Combinatorial Boolean Matrix Multiplication via Graph Decomposition

Eduardo Fernandez, Arya Maheshwari, Andy Wang

December 15, 2023

Abstract

We review a recent breakthrough paper ([AFK⁺23]) that achieves a $n^3/2^{\Omega(\sqrt[7]{\log n})}$ combinatorial algorithm for Boolean Matrix Multiplication (BMM). This is the first combinatorial algorithm that achieves a quasi-polynomial saving over the naïve $O(n^3)$ for Boolean matrix multiplication (previous algorithms had only obtained polylog(n) savings). To provide further context, we also give an overview of important previous combinatorial algorithms for BMM, including the Four Russians' algorithm of [ADKz70] (the first such algorithm with a subcubic runtime) and the work of Bansal and Williams ([BW09]), which introduced the regularity decomposition technique that [AFK$^+$23] build on for their result.

Contents

1 Introduction

Boolean Matrix Multiplication (BMM) is a widely studied problem in algorithm design, with extensive research over the past five decades seeking to improve on the naïve $O(n^3)$ algorithm to multiply two $n \times n$ Boolean matrices. Specifically, the problem of interest is computing the matrix product AB of two Boolean matrices $A, B \in \{0,1\}^{n\times n}$ defined formally by

$$AB_{ik} = \bigvee_{j=1}^{n} A_{ij} \wedge B_{jk}. \tag{1}$$

That is, the (i,k)-th element of the product AB is set to 1 iff there exists an index j where both the i-th row of A and k-th column of B contain a 1. The importance of BMM in computer science is underscored by its far-reaching applications that range from transitive closures [FM71, Fur70] and triangle detection [IR77] in graphs to context-free grammar parsing [Val75] and all-pairs-shortest-path problems [Sei95, GM97, DHZ00].

Compared to standard integer matrix multiplication, BMM is of particular interest from a combinatorial perspective due to its natural interpretation in terms of graphs, phrased as follows. Consider a tripartite graph G defined by vertex tripartition $V = (X, Y, Z)$, such that each part has n vertices and the matrices A and B are adjacency matrices that define the edges from X to Y and Y to Z, respectively. Then, the BMM product $C = AB$ represents the connectivity between X and Z: in particular, it is the adjacency matrix for edges from X to Z if for each $x \in X$, $z \in Z$, the edge $e(x,z)$ is constructed iff x and z share a common neighbor in Y.

Interestingly, despite this combinatorial connection, the fastest current approach to BMM is to just solve the problem as an integer matrix multiplication problem, using so-called algebraic algorithms developed based on Strassen's breakthrough subcubic algorithm from 1969 [Str69]. This line of work is based on recursively multiplying small matrices over a ring and exploiting cancellations to obtain runtime speedups, hence its characterization as "algebraic," and has culminated thus far in a $\tilde{O}(n^{2.3716})$ bound due to [WXXZ23]. However, such procedures suffer from issues related to practical performance, simplicity, and generalizability ([AFK+23]), so researchers have also sought so-called "combinatorial algorithms" that instead aim to exploit the underlying combinatorial structure of BMM neglected in the algebraic approach to the problem. We will discuss in more detail this class of algorithms and the motivation for their development starting in the next section, but broadly, the landmark combinatorial algorithms have been characterized by divide-and-conquer approaches [ADKz70, Cha14, Yu15] (see Section 2.2) or graph decomposition techniques [BW09] (see Section 3.2). While these approaches have yielded polylog savings over cubic time, no combinatorial algorithm has been able to achieve polynomial savings (i.e. a $O(n^{3-\varepsilon})$ bound for any $\varepsilon > 0$), leading to the following conjecture:

Conjecture 1.1. *No "truly subcubic" combinatorial algorithm exists for BMM. That is, for any $\varepsilon > 0$, no combinatorial algorithm for BMM runs in time $O(n^{3-\varepsilon})$.*

In this paper, we survey Abboud et. al's recent breakthrough combinatorial algorithm for BMM [AFK+23] which achieves a (combinatorial) state-of-the-art $n^3/2^{\Omega(\sqrt[7]{\log n})}$ bound via a novel graph decomposition technique. While this still does not achieve a $O(n^{3-\varepsilon})$ bound, it comes much closer than before with the *quasi-polynomial* $2^{\Omega(\sqrt[7]{\log n})}$ speedup (versus previous polylog savings). We state this main result in Theorem 1.1, which is the target that we will build toward in the remainder of our paper (particularly in Section 4).

Theorem 1.1 (Main Result (Theorem 1.2 of [AFK+23]))**.** *There is a deterministic combinatorial algorithm computing the Boolean product of two $n \times n$ matrices in time $n^3/2^{\Omega(\sqrt[7]{\log n})}$.*

We will present the main ideas and techniques that drive this algorithm while also providing more context (that is often omitted in [AFK+23]) to make this survey accessible to a reader without background on the rather technical building blocks used in this new algorithm. As a small technical note that will apply for the rest of the paper, we always assume a word-RAM model with word size $\Theta(\log n)$, and generally ignore polyloglog(n) factors in runtimes unless otherwise stated.

2 Background

To contextualize the result of [AFK$^+$23], we begin by overviewing the motivation for combinatorial algorithms (Section 2.1); the "Four Russians' Algorithm" [ADKz70] and subsequent combinatorial algorithms that it inspired (Section 2.2); and the connection between BMM and triangle detection leveraged by the majority of combinatorial BMM algorithms, including [AFK$^+$23] (Section 2.3). We then cover background on graph regularity decompositions in Section 3 and their role in BMM algorithms before presenting Abboud et. al's algorithm in Section 4.

2.1 Combinatorial algorithms

As previously mentioned, combinatorial algorithms are generally characterized by their lack of the algebraic cancellations used in Strassen's algorithm and its successors, instead applying operations more naturally grounded in the combinatorial structure underlying BMM. That said, there is currently no consensus on a common, precise definition of what counts as a "combinatorial" algorithm. This is in part due to the difficulty in pinning down precise models that are flexible enough to support the operations of existing combinatorial algorithms, yet are limited enough to exclude unrealistic (e.g. quadratic-time) algorithms [DKS18]. For instance, in order to prove lower bounds [DKS18] define specific versions of [Ang76]'s *row-union model* in which a combinatorial algorithm is essentially permitted to take bitwise-ORs of partitions of matrix rows; while this model is able to support the operations of the Four Russians' Algorithm (discussed in Section 2.2), it is unable to simulate more recent algorithms (e.g. [BW09, Cha14, Yu15, AFK$^+$23] in particular) that are still viewed as "combinatorial."

Other works embrace a looser approach to designing combinatorial algorithm. In particular, instead of specifying a precise model, [AFK$^+$23] lay out three key limitations of Strassen-derived algebraic algorithms and argue that the priority should be on *first* finding any other algorithm that can break the subcubic barrier and *then* considering which limitations it improves upon or which combinatorial notions it satisfies. The limitations they highlight are the following:

- **Simplicity**: Strassen's algorithm famously exploits unintuitive algebraic cancellations that provide little insight into the combinatorial behaviors of BMM; indeed, there is a whole line of work interested in explaining why Strassen's algorithm is even possible in the first place [GM17].
- **Practical Efficiency**: Strassen-derived algebraic algorithms are often considered "galactic" in the sense that although they improve upon asymptotic bounds for BMM, they are completely impractical due to excessive leading constant factors. The practicality of Strassen's algorithm itself is somewhat ambiguous—it has seen some use in practice [Lee01, HSHvdG16], but the algorithm still suffers due to poor caching behavior [ABH10].
- **Generalizability**: Perhaps the most compelling theoretical motivation for interest in non-algebraic techniques for BMM is that Strassen's algorithm is not robust in generalizing to BMM-related problems. For instance, triangle enumeration is not possible via Strassen's algorithm due to the algebraic cancellations it requires; on the other hand combinatorial BMM algorithms such as that of [BW09] and [AFK$^+$23] have been able to substantially speed up triangle enumeration, and are in fact state-of-the-art, even compared to algebraic techniques.

We remark that it is natural that a study like [DKS18] aiming to make progress on lower bounds would adhere to a precise, restrictive model of combinatorial algorithms, while a study like [AFK$^+$23] aimed at an algorithm to improve the upper bound would first work from the loosest viewpoint possible.

2.2 Divide and conquer: The Four Russians' Algorithm

Here we briefly review the first subcubic combinatorial algorithm for BMM (the so-called Four Russians' algorithm), discovered by Arlazarov, Dinitz, Kronrod, and Faradzhev and published in 1970, shortly after Strassen's 1969 breakthrough ([ADKz70]). We follow the presentation in [BW09] and [Su21]. Note that the notation Av denotes Boolean matrix-vector multiplication, and $u \vee v$ denotes the bitwise OR of bitvectors u, v. The algorithm is presented in Algorithm 1 below.

