute m entries in AB , we also don't know their locations. But on the other hand, we are promised that AB only has m nonzeros whereas in the subset problem AB could be dense.

Our first main result is an algorithm for fully sparse matrix multiplication that beats the triangle bound, improving the complexity from $O(m^{1.4071})$ to $O(m^{1.3459})$. To our knowledge, this is the first instance of a natural setting where matrix multiplication is *faster than triangle detection*. To state the precise running time, we introduce another constant related to rectangular matrix multiplication: Let μ be the (unique) solution to the equation $\omega(\mu, 1, 1) = 2\mu + 1$. The currently best bounds on μ are $\frac{1}{2} \leq \mu \leq 0.5286$ [GU18], and if $\omega = 2$ then $\mu = \frac{1}{2}$. This constant naturally appears in several algorithms, including various settings of the All-Pairs Shortest Paths (APSP) problem [Zwi99, Zwi02, AY07, CWX21]—most notably, Zwick's algorithm for directed, unweighted APSP in time $O(n^{2+\mu+\epsilon})$ —as well as algorithms for dynamic transitive closure [DI05, San04].

Theorem 1.1 (Fully Sparse Boolean Matrix Multiplication). *Sparse Boolean matrix multiplication is in deterministic time $O(m^{1+\frac{\mu}{1+\mu}+\epsilon}) \leq O(m^{1.3459})$, for all $\epsilon > 0$.*

Our bound is strictly better than the triangle bound as long as $\omega > 2$; otherwise, both bounds become $m^{4/3+\epsilon}$ which is likely to be the right complexity “at the end of days”. Interestingly, our algorithm does not need $\omega = 2$ (or equivalently, $\alpha = 1$ where α is the aforementioned rectangular matrix multiplication exponent) to achieve exponent $4/3$ but already achieves it under the milder condition that $\mu = \frac{1}{2}$ (or equivalently, $\alpha \geq \frac{1}{2}$). In particular, while there are strong barrier results for all currently known approaches to square matrix multiplication (such as the popular *laser method* and its generalizations, applied to the Coppersmith-Winograd tensor) that rule out proving $\omega < 2.3$ [AFG15, BCC⁺16, BCC⁺17, AW18a, AW18b, Alm21, CVZ21], the best-known barriers for rectangular matrix multiplication only rule out proving $\alpha > 0.625$ [AW18a, CGLZ20]. Thus, it is conceivable that further progress on fast rectangular matrix multiplication, using the only current set of techniques, leads to sparse Boolean matrix multiplication in time $m^{4/3}$ via our Theorem 1.1.

Equivalence. Can our algorithm be improved? That is, can we either (1) get closer to $m^{4/3}$ with current $\omega(\cdot, \cdot, \cdot)$ or (2) break the $m^{4/3}$ barrier in the future, e.g. if $\omega = 2$? Our next result shows

that both (1) and (2) are *equivalent* to making progress on a special case of the *all-edge triangle* problem.

Multiple fine-grained complexity conjectures are about triangles in graphs. Underlying all of them is the fundamental statement that “*the only two algorithms for triangle detection are either two-path enumeration or fast matrix multiplication*”. The specific formalization of each hypothesis depends on the family of graphs and the bound achieved when combining these two algorithms. For example, the *Strong Triangle* conjecture [AW14] states that $m^{\frac{2\omega}{\omega+1}-o(1)}$ time is required on m edge graphs, and the *Unbalanced Triangle* hypothesis [KW20] states that in a graph on three unbalanced parts with $m_1 \leq m_2 \leq m_3$ edges between them the time complexity is $(m_1 m_2)^{1/3} m_3^{2/3-o(1)}$ (even if $\omega = 2$). Hypothesis 1.9 in Section 1.3 considers a bound in terms of nodes and edges in some natural regime.⁷

In the *all-edge* version, defined next, we must answer for each edge whether it is in a triangle. This version is appealing because (unlike mere detection) we can often prove its hardness based on other famous conjectures. In particular, reductions from 3-SUM [Pă10, KPP16] and APSP [WX20] establish an $m^{4/3-o(1)}$ lower bound on m edge graphs and serve as a stepping stone for many other reductions [ABKZ22, ABF23, JX23].⁸ In this paper, we need an *unbalanced* all-edge triangle version.

Definition 1.2 (AE-TRIANGLE). *The AE-TRIANGLE(x, y, z, m) problem is to decide, in a given tripartite graph $G = (X, Y, Z, E)$ with $|X| \leq x$, $|Y| \leq y$, $|Z| \leq z$ and $|E| \leq m$, for each edge $(i, j) \in (X \times Z) \cap E$ whether it is part of a triangle in G .*

It is known that the 3-SUM and APSP conjectures imply a matching $n^{2-o(1)}$ lower bound for AE-TRIANGLE($n, n, n, n^{3/2}$) [Pă10, KPP16, WX20]. If we restrict the size of one of the three parts in the tripartite graph to \sqrt{n} we get the AE-TRIANGLE($\sqrt{n}, n, n, n^{3/2}$) problem. Our next theorem shows that (when $\mu = 1/2$) subquadratic time for this problem is possible if and only if the $m^{4/3}$ bound for fully-sparse Boolean matrix multiplication can be broken. It is interesting to note that in the AE-TRIANGLE($\sqrt{n}, n, n, n^{3/2}$) setting only one of the edge-parts is sparse while the other two are dense.

Theorem 1.3 (Equivalence with AE-TRIANGLE). *The following two statements are equivalent in terms of deterministic and randomized algorithms:*

- (1) *There is some $\epsilon > 0$ such that sparse Boolean matrix multiplication is in time $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$.*
- (2) *There is some $\epsilon' > 0$ such that AE-TRIANGLE($n^\mu, n, n, n^{1+\mu}$) is in time $O(n^{1+2\mu-\epsilon'})$.*

Viewed as a hardness result, Theorem 1.3 gives a tight lower bound for sparse matrix multiplication under a natural hypothesis (discussed below) about AE-TRIANGLE. In the other direction, it shows that to break the $m^{4/3}$ barrier for all sparse settings, it is enough to break it in a very specific case where one input matrix is rectangular and dense, one input matrix is square and sparse, and the output matrix is rectangular and dense.

⁷It is possible to unify all of them under a general and formal hypothesis that address all possible choices of the six parameters for the numbers of nodes and edges between the three parts, but the bound becomes too cumbersome to state.

⁸The 3-SUM conjecture states that no $O(n^{2-\epsilon})$ time algorithm can decide, given a set of n integers, if there are three that sum to zero. The APSP conjecture states that no $O(n^{3-\epsilon})$ time algorithm can compute all pair-wise distances in an edge-weighted graph on n nodes.

Integer Matrix Multiplication (Or over any Ring). In the matrix multiplication literature, algorithms for the Boolean case tend to extend seamlessly to handle entries in \mathbf{Z} (or any ring, with arithmetic operations computable in constant time). In the sparse setting, however, prior work [AP09] that beat $m^{3/2}$ faces a fundamental challenge when negative numbers are allowed: Many pairs of nonzero entries $A[i, k]$ and $B[k, j]$ (with the same k) can cancel out, forcing the algorithm to spend too much time without “gaining” any nonzero in the output. Notably, such massive cancellations do not only occur in pathological cases, but are actually inherent to interesting applications of sparse *integer* matrix multiplication; one example to error correction is discussed in Section 1.5.

An exciting feature of our new algorithm is that it can handle integers just as well, at the cost of introducing randomness, giving a dramatic improvement from time $\tilde{O}(m^{1.5})$ [Roc18] to time $O(m^{1.3459})$. In fact, our algorithm extends to *any* ring R . In this case our running time naturally depends on the complexity of dense matrix multiplication over that ring R ; i.e., the constants ω and μ depend on R (see the discussion in Section 2).

Theorem 1.4 (Fully Sparse Matrix Multiplication). *Let R be a ring. Sparse matrix multiplication over R is in randomized time $O(m^{1+\frac{\mu}{1+\mu}+\epsilon})$, for all $\epsilon > 0$ (assuming an oracle for arithmetic operations over R). For $R = \mathbf{Z}$, this running time becomes $O(m^{1.3459})$.*

Since the integer case is only harder than the Boolean case, the bound in Theorem 1.4 is tight unless we break the aforementioned bound of all-edge triangle. In Section 5 we strengthen the equivalence of Theorem 1.3 to show that the integer case is equivalent to a *counting* version of all-edge triangle (Theorem 5.4).

1.3 The General Bound

Recall that our motivating question was about identifying the optimal complexity for any choice of m_{in} vs. m_{out} , i.e., $m_{\text{out}} = m_{\text{in}}^r$ for any $r \in [0, 2]$.⁹ In practice, it would be ideal if there is a single algorithm that is guaranteed to achieve the optimal complexity when run on matrices with m_{in} nonzeros, no matter what m_{out} turns out to be. With a more intricate analysis and under similar assumptions, our new algorithm accomplishes this.

We suggest that the time complexity of sparse matrix multiplication is $m_{\text{in}}^{\sigma(r)}$ when $m_{\text{out}} = m_{\text{in}}^r$, for the following definition of the exponent $\sigma(r) \in [1, 2]$ that *only depends* on the optimal exponent of *dense* rectangular matrix multiplication. (It is not clear that $\sigma(r)$ is well-defined but we prove it in Section 4.3.)

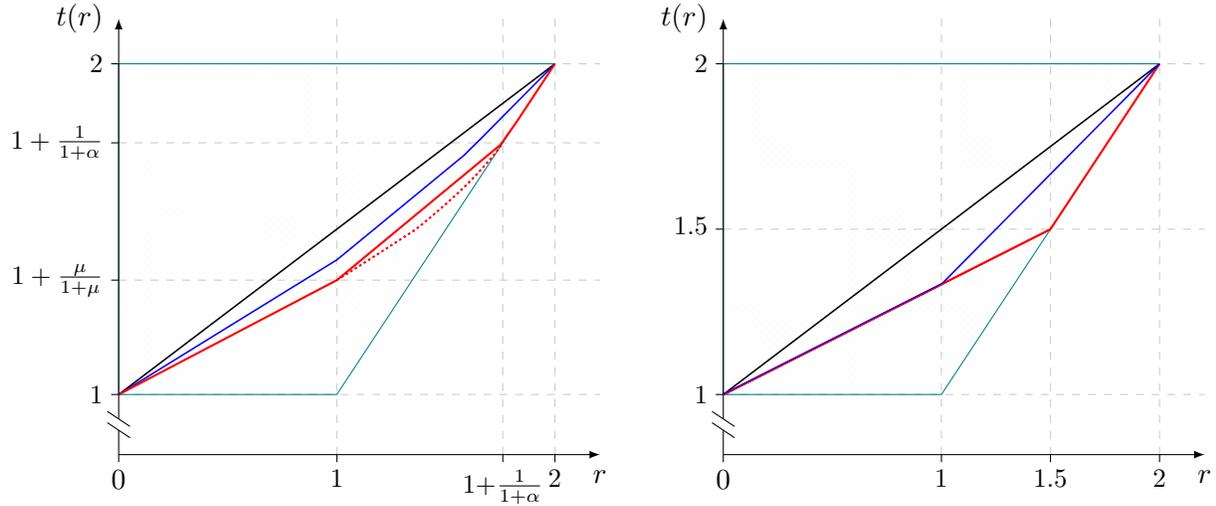
Definition 1.5 (Exponent of Sparse Matrix Multiplication). *Let $r \in [0, 2]$. We define $\sigma(r)$ as the unique solution σ to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$.*

The following are our two main theorems. They generalize Theorems 1.1 and 1.4 from $r = 1$ to any $r \in [0, 2]$; the first gives a deterministic algorithm for the Boolean case and the second gives a randomized algorithm for integers (or any ring).

Theorem 1.6 (Deterministic Sparse Boolean Matrix Multiplication). *Sparse Boolean matrix multiplication with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ is in deterministic time $O(m_{\text{in}}^{\sigma(r)+\epsilon})$, for all $\epsilon > 0, r \in [0, 2]$.*

⁹The two extreme examples that show that $0 \leq r \leq 2$ are the following. If the only nonzero entries are $A[1, k] = 1$ and $B[k, 1] = 1$ for all $k \in [n]$ then $m_{\text{in}} = n$ while $m_{\text{out}} = 1$. If the only nonzero entries are $A[i, 1] = 1$ and $B[1, j] = 1$ for all $i, j \in [n]$ then $m_{\text{in}} = n$ but $m_{\text{out}} = n^2$.

Figure 1. Plots of the exponent $t(r)$ in the running time $O(m_{\text{in}}^{t(r)+\varepsilon})$, for any $\varepsilon > 0$, of sparse Boolean matrix multiplication where $m_{\text{out}} = m_{\text{in}}^r$. The shaded teal region depicts the relevant area between the trivial $\Omega(m_{\text{in}} + m_{\text{out}})$ lower bound and $O(m_{\text{in}}^2)$ upper bound. The black lines depict the $O(m_{\text{in}}\sqrt{m_{\text{out}}})$ upper bound [VGWWZ15, Roc18]. The blue lines depict the upper bound of [AP09] as analyzed by [DHK20]. The red lines depict our bounds on $\sigma(r)$; the thick lines correspond to (1) and the dotted line was obtained via numerical computations. All lines except the blue hold for randomized integer matrix multiplication as well.



(a) The plots for the current bounds on (rectangular) matrix multiplication [GU18, DWZ23].

(b) The plots if $\omega = 2$.

Figure 1 plots our new bound against the previous results highlighted in Table 1 (in the Boolean case) both for (a) the current bounds on $\sigma(r)$ where we beat the state of the art for all r , and (b) for $\omega = 2$ in which case we get an improvement for all $r > 1$ (note that this is the more natural setting). Let us say a few words about how we bound $\sigma(r)$ (all the details are in Section 4.3). If $\omega = 2$ then it is easy to calculate $\omega(a, b, c)$ for any a, b, c and we get that $\sigma(r) = \max\{1 + \frac{r}{3}, r\}$. The situation is more complicated when using the current bounds on $\omega(a, b, c)$ [GU18, DWZ23]. First we prove that $\sigma(r)$ is a convex function of r . Then we apply either of two methods: one algebraic and the other numeric. In the first one, we evaluate $\sigma(r)$ at certain strategic values of r , and then interpolate between these points; the bound we obtain is

$$\sigma(r) \leq \max\left\{1 + r \cdot \frac{\mu}{1 + \mu}, \frac{(2 + \alpha)\mu}{1 + \mu} + r \cdot \frac{1 - \alpha\mu}{1 + \mu}, r\right\}, \quad (1)$$

and it is depicted by the thick red line in Figure 1. In the second method, we feed all known bounds on $\omega(a, b, c)$ [GU18] into a linear program that finds the best bound on $\sigma(r)$; this bound is the dotted red line in Figure 1. In any case, further improvements to our bound within the shaded area are not unconditionally impossible, but would refute the natural hypothesis discussed below.

The next theorem proves that our same bound holds for randomized algorithms in the integer case despite the cancellations; note that in this case the blue line in Figure 1 does not apply and our improvements are bigger.

Theorem 1.7 (Sparse Matrix Multiplication). *Let R be a ring. Sparse matrix multiplication over R with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ is in randomized time $O(m_{\text{in}}^{\sigma(r)+\epsilon})$, for all $\epsilon > 0, r \in [0, 2]$.*

In Section 1.5 we mention one application of Theorem 1.7 that does not follow from the results on the fully sparse setting ($r = 1$), namely to error correction. Interestingly, based on this connection, we discuss that derandomizing our result (for integers) is at least as hard as derandomizing Freivalds’ classical algorithm [Fre79].

Finally, we will isolate a concrete version of the all-edge triangle problem that must be cracked before our sparse matrix multiplication bound can be improved *even for a single r* . We call the following unbalanced version PS-AE-TRIANGLE (“partially sparse” AE-TRIANGLE) because only one edge-part is sparse while the other two are assumed to be dense (since the restriction on the number of edges only applies to one edge-part).

Definition 1.8 (PS-AE-TRIANGLE). *The PS-AE-TRIANGLE(x, y, z, m) problem is to decide, in a given tripartite graph $G = (X, Y, Z, E)$ with $|X| \leq x, |Y| \leq y, |Z| \leq z$ and $|E \cap (Y \times Z)| \leq m$, for each edge $(i, j) \in (X \times Z) \cap E$ whether it is part of a triangle in G .*

Recall from the discussion above Definition 1.2 that the fundamental underlying hypothesis is that the only two algorithms for triangle detection are either two-path enumeration or fast matrix multiplication. For any choice of parameters, it is easy to compute the bound achieved by combining these two algorithms, giving rise to the following hypothesis.

Hypothesis 1.9 (PS-AE-TRIANGLE). *For all $a, b, c \geq 0$, the PS-AE-TRIANGLE(m^a, m^b, m^c, m) problem cannot be solved in time $O(m^{\min\{1+a, \omega(a,b,c)\}-\epsilon})$, for any $\epsilon > 0$.*

The exponent in Definition 1.5 is *the* sparse matrix multiplication exponent unless Hypothesis 1.9 fails.

Theorem 1.10 (Hardness under PS-AE-TRIANGLE). *Let $r \in [0, 2]$. For any $\epsilon > 0$, sparse Boolean matrix multiplication with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ cannot be solved in time $O(m_{\text{in}}^{\sigma(r)-\epsilon})$, unless the PS-AE-TRIANGLE hypothesis fails.*

Interestingly, we do not even need this assumption for the full range of r to give evidence of optimality of our algorithm. Indeed, our algorithm runs in almost-linear time $O(m_{\text{out}}^{1+\epsilon})$ for arbitrary $\epsilon > 0$ —and thus is *unconditionally almost optimal*—when $m_{\text{out}} = \Omega(m_{\text{in}}^{1.762}) = \Omega((m_{\text{in}})^{1+\frac{1}{1\alpha}})$. If $\omega = 2$, the condition simplifies to $m_{\text{out}} \geq \Omega(m_{\text{in}}^{1.5})$.

1.4 Technical Overview

To ease the readability of this paper, we give a concise technical overview of our results. Our algorithms draw from a colorful set of techniques, ranging from basic combinatorial heavy/light decompositions to ideas from sparse recovery, derandomization, and algebra. While none of these ideas are particularly new to the context of matrix multiplication, we manage to refine and combine them in a novel way, leading to our results.