Algorithm 1 Four Russians'

```
 1: procedure FourRussiansBMM($A, B$)
 2:     $\varepsilon \leftarrow 0.1$
 3:     Partition $A$ into blocks $A_{i,j}$ for $i, j \in \left[\frac{n}{\varepsilon \lg n}\right]$, each of size $\varepsilon \lg n \times \varepsilon \lg n$.
 4:     for $i, j \in \left[\frac{n}{\varepsilon \lg n}\right]$ do
 5:         $T_{i,j} \leftarrow$ a lookup table such that for all $v \in \{0,1\}^{\varepsilon \lg n}$, $T_{i,j}[v] = A_{i,j}v$.
 6:     end for
 7:     $S \leftarrow$ a lookup table such that for all $u, v \in \{0,1\}^{\varepsilon \lg n}$, $S[u, v] = u \vee v$.
 8:     For each $k \in [n]$, partition the $k$-th column of $B$ into $\frac{n}{\varepsilon \lg n}$ parts of $\varepsilon \lg n$ consecutive entries.
 9:     $B_j^k \leftarrow$ the $j$-th part of the $k$-th column of $B$ (so $B_j^k \in \{0,1\}^{\varepsilon \lg n}$).
10:     for $i \in \left[\frac{n}{\varepsilon \lg n}\right]$ do
11:         for $k \in [n]$ do
12:             $Q[i, k] \leftarrow$ the all-zeros vector in $\{0,1\}^{\varepsilon \lg n}$.
13:             for $j \in \left[\frac{n}{\varepsilon \lg n}\right]$ do
14:                 $Q[i, k] \leftarrow S(Q[i, k], T_{i,j}[B_j^k])$.
15:             end for
16:         end for
17:     end for
18:     return $Q$.
19: end procedure
```

Although we index Q by $n/(\varepsilon \lg n)$ rows and n columns, note that each entry is a bitvector of length $\varepsilon \lg n$, so we freely identify Q with the corresponding $n \times n$ square matrix.

We claim that Algorithm 1 correctly computes the Boolean matrix product AB and runs in time $O\left(\frac{n^3}{\log^2 n}\right)$[1].

Proof. **Correctness.** Upon termination of the algorithm, by construction, we have that for each $i \in [n/(\varepsilon \lg n)]$ and $k \in [n]$,

$$Q[i, k] = \bigvee_{j=1}^{n/(\varepsilon \lg n)} A_{i,j} B_j^k.$$

It is not hard to see by standard block matrix multiplication that the i, k coordinate of the $n \times n$ matrix corresponding to Q is precisely equal to 1, so the correctness is clear.

Runtime. Each matrix-vector multiplication in line 5 takes time $\varepsilon^2 \log^2 n$ (this is just naïve multiplication of an $\varepsilon \lg n \times \varepsilon \lg n$ matrix by an $\varepsilon \lg n$-length vector).

Since there are $2^{\varepsilon \lg n} = n^\varepsilon$ bitvectors of length $\varepsilon \lg n$, and the for-loop in line 4 executes $\frac{n^2}{\varepsilon^2 \lg^2 n}$ times, the total runtime of the precomputation of the $T_{i,j}$'s is

$$O\left(\varepsilon^2 \log^2 n \cdot n^\varepsilon \cdot \frac{n^2}{\varepsilon^2 \lg^2 n}\right) = O(n^{2+\varepsilon}) = O(n^{2.2}).$$

[1]We should note that the original algorithm as presented by Arlazarov et al. only achieved $\log n$ savings over cubic time, but this was later improved to $\log^2 n$ by precomputation of the bitwise ORs (line 7). This idea is from [Su21], but the improvement was already present in, for example, [BKM95]

By a similar counting argument, the runtime of precomputing table S (line 7) is

$$O((n^{\varepsilon})^2\varepsilon) = O(n^{2\varepsilon}\varepsilon \lg n) = O(n^{0.3}).$$

Here we use that computing the bitwise OR of two bitvectors of length $\varepsilon \lg n$ requires time $O(1)$ in the word-RAM model with $\Theta(\log n)$-sized words.

Since we've already precomputed all the required values in lookup tables that can be accessed in constant time, line 14 can be done in constant time. From the bounds in the three surrounding for-loops, we get that the runtime of lines 10 to 17 is

$$O\left(\frac{n}{\varepsilon \lg n} n \frac{n}{\varepsilon \lg n}\right) = O\left(\frac{n^3}{\log^2 n}\right).$$

Combining these runtimes, we obtain that asymptotically, Algorithm 1 indeed runs in $O\left(\frac{n^3}{\log^2 n}\right)$ time, as desired. ■

Although we will not review their algorithms in detail here, following Bansal and Williams' 2009 breakthrough (see Section 3), Chan ([Cha14]) and Prof. Yu ([Yu15]) achieved $\log^3 n$ and $\log^4 n$ savings, respectively, through more sophisticated divide-and-conquer techniques.

2.3 Reduction to Triangle Detection

Much of the work on combinatorial BMM algorithms in recent years has relied on a standard reduction of Boolean matrix multiplication to the *Triangle Detection* problem – the task of determining whether or not a given input tripartite graph contains a triangle. We include the proof of this reduction here for completeness.

For a comprehensive treatment of subcubic equivalences between various combinatorial problems (all-pairs shortest paths, triangle detection, triangle listing, metric verification, matrix multiplication, among others) see the following paper by Williams and Vassilevska Williams: [WW10]. In this section we adapt the coverage of the reduction in this paper and in the lecture notes [Su21].

First, we prove a useful lemma.

Lemma 1 (Lemma 4.1 in [WW10])**.** *Suppose there is an algorithm that solves Triangle Detection in time $T(n)$ when given an arbitrary tripartite graph $G = (X, Y, Z)$ on n vertices as input. Assume further that $T(n)/n$ is nondecreasing. Then there is an $O(T(n))$-time algorithm that returns a triangle in G iff one exists.*

Proof. This Lemma is cited as folklore in [WW10]. Since their recursive divide-and-conquer proof technique is similar in spirit to the proof of our main Theorem 2.1, we include it here as warmup.

First, divide each of the parts of G into two subparts with cardinality as equal as possible: $(X^{(1)}, X^{(2)})$, $(Y^{(1)}, Y^{(2)})$, and $(Z^{(1)}, Z^{(2)})$. For ease of exposition, suppose $|X|$, $|Y|$, and $|Z|$ are all powers of 2 (to avoid issues with divisibility).

For each of the 8 triples $(X^{(i_1)}, Y^{(i_2)}, Z^{(i_3)})$, run the triangle-detecting algorithm recursively on the graph induced by $X^{(i_1)} \cup Y^{(i_2)} \cup Z^{(i_3)}$. If all of these executions return False (that is, all induced subgraphs are triangle-free), return that the graph is triangle-free. Otherwise, recurse on a *single* induced subgraph on which the algorithm returned True. The base case occurs when the algorithm is called on a graph on three vertices, where we simply check if the three vertices form a triangle in constant time and returns this vertex set if they do.

The correctness of the algorithm is clear: if a triangle exists in G, the triangle-detection algorithm will return True on some triple of subparts, and we will recurse until we find the triangle and return it; if G is triangle-free, then all calls to the triangle detection algorithm will return False.

Denote the runtime of the algorithm by $F(n)$. Then $F(n)$ satisfies the recurrence

$$F(n) = 8T(n) + F(n/2),$$

with $F(k) = O(1)$ for $k \leq 3$. Since we're assuming that $T(n) = ng(n)$ for some nondecreasing g, it follows that

$$T(n) = 2\frac{n}{2}g(n) \geq 2\frac{n}{2}g(n/2) = 2T(n/2).$$

But then from the Master Theorem[2], it follows that $F(n) = O(T(n))$, as desired. ∎

Now we prove the main result.

Theorem 2.1 (Theorem 4.2 in [WW10], Theorem 2.1 in [Su21])**.** *Suppose there is a $T(n)$-time algorithm for Triangle Detection on graphs on n vertices, where $T(n)/n$ is nondecreasing. Then there is an $O(n^2\,T(n^{1/3}))$-time algorithm that computes the Boolean matrix product of $A, B \in \{0,1\}^{n\times n}$.*

Proof. Let G be a tripartite graph with vertex parts X, Y, Z, each of size n. Denote the i-th vertex in each part by X_i, Y_i, and Z_i, respectively. Add an edge (X_i, Z_k) for each $i, k \in [n]$. Also add edges (X_i, Y_j) for each i, j such that $A(i,j) = 1$, and similarly add edges (Y_j, Z_k) for each j, k such that $B(j,k) = 1$.

By Lemma 1, there exists an $O(T(n))$-time algorithm that *finds* a triangle in G if there is one.

Observe that G has a triangle containing vertex Y_j if and only if there exist $i, j, k \in [n]$ such that $A(i,j) = B(j,k) = 1$, which is equivalent to $(AB)(i,k) = 1$. Thus our strategy will be to use the triangle-finding algorithm, combined with a divide-and-conquer technique, to determine if each Y_j is contained in a triangle. This will suffice to compute $(AB)(i,k)$ for all i, k.

Let t be a parameter that we will set later. We will partition each of X, Y, and Z into t subparts (each on n/t vertices), denoted $X^{(1)}, \ldots, X^{(t)}$ in the case of X and similarly for Y and Z.

Initialize a matrix $C \in \{0,1\}^{n\times n}$ to be all zeros.

For each triple $X^{(i_1)}, Y^{(i_2)}, Z^{(i_3)}$, execute the following procedure:

(1) Run the triangle-finding algorithm on the subgraph of G induced by $X^{(i_1)} \cup Y^{(i_2)} \cup Z^{(i_3)}$.