To obtain our algorithms for sparse matrix multiplication, we split the task into two major steps. These steps separately deal with the output- and input-sparsity of the problem. For simplicity we focus on the fully sparse setting ($m_{\text{in}} + m_{\text{out}} \leq m$) in this overview.

Step 1: Output Densification. In the first step, we design a reduction from fully sparse matrix multiplication to *input-sparse* matrix multiplication of $x \times y \times z$ rectangular matrices. That is, after step 1 we can assume that $xz = O(m)$, at the small cost of worsening the running time by a lower-order factor.

The basic idea is to *compress* A to a thinner matrix A' such that (1) the product $A'B$ becomes dense, and (2) we can recover AB from $A'B$. A natural approach to this task, inspired by sparse recovery, is to let $g = \frac{xz}{m}$ and to compress the matrix A by randomly grouping columns into groups of size g . Let A' be the $(x/g) \times y$ matrix obtained by adding all columns in a group. Note that $C' = A'B$ is exactly the matrix obtained from $C = AB$ by applying the same compression. Since the hashing was random, we expect that many entries (i, j) are *isolated* in the sense that $C[i, j]$ is the only nonzero entry in its group in the i -th row. These isolated entries can be efficiently recovered by some more tricks to identify j . For integer matrices, for instance, we can identify j by accessing $C[i, j]$ and $j \cdot C[i, j]$ (both of which can be computed by the previous compression) and taking their quotient.

This main idea—to compress the sparse output matrix to a dense rectangular matrix via sparse recovery techniques—was employed in several previous works [IS09, Pag13, JS15, VGWWZ15]. See also the detailed treatment in [Stö15], including implementations of this idea in the external memory model. Unfortunately, this setup is hard to generalize to arbitrary rings and hard to derandomize. To obtain the full strength of our results, we instead suggest the following twist:

1. First, assume that we know a small superset S of the support $\text{supp}(C) = \{(i, j) : C[i, j] \neq 0\}$ of size at most $|S| \leq 2m$, say. We apply the same grouping as before, and consider an entry $(i, j) \in S$ as *isolated* if there is no other entry $(i, j') \in S$ with j and j' belonging to the same group. The nice insight is that we can check for all pairs $(i, j) \in S$ whether they are isolated, and for isolated pairs recover the corresponding entries $C[i, j]$ *without the need to identify j* . (For the details see Lemma 3.2.)

With some additional effort this step can be derandomized: The key idea is that we can choose the groups *one by one* deterministically, following the method of conditional expectations. We remark that this derandomization is necessary to obtain our deterministic algorithm for Boolean matrix multiplication (Theorem 1.1).

2. Next, we remove the assumption that S is known in advance. For simplicity in this overview, we assume that the matrices A and B contain nonnegative integers. Then we can compute S by *recursively* calling our sparse matrix multiplication algorithm: Let A' be the $(x/2) \times y$ matrix obtained from A by merging and adding up pairs of adjacent rows. We compute the matrix product $C' = A'B$ recursively. Then, we select S to be all positions (i, j) that under the pairing could possibly lead to nonzero entries in C' . It is easy to check that $S \supseteq \text{supp}(C)$ and that $|S| \leq 2m$. Moreover, with each recursive call we half x and therefore the recursion incurs only a logarithmic factor to the running time. (For the details see Lemma 3.11.)

If we allow randomization, this idea indeed generalizes to integers with polylogarithmic overhead (Lemma 3.12) and to arbitrary rings with subpolynomial $2^{\tilde{O}(\sqrt{\log m})}$ overhead (Lemma 3.1).

Step 2: Input-Sparse Matrix Multiplication. Recall that after step 1, we can assume that $xz = O(m)$. It thus remains to solve an instance of input-sparse rectangular $x \times y \times z$ matrix multiplication (Lemma 4.2). In previous works, Yuster and Zwick studied the complexity

of input-sparse $x \times x \times x$ matrix multiplication [YZ05], and Kaplan, Sharir and Verbin studied input-sparse $x \times y \times x$ matrix multiplication [KSV06]. We emphasize that both of these settings do not suffice in our context, as for many cases we indeed have three distinct parameters x, y, z (this happens even in the fully sparse case, due to step 1). Nevertheless, we reuse the same simple algorithmic idea behind their algorithms, namely a heavy/light decomposition. The difficulty here does not lie in the algorithm, but in the analysis.

We may assume without loss of generality that $x \leq z$. Let Δ be a parameter to be determined later. We say that a column k in A is *light* if it contains at most Δ nonzero entries, and *heavy* otherwise. We split the matrix A into two submatrices A_{light} and A_{heavy} consisting of the light and heavy columns, respectively. We similarly split B into B_{light} and B_{heavy} where the former matrix consists exactly of all light rows k . To compute the product AB , it suffices to compute the products $A_{\text{light}}B_{\text{light}}$ and $A_{\text{heavy}}B_{\text{heavy}}$ separately.

We compute the light product $A_{\text{light}}B_{\text{light}}$ in time $O(m\Delta)$ by enumerating, for each nonzero entry $B[k, j]$, the at most Δ relevant entries from A . The heavy product $A_{\text{heavy}}B_{\text{heavy}}$, on the other hand, we compute by fast rectangular matrix multiplication. The running time of this algorithm can be bounded by

$$O\left(m\Delta + \max_{\Delta \leq x \leq m/x} \text{MM}\left(x, \frac{m}{\Delta}, \frac{m}{x}\right)\right),$$

crucially using that after step 1 we have $xz = O(m)$. In [YZ05, KSV06] the authors use coarse bounds on the complexity of rectangular matrix multiplication [HP98] to bound this expression. Our goal is an optimal answer though, so we cannot afford to be lossy in the analysis.

So what is the complexity of $\max_{\Delta \leq x \leq m/x} \text{MM}\left(x, \frac{m}{\Delta}, \frac{m}{x}\right)$? The maximum could be attained at the extreme cases $\text{MM}\left(\Delta, \frac{m}{\Delta}, \frac{m}{\Delta}\right)$ or $\text{MM}\left(\sqrt{m}, \frac{m}{\Delta}, \sqrt{m}\right)$, or anywhere in between. Unfortunately, these extreme cases seem incomparable and not reducible to each other when we treat dense matrix multiplication as a black-box algorithm (to reduce between the extreme cases, we would have to increase one dimension, but decrease another).

As it does not seem possible to us to give a combinatorial answer to this question, we peek behind the curtain of algebraic matrix multiplication. It turns out that, using light insights on the matrix multiplication tensor (Fact 2.5), we can indeed prove that the first extreme case $\text{MM}\left(\Delta, \frac{m}{\Delta}, \frac{m}{\Delta}\right)$ is dominating. Therefore, by setting $\Delta = m^{\frac{\mu}{1+\mu}}$ where μ is the solution to the equation $\omega(\mu, 1, 1) = 1 + 2\mu$, the running time becomes

$$O\left(m \cdot m^{\frac{\mu}{1+\mu}} + \text{MM}\left(m^{\frac{\mu}{1+\mu}}, m^{\frac{1}{1+\mu}}, m^{\frac{1}{1+\mu}}\right)\right) = O\left(m^{1+\frac{\mu}{1+\mu}} + m^{\frac{\omega(\mu, 1, 1)}{1+\mu}}\right) = O\left(m^{1+\frac{\mu}{1+\mu}+\epsilon}\right),$$

for any $\epsilon > 0$. We remark that this part of the proof works for any ring R , too: The light case is purely combinatorial, and in the heavy case we use fast matrix multiplication for that particular ring R .

Equivalence with All-Edges Triangle. Let us also provide some explanation why sparse Boolean matrix multiplication is equivalent to AE-TRIANGLE (Theorem 1.3). One direction is simple: Boolean matrix multiplication can be viewed as the problem of determining, for each pair of nodes $(i, j) \in X \times Z$ in a tripartite graph $G = (X, Y, Z, E)$, whether the pair is connected by a 2-path. To solve AE-TRIANGLE we thus report all pairs (i, j) that are connected by a 2-path and by an edge (which together form a triangle). For the parameters in Theorem 1.3 the computed Boolean matrix product is indeed sparse.

The other direction is more interesting; we give some intuition: $\text{AE-TRIANGLE}(\Delta, \frac{m}{\Delta}, \frac{m}{\Delta}, m)$ is essentially equivalent to the hard case in our previously outlined algorithm, when $x = \Delta$. This does *not* render our reduction straightforward though, as some other cases for $\Delta \leq x \leq m/x$ might be equally hard. To make the reduction work, we prove that $x = \Delta$ is indeed the *hardest* case, which requires very strong numeric bounds on some rectangular matrix multiplication exponents (Lemma 5.3), based on the recent work by Le Gall and Urrutia [GU18].

1.5 Some Applications

Our algorithm could be an appealing choice for anyone using matrix multiplication on sparse data. The applications of matrix multiplication are endless, and in each of them one could now claim a better bound if the input and output happen to be sparse. Let us mention only a few concrete examples from TCS.

Join-Project Queries in Relational Databases. A natural operation in relational databases are *collapsing join-project queries* of two tables (aka *composition* or *set-intersection joins*), see e.g. [AP09, VGWWZ15, DHK20]. Here, we join two tables $R(a, b)$ and $S(b, c)$ on a shared key b , followed by projection on a and c . As an example, given databases such as DBLP, such queries can be used to determine all pairs of researchers who co-authored a paper together.

Answering these queries is naturally *equivalent* to sparse Boolean matrix multiplication [AP09, VGWWZ15, DHK20]: Here, m_{in} denotes the size of the two given tables R and S , while m_{out} denotes the output size of the query. Exploiting this connection, one can answer join-project queries faster than naively evaluating a full join, followed by a projection. Our results conditionally resolve the time complexity of answering these queries.

Transitive Closure in Graphs. Consider the problem of computing the transitive closure of a directed graph G . Denoting by m the (a priori unknown) number of edges of the transitive closure of G , previous output-sensitive algorithms solve the problem in time $\tilde{O}(m^{3/2})$ [VGWWZ15, BCH15]. Using the well-known approach of computing a transitive closure via $O(\log n)$ Boolean matrix products and observing that each of the involved matrices have at most m nonzeros, we obtain the following result as an immediate corollary of Theorem 1.1.

Theorem 1.11 (Transitive Closure). *There is a deterministic algorithm computing the transitive closure of a directed graph in time $O(m^{1+\frac{\mu}{1+\mu}+\epsilon}) = O(m^{1.3459})$, for all $\epsilon > 0$, where m is the number of edges in the transitive closure.*

This running time transfers to recognition of comparability graphs, see [BCH15].

Error Correction. The fact that we obtain (randomized) algorithms even for rings can be used for error correction in matrix products, by a *deliberate* use of cancellations. In this setting, we are given matrices A, B and a possibly faulty matrix product $\tilde{C} \approx AB$. E.g., erroneous entries may have been introduced during transmission of the correctly computed result. The task is to compute the errors $AB - \tilde{C}$ in order to correct \tilde{C} to the true matrix product AB . Let m denote the number of nonzeros in the input matrices A, B, \tilde{C} and let z denote the number of nonzeros in $AB - \tilde{C}$, i.e., the number of errors. If $z \ll m$, can we correct \tilde{C} to the true matrix product faster than computing the product from scratch?

This setting has been studied explicitly e.g. in [GLL⁺17, Kün18, Roc18]. Over rings, error correction can be reduced to sparse matrix multiplication as follows: Given A, B, \tilde{C} with m nonzeros, we can construct A', B' with $O(m)$ nonzeros in time $\tilde{O}(m)$ such that $A'B' = AB - \tilde{C}$, see [Kün18, Proposition 3.1].¹⁰ Thus, correcting z errors over rings reduces to sparse matrix multiplication with $m_{\text{in}} = m$ and $m_{\text{out}} = z$. Hence, from Theorem 1.7, we obtain the following corollary.

Theorem 1.12 (Sparse Matrix Product Correction). *Consider matrices A, B, \tilde{C} with m nonzeros, over some ring R . If \tilde{C} differs from AB in at most $O(m^r)$ entries, where $r \in [0, 2]$, then we can compute AB in randomized time $O(m^{\sigma(r)+\epsilon})$, for all $\epsilon > 0$.*

Roughly speaking, whenever we have fewer errors in \tilde{C} than nonzeros in A, B, AB , we can beat computing AB from scratch, even for sparse matrices. In particular, if $\omega = 2$, we obtain a randomized algorithm correcting up to $z \geq 1$ errors in time $(z + mz^{1/3})^{1+\epsilon}$ for all $\epsilon > 0$.

The above reduction from error correction to sparse matrix multiplication also shows why derandomizing Theorem 1.7 is challenging, as it would derandomize Freivalds' algorithm *even for sparse matrices*: If we could obtain the same running time *deterministically*, we could in particular check, given A, B, \tilde{C} with m nonzeros, whether $\tilde{C} = AB$ in almost-linear time $m^{1+\epsilon}$ as follows. Simply start to run the deterministic algorithm on A', B' as constructed above (see Footnote 10). If $A'B' = 0$, i.e., $\tilde{C} = AB$, it returns an answer within some running time bound $T(m) = m^{1+\epsilon}$ guaranteed for $z = 0$ errors. If the number of steps of the algorithm ever exceeds $T(m)$ or the result matrix is different from 0, we reject, otherwise we accept. This verifies \tilde{C} in deterministic time $m^{1+\epsilon}$, for all $\epsilon > 0$.

Note that Freivalds' algorithm [Fre79] gives a randomized $\tilde{O}(m)$ -time solution, however, a derandomization is still open even for dense matrices, see, e.g., [KS93, NN93, Kün18].

1.6 Outline

The rest of this paper is structured as follows. We give some formal preliminaries and some facts on fast matrix multiplication in Section 2. In Sections 3 and 4 we respectively prove the two steps of our algorithm—the output densification and the input-sparse algorithm. In Section 5 we establish lower bounds and equivalences with the All-Edges Triangle problem. In Section 6 we then conclude with some open problems. Finally, in Appendix A we provide an interesting simple lower bound for Yuster and Zwick's input-sparse matrix multiplication algorithm.

2 Preliminaries

We denote the integers by \mathbf{Z} , the nonnegative integers by \mathbf{N} and we write $[n] = \{1, \dots, n\}$. We write $\tilde{O}(T) = T(\log T)^{O(1)}$ to suppress polylogarithmic factors. We say that an event happens *with high probability* if it happens with probability at least $1 - n^{-c}$, where n is the input size of the problem and c is an arbitrarily large constant. Throughout, by *randomized algorithm* we mean a Monte Carlo randomized algorithm that succeeds with high probability.

¹⁰We sketch the argument here: As discussed in Section 2, we can assume that A, B, \tilde{C} are $m \times m$ matrices. The desired matrices can be obtained as $A' = (A \mid -I)$ and $B' = \begin{pmatrix} B \\ \tilde{C} \end{pmatrix}$. Note that $A'B' = AB - \tilde{C}$ and that A', B' have $O(m)$ nonzeros.

Dense Matrix Multiplication. Let R be a ring. We write $\text{MM}_R(x, y, z)$ to denote the minimum number of arithmetic operations necessary to multiply an $x \times y$ matrix with an $y \times z$ matrix over R (by an arithmetic circuit or an equivalent model [Blä13]). The matrix multiplication exponent $\omega_R(a, b, c)$ is defined as

$$\omega_R(a, b, c) = \inf\{\tau \in \mathbf{R} : \text{MM}_R(\lceil n^a \rceil, \lceil n^b \rceil, \lceil n^c \rceil) = O(n^\tau)\}.$$

Most of the time the ring R is clear and we omit the subscript R .¹¹ We write $\omega = \omega(1, 1, 1)$ for the exponent of square matrix multiplication; for fields of characteristic 0 the current state-of-the-art bounds are $2 \leq \omega \leq 2.3719$ [DWZ23].

Following standard notation, we define two more constants concerned with the complexity of rectangular matrix multiplication: Let $\alpha = \max\{\alpha \in \mathbf{R} : \omega(\alpha, 1, 1) \leq 2\}$ and let μ be the (unique) solution to the equation $\omega(\mu, 1, 1) = 1 + 2\mu$. The best known bounds on α and μ are $0.31389 \leq \alpha \leq 1$ and $\frac{1}{2} \leq \mu \leq 0.5286$ [Gal12, GU18] (for fields of characteristic 0).¹² We routinely rely on the following well-known facts; for proofs, refer for instance to the survey by Bläser [Blä13] or classic work such as [LR83].

Fact 2.1. $\omega(\cdot, \cdot, \cdot)$ is a continuous function.

Fact 2.2. For any $a, b, c \geq 0$, we have $\max\{a + b, a + c, b + c\} \leq \omega(a, b, c) \leq a + b + c$.

Fact 2.3. For any $a, b, c \geq 0$, we have $\omega(a, b, c) = \omega(a, c, b) = \omega(b, a, c) = \omega(b, c, a) = \omega(c, a, b) = \omega(c, b, a)$.

Fact 2.4. For any $a, b, c, \lambda \geq 0$, we have $\omega(\lambda a, \lambda b, \lambda c) = \lambda \cdot \omega(a, b, c)$.

Fact 2.5. Let $a_i, b_i, c_i \geq 0$. Then $\omega(\sum_i a_i, \sum_i b_i, \sum_i c_i) \leq \sum_i \omega(a_i, b_i, c_i)$.

Sparse Matrix Multiplication. This paper is concerned with the *sparse matrix multiplication* problem. For parameters $x, y, z, m_{\text{in}}, m_{\text{out}} \geq 0$, the input consists of two sparsely-represented matrices $A \in \mathbf{Z}^{x \times y}$ and $B \in \mathbf{Z}^{y \times z}$ such that the total number of nonzero entries in A, B is at most m_{in} and the total number of nonzero entries in the product AB is at most m_{out} (note that we do *not* assume that m_{out} is known). The goal is to compute AB . We occasionally write $m = m_{\text{in}} + m_{\text{out}}$.

Throughout we assume that $x, y, z \leq m_{\text{in}}$ for the following reason: Whenever, say $x > m_{\text{in}}$, then there must be a row i in A which does not contain any nonzero entries. We can safely ignore this row in the multiplication as also the i -th row in AB does not contain any nonzero entries. Therefore, after possibly relabeling the rows and columns in A and B , we can assume that $x, y, z \leq m_{\text{in}}$.