(2) If this subgraph is triangle-free, continue to the next iteration (consider a new triple).

(3) Otherwise, suppose the triangle found in the subgraph is (X_i, Y_j, Z_k). Set $C(i,k) = 1$, delete this triangle from the induced subgraph, and recurse on this new subgraph by returning to step (1).

Return C.

First, we prove that the algorithm is correct – that is, $C = AB$. By the observation above, we clearly have that whenever $C(i,k)$ is set to 1 by the algorithm, we indeed have that $(AB)(i,k) = 1$. All that is left to show is that if $C(i,k)$ is *not* set to 1, then $(AB)(i,k) = 0$. Since we exhaustively iterate over *all* triples of vertices in each of the subparts, if $C(i,k) = 0$ upon termination, we conclude that there does not exist $j \in [n]$ such that (X_i, Y_j, Z_k) is a triangle, and therefore indeed $(AB)(i,k) = 0$, as desired.

For the runtime, note that whenever our triangle-finding algorithm returns that the subgraph is triangle-free, we immediately move on to the next iteration of the main for-loop. Thus, the number of times this happens is bounded by t^3, the number of triples the algorithm iterates over. This implies that the runtime contribution of these instances is $O(t^3\,T(3n/t))$.

On the other hand, whenever the triangle-finding algorithm returns a triangle $(X^{(i)}, Y^{(j)}, Z^{(k)})$, we set $C(i,k) = 1$ *and* delete this triangle from the subgraph. This means we set each $C(i,k)$ to 1 at most once. Since there are n^2 entries in C, the runtime contribution from these instances is $O(n^2\,T(3n/t))$.

Thus, the total runtime of the algorithm is $O(n^2\,T(3n/t) + t^3\,T(3n/t))$. To optimize this, we set $t = 3n^{2/3}$, which yields a total runtime of $O(n^2\,T(n^{1/3})$, as desired. ∎

Corollary 2.1.1. *An $O\left(\frac{n^3}{2^{\Omega(\sqrt[7]{\log n})}}\right)$-time algorithm for Triangle Detection implies an algorithm for BMM that also satisfies this asymptotic runtime guarantee.*

[2]Specifically, Case 3 of Theorem 4.1 in the Third Edition of CLRS ([CLRS09])

3 Graph regularity decompositions

The use of graph regularity decompositions in BMM combinatorial algorithms was first introduced by Bansal and Williams in [BW09]. These techniques offered the first substantial improvement in many years for combinatorial BMM algorithms over the classic Four Russians algorithm. Broadly speaking, notions of graph regularity measure the "randomness" of a graph in some sense: the more regular a graph is, the more it behaves like a uniformly random graph does in expectation. Heuristically, this is useful because the properties of random graphs are easy to analyze in expectation, whereas a non-random graph can have arbitrary structures that complicate analyses. The remarkable result of graph regularity decomposition theorems like Szemerédi's regularity lemma is that *any* graph admits a decomposition that partitions the graph into regular pieces, which can then be analyzed using the properties of the relevant regularity notion. In this section we present Szemerédi's regularity lemma and its associated regularity notion to demonstrate some of the general themes present in regularity decompositions. We then turn to the results of [BW09] that demonstrate the utility of regularity decompositions in BMM, and discuss how these ideas compare with the novel regularity decomposition that [AFK+23] introduces to achieve its main result.

3.1 Szemerédi's regularity lemma

Szemerédi's regularity lemma is a classic result that gives robust results on the regularity of arbitrary graphs. It has become a standard tool in various graph theoretical applications [KSSS02], and has yielded improvements on BMM bounds via the Triangle removal lemma [BW09]. Informally, it says that a graph of any size can be partitioned into parts that behave "pseudo-randomly." To make this notion precise, we introduce the notion of **ε-regularity**. Consider an undirected, unweighted graph G with vertices $V(G)$ and edges $E(G)$. For $X, Y \subset V(G)$ (possibly intersecting), we define

$$\begin{aligned} e_G(X,Y) &:= \left|\{(x,y) \in X \times Y \,:\, xy \in E(G)\}\right| \\ d_G(X,Y) &:= \frac{e_G(X,Y)}{|X||Y|}. \end{aligned}$$

Here $d_G(X,Y)$ can be viewed as the edge density between subsets X and Y. The notions of ε-regularity used in Szemerédi's regularity lemma are as follows.

Definition 3.1. For $X, Y \subset V(G)$, the pair (X,Y) is **ε-regular** if for all $A \subset X$ and $B \subset Y$ such that $|A| \geq \varepsilon|X|$ and $|B| \geq \varepsilon|Y|$,

$$\left|d(A,B) - d(X,Y)\right| \leq \varepsilon.$$

Definition 3.2. A partition $\mathcal{P} = \{V_1, \dots, V_k\}$ of $V(G) = V_1 \sqcup \cdots \sqcup V_k$ is **ε-regular** if

$$\sum_{\substack{(i,j)\in[k]^2 \\ (V_i,V_j) \text{ not } \varepsilon\text{-regular}}} |V_i||V_j| \leq \varepsilon|V(G)|^2.$$

Observe that by definition, if a pair $X, Y \subset V(G)$ is not ε-regular, then there must exist some $A \subset X$ and $B \subset Y$ such that $|A| \geq \varepsilon|X|$ and $|B| \geq \varepsilon|Y|$ but $|d(A,B) - d(X,Y)| > \varepsilon$; we call such a pair a "witness" of the irregularity of X and Y. These witnesses play key roles in regularity theorems, particularly in terms of refining partitions to improve regularity, as we will see in the Szemerédi regularity lemma.

Lemma 2 (Szemerédi's regularity lemma)**.** *For all $\varepsilon > 0$, every graph has a ε-regular partition into at most $M(\varepsilon)$ parts, where $M(\varepsilon)$ is a tower $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ of height* $\operatorname{poly}(1/\varepsilon)$.

Note that the size of the decomposition, M, is not dependent on the size of the graph, but instead it only depends on the regularity parameter ε. The proof of Lemma 2 constructs a ε-regular partition of an arbitrary graph by beginning with a trivial partition, and using witnesses of irregularity to refine a partition until it becomes ε-regular (at which point there are no more witnesses). It can be shown that the normalized mean square density, or “energy,” of the partition is contained in $[0, 1]$ but increases by $\text{poly}(\varepsilon)$ each time the partition is refined by irregularity witnesses, so that this process must terminate within some number of steps dependent on ε, yielding a ε-regular partition.

One of the main applications of Szemerédi’s regularity lemma is the graph removal lemma; the Triangle removal lemma is a special case of this result that is pertinent to BMM.

Theorem 3.1 (Triangle removal lemma)**.** *For all $\varepsilon > 0$, every graph on n vertices with at most εn^3 triangles can be made triangle-free by removing at most $f(\varepsilon)n^2$ edges, where $f(\varepsilon) = 1/(\log^* 1/\varepsilon)^\delta$.*

While Szemerédi’s regularity lemma provides powerful graph theoretical results, the sheer size of the partition that it guarantees gives poor practical improvements in algorithms seeking to take advantage of the regularity that the lemma guarantees. Moreover, the size of $M(\varepsilon)$ is nearly tight due to a result by Gowers, so in general, there is no hope of substantial improvements in the size of the decomposition provided by the lemma.

Nonetheless, many of the ideas used in the Szemerédi regularity lemma are used more broadly in regularity decompositions, including results in [BW09] and even in the new result of [AFK$^+$23]. Indeed, the way in which irregularity witnesses generate the regularity decomposition, and the use of density increment in Lemma 2 to terminate the decomposition process are very similar to the main ideas of the regularity decomposition in [AFK$^+$23]. Some of the results that the lemma has proven also have potential for further optimization, even if the lemma itself does not, as we will see briefly in the next section.

3.2 BMM via graph regularity in [BW09]

There are two main results in [BW09] for triangle detection, each of which relies on a different regularity notion. The first result is due to the Triangle removal lemma (which in turn is due to Szemerédi’s regularity lemma), and the second is due to a new, weaker notion of regularity.

a. An algorithmic version of the Triangle removal lemma (which, in turn, is due to Szemerédi’s regularity lemma) yields a randomized combinatorial algorithm for BMM with a runtime of $O\big(n^3 \log(\log^* n)/(\log^2 n(\log^* n)^\delta)\big)$ for some $\delta > 0$, and

b. Weak regularity (a notion used in the Frieze-Kannan regularity lemma in [FK99]) yields a randomized combinatorial algorithm for BMM in time $\hat{O}(n^3/\log^{2.25} n)$, where $\hat{O}$ ignores poly-loglog factors.

The first result is a *very* modest $(\log^* n)^\delta/\log\log^* n$ improvement over the Four Russians’ algorithm. It makes use of the decomposition procedure used to prove Szemerédi’s regularity lemma (described in Section 3.1) to algorithmically obtain an ε-regular partition of a graph, which in turn is applied to the Triangle removal lemma to obtain a sparse (that is, $O(n^2)$ rather than $O(n^3)$) set of edges that can participate in triangles. This allows Bansal and Williams to obtain result (a), but it only results in a small improvement since Szemerédi’s regularity lemma yields such poor bounds on f in the Triangle removal lemma (see Lemma 3.1).