Machine Model. While for dense matrix multiplication algorithms it is most natural to consider an algebraic model of computation, in these models it is unclear how to appropriately handle sparse matrices. Therefore, all of our algorithms assume the word RAM model. For integer matrices, we assume that the word size is $O(\log(xyz) + \log(\Delta))$, where Δ is an upper bound on the largest

¹¹While in principle, $\omega_R(a, b, c)$ can depend on the underlying ring, there is no indication that two rings actually have different complexities. For fields F , it is known that ω_F can only depend on the characteristic of the field F [Sch81].

¹²It is not necessarily obvious that α and μ are well-defined. For α , we remark that since ω is a continuous function (Fact 2.1) the set $\{\alpha : \omega(\alpha, 1, 1) \leq 2\}$ is closed and thus its maximum exists. For μ , we note that both sides of the equation $\omega(\mu, 1, 1) = 1 + 2\mu$ are continuous functions that must meet in some point due to the trivial bounds on ω (Fact 2.2). Using Fact 2.5, one can show that the solution is unique.

entry in the matrices A and B in absolute value. This means that we can perform basic arithmetic and logical operations on indices and entries in constant time. For matrices over general rings R , we assume special memory cells storing the ring elements and the ability to perform arithmetic operations over R by an oracle in constant time. No other information on R is needed.

We remark that we defined $\omega(a, b, c)$ in terms of arithmetic circuits, but their complexity translates to the RAM model. Strictly speaking, however, since $\omega(a, b, c)$ is an *infimum*, we only know that for all $\epsilon > 0$, there is a RAM algorithm to multiply $n^a \times n^b$ by $n^b \times n^c$ matrices in time $O(n^{\omega(a, b, c) + \epsilon})$. For convenience, many research papers that apply fast matrix multiplication in algorithm design ignore this inaccuracy, but we decided to be explicit in this paper.

3 Output Densification

In this section we prove that multiplying input- and output-sparse matrices reduces to multiplying input-sparse rectangular matrices *with dense output*. We refer to this reduction as a *densification*. Specifically, our goal is to prove the following Lemma 3.1:

Lemma 3.1 (Randomized Densification for Arbitrary Rings). *Let R be a ring. Suppose there is an algorithm \mathcal{A} for matrix multiplication over R with input size m_{in} in time $T_{\mathcal{A}}(x, y, z, m_{\text{in}})$. Then there is a randomized algorithm for matrix multiplication over R with input size m_{in} and output size m_{out} in time:*

$$2^{\tilde{O}(\sqrt{\log m_{\text{in}}})} \cdot \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq m_{\text{out}} \cdot 2^{O(\sqrt{\log m_{\text{in}}})}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}).$$

We also derive a *deterministic* densification lemma for *nonnegative* matrices (see Lemma 3.11).

Both densification lemmas are proved in two steps: (1) We design the claimed algorithm for sparse matrix multiplication assuming that we know a small superset S of the *support* $\text{supp}(AB) = \{(i, j) : (AB)[i, j] \neq 0\}$. We solve this subtask in Section 3.1 based on hashing and using ideas from sparse recovery. (2) We remove the assumption that S is given by computing S *recursively*. We provide the details of this second step in Section 3.2.

3.1 Densification via Sparse Recovery

The goal of this section is to prove the following lemma:

Lemma 3.2 (Densification via Sparse Recovery). *Let R be a ring. Suppose there is an algorithm \mathcal{A} for matrix multiplication over R with input size m_{in} in time $T_{\mathcal{A}}(x, y, z, m_{\text{in}})$. Then there is an algorithm $\text{RECOVER}(A, B, S)$ which, given matrices $A \in R^{x \times y}$, $B \in R^{y \times z}$ and a set $S \subseteq [x] \times [z]$ with $\text{supp}(AB) \subseteq S$, computes AB in time*

$$\tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right).$$

If \mathcal{A} is deterministic, then RECOVER is also deterministic.

Algorithm 1 Computes the matrix product C of two sparse matrices $A \in R^{x \times y}$, $B \in R^{y \times z}$ over some ring R , given a small superset $S \subseteq [x] \times [z]$ of the support $\text{supp}(AB)$; see Lemma 3.2. The only difference between the randomized algorithm and its derandomization is in Line 6—in the simple randomized version we use Line 6a and in the deterministic version we use Line 6b.

```

1: procedure RECOVER( $A, B, S$ )
2:   Initialize  $C \in R^{x \times z}$  to be all-zeros
3:   for  $\ell \leftarrow 0, 1, \dots, \lceil \log z \rceil$  do
4:     Let  $A_\ell \in R^{x_\ell \times y}$  denote the restriction of  $A$  to indices  $i$  with  $2^\ell \leq \deg(i) < 2^{\ell+1}$ 
5:     Let  $z_\ell \leftarrow 2^{\ell+2}$ 
6a:    Let  $\mathcal{H}_\ell$  be a set of  $10 \lceil \log m_{\text{in}} \rceil$  random hash functions  $h : [z] \rightarrow [z_\ell]$ 
6b:    Let  $\mathcal{H}_\ell$  be the set of hash functions  $h : [z] \rightarrow [z_\ell]$  computed by Lemma 3.7
7:    for each  $h \in \mathcal{H}_\ell$  do
8:      Let  $B' \in R^{y \times z_\ell}$  denote the matrix defined
          
$$B'[k, b] = \sum_{\substack{j \in [z] \\ h(j)=b}} B[k, j]$$

9:      Compute  $C' \leftarrow A_\ell B'$  with the algorithm  $\mathcal{A}$ 
10:     for each isolated support element  $(i, j) \in S$  (see Definition 3.3) do
11:        $C'[i, j] \leftarrow C'[i, h(j)]$ 
12:   return  $C$ 

```

Consider the pseudocode in Algorithm 1 (and ignore Line 6b for now). Here, for any $i \in [x]$, we define the *degree* $\deg(i) = |\{j : (i, j) \in S\}|$. The algorithm proceeds in *levels* $\ell \leftarrow 0, 1, \dots, \lceil \log z \rceil$ (Line 3). At the ℓ -th level our goal is to recover all entries $C[i, j]$ where i has degree roughly 2^ℓ . More precisely, let A_ℓ denote the submatrix of A restricted to indices $i \in [x]$ with $2^\ell \leq \deg(i) < 2^{\ell+1}$ and let x_ℓ denote the number of such indices (i.e., the number of rows in A_ℓ). Our goal at the ℓ -th level is to recover the submatrix $A_\ell B$ of C .

Simply computing $A_\ell B$ is too expensive (as the product of the outer dimensions $x_\ell \cdot z$ can be much larger than m). Instead, we will hash the matrix B to a substantially thinner matrix $B' \in \mathbf{Z}^{y \times z_\ell}$ by a random hash function $h : [z] \rightarrow [z_\ell]$ where $z_\ell = 2^{\ell+2}$ (in fact, we repeat this step for several random hash functions in Line 6a and Line 7). We define

$$B'[k, b] = \sum_{\substack{j \in [z] \\ h(j)=b}} B[k, j],$$

and compute the product $C' \leftarrow A_\ell B'$ by means of the fast algorithm \mathcal{A} . To make use of C' , we need the following definition:

Definition 3.3 (Isolation). *Let h be a hash function. We say that a pair $(i, j) \in S$ is isolated under h if there is no other $j' \neq j$ with $(i, j') \in S$ and $h(j) = h(j')$. For a set of hash functions \mathcal{H} , we say that $(i, j) \in S$ is isolated under \mathcal{H} if (i, j) is isolated under some $h \in \mathcal{H}$.*

The algorithm computes the set of isolated pairs (i, j) and for each such pair updates $C[i, j] \leftarrow C'[i, h(j)]$. We prove the correctness of this approach in the next lemma:

Lemma 3.4 (Correctness of Algorithm 1). *Assume that for all levels ℓ , and for all pairs $(i, j) \in S$ with $2^\ell \leq \deg(i) < 2^{\ell+1}$, (i, j) is isolated under \mathcal{H}_ℓ . Then Algorithm 1 correctly returns $C = AB$.*

Proof. We prove that the algorithm correctly returns $C[i, j] = (AB)[i, j]$ for all pairs $(i, j) \in [x] \times [z]$. If $(i, j) \notin S$, then we never touch $C[i, j]$ after the initialization to zero. Since $S \supseteq \text{supp}(C)$, the algorithm correctly keeps $C[i, j] = 0$. For the rest of the proof we therefore focus on pairs $(i, j) \in S$, for which we prove the statement in two steps:

- *Isolated pairs are correctly recovered:* Focus on some level ℓ and some iteration of the inner loop in Line 7 and suppose that (i, j) satisfies $2^\ell \leq \deg(i) < 2^{\ell+1}$. We show that if (i, j) is isolated under h , then we correctly recover $C[i, j] \leftarrow C'[i, h(j)]$ in Line 11. The proof is a simple calculation. Recall that by the degree assumption, i is part of the restricted matrix A_ℓ . Therefore:

$$\begin{aligned} C'[i, h(j)] &= \sum_{k \in [y]} A_\ell[i, k] \cdot B'[k, h(j)] \\ &= \sum_{k \in [y]} A_\ell[i, k] \cdot \sum_{\substack{j' \in [z] \\ h(j) = h(j')}} B[k, j'] \\ &= \sum_{\substack{j' \in [z] \\ h(j) = h(j')}} (A_\ell B)[i, j']. \end{aligned}$$

Since $(i, j) \in S$ is isolated, any other pair $(i, j') \in \text{supp}(AB) \subseteq S$ must satisfy that $h(j) \neq h(j')$. Therefore, the unique term in the sum is $(A_\ell B)[i, j]$.

- *All pairs are correctly recovered.* We use the assumption of the lemma statement: For any $(i, j) \in S$ with $2^\ell \leq \deg(i) < 2^{\ell+1}$ there is some hash function $h \in \mathcal{H}_\ell$ under which (i, j) is isolated. By the previous bullet, we correctly recover $C[i, j]$ in this iteration. In all iterations where (i, j) is not isolated, we do not change $C[i, j]$. \square

To complete the correctness proof, we need to justify the assumption that all pairs $(i, j) \in S$ are isolated. We devote the following two Sections 3.1.1 and 3.1.2 to this task, and for now focus on the running time.

Lemma 3.5 (Running Time of Algorithm 1). *Algorithm 1 runs in time*

$$\tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right).$$

Proof. We can precompute the degrees in time $O(|S|)$ and then compute the matrices $A_0, \dots, A_{\lfloor \log z \rfloor}$ in time $O(m_{\text{in}})$. We again analyze a fixed level ℓ ; the total number of levels is $\log z \leq \log m_{\text{in}}$ and can therefore be neglected. As we will see in Sections 3.1.1 and 3.1.2, we can select \mathcal{H} in time $\tilde{O}(m_{\text{in}})$. Moreover, we store all hash functions explicitly and thus evaluating any hash function takes constant time. In each of the $O(\log m_{\text{in}})$ iterations, we construct the matrix B' in time $O(m_{\text{in}})$ by a single pass over the nonzero entries of B . To compute the matrix product $A_\ell \cdot B'$, we use the algorithm \mathcal{A} . Note that at most a $2^{-\ell}$ -fraction of $[x]$ has degree at least 2^ℓ , and therefore

$x_\ell \leq \frac{|S|}{2^\ell}$. It follows that $x_\ell \cdot z_\ell \leq \frac{|S|}{2^\ell} \cdot 2^{\ell+2} = 4|S|$ and thus the running time of \mathcal{A} can be bounded by

$$T_{\mathcal{A}}(x_\ell, y, z_\ell, m_{\text{in}}) \leq \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}).$$

In time $O(|S|)$ we can moreover filter the isolated elements in S , and run the recovery loop in Line 7. In total, the running time becomes

$$\tilde{O} \left(|S| + \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right) = \tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right);$$

here, the term $|S|$ is dominated since in the worst-case \mathcal{A} must return an output of size up to $|S|$. \square

3.1.1 Randomized Isolation

We quickly show that it is easy to satisfy the required isolation property if \mathcal{H} is a set of logarithmically many random hash functions (as sampled in Line 6a).

Lemma 3.6 (Randomized Isolation). *Fix any level ℓ and let \mathcal{H}_ℓ be a set of $10 \lceil \log m_{\text{in}} \rceil$ random hash functions $h : [z] \rightarrow [z_\ell]$ (as in Line 6a). With high probability, all pairs $(i, j) \in S$ with $\deg(i) < 2^{\ell+1}$ are isolated under some hash function $h \in \mathcal{H}$.*

Proof. We first focus on a single pair $(i, j) \in S$ and a single random hash function $h : [z] \rightarrow [z_\ell]$, and argue that (i, j) is isolated under h with probability at least $\frac{1}{2}$. Let $J = \{j' : (i, j') \in S\} \setminus \{j\}$ and note that $|J| \leq \deg(i) < 2^{\ell+1}$. The error event is that there is some $j' \in J$ with $h(j) = h(j')$. For any fixed j, j' this happens with probability at most $\frac{1}{z_\ell} = \frac{1}{2^{\ell+2}}$. Taking a union bound over the at most $2^{\ell+1}$ elements in J , we obtain that (i, j) is not isolated with probability at most $\frac{2^{\ell+1}}{2^{\ell+2}} = \frac{1}{2}$.

Since we pick the $10 \lceil \log m_{\text{in}} \rceil$ hash functions in \mathcal{H}_ℓ independently, it follows that (i, j) is isolated under \mathcal{H}_ℓ with probability at least $1 - 2^{-10 \log m_{\text{in}}} = 1 - m_{\text{in}}^{-10}$. By a union bound over the at most $xz \leq m_{\text{in}}^2$ many elements in S , we get that all pairs in S are isolated with probability at least $1 - m_{\text{in}}^{-8}$. (The constant 8 can be chosen arbitrarily larger.) \square

3.1.2 Deterministic Isolation

In this section we derandomize the previous algorithm. Note that the only place where we use randomness is to guarantee that each pair (i, j) is isolated (at the appropriate level). We will therefore replace the sets \mathcal{H}_ℓ of random hash functions by hash functions constructed by the following deterministic algorithm:

Lemma 3.7 (Deterministic Isolation). *Let $S \subseteq [x] \times [z]$ be such that for all $i \in [x]$, $|\{j : (i, j) \in S\}| \leq s$ (i.e., each row in S has at most s entries). We can compute a set of hash functions $\mathcal{H} \subseteq \{h : [z] \rightarrow [2s]\}$ with the following properties:*

1. All pairs $(i, j) \in S$ are isolated under \mathcal{H} .
(That is, there is some $h \in \mathcal{H}$ such that for all $j' \neq j$ with $(i, j') \in S$, we have $h(j) \neq h(j')$).
2. $|\mathcal{H}| \leq \log |S| + 1$.

Algorithm 2 A deterministic algorithm to find a small set of hash functions \mathcal{H} isolating the given set $S \subseteq [x] \times [z]$; see Lemma 3.7.

```

1: Let  $\mathcal{H} \leftarrow \emptyset$ 
2: while  $S_{\mathcal{H}} \neq \emptyset$  do
3:   Let  $h$  be a function returning  $\perp$  for every input
4:   for  $j \leftarrow 1, \dots, z$  do
5:     Find a bucket  $b^*$  with  $|C_{\mathcal{H}}(h[j \mapsto b^*])| \leq \mathbf{E}_{b \in [2s]} |C_{\mathcal{H}}(h[j \mapsto b])|$ 
6:      $h(j) \leftarrow b^*$ 
7:      $\mathcal{H} \leftarrow \mathcal{H} \cup \{h\}$ 
8: return  $\mathcal{H}$ 

```

3. The algorithm to compute \mathcal{H} is deterministic and runs in time $\tilde{O}(z + |S|)$.

It is easy to verify that by replacing Line 6a with Line 6b in Algorithm 1 (and using Lemma 3.7 in place of Lemma 3.6 in the analysis), we preserve the correctness of the algorithm. For this reason, in this section we focus on proving Lemma 3.7. At the end of this section we further remark how the running time of Algorithm 1 is affected.

An Inefficient Algorithm. Lemma 3.7 is proven using the method of conditional expectations. We pick the set of hash functions \mathcal{H} one by one, and for each hash function h we assign the values $h(1), \dots, h(z)$ one by one. The key idea is that in each step we select the current value $h(i)$ to be *at least as good* as what we expect from a random choice. To precisely state what we mean by *good*, we start with some terminology.

For a set of hash functions \mathcal{H} , let $S_{\mathcal{H}} \subseteq S$ denote the subset of pairs (i, j) that are *not* isolated under \mathcal{H} ; we will refer to these pairs as *active*. In this language our goal is to find a set of hash functions \mathcal{H} with zero active pairs. To this end, we start with $\mathcal{H} \leftarrow \emptyset$ and proceed in several *rounds*. In each round, our goal is to select a hash function that halves the number of active pairs.

Next, we describe how to select such a hash function h . Let $C_{\mathcal{H}}$ denote the set of all triples (i, j, j') with $j \neq j'$, $(i, j) \in S_{\mathcal{H}}$ and $(i, j') \in S$. For a hash function $h : [z] \rightarrow [2s] \cup \{\perp\}$, let $C_{\mathcal{H}}(h) \subseteq C_{\mathcal{H}}$ denote the subset of triples (i, j, j') with $h(j) = h(j') \neq \perp$. We refer to the elements of $C_{\mathcal{H}}(h)$ as the *collisions under h* . Intuitively, a collision (i, j, j') is a witness that (i, j) is not isolated under h . Our approach is to select a next hash function causing at most $\frac{1}{2}|S_{\mathcal{H}}|$ collisions. More precisely, we start from a hash function initialized to some dummy value $h(1) = \dots = h(z) = \perp$. Then, we enumerate $j \leftarrow 1, \dots, z$ and assign $h(j) \leftarrow b^*$ for some bucket b^* that causes at most as many new collisions as a random bucket would cause in expectation, i.e.,

$$|C_{\mathcal{H}}(h[j \mapsto b^*])| \leq \mathbf{E}_{b \in [2s]} |C_{\mathcal{H}}(h[j \mapsto b])|.$$

Here, we write $h[j \mapsto b]$ to denote the updated hash function that maps j to b . We summarize this algorithm in Algorithm 2, ignoring for now how to find such a bucket b^* (in Line 5). Instead, we prove that this algorithm indeed leads to the claimed properties 1 and 2.

Lemma 3.8 (Correctness of Algorithm 2). *Algorithm 2 returns a set of hash functions \mathcal{H} that satisfies properties 1 and 2 of Lemma 3.7:*

1. All pairs $(i, j) \in S$ are isolated under \mathcal{H} .
(That is, there is some $h \in \mathcal{H}$ such that for all $j' \neq j$ with $(i, j') \in S$, we have $h(j) \neq h(j')$).

2. $|\mathcal{H}| \leq \log |S| + 1$.

Proof. We prove that the number of active elements $|S_{\mathcal{H}}|$ halves after each round. From this statement both properties follow immediately: Since initially $|S_0| = |S|$, this process stops after at most $\log |S| + 1$ rounds which proves property 2. Moreover, as soon as the algorithm terminates we have that $S_{\mathcal{H}} = \emptyset$ and therefore all pairs in S are isolated under \mathcal{H} , by definition.

Focus on any round; we prove that Algorithm 2 selects a hash function $h : [z] \rightarrow [2s]$ which causes at most $\frac{1}{2}|S_{\mathcal{H}}|$ collisions, i.e., $|C_{\mathcal{H}}(h)| \leq \frac{1}{2}|S_{\mathcal{H}}|$. As any non-isolated active element leads to at least one collision, this entails that indeed the number of active elements halves after this round, i.e., $|S_{\mathcal{H} \cup \{h\}}| \leq \frac{1}{2}|S_{\mathcal{H}}|$.

Consider any triple $(i, j, j') \in C_{\mathcal{H}}$; when do we decide whether (i, j, j') becomes a collision? We claim that this decision happens in exactly the $\ell = \max\{j, j'\}$ -th iteration of the inner loop. Before that iteration we have not yet fixed the hash value $h(\max\{j, j'\})$ and after that iteration we will never change the hash values $h(j)$ and $h(j')$. Let $C_{\mathcal{H}}^{(\ell)}$ denote the subset of triples (i, j, j') with $\max\{j, j'\} = \ell$, and let $C_{\mathcal{H}}^{(\ell)}(h) = C_{\mathcal{H}}^{(\ell)} \cap C_{\mathcal{H}}(h)$. With this insight in mind, focus on the j -th iteration of the inner loop. For any bucket b , we have that

$$|C_{\mathcal{H}}(h[j \mapsto b])| = |C_{\mathcal{H}}(h)| + |C_{\mathcal{H}}^{(j)}(h[j \mapsto b])|.$$

Therefore, we select a bucket b^* with

$$\begin{aligned} |C_{\mathcal{H}}(h)| + |C_{\mathcal{H}}^{(j)}(h[j \mapsto b^*])| &\leq \mathbf{E}_{b \in [2s]} \left(|C_{\mathcal{H}}(h)| + |C_{\mathcal{H}}^{(j)}(h[j \mapsto b])| \right) \\ &= |C_{\mathcal{H}}(h)| + \mathbf{E}_{b \in [2s]} \left(|C_{\mathcal{H}}^{(j)}(h[j \mapsto b])| \right) \\ &= |C_{\mathcal{H}}(h)| + \frac{|C_{\mathcal{H}}^{(j)}|}{2s}. \end{aligned}$$

In the last step we have used that for any triple $(i, j, j') \in C_{\mathcal{H}}^{(\ell)}$, there is exactly one choice for $h(\ell)$ which turns this triple into a collision (namely, $h(\ell) \leftarrow h(\min\{j, j'\})$). This calculation shows that the number of *new* collisions in the j -th round is at most $\frac{1}{2s}|C_{\mathcal{H}}^{(j)}|$. Thus, the total number of collisions for the hash function h selected in the current round is

$$|C_{\mathcal{H}}(h)| \leq \sum_{j=1}^z \frac{|C_{\mathcal{H}}^{(j)}|}{2s} = \frac{|C_{\mathcal{H}}|}{2s} \leq \frac{|S_{\mathcal{H}}| \cdot s}{2s} = \frac{|S_{\mathcal{H}}|}{2}.$$

This is precisely what we set out to prove. □

How To Efficiently Select Good Buckets. In the previous lemma we have completely analyzed Algorithm 2—except that we left open *how* to find the good buckets b^* in Line 5. It is possible to simply test all options $b \leftarrow 1, \dots, 2s$, but this is too inefficient for our purposes. We now describe a faster approach leading to the claimed running time. The idea is to find b^* via binary search. We maintain the following data:

- A matrix $M \in \mathbf{N}^{x \times 2s}$ where $M[i, b] = |\{(i, j) \in S : h(j) = b\}|$.
(The entry $M[i, b]$ counts for how many pairs (i, j) we have hashed j to the bucket b .)

- A matrix $M_{\mathcal{H}} \in \mathbf{N}^{x \times 2s}$ where $M_{\mathcal{H}}[i, b] = |\{(i, j) \in S_{\mathcal{H}} : h(j) = b\}|$.
(The entry $M_{\mathcal{H}}[i, b]$ counts for how many *active* pairs (i, j) we have hashed j to the bucket b .)
- Moreover, we maintain segment trees on the rows of the matrices M and $M_{\mathcal{H}}$ such that, given $i \in [x]$ and an interval $B \subseteq [2s]$, we can compute $\sum_{b \in B} M[i, b]$ and $\sum_{b \in B} M_{\mathcal{H}}[i, b]$ in polylogarithmic time.

This data is easy to maintain: As the hash function is initialized to \perp , we initially set both matrices to be all-zeros. Whenever we assign $h(j) \leftarrow b^*$ we increment all entries $M[i, b]$ where $(i, j) \in S$ and all entries $M_{\mathcal{H}}[i, b]$ where $(i, j) \in S_{\mathcal{H}}$. We update the segment tree accordingly. The implementation of the binary search hinges on the following lemma:

Lemma 3.9. *Focus on any round of Algorithm 2. In the j -th iteration of the inner loop, we have that*

$$|C_{\mathcal{H}}(h[j \mapsto b])| = |C_{\mathcal{H}}(h)| + \sum_{\substack{i \in [x] \\ (i, j) \in S_{\mathcal{H}}}} M[i, b] + \sum_{\substack{i \in [x] \\ (i, j) \in S}} M_{\mathcal{H}}[i, b].$$

Proof. Focus on any round of Algorithm 2. In the ℓ -th iteration of the inner loop we have already assigned $h(1), \dots, h(\ell - 1) \in [2s]$, but the values $h(\ell), \dots, h(z)$ are still set to \perp . The proof is by the following simple yet tedious calculation:

$$\begin{aligned} |C_{\mathcal{H}}(h[\ell \mapsto b])| &= |C_{\mathcal{H}}(h)| + |C_{\mathcal{H}}^{(\ell)}(h[\ell \mapsto b])| \\ &= |C_{\mathcal{H}}(h)| + |\{(i, j, j') \in C_{\mathcal{H}}^{(\ell)} : h(\min\{j, j'\}) = b\}| \\ &= |C_{\mathcal{H}}(h)| + |\{(i, j, j') \in C_{\mathcal{H}} : j' < j = \ell, h(j') = b\}| \\ &\quad + |\{(i, j, j') \in C_{\mathcal{H}} : j < j' = \ell, h(j) = b\}| \\ &= |C_{\mathcal{H}}(h)| + \sum_{\substack{i \in [x] \\ (i, \ell) \in S_{\mathcal{H}}}} |\{(i, j') \in S : h(j') = b\}| + \sum_{\substack{i \in [x] \\ (i, \ell) \in S}} |\{(i, j) \in S_{\mathcal{H}} : h(j) = b\}| \\ &= |C_{\mathcal{H}}(h)| + \sum_{\substack{i \in [x] \\ (i, \ell) \in S_{\mathcal{H}}}} M[i, b] + \sum_{\substack{i \in [x] \\ (i, \ell) \in S}} M_{\mathcal{H}}[i, b]. \quad \square \end{aligned}$$

Lemma 3.10 (Running Time of Algorithm 2). *There is an implementation of Line 5 in Algorithm 2 such that the total running time becomes $\tilde{O}(z + |S|)$.*

Proof. Focus on any round of Algorithm 2, and the j -th iteration of the inner loop. We find a bucket b^* satisfying $|C_{\mathcal{H}}(h[j \mapsto b^*])| \leq \mathbf{E}_{b \in [2s]} |C_{\mathcal{H}}(h[j \mapsto b])|$ by binary search as follows. We maintain a search interval B and the invariant

$$\mathbf{E}_{b \in B} |C_{\mathcal{H}}(h[j \mapsto b])| \leq \mathbf{E}_{b \in [2s]} |C_{\mathcal{H}}(h[j \mapsto b])|.$$

Initially, $B = [2s]$ and thus the invariant holds trivially. In each step, we split $B = B_1 \cup B_2$ into roughly equal-sized parts and recur on the half k with the smaller value $\mathbf{E}_{b \in B_k} |C_{\mathcal{H}}(h[j \mapsto b])|$. It is

clear that this half maintains the invariant. To compute these values, we use that for any interval we can express

$$\begin{aligned} \mathbf{E}_{b \in B} |C_{\mathcal{H}}(h[j \mapsto b])| &= \frac{1}{|B|} \sum_{b \in B} |C_{\mathcal{H}}(h[j \mapsto b])| \\ &= |C_{\mathcal{H}}(h)| + \frac{1}{|B|} \sum_{\substack{i \in [x] \\ (i,j) \in S_{\mathcal{H}}}} \sum_{b \in B} M[i, b] + \frac{1}{|B|} \sum_{\substack{i \in [x] \\ (i,j) \in S}} \sum_{b \in B} M_{\mathcal{H}}[i, b] \end{aligned}$$

by Lemma 3.9. The first term in the sum is the same for both halves B_1 and B_2 and can be ignored. The second and third terms can be computed in time $|\{i : (i, j) \in S\}| \cdot \text{polylog}(s)$, using that the inner sums can be evaluated in polylogarithmic time by the segment trees.

Let us analyze the total running time. The algorithm runs for $O(\log |S|)$ rounds. In each round, we run z iterations, where the j -th iteration takes time $O(1 + |\{i : (i, j) \in S\}| \cdot \text{polylog}(s))$. Therefore, across all iterations the running time is bounded by $O(z + |S| \text{polylog}(s)) = \tilde{O}(z + |S|)$. The total number of updates to the matrices M and $M_{\mathcal{H}}$ is $|S|$ each, and therefore the overhead due to maintaining M , $M_{\mathcal{H}}$ and the segment tree becomes $\tilde{O}(|S|)$, too. \square

We remark that the application of Lemma 3.7 in the recovery algorithm (Algorithm 1) incurs a running time overhead of $\sum_{\ell} \tilde{O}(z + x_{\ell} z_{\ell}) = \sum_{\ell} \tilde{O}(z + \frac{|S|}{2^{\ell}} \cdot 2^{\ell}) = \tilde{O}(m_{\text{in}} + |S|)$. This does not increase the running time asymptotically.

3.2 Approximating the Support

We are ready to complete the proofs of the densification lemmas. By the previous section it suffices to compute a small over-approximation of the support $\text{supp}(AB)$, and the main goal of this section is to prove that this approximation can be computed recursively.

Warm-Up: Nonnegative Integer Matrices. We start with the following easy deterministic densification for nonnegative integer matrices (in particular, for Boolean matrices). The advantage of nonnegative matrices is that we can conveniently avoid cancellations.

Lemma 3.11 (Deterministic Densification for Nonnegative Matrices). *Suppose there is a deterministic algorithm \mathcal{A} for nonnegative integer matrix multiplication with input size m_{in} in time $T_{\mathcal{A}}(x, y, z, m_{\text{in}})$. Then there is a deterministic algorithm for nonnegative integer matrix multiplication with input size m_{in} and output size m_{out} in time:*

$$\tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 8m_{\text{out}}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right).$$

Proof. Throughout, let $A \in \mathbf{N}^{x \times y}$ and $B \in \mathbf{N}^{y \times z}$ denote the given matrices. Our goal is to compute their product $C = AB$ by an algorithm $\text{MULTIPLYNONNEGATIVE}(A, B)$; see the pseudocode in Algorithm 3. If $x \leq 1$, then the problem deforms to a vector-matrix multiplication that is trivially solvable in linear time $O(m_{\text{in}})$. So assume that $x > 1$. By Lemma 3.2 it suffices to compute a set $S \subseteq [x] \times [z]$ satisfying the following two properties:

Algorithm 3 A deterministic algorithm to multiply sparse nonnegative matrices $A \in \mathbf{N}^{x \times y}$, $B \in \mathbf{N}^{y \times z}$; see Lemma 3.11.

```

1: procedure MULTIPLYNONNEGATIVE( $A, B$ )
2:   if  $x \leq 1$  then
3:     Solve the problem naively in time  $O(m_{\text{in}})$ 
4:   else
5:     Let  $A'$  be the  $\lceil \frac{x}{2} \rceil \times y$  matrix defined by  $A'[i, k] = A[2i - 1, k] + A[2i, k]$ 
6:     Compute  $C' \leftarrow$  MULTIPLYNONNEGATIVE( $A', B$ ) recursively
7:     Compute  $S \leftarrow \{(2i - 1, j), (2i, j) : (i, j) \in \text{supp}(C')\}$ 
8:     return RECOVER( $A, B, S$ )

```

(i) $S \supseteq \text{supp}(C)$, and

(ii) $|S| \leq 2|\text{supp}(C)| \leq 2m_{\text{out}}$.

We will compute this set S by calling MULTIPLYNONNEGATIVE recursively. Specifically, let A' be the $\lceil \frac{x}{2} \rceil \times y$ matrix defined by $A'[i, k] = A[2i - 1, k] + A[2i, k]$. (In case that there is an overflow and $2i > x$, simply treat $A[2i, k]$ as zero.) We recursively compute $C' = A'B$, and let $S = \{(2i - 1, j), (2i, j) : (i, j) \in \text{supp}(C')\}$. The interesting part is to prove the two properties.

(i) By definition we have that

$$\begin{aligned}
C'[i, j] &= \sum_{k \in [y]} A'[i, k] \cdot B[k, j] \\
&= \sum_{k \in [y]} (A[2i - 1, k] + A[2i, k]) \cdot B[k, j] \\
&= C[2i - 1, j] + C[2i, j].
\end{aligned}$$

Hence, whenever $(2i - 1, j) \in \text{supp}(C)$ or $(2i, j) \in \text{supp}(C)$ then have $C'[i, j] > 0$ —here, we use that the matrices are nonnegative. In this case, it follows that $(i, j) \in \text{supp}(C')$ and that $(2i - 1, j), (2i, j) \in S$.

(ii) From the same calculation one can see that $(i, j) \in \text{supp}(C')$ only if at least one of the pairs $(2i - 1, j)$ and $(2i, j)$ is part of $\text{supp}(C)$. It follows that $|S| = 2|\text{supp}(C')| \leq 2|\text{supp}(C)| \leq 2m_{\text{out}}$, which proves property (ii).

It remains to analyze the running time. The running time is dominated by the call to RECOVER with running time bounded by

$$\tilde{O} \left(|S| + \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right) = \tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 8m_{\text{out}}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right),$$

where the term $|S| = O(m_{\text{out}})$ is dominated by the running time of \mathcal{A} . MULTIPLYNONNEGATIVE reaches a recursion depth of at most $O(\log x) = O(\log m_{\text{in}})$ which only incurs a logarithmic factor. \square

Integer Matrices. This approach is promising to extend to integers and arbitrary rings, but there is a difficulty: Suppose that in the product matrix $C = AB$ there are two entries $C[2i - 1, j] = -C[2i, j]$. Then, if we compute A' and C' as before, these two entries *cancel* leading to $C'[i, j] = 0$. We have therefore lost the support elements $(2i - 1, j), (2i, j)$ and the set S is incorrect. While it is hard to deal with these cancellations deterministically, we show how to remedy this state of affairs by exploiting randomization.

Lemma 3.12 (Randomized Densification for Integer Matrices). *Suppose there is an algorithm \mathcal{A} for integer matrix multiplication with input size m_{in} in time $T_{\mathcal{A}}(x, y, z, m_{\text{in}})$. Then there is a randomized algorithm for integer matrix multiplication with input size m_{in} and output size m_{out} in time:*

$$\tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 8m_{\text{out}}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right).$$

Proof. This proof is almost exactly as in Lemma 3.11, except for the following difference: We define the matrix A' by $A'[i, k] = A[2i - 1, k] + r \cdot A[2i, k]$, where r is a random integer in the range $[m_{\text{in}}^{10}]$. The only error event is that there are nonzero entries $C[2i - 1, j]$ and $C[2i, j]$ such that $C[2i - 1, j] + r \cdot C[2i, j] = 0$ (in which case we incorrectly miss the support elements $(2i - 1, j)$ and $(2i, j)$ in S). For any such pair, there is at most one bad value of r and thus, this event happens with probability at most m_{in}^{-10} . Taking a union bound over the at most $xz \leq m_{\text{in}}^2$ pairs, the error event happens with small probability m_{in}^{-8} . (The constant 8 can be arbitrarily larger.)