Bansal and Williams note, however, that improvements in the Triangle removal lemma could improve the savings from this result. In fact, known lower bounds for the Triangle removal lemma are still poorly developed, which leaves room for further improvements to the Triangle removal lemma. In particular, as of [BW09], the best known lower bound on $f(\varepsilon)$ in Lemma 3.1 is still $2^{-O(\sqrt{\log(1/\varepsilon)})}$, which, if achieved, could still imply a $n^3/2^{\Theta(\sqrt{\log n})}$ time BMM algorithm—this bound is even better than the current state-of-the-art runtime of $n^3/2^{\Omega(\sqrt[7]{\log n})}$ achieved in [AFK$^+$23].

Bansal and Williams also use another notion of regularity due to Frieze and Kannan, known as *weak regularity*, to achieve better concrete bounds in result b. Compared to the Four Russians algorithm, they achieve additional savings of $\log^{0.25} n$ (ignoring poly-loglog factors). This improvement is possible because weak regularity admits a much more efficient decomposition than ε-regularity in Szemerédi's regularity lemma: Frieze and Kannan show that an arbitrary graph can be decomposed into an ε-pseudoregular (the analog of ε-regularity for weak regularity) partition of size $2^{1/\varepsilon^2}$ [FK99], compared to the tower of 2's of height poly(ε) in Szemerédi's regularity lemma.

This demonstrates one of the key ideas in adapting regularity decomposition techniques for use in algorithms: there is a fundamental tradeoff between the strength of the regularity guaranteed by the decomposition and the practical effiency of the decomposition [AFK$^+$23]. Stronger regularity notions provide better guarantees but require larger decompositions, and inversely. For instance, Szemerédi's regularity lemma provides very strong (and tight) guarantees, but it is overkill for BMM, since it is only used to achieve the Triangle removal lemma (which has tighter implementations from other techniques). The key contribution of [AFK$^+$23] is a new regularity decomposition that better optimizes this tradeoff, based on a substantially different idea of regularity compared to those of Szemerédi's regularity lemma and Frieze-Kannan weak regularity known as *grid regularity*, introduced by Kelley, Lovett, and Meka [KLM23]. Grid regularity is weaker than the notions used in [BW09], but it is strong enough to be used in triangle detection and can be used with much smaller decompositions.

4 A new graph decomposition in [AFK$^+$23]

The main result we survey in this paper is [AFK$^+$23], which achieves the first super-polylogarithmic runtime savings over n^3 for the Triangle Detection problem, and hence also a state-of-the-art combinatorial algorithm for BMM by 2.1.

4.1 Preliminaries

In this section we introduce some notation and conventions used throughout [AFK$^+$23] (and hence throughout the rest of this section). This material is entirely from Section 2.2 of [AFK$^+$23].

Unless otherwise specified, X, Y, and Z are sets and A, B, and C are matrices (in $\mathbb{R}_{\geq 0}^{|X|\times|Y|}$, $\mathbb{R}_{\geq 0}^{|Y|\times|Z|}$, and $\mathbb{R}_{\geq 0}^{|X|\times|Z|}$, respectively).

AB denotes the standard matrix product, and $A \circ B$ denotes a slightly modified matrix product:

$$(A \circ B)(x, z) = \mathbb{E}_{y \in Y} A(x, y)B(y, z) = \frac{1}{|Y|} \sum_{y \in Y} A(x, y)B(y, z) = \frac{1}{|Y|}(AB)(x, z).$$

We will very often freely identify boolean matrices $A \in \{0, 1\}^{X\times Y}$ with the corresponding bipartite graphs with parts X, Y and edges (x, y) present iff $A(x, y) = 1$. For $X^* \subseteq X$ and $Y^* \subseteq Y$, $A[X^*, Y^*]$ denotes the bipartite subgraph of A induced by the nodes in X^* and Y^*.

Unless otherwise specified, $\mathbb{P}$ and $\mathbb{E}$ assume uniform random sampling over objects of interest. For example, the *density* of A is

$$\mathbb{E}[A] = \mathbb{E}_{\substack{x \in X\\ y \in Y}} A(x, y) = \frac{1}{|X||Y|} \sum_{\substack{x \in X\\ y \in Y}} A(x, y).$$

That is, if $x \in X$ is sampled uniformly, as is $y \in Y$ (independently), then $\mathbb{E}[A]$ is the probability that (x, y) is an edge in A.

We also define the relative degree of a node $x \in X$:

$$\deg_A(x) = \mathbb{E}_{y \in Y} A(x, y) = \frac{1}{|Y|} \sum_{y \in Y} A(x, y).$$

The definition of $\deg_A(y)$ for $y \in Y$ is completely symmetric.

In other words, out of all possible edges (x, y) that could exist in the bipartite graph A, a $\deg_A(x)$ fraction of them are actually present.

A is *ε-left-min-degree* (or just *ε-min-degree*) if $\min_{x\in X}\deg_A(x) \geq (1-\varepsilon)\mathbb{E}[A]$. Intuitively, A satisfies this condition if no node in X has a relative degree that is "too low", compared to the overall density of the entire graph.

Finally, for $\alpha,\varepsilon,\delta > 0$, we say that A is $(\alpha,\varepsilon,\delta)$-*uniform* if

$$\mathbb{P}_{(x,y)\in X\times Y}[(1-\varepsilon)\alpha \leq A(x,y) \leq (1+\varepsilon)\alpha] \geq 1-\delta.$$

In other words, A satisfies this uniformity condition if at least a $1-\delta$ fraction of the entries in A lie between $(1-\varepsilon)\alpha$ and $(1+\varepsilon)\alpha$. The α parameter seems superfluous for Boolean matrices whose values are in $\{0,1\}$, but we use this notion of uniformity in connection with Theorem 4.1, which applies to general matrices with nonnegative entries, so we keep the parameters as they are.

4.2 Grid regularity and matrix products

[AFK$^+$23] builds on the work of [BW09] in that the triangle detection algorithm the authors present relies on a regularity decomposition of a graph. A key innovation in [AFK$^+$23] is using *grid regularity* – rather than Szemerédi regularity or Frieze-Kannan regularity – to guide their decomposition . In this section we develop this notion of regularity and connect it to matrix products. This material is from Sections 2.3 and 2.4 of [AFK$^+$23], with some results from [KLM23].

To start, given $k,\ell \geq 1$, we define the (k,ℓ)-grid norm[3] of a nonnegative matrix $A \in \mathbb{R}_{\geq 0}^{X\times Y}$:

$$\|A\|_{U(k,\ell)} = \left(\mathbb{E}_{\substack{x_1,\ldots,x_k\in X\\ y_1,\ldots,y_\ell\in Y}} \prod_{\substack{i\in[k]\\ j\in[\ell]}} A(x_i,y_j)\right)^{\frac{1}{k\ell}}.$$

Note that for a fixed k-tuple $x_1,\ldots,x_k \in X$ we can write

$$\mathbb{E}_{y_1,\ldots,y_\ell} \prod_{i\in[k],j\in[\ell]} A(x_i,y_j) = \mathbb{E}_{y_1}\ldots\mathbb{E}_{y_\ell} \prod_{i\in[k],j\in[\ell]} A(x_i,y_j) = \left(\mathbb{E}_{y\in Y}\prod_{i\in[k]} A(x_i,y)\right)^{\ell}.$$

Thus, raising the original (k,ℓ)-norm to the $k\ell$ power, we can write

$$\|A\|_{U(k,\ell)}^{k\ell} = \mathbb{E}_{x_1,\ldots,x_k\in X}\left(\mathbb{E}_{y\in Y}\prod_{i\in[k]} A(x_i,y)\right)^{\ell},$$

which will be useful for us later.

For the rest of this section, suppose A is a Boolean matrix. Then note that for a given pair of tuples $x_1,\ldots,x_k$ and $y_1,\ldots,y_\ell$, each with distinct elements, the product $\prod_{i,j} A(x_i,y_j)$ is 1 if and only if the subgraph induced by these vertices is a (k,ℓ)-*biclique* – that is, (x_i,y_j) is an edge for all i,j. Thus, we can roughly think of the grid norm as counting the number of (k,ℓ)-bicliques that occur as a subgraph of A, followed by some normalization, with the technicality that some of the nodes of each biclique might coincide.

Observe that from comparing definitions, we have $\|A\|_{U(1,1)} = \mathbb{E}[A]$, the density of A. Moreover, across all graphs A, clearly $\|A\|_{U(|X|,|Y|)}$ is maximized when $A = K_{|X|,|Y|}$, the complete biclique on vertex parts X,Y. In this case, $\|A\|_{U(|X|,|Y|)} = 1$, since $A(x,y) = 1$ for all $(x,y)\in X\times Y$.

We also have the following useful lemma:

Lemma 3 (Claim 4.2 in [KLM23]). *If $\ell \leq \ell'$, then for any $A \in \mathbb{R}_{\geq 0}^{X\times Y}$,*

$$\|A\|_{U(k,\ell)} \leq \|A\|_{U(k,\ell')}.$$

[3]Note that despite the terminology used, the grid norm is not necessarily a norm (i.e., it doesn't necessarily satisfy the required triangle inequality).

We omit the proof, but note that by symmetry, the analogous claim that whenever $k \leq k'$ we have $\|A\|_{U(k,\ell)} \leq \|A\|_{U(k',\ell)}$ is also true.