There is a subtle consequence: With each recursive call, we increase the entries in A by a factor $\text{poly}(m_{\text{in}})$. Therefore, at the deepest level of the recursion, the numbers have increased by a factor $m_{\text{in}}^{O(\log m_{\text{in}})}$. To represent such numbers, we need $O(\log m_{\text{in}})$ memory cells per number, which worsens the time complexity by a polylogarithmic factor. \square

Matrices over Arbitrary Rings. So far, we have exploited the specific structure of the integers (namely, that any univariate linear equation has at most one integer solution, which is most likely not the randomly chosen value r). For arbitrary rings, we can neither assume any structure, nor that we have access to random ring elements. Densification is still possible, using the following simple key lemma:

Lemma 3.13. *Let R be a ring and let $a_1, \dots, a_w \in R$ at least one of which is nonzero. For a uniformly random subset $I \subseteq [w]$, the probability that $\sum_{i \in I} a_i = 0$ is at most $\frac{1}{2}$.*

Proof. Without loss of generality, assume that $a_1 \neq 0$. Writing $I_1 = I \cap \{1\}$, $I_2 = I \cap \{2, \dots, w\}$, note that the random sets I_1 and I_2 are independently selected. We therefore consider I_2 fixed and only rely on the randomness of I_1 . The condition $\sum_{i \in I} a_i = 0$ can be equivalently rewritten as $a_1 |I_1| = -\sum_{i \in I_2} a_i$ where $|I_1|$ is either 0 or 1 with equal probabilities $\frac{1}{2}$. We thus distinguish two cases: If $-\sum_{i \in I_2} a_i = 0$, then we succeed in the event that I_1 is nonempty, and if $-\sum_{i \in I_2} a_i \neq 0$, then we succeed in the event that I_1 is empty. \square

This lemma leads to the following densification result for arbitrary rings:

Lemma 3.1 (Randomized Densification for Arbitrary Rings). *Let R be a ring. Suppose there is an algorithm \mathcal{A} for matrix multiplication over R with input size m_{in} in time $T_{\mathcal{A}}(x, y, z, m_{\text{in}})$. Then*

Algorithm 4 A randomized algorithm to multiply sparse matrices $A \in R^{x \times y}$, $B \in R^{y \times z}$ for general rings R ; see Lemma 3.1. In the algorithm, we use the parameter $w = 2^{\sqrt{\log x}}$.

```

1: procedure MULTIPLY( $A, B$ )
2:   if  $x \leq 1$  then
3:     Solve the problem naively in time  $O(m_{\text{in}})$ 
4:   else
5:     for  $\ell \leftarrow 1, \dots, L := 10 \lceil \log(m_{\text{in}}) \rceil$  do
6:       Sample a random subset  $I_\ell \subseteq [w]$ 
7:       Let  $A_\ell$  be the  $\lceil \frac{x}{w} \rceil \times y$  matrix defined by
           
$$A_\ell[i, k] = \sum_{i' \in I_\ell} A[(i-1)w + i', k]$$

8:        $C_\ell \leftarrow \text{MULTIPLY}(A_\ell, B)$ 
9:        $S_\ell \leftarrow \{((i-1)w + i', j) : (i, j) \in \text{supp}(C_\ell), i' \in [w]\}$ 
10:       $S \leftarrow S_1 \cup \dots \cup S_L$ 
11:   return RECOVER( $A, B, S$ )

```

there is a randomized algorithm for matrix multiplication over R with input size m_{in} and output size m_{out} in time:

$$2^{\tilde{O}(\sqrt{\log m_{\text{in}}})} \cdot \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq m_{\text{out}} \cdot 2^{O(\sqrt{\log m_{\text{in}}})}}} T_A(x', y, z', m_{\text{in}}).$$

Proof. The idea is very similar to Lemma 3.11. We design an algorithm MULTIPLY; see the pseudocode in Algorithm 4. If $x \leq 1$, then the problem is trivially solvable in time $O(m_{\text{in}})$, so suppose otherwise. Let w be a parameter to be fixed later. As before, our goal is to compute a set $S \subseteq [x] \times [z]$ satisfying the following two properties so that the matrix product can be computed by RECOVER (Lemma 3.2):

- (i) $S \supseteq \text{supp}(C)$ with high probability, and
- (ii) $|S| \leq w \cdot |\text{supp}(C)| \leq w m_{\text{out}}$.

To compute S , we repeat the following steps for $\ell \leftarrow 1, \dots, L := 10 \lceil \log(m_{\text{in}}) \rceil$: Sample a uniformly random subset $I_\ell \subseteq [w]$, and let A_ℓ be the $\lceil \frac{x}{w} \rceil \times y$ matrix defined by $A'_\ell[i, k] = \sum_{i' \in I_\ell} A[(i-1)w + i', k]$ (again we ignore overflows and assume that A is zero outside of its bounds). By calling MULTIPLY recursively, we compute the matrix product $C' = A'_\ell B$. Then we pick $S_\ell = \{((i-1)w + i', j) : (i, j) \in \text{supp}(C'), i' \in [w]\}$. Finally, take $S = S_1 \cup \dots \cup S_L$. We show that both properties (i) and (ii) are satisfied by this construction:

- (i) Consider any support element from $\text{supp}(C)$, written as $((i-1)w + i', j)$ for $i \in [x]$, $i' \in [w]$, $j \in [z]$. We prove that with high probability, $((i-1)w + i', j) \in S$. First, focus on any

repetition ℓ . By the definition of A_ℓ and C_ℓ , we have

$$\begin{aligned} C_\ell[i, j] &= \sum_{k \in [y]} A_\ell[i, k] \cdot B[k, j] \\ &= \sum_{k \in [y]} \left(\sum_{i' \in I_\ell} A[(i-1)w + i'] \right) \cdot B[k, j] \\ &= \sum_{i' \in I_\ell} C[(i-1)w + i', j]. \end{aligned}$$

Recall that $I_\ell \subseteq [w]$ is a uniformly random subset and that at least one of the terms in the sum is nonzero (namely, $C[(i-1)w + i', j]$). Therefore, by Lemma 3.13, $C_\ell[i, j]$ is nonzero with probability at most $\frac{1}{2}$. Recall that in this case we include $((i-1)w + i', j)$ in S_ℓ . It follows that $((i-1)w + i', j) \in S$ with high probability $1 - 2^{-L} \geq 1 - m_{\text{in}}^{-10}$.

- (ii) From the same calculation we conclude that $(i, j) \in \text{supp}(C_\ell)$ only if there is some $i' \in [w]$ such that $((i-1)w + i', j) \in \text{supp}(C)$. It follows that all sets S_1, \dots, S_L (and thus also S) are contained in a superset of $\text{supp}(C)$ of size at most $w \cdot |\text{supp}(C)| = wm_{\text{out}}$.

This completes the correctness proof, but it remains to analyze the running time. Ignoring the cost of the recursive calls, the running time of MULTIPLY is dominated by calling RECOVER. Each such call runs in time

$$\tilde{O} \left(|S| + \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4|S|}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right) = \tilde{O} \left(\max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4wm_{\text{out}}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \right).$$

However, this time the recursive calls are considerably more costly than in Lemma 3.11: The recursion reaches depth $O(\log_w(x)) = O(\log_w(m_{\text{in}}))$ and each node in the recursion tree branches with degree $L = O(\log m_{\text{in}})$. It follows that the number of nodes in the recursion tree is bounded by $(\log m_{\text{in}})^{O(\log_w m_{\text{in}})}$. Thus, by setting $w = 2^{\sqrt{\log m_{\text{in}}}}$, the total running time is

$$\begin{aligned} &(\log m_{\text{in}})^{O(\log_w m_{\text{in}})} \cdot \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq 4wm_{\text{out}}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}) \\ &= 2^{O(\sqrt{\log m_{\text{in}}} \log \log m_{\text{in}})} \cdot \max_{\substack{x' \leq x, z' \leq z \\ x' \cdot z' \leq m_{\text{out}} \cdot 2^{\sqrt{\log m_{\text{in}}} + 2}}} T_{\mathcal{A}}(x', y, z', m_{\text{in}}), \end{aligned}$$

which is as claimed. □

4 The Complete Algorithm

In this section we complete our algorithm for sparse matrix multiplication. We structure this section as follows: We first give the heavy/light algorithm for input-sparse matrix multiplication in Section 4.1, and combine this algorithm with the densification lemmas to obtain our main results

in Section 4.2. Both algorithms rely on our sparse matrix multiplication exponent $\sigma(r)$ as defined in the following definition:¹³

Definition 1.5 (Exponent of Sparse Matrix Multiplication). *Let $r \in [0, 2]$. We define $\sigma(r)$ as the unique solution σ to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$.*

In Section 4.3 we analyze this exponent $\sigma(r)$ in detail: We argue that $\sigma(r)$ is well-defined, and derive readable upper bounds on $\sigma(r)$.

4.1 Input-Sparse Matrix Multiplication

The goal of this section is to design a heavy/light algorithm for input-sparse matrix multiplication. The analysis of this algorithm (in the heavy case) hinges on the following lemma:

Lemma 4.1. *Let $a, b, c \geq 0$ with $a \leq c$. Then, for any $0 \leq \delta \leq \frac{c-a}{2}$, it holds that $\omega(a + \delta, b, c - \delta) \leq \omega(a, b, c)$.*

Proof. If $a = c$, then also $\delta = 0$ and the claim is trivial. So assume that $c - a > 0$ and let $\epsilon = \frac{\delta}{c-a}$. By Facts 2.3 and 2.4, we have that $\omega((1 - \epsilon)a, (1 - \epsilon)b, (1 - \epsilon)c) = (1 - \epsilon)\omega(a, b, c)$ and $\omega(\epsilon c, \epsilon b, \epsilon a) = \epsilon\omega(c, b, a) = \epsilon\omega(a, b, c)$. Thus, by Fact 2.5 it follows that

$$\begin{aligned} \omega(a + \delta, b, c - \delta) &= \omega(a + \epsilon(c - a), b, c - \epsilon(c - a)) \\ &= \omega((1 - \epsilon)a + \epsilon c, (1 - \epsilon)b + \epsilon b, (1 - \epsilon)c + \epsilon a) \\ &\leq \omega((1 - \epsilon)a, (1 - \epsilon)b, (1 - \epsilon)c) + \omega(\epsilon c, \epsilon b, \epsilon a) \\ &= (1 - \epsilon)\omega(a, b, c) + \epsilon\omega(a, b, c) \\ &= \omega(a, b, c), \end{aligned}$$

which proves the claim. □

Lemma 4.2 (Deterministic Input-Sparse Matrix Multiplication). *Let R be a ring and let $r \in [0, 2]$. Given two matrices $A \in R^{x \times y}$ and $B \in R^{y \times z}$ with input size m_{in} and $xz \leq m_{\text{in}}^r$, we can compute their product AB in deterministic time $O(m_{\text{in}}^{\sigma(r)+\epsilon})$, for any $\epsilon > 0$.*

Proof. We assume by symmetry that $x \leq z$ (otherwise we use the identity $AB = (B^T A^T)^T$ to compute AB and thereby exchange x and z). Throughout, let $\sigma = \sigma(r)$ (and recall that $\sigma \geq 1$).

We apply a heavy-light approach: We say that $k \in [y]$ is *light* if $|\{i : A[i, k] \neq 0\}| \leq m_{\text{in}}^{\sigma-1}$, and *heavy* otherwise. Let y_1 and y_2 denote the number of light and heavy indices, respectively. We subdivide A into submatrices $A_1 \in \mathbf{Z}^{x \times y_1}$ and $A_2 \in \mathbf{Z}^{x \times y_2}$, where the light indices participate in A_1 and the heavy indices participate in A_2 . Subdivide B similarly into $B_1 \in \mathbf{Z}^{y_1 \times z}$ and $B_2 \in \mathbf{Z}^{y_2 \times z}$. Then the algorithm executes the following two steps:

1. Compute $r_1 = A_1 \cdot B_1$ exploiting the sparsities of A_1 and B_1 . Specifically:
 - Initialize r_1 as an all-zero $x \times z$ matrix
 - **for** $(k, j) \in [y_1] \times [z]$ with $B_1[k, j] \neq 0$ **do**
 - **for** $i \in [x]$ with $A_1[i, k] \neq 0$ **do**

¹³Strictly speaking, $\sigma(r)$ also depends on the underlying ring. That is, for a ring R we define $\sigma_R(r)$ as the unique solution to the equation $\omega_R(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$. As for ω , we typically omit the subscript when the ring R is clear.

$$- \quad r_1[i, j] \leftarrow r_1[i, j] + A_1[i, k] \cdot B_1[k, j]$$

2. Compute $r_2 = A_2 \cdot B_2$ using fast rectangular matrix multiplication (ignoring the assumption that A_2 and B_2 are sparse).

We report $C = r_1 + r_2$ as the output.

The correctness is fairly obvious: For any subdivision of $[y]$ we have that $A \cdot B = A_1 \cdot B_1 + A_2 \cdot B_2$. It is easy to verify that step 1 correctly computes $r_1 = A_1 \cdot B_1$ and immediate that step 2 correctly computes $r_2 = A_2 \cdot B_2$.

The interesting part is to analyze the running time. We start with step 1. The outer loop runs for at most m_{in} iterations (since B contains at most m_{in} nonzero entries). And for any k , the inner loop runs for at most $m_{\text{in}}^{\sigma-1}$ iterations since A_1 corresponds to the light indices. Therefore, the running time of step 1 is bounded by $O(m_{\text{in}}^\sigma)$.

For the second step, we start with some observations: First, if there is no heavy index (i.e., $y_2 = 0$) then step 2 is trivial. We may therefore assume that there is at least one heavy index which implies that $x \geq m_{\text{in}}^{\sigma-1}$. Second, using the assumptions that $xz \leq m_{\text{in}}^r$ and $x \leq z$, there is some $0 \leq \delta \leq 1 + \frac{r}{2} - \sigma$ such that $x \leq m_{\text{in}}^{\sigma-1+\delta}$ and $z \leq m_{\text{in}}^{1+r-\sigma-\delta}$. Third, the number of heavy indices is bounded by $y_2 \leq m_{\text{in}}/m_{\text{in}}^{\sigma-1} = m_{\text{in}}^{2-\sigma}$. Recall that we compute the matrix product $r_2 = A_2 \cdot B_2$ using fast matrix multiplication. The running time of this step is thus $O(m_{\text{in}}^{\omega(\sigma-1+\delta, 2-\sigma, 1+r-\sigma-\delta)+\epsilon})$, for all $\epsilon > 0$. This exponent can be bounded using Lemma 4.1:

$$\omega(\sigma - 1 + \delta, 2 - \sigma, 1 + r - \sigma - \delta) \leq \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma.$$

In the final step we have applied the definition of $\sigma = \sigma(r)$. In summary, the second step runs in time $O(m_{\text{in}}^{\sigma+\epsilon})$, which completes the proof. \square

4.2 Sparse Matrix Multiplication

We obtain our main results by a combination of the densification technique from the previous Section 3 with the input-sparse algorithm. Specifically, the proofs of our Main Theorems 1.6 and 1.7 are a straightforward combination of Lemma 4.2 with the densification algorithms from Lemmas 3.1 and 3.11:

Theorem 1.6 (Deterministic Sparse Boolean Matrix Multiplication). *Sparse Boolean matrix multiplication with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ is in deterministic time $O(m_{\text{in}}^{\sigma(r)+\epsilon})$, for all $\epsilon > 0, r \in [0, 2]$.*

Theorem 1.7 (Sparse Matrix Multiplication). *Let R be a ring. Sparse matrix multiplication over R with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ is in randomized time $O(m_{\text{in}}^{\sigma(r)+\epsilon})$, for all $\epsilon > 0, r \in [0, 2]$.*

We will later argue that these theorems are conditionally optimal, but at this point it is far from clear how to even interpret the running time $m_{\text{in}}^{\sigma(r)+\epsilon}$. Therefore, the next section is devoted to a detailed analysis of $\sigma(r)$.

4.3 The Sparse Matrix Multiplication Exponent

The Sparse Matrix Multiplication Exponent Is Well-Defined. Recall that we define $\sigma(r)$ as the unique solution to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$. However, it is a priori not clear that a solution σ to this equation even exists. Therefore, our first goal is to prove that $\sigma(r)$ is well-defined in the sense that there indeed exists unique solution (see Lemma 4.4).

Lemma 4.3. *For any $r \in [0, 2]$, the function $\sigma - \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma)$ is strictly increasing for $\sigma \in [1, 1 + \frac{r}{2}]$.*

Proof. Let $f(\sigma) = \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma)$. To prove the claim it suffices to show that f is nonincreasing. And indeed, for any $\sigma, \sigma' \in [1, 1 + \frac{r}{2}]$ with $\sigma \leq \sigma'$ we have

$$\begin{aligned} & \omega(\sigma' - 1, 2 - \sigma', 1 + r - \sigma') \\ & \leq \omega(\sigma' - 1, 2 - \sigma, 1 + r - \sigma') \\ & = \omega(\sigma - 1 + (\sigma' - \sigma), 2 - \sigma, 1 + r - \sigma - (\sigma' - \sigma)) \\ & \leq \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma). \end{aligned}$$

Here, in the last step, we have applied Lemma 4.1 with $\delta = \sigma' - \sigma$; note that this choice satisfies the required condition $\delta \leq \frac{1+r-\sigma-(\sigma-1)}{2} = 1 + \frac{r}{2} - \sigma$. \square

Lemma 4.4. *For any $r \in [0, 2]$, there is a unique solution to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$.*

Proof. Fix $r \in [0, 2]$. We start with an observation: By the trivial bounds $a + b \leq \omega(a, b, c) \leq a + b + c$, we have the following lower and upper bounds:

$$1 = \sigma - 1 + 2 - \sigma \leq \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) \leq \sigma - 1 + 2 - \sigma + 1 + r - \sigma = 2 + r - \sigma.$$

It follows that any feasible solution to the equation $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$ must satisfy $1 \leq \sigma \leq 2 + r - \sigma$ and thus lies in the range $[1, 1 + \frac{r}{2}]$. Next, by the previous lemma the function $g(\sigma) = \sigma - \omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma)$ is strictly increasing on $[1, 1 + \frac{r}{2}]$. Moreover, since $\omega(\cdot, \cdot, \cdot)$ is known to be a continuous function (Fact 2.1), it follows that also g is continuous. Finally, using again the trivial bounds $b + c \leq \omega(a, b, c) \leq a + b + c$, we can bound g at

$$g(1) = 1 - \omega(0, 1, r) \leq 1 - (1 + r) = -r \leq 0,$$

and

$$g(1 + \frac{r}{2}) = 1 + \frac{r}{2} - \omega(\frac{r}{2}, 1 - \frac{r}{2}, \frac{r}{2}) \geq 1 + \frac{r}{2} - (\frac{r}{2} + 1 - \frac{r}{2} + \frac{r}{2}) = 0.$$

All in all, we showed that g is continuous and strictly increasing on $[1, 1 + \frac{r}{2}]$ and that $g(1) \leq 0$ and $g(1 + \frac{r}{2}) \geq 0$. It follows that there is some $\sigma \in [1, 1 + \frac{r}{2}]$ with $g(\sigma) = 0$, or equivalently, $\omega(\sigma - 1, 2 - \sigma, 1 + r - \sigma) = \sigma$. \square

Algebraic Upper Bounds for $\sigma(r)$. We have established that $\sigma(r)$ is well-defined, but it still remains hard to grasp how $\sigma(r)$ behaves as a function of r . For this reason, we will now derive some explicit upper bounds on $\sigma(r)$. Our strategy is as follows: We first prove that $\sigma(r)$ is a convex function of r . Then, we evaluate $\sigma(r)$ at certain strategic points and conclude by the convexity that the linear interpolation between these points is an upper bound on σ .

Lemma 4.5 (Convexity). *The function $\sigma(r)$ is convex.*

Proof. Fix any $r_1, r_2 \in [0, 2]$ and any $\lambda \in [0, 1]$; we show that

$$\sigma(\lambda r_1 + (1 - \lambda)r_2) \leq \lambda\sigma(r_1) + (1 - \lambda)\sigma(r_2) =: \sigma^*.$$

We start with the following calculation that uses the subadditivity—or equivalently, convexity—of ω (see Fact 2.5):

$$\begin{aligned} & \omega(\sigma^* - 1, 2 - \sigma^*, 1 + \lambda r_1 + (1 - \lambda)r_2 - \sigma^*) \\ &= \omega(\lambda(\sigma(r_1) - 1) + (1 - \lambda)(\sigma(r_2) - 1), \\ & \quad \lambda(2 - \sigma(r_1)) + (1 - \lambda)(2 - \sigma(r_2)), \\ & \quad \lambda(1 + r_1 - \sigma(r_1)) + (1 - \lambda)(1 + r_2 - \sigma(r_2))) \\ &\leq \lambda \cdot \omega(\sigma(r_1) - 1, 2 - \sigma(r_1), 1 + r_1 - \sigma(r_1)) \\ & \quad + (1 - \lambda) \cdot \omega(\sigma(r_2) - 1, 2 - \sigma(r_2), 1 + r_2 - \sigma(r_2)) \\ &= \lambda\sigma(r_1) + (1 - \lambda)\sigma(r_2) \\ &= \sigma^*. \end{aligned}$$

By Lemma 4.3 the function $g(\sigma) = \omega(\sigma - 1, 2 - \sigma, 1 + \lambda r_1 + (1 - \lambda)r_2 - \sigma)$ is strictly increasing. Since we just proved that $g(\sigma^*) \geq 0$, it follows that the unique zero $\sigma(\lambda r_1 + (1 - \lambda)r_2)$ of g satisfies $\sigma(\lambda r_1 + (1 - \lambda)r_2) \leq \sigma^*$. \square

Lemma 4.6 (Trivial Bounds). $\max\{1, r\} \leq \sigma(r) \leq 1 + \frac{r}{2}$.

Proof. Recall that $\sigma(r) = \omega(\sigma(r) - 1, 2 - \sigma(r), 1 + r - \sigma(r))$. On the one hand, from the trivial lower bound $\max\{a + b, a + c\} \leq \omega(a, b, c)$ (Fact 2.2), we obtain $\sigma(r) \geq \max\{\sigma(r) - 1 + 2 - \sigma(r), \sigma(r) - 1 + 1 + r - \sigma(r)\} = \max\{1, r\}$. On the other hand, the trivial upper bound $\omega(a, b, c) \leq a + b + c$ (Fact 2.2) entails that $\sigma(r) \leq \sigma(r) - 1 + 2 - \sigma(r) + 1 + r - \sigma(r)$ which can be rewritten as $\sigma(r) \leq 1 + \frac{r}{2}$. \square

Note that these trivial bounds already imply tight values $\sigma(0) = 1$ and $\sigma(2) = 2$. We continue to evaluate $\sigma(r)$ for more points. For the following two lemmas, recall that we define the rectangular matrix multiplication constant μ as the unique solution to $\omega(\mu, 1, 1) \leq 1 + 2\mu$, and $\alpha = \max\{\alpha : \omega(\alpha, 1, 1) = 2\}$.

Lemma 4.7. $\sigma(1) = 1 + \frac{\mu}{1 + \mu}$.

Proof. Recall that $\sigma = \sigma(1)$ is the unique solution to the equation (1) $\omega(\sigma - 1, 2 - \sigma, 2 - \sigma) = \sigma$ and that μ is the unique solution to the equation (2) $\omega(\mu, 1, 1) = 1 + 2\mu$, or equivalently (3) $\omega(\frac{\mu}{1 + \mu}, \frac{1}{1 + \mu}, \frac{1}{1 + \mu}) = \frac{1 + 2\mu}{1 + \mu}$. By substituting σ by $1 + \frac{\mu}{1 + \mu} = \frac{1 + 2\mu}{1 + \mu}$ in (1), we find that (1) and (3) are equivalent. \square

Lemma 4.8. $\sigma(1 + \frac{1}{1 + \alpha}) = 1 + \frac{1}{1 + \alpha}$.

Proof. We have to prove that $\omega(1 + \frac{1}{1 + \alpha} - 1, 2 - (1 + \frac{1}{1 + \alpha}), 2 + \frac{1}{1 + \alpha} - (1 + \frac{1}{1 + \alpha})) = \omega(\frac{1}{1 + \alpha}, \frac{\alpha}{1 + \alpha}, 1) = 1 + \frac{1}{1 + \alpha}$. On the one hand, from the trivial lower bound $a + c \leq \omega(a, b, c)$ (Fact 2.2), we get that $\omega(\frac{1}{1 + \alpha}, \frac{\alpha}{1 + \alpha}, 1) \geq 1 + \frac{1}{1 + \alpha}$. On the other hand, by Facts 2.4 and 2.5 we have that

$$\omega\left(\frac{1}{1 + \alpha}, \frac{\alpha}{1 + \alpha}, 1\right) \leq \omega\left(\frac{1}{1 + \alpha}, \frac{\alpha}{1 + \alpha}, \frac{1}{1 + \alpha}\right) + \frac{\alpha}{1 + \alpha} = \frac{2}{1 + \alpha} + \frac{1}{1 + \alpha} = 1 + \frac{1}{1 + \alpha}. \quad \square$$

Table 2. Numerical bounds on the sparse matrix multiplication exponent $\sigma(r)$, based on the dense rectangular matrix multiplication bounds by Le Gall and Urrutia [GU18].

r	$\sigma(r)$								
0.00	1.0000	0.45	1.1539	0.85	1.2921	1.25	1.4513	1.65	1.6720
0.05	1.0171	0.50	1.1710	0.90	1.3099	1.30	1.4728	1.70	1.7091
0.10	1.0342	0.55	1.1881	0.95	1.3277	1.35	1.4943	1.75	1.7505
0.15	1.0513	0.60	1.2052	1.00	1.3458	1.40	1.5199	1.80	1.8000
0.20	1.0684	0.65	1.2223	1.05	1.3665	1.45	1.5476	1.85	1.8500
0.25	1.0855	0.70	1.2396	1.10	1.3875	1.50	1.5761	1.90	1.9000
0.30	1.1026	0.75	1.2569	1.15	1.4086	1.55	1.6060	1.95	1.9500
0.35	1.1197	0.80	1.2744	1.20	1.4299	1.60	1.6378	2.00	2.0000
0.40	1.1368								

In summary, we have the identities $\sigma(0) = 1$, $\sigma(1) = 1 + \frac{\mu}{1+\mu}$, $\sigma(1 + \frac{1}{1+\alpha}) = 1 + \frac{1}{1+\alpha}$, $\sigma(2) = 2$ by Lemmas 4.6 to 4.8. By the convexity of $\sigma(r)$ it follows that $\sigma(r)$ is upper-bounded by the line segments that interpolate between these four points:

Lemma 4.9. $\sigma(r) \leq \max\{1 + r \cdot \frac{\mu}{1+\mu}, \frac{(2+\alpha)\mu}{1+\mu} + r \cdot \frac{1-\alpha\mu}{1+\mu}, r\}$, for any $r \in [0, 2]$.

Numerical Upper Bounds for $\sigma(r)$. The previous consideration lead to readable upper bounds on $\sigma(r)$, but it is actually possible to improve these upper bounds using the currently best *numerical* bounds on the complexity of rectangular matrix multiplication [GU18]. We give our results in Table 2.

To achieve these results, suppose that we know a set of bounds of the form $\{\omega(a_i, b_i, c_i) \leq \omega_i\}_{i=1}^n$. We take [GU18, Table 3] as the basis for our bounds in Table 2. Then, consider the following linear program in the variables $\sigma, \lambda_1, \dots, \lambda_n$:

$$\begin{aligned}
 & \text{minimize} && \sigma, \\
 & \text{subject to} && \sum_{i=1}^n \lambda_i \omega_i = \sigma, \\
 & && \sum_{i=1}^n \lambda_i a_i = 1 + \sigma, \\
 & && \sum_{i=1}^n \lambda_i b_i = 2 - \sigma, \\
 & && \sum_{i=1}^n \lambda_i c_i = 1 + r - \sigma, \\
 & && \sigma, \lambda_1, \dots, \lambda_n \geq 0.
 \end{aligned}$$

Using Fact 2.5, it is easy to show that any solution σ^* to this linear program yields an upper bound of the form $\sigma(r) \leq \sigma^*$.

Bounds when $\omega = 2$. Finally, suppose that $\omega = 2$. In this case, we have $\omega(a, b, c) = \max\{a + b, a + c, b + c\}$. It follows that $\sigma(r)$ is the unique solution to the equation $\max\{1, r, 3 + r - 2\sigma\} = \sigma$, and the following lemma is immediate:

Lemma 4.10. *If $\omega = 2$, then $\sigma(r) = \max\{1 + \frac{r}{3}, r\}$.*

5 Relation to All-Edges Triangle

In this section we prove conditional lower bounds for sparse (Boolean) matrix multiplication under a hypothesis (Hypothesis 1.9) about the All-Edges Triangle problem (AE-TRIANGLE, Definition 1.2). We also prove that in the *fully sparse* setting (where $m_{\text{in}} + m_{\text{out}} \leq m$), sparse (Boolean) matrix multiplication is *equivalent* to a certain parameterization of AE-TRIANGLE. We start with a recap of the relevant definitions.

Definition 1.2 (AE-TRIANGLE). *The AE-TRIANGLE(x, y, z, m) problem is to decide, in a given tripartite graph $G = (X, Y, Z, E)$ with $|X| \leq x$, $|Y| \leq y$, $|Z| \leq z$ and $|E| \leq m$, for each edge $(i, j) \in (X \times Z) \cap E$ whether it is part of a triangle in G .*

Definition 5.1 (#AE-TRIANGLE). *The #AE-TRIANGLE(x, y, z, m) problem is to count, in a given tripartite graph $G = (X, Y, Z, E)$ with $|X| \leq x$, $|Y| \leq y$, $|Z| \leq z$ and $|E| \leq m$, for each edge $(i, j) \in (X \times Z) \cap E$ how many triangles it is part of.*

Definition 1.8 (PS-AE-TRIANGLE). *The PS-AE-TRIANGLE(x, y, z, m) problem is to decide, in a given tripartite graph $G = (X, Y, Z, E)$ with $|X| \leq x$, $|Y| \leq y$, $|Z| \leq z$ and $|E \cap (Y \times Z)| \leq m$, for each edge $(i, j) \in (X \times Z) \cap E$ whether it is part of a triangle in G .*

Hypothesis 1.9 (PS-AE-TRIANGLE). *For all $a, b, c \geq 0$, the PS-AE-TRIANGLE(m^a, m^b, m^c, m) problem cannot be solved in time $O(m^{\min\{1+a, \omega(a,b,c)\}-\epsilon})$, for any $\epsilon > 0$.*

Recall that this hypothesis morally expresses that the best way to detect triangles (for all edges) in a graph is to combine two algorithms: Fast matrix multiplication, and enumerating all 2-paths in the graph. Our hardness result is that, under this hypothesis, sparse matrix multiplication has exponent $\sigma(r)$:

Theorem 1.10 (Hardness under PS-AE-TRIANGLE). *Let $r \in [0, 2]$. For any $\epsilon > 0$, sparse Boolean matrix multiplication with input size m_{in} and output size $m_{\text{out}} = m_{\text{in}}^r$ cannot be solved in time $O(m_{\text{in}}^{\sigma(r)-\epsilon})$, unless the PS-AE-TRIANGLE hypothesis fails.*

Proof. Fix $r \in [0, 2]$, let $\sigma = \sigma(r)$, and suppose that sparse Boolean matrix multiplication is in time $O(m_{\text{in}}^{\sigma-\epsilon})$. We prove that the PS-AE-TRIANGLE(m^a, m^b, m^c, m) problem for $a = \sigma - 1$, $b = 2 - \sigma$, $c = 1 + r - \sigma$ can be solved polynomially faster than $m^{\min\{1+a, \omega(a,b,c)\}}$. Let $G = (X, Y, Z, E)$ be a given instance of PS-AE-TRIANGLE(m^a, m^b, m^c, m). We solve this instance as follows: Let A be the adjacency matrix of the bipartite subgraph with vertices $X \cup Y$, and let B be the adjacency matrix of the bipartite subgraph with vertices $Y \cup Z$. We compute the matrix product AB using the efficient algorithm for sparse Boolean matrix multiplication—note that the input size is at most $xy + m = O(m)$ and the output size is at most $xz = m^r$ which yields the correct input-to-output ratio (possibly after padding). We report all edges $(i, j) \in (X \times Z) \cap E$ with $(AB)[i, j]$.

The correctness is easy: For any pair $(i, j) \in X \times Z$ we have $(AB)[i, j] = 1$ if and only if there is a 2-path in G from i to j via some node $k \in Y$. Thus, the algorithm reports exactly all pairs $(i, j) \in X \times Z$ for which there is 2-path (i, k, j) and there is an edge $(i, j) \in E$. This is the set of pairs involved in a triangle.

Next, we analyze the running time. The dominating step is the sparse Boolean matrix multiplication in time $O(m^{\sigma-\epsilon})$; afterwards, the reporting step runs in negligible time $O(yz) = O(m)$.

But recall that σ satisfies the equation $\omega(\sigma + 1, 2 - \sigma, 1 + r - \sigma) = \sigma$, and thus the running time can be written as

$$O(m^{\sigma-\epsilon}) = O(m^{\min\{\sigma, \omega(\sigma+1, 2-\sigma, 1+r-\sigma)\}-\epsilon}) = O(m^{\min\{1+a, \omega(a,b,c)\}-\epsilon}),$$

which contradicts the PS-AE-TRIANGLE hypothesis. \square

5.1 Equivalence with All-Edges Triangle in the Fully Sparse Setting

In this section we prove that in the fully sparse setting (i.e., when we measure the combined input plus output sparsity $m = m_{\text{in}} + m_{\text{out}}$), the sparse matrix multiplication problem is *equivalent* to a certain parameterization of AE-TRIANGLE. We start with the following lemmas, based on Le Gall and Urrutia's bounds on rectangular matrix multiplication:

Lemma 5.2 ([GU18, Table 3]). $\omega(1, 1.3, 1) \leq 2.6217$ and $\omega(1, 1.4, 1) \leq 2.7085$.

Lemma 5.3. For any $0 \leq \delta \leq \frac{1-\mu}{2}$, it holds that $\omega(\mu + \delta, 1, 1 - \delta) \leq 1 + 2\mu - 0.02\delta$.

Proof. Let $\gamma = 1 - \frac{\delta}{1-\mu}$ and note that $\delta \leq \frac{1}{2} \leq \gamma \leq 1 - \delta$. By the subadditivity of ω (see Fact 2.5), we have that

$$\begin{aligned} \omega(\mu + \delta, 1, 1 - \delta) &= \omega(1 - (1 - \mu)\gamma, \quad 1, \quad \mu + (1 - \mu)\gamma) \\ &\leq \omega(\left(\gamma - \delta\right)\mu, \quad \gamma - \delta, \quad \gamma - \delta) \\ &\quad + \omega(1 - \gamma - \delta, \quad 1 - \gamma - \delta, \quad (1 - \gamma - \delta)\mu) \\ &\quad + \omega((1 + \mu)\delta, \quad 2\delta, \quad (1 + \mu)\delta). \end{aligned}$$

We can bound these three terms by $(\gamma - \delta)(1 + 2\mu)$, $(1 - \gamma - \delta)(1 + 2\mu)$ and $(1 + \mu)\delta \cdot \omega(1, \frac{2}{1+\mu}, 1)$, respectively. Further, we numerically bound $\omega(1, \frac{2}{1+\mu}, 1)$ as follows. Since $\mu \geq \frac{1}{2}$, we have $\omega(1, \frac{2}{1+\mu}, 1) \leq \omega(1, \frac{4}{3}, 1)$. Next, we use that $\omega(\cdot, \cdot, \cdot)$ is convex and can thus be upper-bounded by any linear interpolation between two points. Specifically, we can bound

$$\omega(1, \frac{4}{3}, 1) \leq \frac{2}{3} \cdot \omega(1, 1.3, 1) + \frac{1}{3} \cdot \omega(1, 1.4, 1) = \frac{2}{3} \cdot 2.6217 + \frac{1}{3} \cdot 2.7085 \leq 2.6507$$

by the bounds from the previous Lemma 5.2. Therefore:

$$\begin{aligned} \omega(\mu + \delta, 1, 1 - \delta) &\leq (\gamma - \delta)(1 + 2\mu) + (1 - \gamma - \delta)(1 + 2\mu) + (1 + \mu)\delta \cdot \omega(1, \frac{2}{1+\mu}, 1) \\ &\leq 1 + 2\mu - 2\delta(1 + 2\mu) + \delta(1 + \mu) \cdot 2.6507 \\ &= 1 + 2\mu - \delta(2 + 4\mu - 2.6507 - 2.6507\mu) \\ &= 1 + 2\mu - \delta(1.3493\mu - 0.6507) \\ &\leq 1 + 2\mu - \delta(\frac{1}{2} \cdot 1.3493 - 0.6507) \\ &\leq 1 + 2\mu - 0.0240\delta, \end{aligned}$$

which completes the proof. \square

Theorem 1.3 (Equivalence with AE-TRIANGLE). *The following two statements are equivalent in terms of deterministic and randomized algorithms:*

- (1) *There is some $\epsilon > 0$ such that sparse Boolean matrix multiplication is in time $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$.*
- (2) *There is some $\epsilon' > 0$ such that AE-TRIANGLE($n^\mu, n, n, n^{1+\mu}$) is in time $O(n^{1+2\mu-\epsilon'})$.*

Proof: (1) implies (2). This part is very similar to Theorem 1.10, and due to the similarity we only sketch this part. We use sparse Boolean matrix multiplication to compute all pairs of nodes $(i, j) \in X \times Z$ connected by a 2-path, and return all such pairs that are additionally connected by an edge $(i, j) \in E$. The time complexity is dominated by the Boolean matrix multiplication with input size $n^{1+\mu}$ and output size $n^{1+\mu}$ running in time $O((n^{1+\mu})^{1+\frac{\mu}{1+\mu}-\epsilon}) = O(n^{1+2\mu-\epsilon})$.