Combining the Lemma and the observation above, we get that for any graph A and any $k, \ell \in [n]$,

$$\mathbb{E}[A] \leq \|A\|_{U(k,\ell)} \leq 1.$$

To see why it is reasonable to interpret the grid norm as a measure of pseudo-randomness, consider the following: For concreteness, take $n = |X| = |Y| = 20$ and $k = \ell = 4$.

Let $A \in \{0,1\}^{X \times Y}$ be a Boolean matrix such that the $n/2 \times n/2$ top-left block is all 1's, but all other entries are zero. Then $\mathbb{E}[A] = 1/4$, and the number of (k,ℓ)-bicliques is exactly $\binom{10}{4}^2 > 44{,}000$.

Let $B \in \{0,1\}^{X \times Y}$ be a purely random bipartite graph, where each edge (x,y) is present independently and with probability $p = \mathbb{E}[A] = 1/4$. Then note that in expectation, the density of B is clearly equal to $\mathbb{E}[A]$ and the number of (k,ℓ)-bicliques in B is

$$p^{k\ell}\binom{n}{k}\binom{n}{\ell} = \left(\frac{1}{4}\right)^{16}\binom{20}{4}^2 < 1.$$

Of course, this is only a specially constructed example and we don't take into account bicliques with repeated nodes (which the original definition of the (k,ℓ) grid norm does consider) or normalization (exponentiating to $1/(k\ell)$), but this case still illustrates an important general takeaway: a highly structured graph (like A above) will tend to have a larger grid norm than a random-like graph (like B), *even if both graphs have the same edge density.*

With this motivation in mind, we say that A is (ε, k, ℓ)-regular if its (k,ℓ)-grid norm is not too large relative to its density – that is, if

$$\|A\|_{U(k,\ell)} \leq (1+\varepsilon)\mathbb{E}[A].$$

Grid norms turn out to be extremely useful due to a recent result by Kelley, Lovett and Meka ([KLM23]) that links the regularity of two graphs to the product of their corresponding matrices:

Theorem 4.1 (Theorem 2.1 in [AFK+23], Lemma 4.8 in [KLM23])**.** *Let $A \in \mathbb{R}_{\geq 0}^{X \times Y}$ and $B \in \mathbb{R}_{\geq 0}^{Y \times Z}$, let $\varepsilon \in (0, \frac{1}{80})$, let $d \geq 2/\varepsilon$ and assume that*

(a) A and B^T are $(\varepsilon, 2, d)$-regular, and

(b) A and B^T are ε-min-degree.

Then $A \circ B$ is $(\mathbb{E}[A]\mathbb{E}[B], 80\varepsilon, 2^{-\varepsilon d/2})$-uniform.

We omit the proof here, which involves a combination of analytic and probabilistic arguments. When $A, B \in \{0,1\}^{n \times n}$, we can very roughly think of Theorem 4.1 as saying that if A and B behave pseudo-randomly (i.e. they satisfy the grid-regularity and min-degree conditions), then a large fraction of the entries in the *product* matrix AB are 1's (a consequence of satisfying the uniformity condition). This structural result is the starting point of the decomposition that is the focus of the rest of the paper.

4.3 Easy case: regular edge parts

In this section we connect the ideas in Section 4.2 to the Triangle Detection problem in order to showcase the power of regularity. Suppose $A \in \{0,1\}^{X \times Y}$, $B \in \{0,1\}^{Y \times Z}$, and $C \in \{0,1\}^{X \times Z}$, and our task is to detect a triangle in the tripartite graph G with vertex parts (X, Y, Z) in subcubic time. For this section, we assume a "best case" scenario where A and B satisfy strong regularity properties. More specifically, we assume they satisfy the conditions in Theorem 4.1: Let $\varepsilon = \frac{1}{160}$ and $d \geq 2/\varepsilon$. Then we assume that

(1) A and B^T are $(\varepsilon, 2, d)$-regular and

(2) A and B^T are ε-min-degree.

Note that there is a triangle in G if there are vertices $x \in X, y \in Y, z \in Z$ such that $A(x,z) = B(y,z) = C(x,z) = 1$. In terms of Boolean matrix multiplication, this is equivalent to the existence of a pair of indices (i,k) such that $(AB)(i,k) = C(i,k) = 1$.

First, if either A or B is the zero matrix (which we can check in $O(n^2)$ time), return False – clearly there is no triangle in G in this case.

We split the triangle detection task into two cases.

First, suppose C is *sparse* – that is, $\mathbb{E}[C] \leq 2^{-\varepsilon d/2}$. In this case our strategy is straightforward brute force: enumerate all $\frac{n^2}{2^{O(d)}}$ edges in C and iterate over all n nodes in Y to check in constant time for each node if the C edge is part of a triangle. This gives an $n^3/2^{O(d)}$ algorithm.

For the remaining case, suppose C is dense, so $\mathbb{E}[C] > 2^{-\varepsilon d/2}$. In this case, we can just output True – that is, G does contain a triangle.

To see why, note that by Theorem 4.1 properties (1) and (2) together guarantee that $A \circ B$ is $(\mathbb{E}[A]\mathbb{E}[B], \frac{1}{2}, 2^{-\varepsilon d/2})$-uniform. Unpacking the definition of uniformity from Section 4.1, this says that at least a $1 - 2^{-\varepsilon d/2}$-fraction of the entries in $A \circ B$ lie in the interval $[\frac{1}{2}\mathbb{E}[A]\mathbb{E}[B], \frac{3}{2}\mathbb{E}[A]\mathbb{E}[B]]$. In particular, this means that less than a $2^{-\varepsilon d/2}$-fraction of the entries in $A \circ B$ are 0 (by the assumption that neither $\mathbb{E}[A]$ nor $\mathbb{E}[B]$ are 0).

Since each entry in $A \circ B$ is just a positive scaling of the corresponding entry in AB, we get that $\mathbb{E}[AB] \geq 1 - 2^{-\varepsilon d/2}$. Comparing $\mathbb{E}[AB]$ and $\mathbb{E}[C]$, the Pigeonhole Principle implies that AB and C must have a nonzero entry in common, which finishes the proof.

Of course, in general, A and B are arbitrary matrices, which might not satisfy any nice regularity properties. The authors in [AFK+23] circumvent this issue by proving that it is possible to *simultaneously decompose* A and B into matrices that satisfy strong-enough regularity properties, and to do so in an efficient manner. Thus, our task for the remainder of Section 4 is to explain how it is possible to execute this decomposition in a way that yields super-polylogarithmic savings for Triangle Detection.

We defer discussion of the *full* decomposition of A and B to Section 4.7. To streamline the presentation, in Sections 4.4 through 4.6 we review the analogous decomposition of a *single* bipartite graph A, which we formally state now:

Theorem 4.2 (Theorem 3.3 in [AFK+23])**.** *Let $A \in \{0,1\}^{X \times Y}$, $\varepsilon \in (0,1)$, and $d \geq 1$. Then there is an algorithm* $\textsc{ADecomposition}(X, Y, A, \varepsilon, d)$ *computing tuples $\{(X_\ell, Y_\ell, A_\ell)\}_{\ell=1}^{L}$ with $X_\ell \subseteq X$, $Y_\ell \subseteq Y$, and $A_\ell \in \{0,1\}^{X_\ell \times Y_\ell}$ such that:*

(1) $A = \sum_{\ell=1}^{L} A_\ell$.

(2) For all $\ell \in [L]$, either

(i) $\mathbb{E}[A_\ell] \leq 2^{-d}$, *or*

(ii) A_ℓ is $(\varepsilon, 2, d)$-regular and ε-min-degree.

(3) $\sum_{\ell=1}^{L} |X_\ell||Y_\ell| \leq (d+2)|X||Y|$.

(4) $L \leq \exp(d^3 \mathrm{poly}(\varepsilon^{-1})$ *and* $\min_\ell |X_\ell||Y_\ell| \geq \exp(-d^3\mathrm{poly}(\varepsilon^{-1}))|X||Y|$.

Moreover, this algorithm is deterministic and runs in time $n^2\exp(d^3\mathrm{poly}(\varepsilon^{-1}))$ (where here $n = |X| + |Y|$).

Focusing on properties (1) and (2) for now, we will intuitively seek to decompose A into disjoint parts, each of which is either sparse or satisfies the conditions of our workhorse Theorem 4.1. Just like in the "easy case" explored above, eventually (in the full decomposition of Section 4.7) in the sparse case we will be able to use brute force, and in the regular case we will exploit uniformity. Thus the next two sections are devoted to explaining how we find large enough subgraphs that are either sparse or *both* ε-min-degree (Section 4.4) and $(\varepsilon, 2, d)$-regular (Section 4.5).

4.4 Enforcing ε-min-degree

We start with the algorithm that will enable us to obtain a large ε-min-degree subgraph from a given graph $A \in \{0,1\}^{X \times Y}$. The key idea behind both this algorithm and the one for enforcing regularity (Section 4.5) is that we will find a large subgraph of $A' \subseteq A$ that either (a) directly satisfies our desired property (whether ε-min-degree or (ε, k, ℓ)-regularity), or otherwise (b) has a considerably increased density, i.e. a *density increment*. The density increment case does not ultimately pose a problem for achieving the desired property because — as we will see more formally when applying these subroutines in Section 4.6 — the number of times such an increment can occur is limited due to bounds on the density. This crucial fact means that we can simply recurse on the higher-density subgraph A' until we inevitably exit in case (a) as desired. We will see that the density increments are indeed sufficient to bound the depth of this recursion and ensure that the final subgraph is still large enough for our purposes.