(2) implies (1). Assume that there is an algorithm \mathcal{A} for AE-TRIANGLE($n^\mu, n, n, n^{1+\mu}$) in time $O(n^{1+2\mu-\epsilon'})$ for some $\epsilon' > 0$. We design an efficient algorithm for sparse Boolean matrix multiplication. Let $A \in \{0, 1\}^{x \times y}$ and $B \in \{0, 1\}^{y \times z}$ be a given instance. By densification (Lemma 3.11), we can assume that $xz \leq 8m$ and by symmetry we assume that $x \leq z$.

Let $\epsilon > 0$ be a parameter to be fixed later. In the same spirit as Lemma 4.2, we say that an index $k \in [y]$ is *light* if $|\{i : A[i, k] \neq 0\}| \leq m^{\frac{\mu}{1+\mu}-\epsilon}$, and *heavy* otherwise. We let y_1 and y_2 denote the number of light and heavy indices, respectively, and subdivide A into submatrices $A_1 \in \{0, 1\}^{x \times y_1}$ and $A_2 \in \{0, 1\}^{x \times y_2}$, where the light indices participate in A_1 and the heavy indices participate in A_2 . We similarly subdivide B into $B_1 \in \{0, 1\}^{y_1 \times z}$ and $B_2 \in \{0, 1\}^{y_2 \times z}$. Then we run the following two steps:

1. Compute $C_1 = A_1 \cdot B_1$ exploiting the sparsities of A_1 and B_1 exactly as in Lemma 4.2 (by enumerating all 2-paths $(i, k, j) \in [x] \times [y_1] \times [z]$ with $A_1[i, k] = B_1[k, j] = 1$).
2. To compute $C_2 = A_2 \cdot B_2$, we distinguish the following two cases:
 - 2a. If $x \geq m^{\frac{\mu}{1+\mu}+300\epsilon}$, then compute $A_2 \cdot B_2$ using fast matrix multiplication (ignoring the assumption that A_2 and B_2 are m -sparse).
 - 2b. If $x < m^{\frac{\mu}{1+\mu}+300\epsilon}$, then we compute $A_2 \cdot B_2$ with the help of algorithm \mathcal{A} . Let $G = (X, Y, Z, E)$ be a tripartite graph with vertex parts X, Y, Z of sizes $|X| = x$, $|Y| = y_2$, $|Z| = z$. Add edges in $X \times Y$ as specified by A_2 , add edges in $Y \times Z$ as specified by B_2 (i.e., view A_2 and B_2 as the bi-adjacency matrices for these respective parts), and add all edges in $X \times Z$. Run \mathcal{A} on this instance G . We let $C_2 \in \{0, 1\}^{x \times z}$ be the matrix with $C_2[i, j] = 1$ if and only if the edge $(i, j) \in X \times Z$ participated in a triangle in G .

Finally, report $C = C_1 + C_2$ as the output.

Due to their similarity to Lemma 4.2 we omit the correctness proof of steps 1 and 2a, and only analyze step 2b. By the construction of the graph G , there is a 2-path between two nodes $i \in X$ and $j \in Z$ if and only if there is some node $k \in Y$ with $A[i, k] = B[k, j] = 1$. Since the graph contains *all* edges (i, j) any such 2-path can be completed to a triangle. Therefore, G indeed contains a triangle involving (i, j) if and only if $C_2[i, j] = 1$.

Next, focus on the running time. As before, step 1 runs in time $O(m \cdot m^{\frac{\mu}{1+\mu}-\epsilon}) = O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$. To bound the running time of step 2, we may assume that $x \geq m^{\frac{\mu}{1+\mu}-\epsilon}$ (as otherwise there is no heavy index k which renders step 2 trivial) and that $y_2 \leq m/m^{\frac{\mu}{1+\mu}-\epsilon} = m^{\frac{1}{1+\mu}+\epsilon}$.

We start with step 2a: By the three assumptions that $x \geq m^{\frac{\mu}{1+\mu}+300\epsilon}$, that $xz \leq 8m$ and that $x \leq z$, there must be some constant δ with $300\epsilon \leq \delta \leq \frac{1-\mu}{2}$ such that $x \leq m^{\frac{\mu+\delta}{1+\mu}+o(1)}$ and $z \leq m^{\frac{1-\delta}{1+\mu}+o(1)}$. Therefore, it takes time $O(m^{\omega(\frac{\mu+\delta}{1+\mu}, \frac{1}{1+\mu}+\epsilon, \frac{1-\delta}{1+\mu})+\epsilon})$, say, to compute C_2 by fast matrix multiplication. Using Lemma 5.3 and Facts 2.4 and 2.5 this exponent can be bounded by

$$\begin{aligned} & \omega\left(\frac{\mu+\delta}{1+\mu}, \frac{1}{1+\mu}+\epsilon, \frac{1-\delta}{1+\mu}-\delta\right) + \epsilon \\ & \leq \frac{\omega(\mu+\delta, 1, 1-\delta)}{1+\mu} + 2\epsilon \\ & \leq \frac{1+2\mu-0.02\delta}{1+\mu} + 2\epsilon \\ & \leq \frac{1+2\mu}{1+\mu} + -0.01\delta + 2\epsilon \\ & \leq 1 + \frac{\mu}{1+\mu} - \epsilon. \end{aligned}$$

It remains to analyze the running time of step 2b. Recall that we can assume that $y_2 \leq m^{\frac{1}{1+\mu}+\epsilon}$ and that $x \geq m^{\frac{\mu}{1+\mu}-\epsilon}$ and thus $z \leq m^{\frac{1}{1+\mu}+\epsilon}$. By step 2b we further have $x < m^{\frac{\mu}{1+\mu}+300\epsilon}$. Let $n = m^{\frac{1}{1+\mu}+600\epsilon}$, then these bounds imply that vertex parts in the graph G have sizes $|X| = x \leq n^\mu$, $|Y| = y_2 \leq n$, $|Z| = z \leq n$. Moreover, the number of edges in the graph G is at most $m + xz \leq O(m) = O(n^{1+\mu})$. Therefore, the graph G is an instance of $\text{AE-TRIANGLE}(n^\mu, n, n, O(n^{1+\mu}))$ and can be solved by \mathcal{A} in time

$$O(n^{1+2\mu-\epsilon'}) = O(m^{(\frac{1}{1+\mu}+600\epsilon)(1+2\mu-\epsilon')}) = O(m^{1+\frac{\mu}{1+\mu}+1800\epsilon-\frac{1}{2}\epsilon'}).$$

(In the last step we used the trivial bounds $\frac{1}{2} \leq \mu \leq 1$.) Setting $\epsilon = \frac{\epsilon'}{3602}$, this becomes $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$ and also the total running time is $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$. \square

In fact, this equivalence between Boolean matrix multiplication and AE-TRIANGLE can be adapted to an equivalence between *integer* matrix multiplication and $\#\text{AE-TRIANGLE}$:

Theorem 5.4 (Equivalence with $\#\text{AE-TRIANGLE}$). *The following four statements are equivalent in terms of randomized algorithms:*

- (1) *There is some $\epsilon > 0$ such that sparse integer matrix multiplication (with entries bounded by $\text{poly}(m)$) is in time $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$.*
- (2) *There is some $\epsilon > 0$ such that sparse nonnegative integer matrix multiplication (with entries bounded by $\text{poly}(m)$) is in time $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$.*
- (3) *There is some $\epsilon > 0$ such that sparse integer matrix multiplication of $\{0, 1\}$ -matrices is in time $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$.*
- (4) *There is some $\epsilon' > 0$ such that $\#\text{ALLEDESTRIANGLE}(n^\mu, n, n, n^{1+\mu})$ is in time $O(n^{1+2\mu-\epsilon'})$.*

Proof: (1) implies (4). This part of the proof is again very similar to Lemma 4.2. The only difference is that since integer matrix multiplication supports to *count* the number of 2-paths between two nodes i and j , we can also count the number of triangles involving (i, j) .

(4) *implies* (3). This part of the proof is similar to Theorem 1.3. The steps 1 and 2a are already computing the integer-valued matrices C_1 and C_2 . Since we restrict the matrices to have entries $\{0, 1\}$, in step 2b we can exactly express the matrix multiplication problem as an instance of #AE-TRIANGLE. We omit further details and instead focus on the new aspects of this proof.

(3) *implies* (2). Assume that for some $\epsilon > 0$ there is an $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$ -time algorithm for sparse matrix multiplication of $\{0, 1\}$ -matrices. We give an algorithm to efficiently multiply two *nonnegative* matrices A, B with entries bounded by m^c for some constant c . Let $L = \lceil c \log(m) \rceil$, and construct the $\{0, 1\}$ -matrices A_0, \dots, A_L , where $A_\ell[i, j] = 1$ if and only if the ℓ -th bit of $A[i, j]$ is one. We similarly construct B_0, \dots, B_L . Note that

$$A = \sum_{\ell=0}^L 2^\ell A_\ell, \quad B = \sum_{\ell=0}^L 2^\ell B_\ell,$$

and thus

$$AB = \left(\sum_{\ell=0}^L 2^\ell A_\ell \right) \left(\sum_{\ell=0}^L 2^\ell B_\ell \right) = \sum_{\ell_1=0}^L \sum_{\ell_2=0}^L 2^{\ell_1+\ell_2} A_{\ell_1} B_{\ell_2}.$$

Note that the sparsity of A_{ℓ_1} , B_{ℓ_2} and $A_{\ell_1} B_{\ell_2}$ does not blow up—more precisely, $\text{supp}(A_{\ell_1}) \subseteq \text{supp}(A)$, $\text{supp}(B_{\ell_2}) \subseteq \text{supp}(B)$ and $\text{supp}(A_{\ell_1} B_{\ell_2}) \subseteq \text{supp}(AB)$ for all ℓ_1, ℓ_2 . Hence, we can compute the $L^2 = O(\log^2 m)$ matrix products $A_{\ell_1} B_{\ell_2}$ in time $\tilde{O}(m^{1+\frac{\mu}{1+\mu}-\epsilon})$, and obtain AB by the previous equation.

(2) *implies* (1). Assume that for some $\epsilon > 0$ there is an $O(m^{1+\frac{\mu}{1+\mu}-\epsilon})$ -time algorithm for sparse nonnegative matrix multiplication. We give an algorithm to efficiently multiply two *integer* matrices $A \in \mathbf{Z}^{x \times y}$ and $B \in \mathbf{Z}^{y \times z}$ (with possibly negative entries). By the randomized densification from Lemma 3.12, we may assume that $xz \leq m^{1+o(1)}$ at the cost of worsening the running time by a polylogarithmic factor.

Let Δ denote the largest entry in A and B in absolute value. We define two nonnegative matrices A_0, A_1 as follows:

$$A_0[i, j] = \begin{cases} A[i, j] + \Delta & \text{if } A[i, j] \neq 0, \\ 0 & \text{otherwise,} \end{cases} \quad A_1[i, j] = \begin{cases} \Delta & \text{if } A[i, j] \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $A = A_0 - A_1$. For similarly defined nonnegative matrices B_0, B_1 we have $B = B_0 - B_1$. It follows that

$$AB = (A_0 - A_1)(B_0 - B_1) = A_0 B_0 - A_0 B_1 - A_1 B_0 + A_1 B_1.$$

Since $\text{supp}(A_0), \text{supp}(A_1) \subseteq \text{supp}(A)$ and similarly $\text{supp}(B_0), \text{supp}(B_1) \subseteq \text{supp}(B)$, the number of nonzero entries in A_0, A_1, B_0, B_1 is bounded by m . Additionally, since all four products have dimensions $x \times z$ and we assumed that $xz \leq m^{1+o(1)}$, the output size is trivially bounded by $m^{1+o(1)}$. Therefore, we can compute the four matrix products in time $m^{1+\frac{\mu}{1+\mu}-\epsilon+o(1)} \leq O(m^{1+\frac{\mu}{1+\mu}-\epsilon/2})$, and compute AB by the previous equation. \square

6 Conclusions and Open Problems

We conclude with two important open questions.

Almost-Linear Time? If $\omega = 2$ then matrix multiplication can be solved in linear time in the *dense* case, but the $m^{4/3}$ barrier persists in the fully sparse case $m = m_{\text{in}} + m_{\text{out}}$. The non-existence of linear-time algorithms for sparse matrix multiplication has been used as hardness assumption in [BGS20]. Theorem 1.3 shows that future research can focus on a concrete special case, namely the AE-TRIANGLE($\sqrt{n}, n, n, n^{3/2}$) problem. Any new techniques either for algorithms or for reductions (from other famous problems) should be tested against it.

Derandomization. Can we solve integer matrix multiplication in time $O(m^{1+\frac{\mu}{1+\mu}+\epsilon})$ *deterministically*? A full derandomization of our $O(m_{\text{in}}^{\sigma(r)+\epsilon})$ -time algorithm for $m_{\text{out}} = \Theta(m_{\text{in}}^r)$ for all $r \in [0, 2]$ is certainly challenging, as it would imply a deterministic $O(m^{1+\epsilon})$ -time algorithm for verifying whether three given matrices A, B, C with m nonzeros satisfy $C = AB$, see Section 1.5. Such a derandomization of Freivalds’ algorithm, which even applies to sparse matrices, is perhaps too strong to hope for. The next best goal is to obtain a derandomization in the relaxed setting (pursued, e.g., in [Kut13, Kün18]) in which we are given a close estimate b that satisfies $m_{\text{out}} \leq b \leq O(m_{\text{out}})$.

References

- [ABF23] Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-SUM lower bounds for approximate distance oracles via additive combinatorics. In *55th Annual ACM Symposium on Theory of Computing (STOC 2023)*. ACM, 2023. To appear. doi:10.48550/arXiv.2211.07058. 5
- [ABKZ22] Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in P via short cycle removal: Cycle detection, distance oracles, and beyond. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM Symposium on Theory of Computing (STOC 2022)*, pages 1487–1500. ACM, 2022. doi:10.1145/3519935.3520066. 5
- [ADKF70] Vladimir L’vovich Arlazarov, Yefim A. Dinitz, M. Kronrod, and Igor Aleksandrovich Faradzhev. On economical construction of the transitive closure of an oriented graph. In *Doklady Akademii Nauk*, volume 194, pages 487–488. Russian Academy of Sciences, 1970. 2
- [AFG15] Andris Ambainis, Yuval Filmus, and François Le Gall. Fast matrix multiplication: Limitations of the coppersmith-winograd method. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 585–593. ACM, 2015. doi:10.1145/2746539.2746554. 4
- [Alm21] Josh Alman. Limits on the universal method for matrix multiplication. *Theory Comput.*, 17:1–30, 2021. URL: <https://theoryofcomputing.org/articles/v017a001/>. 4

- [AP09] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In Ronald Fagin, editor, *12th International Conference on Database Theory (ICDT 2009)*, volume 361 of *ACM International Conference Proceeding Series*, pages 121–126. ACM, 2009. doi:10.1145/1514894.1514909. 2, 3, 4, 6, 7, 11
- [AR15] Andrew Arnold and Daniel S. Roche. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In Kazuhiro Yokoyama, Steve Linton, and Daniel Robertz, editors, *40th International Symposium on Symbolic and Algebraic Computation (ISSAC 2015)*, pages 29–36. ACM, 2015. doi:10.1145/2755996.2756653. 4
- [AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53. 5
- [AW18a] Josh Alman and Virginia Vassilevska Williams. Further limitations of the known approaches for matrix multiplication. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *LIPICs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ITCS.2018.25. 4
- [AW18b] Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In Mikkel Thorup, editor, *59th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2018)*, pages 580–591. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00061. 4
- [AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32. 1
- [AY07] Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *15th Annual European Symposium on Algorithms (ESA 2007)*, volume 4698 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2007. doi:10.1007/978-3-540-75520-3_17. 4
- [AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189. 4, 46
- [BCC⁺16] Jonah Blasiak, Thomas Church, Henry Cohn, Joshua A. Grochow, and Chris Umans. On cap sets and the group-theoretic approach to matrix multiplication. *CoRR*, abs/1605.06702, 2016. URL: <http://arxiv.org/abs/1605.06702>, arXiv:1605.06702. 4

- [BCC⁺17] Jonah Blasiak, Thomas Church, Henry Cohn, Joshua A. Grochow, and Chris Umans. Which groups are amenable to proving exponent two for matrix multiplication? *CoRR*, abs/1712.02302, 2017. URL: <http://arxiv.org/abs/1712.02302>, [arXiv:1712.02302](https://arxiv.org/abs/1712.02302). 4
- [BCH15] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. In Pierluigi Crescenzi and Michele Loreti, editors, *16th Italian Conference on Theoretical Computer Science (ICTCS 2015)*, volume 322 of *Electronic Notes in Theoretical Computer Science*, pages 51–67. Elsevier, 2015. doi:10.1016/j.entcs.2016.03.005. 11
- [BCRL79] Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. $o(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Inf. Process. Lett.*, 8(5):234–235, 1979. doi:10.1016/0020-0190(79)90113-3. 1
- [BFN21] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing (STOC 2021)*, pages 1711–1724. ACM, 2021. doi:10.1145/3406325.3451090. 4
- [BFN22] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and Las Vegas algorithms for sparse nonnegative convolution. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 3069–3090. SIAM, 2022. doi:10.1137/1.9781611977073.119. 4
- [BGS20] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: A tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636. 36
- [Blä99] Markus Bläser. A $5/2n^2$ -lower bound for the rank of $n \times n$ matrix multiplication over arbitrary fields. In *40th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1999)*, pages 45–50. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814576. 1
- [Blä13] Markus Bläser. Fast matrix multiplication. *Theory Comput.*, 5:1–60, 2013. doi:10.4086/toc.gs.2013.005. 1, 13
- [BS06] Harry Buhrman and Robert Spalek. Quantum verification of matrix products. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 880–889. ACM Press, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109654>. 2, 3
- [BW12] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory Comput.*, 8(1):69–94, 2012. doi:10.4086/toc.2012.v008a004. 2
- [CGLZ20] Matthias Christandl, François Le Gall, Vladimir Lysikov, and Jeroen Zuiddam. Barriers for rectangular matrix multiplication. *CoRR*, abs/2003.03019, 2020. URL: <https://arxiv.org/abs/2003.03019>, [arXiv:2003.03019](https://arxiv.org/abs/2003.03019). 4