We first precisely state the result that we can find a large subgraph that is either ε-min-degree or a density increment via (one call to) such an algorithm.

Lemma 4. *(Lemma 5.1 in [AFK$^+$23]) Let $A \in \{0,1\}^{X \times Y}$ and let $0 < \varepsilon, \gamma < 1$. There is a deterministic $O(|X||Y|)$-time algorithm* MinDegree$(X, Y, A, \varepsilon, \gamma)$ *that computes a subset $X' \subset X$ (inducing the subgraph $A' = A[X', Y]$) such that $|X'| \geq \lfloor (1-\gamma)|X| \rfloor$, and either*

(1) A' is ε-min-degree and $\mathbb{E}[A'] \geq \mathbb{E}[A]$, or

(2) $\mathbb{E}[A'] \geq (1+\gamma\varepsilon)\mathbb{E}[A]$ (a density increment).

The MinDegree algorithm that accomplishes this task is presented in Algorithm 2, slightly expanded from the version in [AFK$^+$23] (Algorithm 5.1) to include some additional details for concreteness.[4] In words, given $A \in \{0,1\}^{X \times Y}$, the algorithm iteratively computes a subset $X' \subseteq X$ from X (inducing the subgraph $A' = A[X', Y]$) by iteratively removing vertices x from X' in increasing order of $\deg_A(x)$. The algorithm terminates when the size of the current X' is no larger than $(1-\gamma)|X|$ or when the next x (in the sorted order) satisfies $\deg_{A'}(x) \geq (1-\varepsilon)\mathbb{E}[A']$, at which point we return the set X'.

Algorithm 2 Find ε-min-degree or density increment subgraph

```
1: procedure MinDegree($X, Y, A, \varepsilon, \gamma$)
2:     Let $X' \leftarrow X$, $A' \leftarrow A$.
3:     Precompute $\deg_A(x)$ for each $x \in X'$ and sort $X'$ in increasing order of $\deg_A(x)$
4:     Let $x \leftarrow$ head (first element) of $X'$
5:     while $|X'| > (1-\gamma)|X|$ and $\deg_{A'}(x) < (1-\varepsilon)\mathbb{E}[A']$ do
6:         Update $X' \leftarrow X' \setminus \{x\}$, $A' \leftarrow A[X', Y]$
7:         $x \leftarrow$ head of $X'$
8:     end while
9:     return $X'$
10: end procedure
```

The core reasons behind this algorithm's correctness come from the definitions of density $\mathbb{E}[A]$, (relative) degree $\deg_A(x)$, and how these quantities are related. We distill this into three key facts that provide the foundation for our proof of the lemma.

Fact 1. For any $A' = A[X', Y]$ induced by some $X' \subset X$, $\deg_{A'}(x) = \deg_A(x)$, for any $x \in X'$.

Fact 2. $\mathbb{E}[A'] = \frac{1}{|X'|}\sum_{x \in X'} \deg_{A'}(x)$. Hence removing some x from X' s.t. $\deg_{A'}(x) < \mathbb{E}[A']$ will cause an *increase* in $\mathbb{E}[A']$.

Fact 3. For $X = X_1 \sqcup X_2$ and $A_1 := A[X_1, Y]$, $A_2 := A[X_2, Y]$: $\mathbb{E}[A] = \frac{|X_1|}{|X|}\mathbb{E}[A_1] + \frac{|X_2|}{|X|}\mathbb{E}[A_2]$.

[4]The two presentations are equivalent because to check if there exists some node $x \in X'$ with $\deg_{A'}(x)$ smaller than some threshold (as in [AFK$^+$23]), it suffices to check the node with the smallest $\deg_A(x)$ value (as in our algorithm), since $\deg_A(x) = \deg_{A'}(x)$ (see our "Fact 1"). Indeed the runtime analysis in [AFK$^+$23] relies on implementing our presentation of the algorithm.

Fact 1 holds due to the definition $\deg_A(x) = \frac{1}{|Y|}\sum_{y\in Y} A(x,y)$, since the set Y does not change for the subgraph (submatrix) $A' = A[X', Y]$. Fact 2 characterizes the density of A as an average over $x \in X$ of the degrees $\deg_A(x)$ (and then applying this to A'), which follows from the definitions of density and degree; the second part of Fact 2 in turn follows from this characterization. Finally, Fact 3 states that for a partition $A = A_1 \sqcup A_2$, the density of $\mathbb{E}[A]$ can be written as the size-weighted average of the densities $\mathbb{E}[A_1]$ and $\mathbb{E}[A_2]$; this can be derived from Fact 1 and 2 or directly from the definition of density. We will see the utility of these facts in the proof below.

Proof of Lemma. **Runtime:** It is easy to see that we can precompute $\deg_A(x)$ for each $x \in X'$ (where X' is initialized to X) and then sort X' overall in time $O(|X||Y|)$. Then, observe that the loop repeats at most $O(|X|)$ times since it removes a vertex from X' every time. Furthermore, it takes constant work in each iteration to check $\deg_{A'}(x) = \deg_A(x) < (1-\varepsilon)\mathbb{E}[A']$ (using Fact 1 and updating $\mathbb{E}[A']$ using the first part of Fact 2) and to remove x from X' in the body of the loop (if we store X' with an appropriate data structure e.g. queue after sorting). Thus the overall runtime of MinDegree is $O(|X||Y|)$.

Correctness: First observe that the returned X' always satisfies $|X'| \geq \lfloor (1-\gamma)|X| \rfloor$: either the loop condition $|X'| > (1-\gamma)|X|$ always holds, or the loop terminates in the first iteration where $|X'| \leq (1-\gamma)|X|$, in which case the claim $|X'| \geq \lfloor (1-\gamma)|X| \rfloor$ follows from noticing that $|X'|$ only decreases by 1 in every iteration of the loop.

We now consider the two cases in which we terminate the loop. We complete the proof by showing that in Case 1 below, we satisfy Case 1 of the lemma (ε-min-degree and $\mathbb{E}[A'] \geq \mathbb{E}[A]$), and similarly in Case 2 below we satisfy Case 2 of the lemma (density increment).

Case 1: $\underline{deg_{A'}(x) \geq (1-\varepsilon)\mathbb{E}[A']}$. Observe that this termination condition directly implies A' is ε-min-degree by definition, where we use Fact 1 and the fact that we sort X' in increasing order of $\deg_A(x) = \deg_{A'}(x)$ (so *all* remaining $x \in X'$ satisfy $deg_{A'}(x) \geq (1-\varepsilon)\mathbb{E}[A']$). Meanwhile, we can also conclude $\mathbb{E}[A'] \geq \mathbb{E}[A]$ by repeatedly applying the second part of Fact 2 to each step where we update X' by removal.

Case 2: $\underline{|X'| \leq (1-\gamma)|X|}$. Letting $\hat{X} = X \setminus X'$, this condition implies that $|\hat{X}| \geq \gamma|X|$. The intuition for this case is that we have removed sufficiently many low-degree vertices (which form $\hat{X}$) to guarantee, applying Fact 3, that the remaining part X' must have a higher-than-average density.

Formally, let $\hat{A} = A[\hat{X}, Y]$. Fact 2 implies that $\mathbb{E}[A']$ only increases throughout the algorithm, and hence any x removed in the algorithm satisfies $\deg_{\hat{A}}(x) = \deg_{A'}(x) < (1-\varepsilon)\mathbb{E}[A']$ for the final A' (note that we have used Fact 1 here). Thus, applying Fact 2, $\mathbb{E}[\hat{A}] = \frac{1}{|\hat{X}|}\sum_{x\in\hat{X}} \deg_{\hat{A}}(x) < (1-\varepsilon)\mathbb{E}[A']$, and so

$$\begin{aligned}
\mathbb{E}[A] &= \frac{|X'|}{|X|}\mathbb{E}[A'] + \frac{|\hat{X}|}{|X|}\mathbb{E}[\hat{A}] && \text{by Fact 3}\\
&\leq \frac{|X'|}{|X|}\mathbb{E}[A'] + \frac{|\hat{X}|}{|X|}(1-\varepsilon)\mathbb{E}[A'] \\
&= \mathbb{E}[A'](1-\varepsilon\frac{|\hat{X}|}{|X|}) && \text{since } \frac{|X'|}{|X|} + \frac{|\hat{X}|}{|X|} = 1\\
&\leq \mathbb{E}[A'](1-\gamma\varepsilon)\\
&\Rightarrow \mathbb{E}[A'] \geq (1+\gamma\varepsilon)\mathbb{E}[A] && \text{by dividing by } (1-\gamma\varepsilon)
\end{aligned}$$

where the last statement holds because $0 < \gamma\varepsilon < 1$ and so $\frac{1}{1-\gamma\varepsilon} > 1 + \gamma\varepsilon$. ■

4.5 Enforcing regularity via "sifting"

In order to enforce regularity in the decomposition process, we rely on a "sifting" algorithm to either certify that a given graph is regular, or else find a denser subgraph to recurse upon to find a regular subgraph. As in Section 4.4, this density increment allows us to bound the number or recursions necessary to achieve regularity. The authors of [KLM23] give a proof of existence for the sifting theorem that yields a randomized algorithm for sifting, but in order to derandomize the sifting algorithm, [AFK$^+$23] uses a different approach. Formally, their main result is as follows.

Theorem 4.3 (Sifting, Theorem 3.2 in [AFK$^+$23])**.** *Let $A \in \{0,1\}^{X\times Y}$, let $\varepsilon \in (0,1)$ and $k, \ell \geq 1$. There is an algorithm* $\textsc{Sift}(X, Y, A, \varepsilon, k, \ell)$ *that returns either*

a. "regular," in which case A is (ε, k, ℓ)-regular, or

b. sets $X' \subseteq X, Y' \subseteq Y$ with $|X'||Y'| \geq \frac{\varepsilon}{16} \cdot \mathbb{E}[A]^{k\ell} \cdot |X||Y|$ and $\mathbb{E}[A[X', Y']] \geq (1 + \frac{\varepsilon}{2})\mathbb{E}[A]$.

The algorithm is determinstic and runs in time $n^2 \cdot (\varepsilon\mathbb{E}[A]/k)^{-O(k\ell(k+\ell))}$, where $n = |X| + |Y|$.

The role that the sets X', Y' play in this theorem are analogous to the "witnesses" of irregularity seen in the notion of ε-regularity used in Szemerédi's regularity lemma; indeed, if A is not (ε, k, ℓ)-regular, then we are guaranteed to have sets X', Y' with unusually high density, and the theorem promises an algorithm to find these sets. Similarly to Szemerédi's regularity decomposition, the witnesses in the sifting algorithm are used to refine the search for a regular subgraph. Note, however, that the algorithm may return some "false negatives"; that is, even if A is (ε, k, ℓ)-regular, the algorithm may still find and return X', Y' with a density increment. This is not an issue, though, since the density increment still guarantees that regularity can be enforced within a limited number of recursions.

The main tool that [AFK$^+$23] uses to achieve the algorithm in Theorem 4.3 is the following recursive sifting lemma that, given an irregular graph, either gives (a) evidence of irregularity in the form of a large subset of high-degree vertices, or provides (b) a subgraph that demonstrates "stronger" irregularity. The first case allows the algorithm to report X', Y' that satisfy Theorem 4.3; the second case gives the algorithm a subgraph in which it can continue searching for X', Y'. In the lemma and following discussion, we define $Y_x := \{y \in Y \,:\, A(x, y) = 1\}$ and $A_x = A[X, Y_x]$.

Lemma 5 (Recursive Sifting, Lemma 4.2 in [AFK$^+$23])**.** *Let $A \in \{0,1\}^{X\times Y}$, let $\delta, \varepsilon > 0$ and $k, \ell \geq 1$ and assume that $\|A\|_{U(k,\ell)} \geq (1+\varepsilon)\delta$. Then one of the following two cases applies:*

a. $\big|\{x \in X \,:\, \deg_A(x) \geq \delta\}\big| \geq \frac{\varepsilon}{2} \cdot \delta^{k\ell} \cdot |X|$, or

b. $k > 1$ and there is some $x \in X$ such that:

- *$\deg_A(x) \geq \delta^k$, and*
- *$\|A_x\|_{U(k-1,\ell)} \geq (1+\varepsilon)\delta$.*

The irregularity in the subgraph A_x promised by the lemma is stronger in the sense that it demonstrates the same irregularity as A, except with respect to the $U(k-1, \ell)$ grid norm rather than the $U(k, \ell)$ grid norm, which is strictly harder (see Lemma 3). When $k = 1$, the lemma guarantees that case (a) applies. Note that, just as in Theorem 4.3, neither case actually proves irregularity.

This lemma is proved in [AFK$^+$23] by showing that for $k > 1$, if case (a) does not apply, then a randomly chosen vertex in X satisfies the properties of x in case (b) with nonzero probability. For the sake of brevity, we omit the details here, but the lemma is almost immediate from the definition of the grid norm. The parameter δ in Lemma 5 is commonly just $\mathbb{E}[A]$, possibly scaled by some factor; we use δ for the sake of generality and presentation. We will discuss the utility of this lemma more in the proof of Theorem 4.3.

To make use of Lemma 5, we need one final tool to approximately calculate the grid norm of A_x for vertices $x \in X$, given by Lemma 6. If case (b) of Lemma 5 applies, this grid approximation tool will allow us to algorithmically identify an $x \in X$ that (approximately) satisfies the promised properties. The main complexity in derandomizing the sifting algorithm lies in making this grid norm approximation algorithm deterministic. We defer details of Lemma 6 to Section 4.5.1.

Lemma 6 (Deterministic regularity approximation, Lemma 4.3 in [AFK$^+$23]). *Let $A \in \{0,1\}^{X \times Y}$, let $\alpha > 0$ and $k, \ell \geq 1$. There is a determinstic algorithm that computes, for all $x \in X$, an approximation v_x satisfying that $v_x = \|A_x\|_{U(k,\ell)} \pm (\alpha / \deg_A(x)^{\frac{1}{k}})$, and runs in time $n^2 \cdot \alpha^{-O(k\ell(k+\ell))}$, where $n = |X| + |Y|$.*

We now turn to the proof of Theorem 4.3.

Proof. (Theorem 4.3) We will show that the Sift procedure in algorithm 3 accomplishes the task of

Algorithm 3 Sift a graph (Algorithm 4.1 in [AFK$^+$23])

```
1: procedure Sift'($X, Y, A, \delta, \varepsilon, k, \ell$)
2:     Let $X' = \{x \in X : \deg_A(x) \geq \delta\}$.
3:     if $|X'| \geq \frac{\varepsilon}{2} \cdot \delta^{k\ell} \cdot |X|$ then
4:         return $X', Y$
5:     end if
6:     if $k = 1$ then
7:         return "regular"
8:     else
9:         Compute approximations $v_x$ of $\|A_x\|_{U(k-1,\ell)}$ by Lemma 6 with $\alpha = \frac{\varepsilon\delta^2}{2k^2}$
10:        Select $x \in X$ with $\deg_A(x) \geq \delta^k$ maximizing $v_x$
11:        return Sift'($X, Y_x, A_x, \delta, \varepsilon \cdot (1 - \frac{1}{k^2}), k-1, \ell$)
12:    end if
13: end procedure
14:
15: procedure Sift($X, Y, A, \varepsilon, k, l$)
16:    return Sift'($X, Y, A, (1 + \frac{\varepsilon}{2})\mathbb{E}[A], \frac{\varepsilon}{4}, k, \ell$)
17: end procedure
```

Theorem 4.3 in the desired time complexity. Without loss of generality, we can assume that $k \leq \ell$; otherwise we could analyze $A^\top$ and swap X, Y and k, ℓ. The procedure Sift is simply a wrapper for the Sift'($X, Y, A, \delta, \varepsilon, k, \ell$) routine, which accomplishes the task to

a. either return "regular," in which case $\|A\|_{U(k,\ell)} \leq (1+\varepsilon)\delta$, or

b. return $X' \subseteq X, Y' \subseteq Y$ such that $|X'||Y'| \geq \frac{\varepsilon}{4} \cdot \delta^{k\ell} \cdot |X||Y|$ and $\mathbb{E}[A[X', Y']] \geq \delta$.

Henceforth we only consider $\delta \leq 1$, since otherwise the task is trivial; we can always return "regular" using the trivial bound $\|A\|_{U(k,\ell)} \leq 1 \leq (1+\varepsilon)\delta$.

Calling Sift' with the parameters $\delta' \leftarrow (1+\frac{\varepsilon}{2})\mathbb{E}[A]$ and $\varepsilon' \leftarrow \frac{\varepsilon}{2}$, then, solves the task of Sift, since $\delta' \geq \mathbb{E}[A]$ and $(1+\varepsilon')\delta' \leq (1+\frac{\varepsilon}{4})(1+\frac{\varepsilon}{2})\mathbb{E}[A] \leq (1+\varepsilon)\mathbb{E}[A]$ for $\varepsilon \leq 1$. Thus for correctness it suffices to show that the procedure Sift' is implemented correctly. The Sift' procedure first attempts to identify a large set of high-degree vertices X'; if successful, then the algorithm can simply return X', Y to satisfy Theorem 4.3. Otherwise, the procedure applies Lemma 5: if $k = 1$, then the graph must be regular; otherwise, we can continue searching for X', Y' in the most irregular subgraph $A_x = A[X, Y_x]$ that is not too small (i.e. $\deg_A(x) \geq \delta^k$). Case b guarantees that we will be able to find such an irregular subgraph. Each time we recurse, the size of Y may decrease by a factor of δ^k since we restrict the graph to Y_x where $|Y_x|/|Y| = \deg_A(x) \geq \delta^k$, but the required size of X' also increases by a factor of $\delta^{-\ell} \geq \delta^{-k}$. This prevents the size of the subgraph $A[X', Y']$ from becoming too small.

Correctness: We prove the following claims:

i. if the algorithm reports X', Y', then $|X'||Y'| \geq \frac{\varepsilon}{4} \cdot \delta^{k\ell} \cdot |X||Y|$ and $\mathbb{E}[A[X', Y']] \geq \delta$, and

ii. if the graph is irregular, then the graph reports some X', Y' rather than reporting "regular." (Note that it does not matter what the algorithm reports if the graph is actually regular.)