- [CH02] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In John H. Reif, editor, *34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 592–601. ACM, 2002. doi:[10.1145/509907.509992](https://doi.org/10.1145/509907.509992). 4
- [Cha15] Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 212–217. SIAM, 2015. doi:[10.1137/1.9781611973730.16](https://doi.org/10.1137/1.9781611973730.16). 2
- [CKSU05] Henry Cohn, Robert D. Kleinberg, Balázs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 379–388. IEEE Computer Society, 2005. doi:[10.1109/SFCS.2005.39](https://doi.org/10.1109/SFCS.2005.39). 1
- [CL15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 31–40. ACM, 2015. doi:[10.1145/2746539.2746568](https://doi.org/10.1145/2746539.2746568). 4
- [Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:[10.1137/0211037](https://doi.org/10.1137/0211037). 2
- [Cop97] Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997. doi:[10.1006/jcom.1997.0438](https://doi.org/10.1006/jcom.1997.0438). 2
- [CU03] Henry Cohn and Christopher Umans. A group-theoretic approach to fast matrix multiplication. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 438–449. IEEE Computer Society, 2003. doi:[10.1109/SFCS.2003.1238217](https://doi.org/10.1109/SFCS.2003.1238217). 1
- [CU13] Henry Cohn and Christopher Umans. Fast matrix multiplication using coherent configurations. In Sanjeev Khanna, editor, *24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1074–1087. SIAM, 2013. doi:[10.1137/1.9781611973105.77](https://doi.org/10.1137/1.9781611973105.77). 1
- [CVZ21] Matthias Christandl, Péter Vrana, and Jeroen Zuiddam. Barriers for fast matrix multiplication from irreversibility. *Theory Comput.*, 17:1–32, 2021. URL: <https://theoryofcomputing.org/articles/v017a002/>. 4
- [CW82] Don Coppersmith and Shmuel Winograd. On the asymptotic complexity of matrix multiplication. *SIAM J. Comput.*, 11(3):472–492, 1982. doi:[10.1137/0211038](https://doi.org/10.1137/0211038). 1
- [CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:[10.1016/S0747-7171\(08\)80013-2](https://doi.org/10.1016/S0747-7171(08)80013-2). 1
- [CWX21] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Algorithms, reductions and equivalences for small weight variants of all-pairs shortest paths. In

- Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPICs*, pages 47:1–47:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.47. 4
- [DHK20] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *International Conference on Management of Data (SIGMOD 2020)*, pages 1213–1223. ACM, 2020. doi:10.1145/3318464.3380607. 2, 3, 7, 11
- [DI05] Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on dags: Breaking through the n^2 barrier. *J. ACM*, 52(2):147–156, 2005. doi:10.1145/1059513.1059514. 4
- [DKS18] Debarati Das, Michal Koucký, and Michael E. Saks. Lower bounds for combinatorial algorithms for boolean matrix multiplication. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *LIPICs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.23. 2
- [DWZ23] Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2023)*. IEEE Computer Society, 2023. To appear. URL: <https://doi.org/10.48550/arXiv.2210.10173>. 1, 3, 7, 13, 44
- [Fre79] Rusins Freivalds. Fast probabilistic algorithms. In *8th International Symposium on Mathematical Foundations of Computer Science (MFCS 1979)*, pages 57–69, 1979. doi:10.1007/3-540-09526-8_5. 8, 12
- [Gal12] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 514–523. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.80. 2, 3, 13
- [GJC⁺23] Jianhua Gao, Weixing Ji, Fangli Chang, Shiyu Han, Bingxin Wei, Zeming Liu, and Yizhuo Wang. A systematic survey of general sparse matrix-matrix multiplication. *ACM Comput. Surv.*, 55(12), 2023. doi:10.1145/3571157. 1
- [GKLP16] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How hard is it to find (honest) witnesses? In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPICs*, pages 45:1–45:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.45. 4
- [GLL⁺17] Leszek Gasieniec, Christos Levcopoulos, Andrzej Lingas, Rasmus Pagh, and Takeshi Tokuyama. Efficiently correcting matrix products. *Algorithmica*, 79(2):428–443, 2017. doi:10.1007/s00453-016-0202-3. 2, 3, 12

- [GU18] Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67. 2, 3, 4, 7, 11, 13, 30, 32, 44
- [Gus78] Fred G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4(3):250–269, 1978. doi:10.1145/355791.355796. 1, 3
- [HP98] Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complex.*, 14(2):257–299, 1998. doi:10.1006/jcom.1998.0476. 2, 10
- [IS09] Mark A. Iwen and Craig V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Inf. Process. Lett.*, 109(10):468–471, 2009. doi:10.1016/j.ipl.2009.01.010. 2, 3, 9
- [JS15] Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In Nikhil Bansal and Irene Finocchi, editors, *23th Annual European Symposium on Algorithms (ESA 2015)*, volume 9294 of *Lecture Notes in Computer Science*, pages 766–778. Springer, 2015. doi:10.1007/978-3-662-48350-3_64. 2, 3, 9
- [JX23] Ce Jin and Yinzhan Xu. Removing additive structure in 3SUM-based reductions. In *55th Annual ACM Symposium on Theory of Computing (STOC 2023)*. ACM, 2023. To appear. doi:10.48550/arXiv.2211.07058. 5
- [KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.ch89. 5
- [KS93] Tracy Kimbrel and Rakesh K. Sinha. A probabilistic algorithm for verifying matrix products using $o(n^2)$ time and $\log_2 n + o(1)$ random bits. *Inf. Process. Lett.*, 45(2):107–110, 1993. doi:10.1016/0020-0190(93)90224-W. 12
- [KSV06] Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In Nina Amenta and Otfried Cheong, editors, *22nd Annual Symposium on Computational Geometry (SoCG 2006)*, pages 52–60. ACM, 2006. doi:10.1145/1137856.1137866. 1, 10
- [Kün18] Marvin Künnemann. On nondeterministic derandomization of Freivalds’ algorithm: Consequences, avenues and algorithmic progress. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPICs*, pages 56:1–56:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.56. 2, 3, 12, 36
- [Kut13] Konstantin Kutzkov. Deterministic algorithms for skewed matrix products. In Nat-acha Portier and Thomas Wilke, editors, *30th Annual Symposium on Theoretical*

- Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPICs*, pages 466–477. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.STACS.2013.466. 2, 3, 36
- [KW20] Tsvi Kopelowitz and Virginia Vassilevska Williams. Towards optimal set-disjointness and set-intersection data structures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPICs*, pages 74:1–74:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.74. 5
- [Lan14] Joseph M. Landsberg. New lower bounds for the rank of matrix multiplication. *SIAM J. Comput.*, 43(1):144–149, 2014. doi:10.1137/120880276. 1
- [Lin11] Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. *Algorithmica*, 61(1):36–50, 2011. doi:10.1007/s00453-010-9441-x. 2, 3
- [LM18] Joseph M. Landsberg and Mateusz Michałek. A lower bound for the border rank of matrix multiplication. *International Mathematics Research Notices*, 2018(15):4722–4733, 2018. 1
- [LPW20] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *LIPICs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ITCS.2020.53. 45
- [LR83] Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theor. Comput. Sci.*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3. 13
- [Nak20] Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020. doi:10.1109/TIT.2020.2989385. 4
- [NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. doi:10.1137/0222053. 12
- [Pag13] Rasmus Pagh. Compressed matrix multiplication. *ACM Trans. Comput. Theory*, 5(3):9:1–9:17, 2013. doi:10.1145/2493252.2493254. 2, 3, 9
- [Pan78] Victor Y. Pan. Strassen’s algorithm is not optimal: Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *19th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1978)*, pages 166–176. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.34. 1
- [Pan80] Victor Y. Pan. New fast algorithms for matrix operations. *SIAM J. Comput.*, 9(2):321–342, 1980. doi:10.1137/0209027. 1

- [Pă10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *42nd Annual ACM Symposium on Theory of Computing (STOC 2010)*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772. 5
- [Raz03] Ran Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369, 2003. doi:10.1137/S0097539702402147. 1
- [Roc18] Daniel S. Roche. What can (and can’t) we do with sparse polynomials? In Manuel Kauers, Alexey Ovchinnikov, and Éric Schost, editors, *43th International Symposium on Symbolic and Algebraic Computation (ISSAC 2018)*, pages 25–30. ACM, 2018. doi:10.1145/3208976.3209027. 2, 3, 6, 7, 12
- [Rom82] Francesco Romani. Some properties of disjoint sums of tensors related to matrix multiplication. *SIAM J. Comput.*, 11(2):263–267, 1982. doi:10.1137/0211020. 1
- [San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 509–517. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.25. 4
- [Sch81] Arnold Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981. doi:10.1137/0210032. 1, 13
- [Sch82] Amir Schorr. Fast algorithm for sparse matrix multiplication. *Inf. Process. Lett.*, 15(2):87–89, 1982. doi:10.1016/0020-0190(82)90114-4. 1
- [Shp03] Amir Shpilka. Lower bounds for matrix product. *SIAM J. Comput.*, 32(5):1185–1200, 2003. doi:10.1137/S0097539702405954. 1
- [Sto10] Andrew J. Stothers. *On the complexity of matrix multiplication*. PhD thesis, University of Edinburgh, 2010. 1
- [Stö15] Morten Stöckel. *Randomized Primitives for Big Data Processing*. PhD thesis, IT-Universitetet i København, Denmark, 2015. 9
- [Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969. doi:10.1007/BF02165411. 1
- [Str86] Volker Strassen. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *27th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1986)*, pages 49–54. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.52. 1
- [VGWWZ15] Dirk Van Gucht, Ryan Williams, David P. Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In Tova Milo and Diego Calvanese, editors, *34th ACM Symposium on Principles of Database Systems (PODS 2015)*, pages 199–212. ACM, 2015. doi:10.1145/2745754.2745779. 2, 3, 7, 9, 11

- [VWY06] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding the smallest H -subgraph in real weighted graphs and related problems. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *33th International Colloquium on Automata, Languages, and Programming (ICALP 2006)*, volume 4051 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2006. doi:[10.1007/11786986_24](https://doi.org/10.1007/11786986_24). 45
- [Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing (STOC 2012)*, pages 887–898. ACM, 2012. doi:[10.1145/2213977.2214056](https://doi.org/10.1145/2213977.2214056). 1
- [WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:[10.1145/3186893](https://doi.org/10.1145/3186893). 2, 3, 4
- [WX20] Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2020)*, pages 786–797. IEEE, 2020. doi:[10.1109/FOCS46700.2020.00078](https://doi.org/10.1109/FOCS46700.2020.00078). 5, 45
- [WY14] Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In Chandra Chekuri, editor, *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1867–1877. SIAM, 2014. doi:[10.1137/1.9781611973402.135](https://doi.org/10.1137/1.9781611973402.135). 2, 3
- [Yu18] Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Inf. Comput.*, 261:240–247, 2018. doi:[10.1016/j.ic.2018.02.006](https://doi.org/10.1016/j.ic.2018.02.006). 2
- [YZ05] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:[10.1145/1077464.1077466](https://doi.org/10.1145/1077464.1077466). 1, 3, 10, 44
- [Zwi99] Uri Zwick. All pairs lightest shortest paths. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *31st Annual ACM Symposium on Theory of Computing (STOC 1999)*, pages 61–69. ACM, 1999. doi:[10.1145/301250.301271](https://doi.org/10.1145/301250.301271). 4
- [Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:[10.1145/567112.567114](https://doi.org/10.1145/567112.567114). 4

A Yuster-Zwick Is Conditionally Optimal

In this short section we provide some evidence that Yuster and Zwick’s algorithm [YZ05] for input-sparse matrix multiplication is optimal. Recall that their algorithm computes the product of two sparse $n \times n$ matrices with at most m nonzeros (and without any bound on the number of nonzeros in the output matrix) in time $m^{\frac{2\omega-4}{\omega-1-\alpha}} n^{\frac{2-\alpha\omega}{\omega-1-\alpha} + o(1)}$. For the current values of $\omega \leq 2.3719$ [DWZ23] and $\alpha \geq 0.3139$ [GU18], this running time becomes $O(m^{0.704} n^{1.186})$. Note that if $\omega = 2$, the Yuster-Zwick algorithm becomes irrelevant as we can solve the problem in optimal time $n^{2 \pm o(1)}$. We will

therefore assume throughout this section that $\omega > 2$. It is easy to check that the Yuster-Zwick algorithm beats the matrix multiplication time n^ω in the regime with at most $m \ll n^{\frac{1+\omega}{2}}$ nonzeros. For exactly $m = \Theta(n^{\frac{1+\omega}{2}})$ nonzeros, their algorithm recovers the matrix multiplication running time. In this section, based on the previous work of the fine-grained complexity community, we show that improving upon Yuster-Zwick in the regime $m = \Theta(n^{\frac{1+\omega}{2}})$ would contradict some recent fine-grained assumptions. Consider the following problem:

Definition A.1 (Monochromatic All-Edges Triangle). *The Monochromatic All-Edges Triangle problem is, given an edge-colored graph, to decide for each edge whether it is part of a monochromatic triangle (i.e., a triangle in which all three edges have the same color).*

The Monochromatic All-Edges Triangle problem can be solved in time $n^{\frac{3+\omega}{2}+o(1)}$ (this is typically called an *intermediate* running time—between fast matrix multiplication and cubic-time brute-force), and it is a recent conjecture that this time is optimal (up to subpolynomial factors) [LPW20, VWY06, WX20]:

Hypothesis A.2 (Monochromatic All-Edges Triangle). *The Monochromatic All-Edges Triangle problem cannot be solved in time $O(n^{\frac{3+\omega}{2}-\epsilon})$, for any $\epsilon > 0$.*

Evidence for this hypothesis is that any improvement for Monochromatic All-Edges Triangle beyond the $n^{\frac{3+\omega}{2}}$ barrier carries over to other well-studied intermediate problems, including the (min, max)-Product, \exists Dominance Product, \exists Equality Product, and many more [WX20] (assuming that $\omega > 2$). While this hypothesis is much more recent than many other well-established conjectures in fine-grained complexity (and may therefore seem less believable), it can certainly be viewed as an important algorithmic barrier that needs to be overcome to make progress on several interesting problems.

Based on similar reductions as in [LPW20], we prove that the Monochromatic All-Edges Triangle hypothesis implies that Yuster and Zwick’s algorithm is optimal. The proof is simple, but we are not aware of any prior references.

Lemma A.3. *Assume that $\omega > 2$. Then the Boolean matrix product of two $n \times n$ matrices with at most $O(n^{\frac{1+\omega}{2}})$ nonzero entries cannot be computed in time $O(n^{\omega-\epsilon})$, for any $\epsilon > 0$, unless the Monochromatic All-Edges Triangle Hypothesis fails.*

Proof. We design a reduction from the Monochromatic All-Edges Triangle problem. We will treat all colors separately. So fix any color χ , let G_χ denote the subgraph with edges colored with χ , and let m_χ denote the number of edges in G_χ . We distinguish the following three cases, based on the frequency m_χ . Throughout, let $\delta > 0$ be a parameter to be fixed later.

- If $n^{\frac{1+\omega}{2}+\delta} \leq m_\chi$: Solve the All-Edges Triangle problem on G_χ in time $n^{\omega+o(1)}$ using fast (Boolean) matrix multiplication.
- If $n^{\frac{1+\omega}{2}-\delta} \leq m_\chi \leq n^{\frac{1+\omega}{2}+\delta}$: By adding $n' = n^{1+\delta}$ isolated dummy nodes to the graph G_χ , we obtain a larger graph G'_χ with $O(n')$ nodes and

$$m_\chi \leq n^{\frac{1+\omega}{2}+\delta} \leq n^{(1+\delta)\cdot\frac{1+\omega}{2}} \leq O((n')^{\frac{1+\omega}{2}})$$

edges. We can use the oracle for input-sparse matrix multiplication to solve the All-Edges Triangle problem on that graph in time $O((n')^{\omega-\epsilon}) \leq O(n^{\omega+\delta\omega-\epsilon}) \leq O(n^{\omega+3\delta-\epsilon})$.

- If $m_\chi \leq n^{\frac{1+\omega}{2}-\delta}$: Solve the All-Edges Triangle problem on G_χ in time $(m_\chi)^{\frac{2\omega}{\omega+1}+o(1)}$ [AYZ97].

This completes the description of the algorithm. It remains to bound the running time. Clearly there are at most $n^2/n^{\frac{1+\omega}{2}+\delta} = n^{\frac{3-\omega}{2}-\delta}$ many colors falling into the first category. Solving each such color in time $n^{\omega+o(1)}$ takes time $n^{\frac{\omega+3}{2}-\delta+o(1)}$ in total. Similarly, there can be at most $n^{\frac{3-\omega}{2}+\delta}$ colors falling into the second category. For each such color we spend time $O(n^{\omega+3\delta-\epsilon})$, and thus the total time for the second case is $O(n^{\frac{\omega+3}{2}+4\delta-\epsilon})$. Finally, discarding all colors from the first two categories, the remaining colors satisfy that $m_\chi \leq n^{\frac{1+\omega}{2}-\delta}$ and that $\sum_\chi m_\chi \leq n^2$. Thus, the time for the third case is at most

$$\sum_\chi (m_\chi)^{\frac{2\omega}{\omega+1}+o(1)} = \sum_\chi m_\chi \cdot (m_\chi)^{\frac{\omega-1}{\omega+1}+o(1)} \leq n^2 \cdot (n^{\frac{1+\omega}{2}-\delta})^{\frac{\omega-1}{\omega+1}+o(1)} \leq n^{\frac{3+\omega}{2}-\frac{\delta}{4}+o(1)}.$$

All in all, the algorithm runs in time $n^{\frac{3+\omega}{2}-\min(\delta, \epsilon-4\delta, \frac{\delta}{4})+o(1)}$. The claim follows for $0 < \delta < \frac{\epsilon}{4}$. \square