Together these two claims prove the correctness of algorithm 3. For the first claim, we only return X', Y' on line 4 of the program, and then $\deg_A(x) \geq \delta$ for all $x \in X'$ (the parameter δ is the same in all recursions), so it follows that $\mathbb{E}[A[X', Y']] \geq \delta$. We prove claim (i.) by induction on the claim

$$|X'||Y'| \geq \frac{\varepsilon}{2} \cdot \prod_{i=2}^{k} \left(1 - \frac{1}{i^2}\right) \cdot \delta^{k\ell} \cdot |X||Y|,$$

over $k \geq 1$, where X', Y' are the returned sets of Sift'$(X, Y, A, \delta, \varepsilon, k, \ell)$ (note that $\prod_{i=2}^{k}(1 - \frac{1}{i^2}) = \frac{k+1}{2k} \geq \frac{1}{2}$. If the procedure returns X', Y' on line 4 with $|X'| \geq \frac{\varepsilon}{2} \cdot \delta^{k\ell} \cdot |X|$ and $Y' = Y$, then $|X'||Y'| \geq \frac{\varepsilon}{2} \cdot \delta^{k\ell}|X||Y|$, as desired (in fact this holds when returning immediately for any k). In the base case $k = 1$, X', Y' can only be obtained from line 4, so the base case holds. Suppose that the inductive hypothesis holds for some $k-1 \geq 1$, and consider the X', Y' returned for parameter k. The sets X', Y' are either obtained from line 4, in which case the claim holds, or they are obtained from line 11, i.e., Sift'$(X, Y_x, A_x, \delta, \varepsilon \cdot (1 - \frac{1}{k^2}), k-1, \ell)$. In the latter case,

$$|X'||Y'| \geq \frac{\varepsilon}{2}\left(1 - \frac{1}{k^2}\right) \cdot \prod_{i=2}^{k-1}\left(1 - \frac{1}{i^2}\right) \cdot \delta^{(k-1)l} \cdot |X||Y_x| \geq \frac{\varepsilon}{2} \cdot \prod_{i=2}^{k}\left(1 - \frac{1}{i^2}\right) \cdot \delta^{k\ell} \cdot |X||Y|,$$

using the fact that $|Y_x| \geq \delta^k |Y|$, $\ell \leq k$, and $\delta \leq 1$.

Now to prove claim (ii.), suppose that the graph A is irregular, i.e., $\|A\|_{U(k,\ell)} \geq (1+\varepsilon)\delta$. If $k = 1$, then Lemma 5 guarantees that line 4 executes. Otherwise, if $k > 1$, Lemma 5 guarantees that either line 4 is executed, or case (b) 5 applies and there exists some $x \in X$ such that $\deg_A(x) \geq \delta^k$ and $\|A_x\|_{U(k-1,\ell)} \geq (1+\varepsilon)\delta$. We choose x such that $\deg_A(x) \geq \delta^k$ and $\|A_x\|_{U(k-1,\ell)}$ is maximized with error $\alpha / \deg_A(x)^{\frac{1}{k}} \leq \alpha/\delta = \frac{\varepsilon\delta}{2k^2}$. Thus $\|A_x\|_{U(k-1,\ell)} \geq (1+\varepsilon)\delta - \frac{\varepsilon\delta}{k^2} = \big(1 + \varepsilon \cdot \big(1 - \frac{1}{k^2}\big)\big)$. Since we recurse with $\varepsilon' = \varepsilon \cdot (1 - \frac{1}{k^2})$, the recursive call still deals with an irregular graph, so we can apply induction to argue that the algorithm must return some X', Y'.

Runtime: The runtime of the algorithm is straightforward due to the k-step simple recursion scheme. It is mainly bottlenecked by the complexity of the grid approximation algorithm. With at most k levels of recursion, and each recursion taking time $n^2(\alpha)^{-O(k\ell(k+\ell))} = n^2(k/\varepsilon\delta)^{O(k\ell(k+\ell))}$ to compute the grid approximations, the total running time is still $n^2(k/\varepsilon\delta)^{O(k\ell(k+\ell))}$, as the extra k factor gets absorbed into the $(k/\varepsilon\delta)^{O(k\ell(k+\ell))}$ factor. All other tasks in each recursion layer can be performed in $O(n^2)$ time. ■

4.5.1 Derandomizing sifting via deterministic regularity approximation

We now turn to proving Lemma 6: recall that (efficiently) derandomizing the approximation of the grid norm $\|A_x\|_{U(k,\ell)}$ is required in order for our final algorithm (Theorem 1.1) to be deterministic as desired. In this section we overview the use of *oblivious samplers* to obtain Lemma 6. We will highlight the key ideas and intuition while omitting the technical details and full proofs, which can be found in Section 4.1 in [AFK+23] and existing literature on oblivious samplers (e.g. [GW94]).

We start by showing why a naïve deterministic algorithm to compute the grid norm does *not* suffice, motivating the need for more involved machinery like oblivious samplers (defined shortly). Recall that for $A_x = A[X, Y_x]$, we can write $\|A_x\|_{U(k,\ell)}^{k\ell} = \mathbb{E}_{x_1,\ldots,x_k \in X}[\mathbb{E}_{y \in Y_x}\Pi_{i\in[k]}A(x_i, y)]$. The direct approach for computing the expectations requires enumerating all k-tuples $x_1, \ldots, x_k \in X$ and all $y \in Y_x$ (n.b. assuming WLOG $k \leq \ell$, this formulation of the grid norm yields the best naïve approach). This takes time $|X|^k|Y_x|$, which can be as large as n^{k+1}. The Sift procedure will eventually be applied in the decomposition (Section 4.6) with $k = 2$ and $\ell = d = \Theta(\sqrt[7]{\log n})$ indeed greater than k, so the naïve approach would take $n^{k+1} = n^3$ time — already failing our attempt to obtain a subcubic algorithm. Instead, by using oblivious samplers to approximate the norm in $n^2 \cdot \alpha^{-O(k\ell(k+\ell))}$ time, we will ultimately obtain a Sift running faster in $n^2 \cdot \exp\big(d^3\text{poly}(\varepsilon^{-1})\big)$ time based on our choice of α in the proof of Theorem 4.3 and setting $k = 2$ and $\ell = d$.

We now formally define oblivious samplers. Informally, the next lemma says the following: given a set X, we can efficiently find a not-too-large family $\mathcal{S}$ of subsets $S \subseteq X$, each not too large, such that the expected result from applying functions to a random element drawn from X does not change significantly if we instead randomly draw the element from a randomly chosen subset $S \in \mathcal{S}$.

Lemma 7 (Oblivious Sampling: [GW94], Lemma 4.4. in [AFK+23])**.** *Let X be a set and let $\delta, \varepsilon > 0$. There is a deterministic algorithm computing, in time $|X| \cdot \operatorname{poly}\left(\varepsilon^{-1}, \delta^{-1}, \log |X|\right)$, a family $\mathcal{S}$ of subsets $S \subseteq X$ such that*

a. $|S| \leq |X| \cdot \operatorname{poly}\left(\varepsilon^{-1}, \delta^{-1}\right)$,

b. $|\mathcal{S}| \leq \operatorname{poly}\left(\varepsilon^{-1}, \delta^{-1}\right)$ *for all* $S \in S$,

c. For every function $f : X \to [0,1]$,

$$\mathop{\mathbb{P}}_{S\in\mathcal{S}}[\mathop{\mathbb{E}}_{x\in X} f(x) = \mathop{\mathbb{E}}_{x\in S} f(x) \pm \varepsilon] \geq 1 - \delta.$$

We call S an (ε, δ)-oblivious sampler of X.

The utility of oblivious samplers comes from the next lemma, which states that for S, T drawn from oblivious samplers of X, Y respectively, the grid norm of any $A \in \{0,1\}^{X\times Y}$ can be approximated by the expected grid norm $A[S,T]$ up to a reasonably small additive error.

Lemma 8 (Lemma 4.6 in [AFK+23])**.** *Let $A \in \{0,1\}^{X\times Y}$, let $\delta, e > 0$ and $k, \ell \geq 1$, and let $\mathcal{S}, \mathcal{T}$ be (ε, δ)-oblivious samplers of X and Y, respectively. Then*

$$\|A\|_{U(k,\ell)}^{k\ell} = \mathop{\mathbb{E}}_{\substack{S\in\mathcal{S}\\ T\in\mathcal{T}}} \|A[S,T]\|_{U(k,\ell)}^{k\ell} \pm (2ek + 2e\ell + 2\delta).$$

We omit the full proof of this lemma but sketch the following main ideas. By defining $f(x) = \prod_{j\in[\ell]} A(x, y_j)$ for fixed $\vec{y}$, we can bound the probability of $(\mathbb{E}_{x\in S} f(x))^k$ deviating from $(\mathbb{E}_{x\in X} f(x))^k$ using property (c) of oblivious sampling. We can then turn this into a (nonrandom) bound (nonrandom) bound on the expectation over drawing $S \in \mathcal{S}$ using some probability bounding rules. Recalling the definition of grid norm as an expectation of the product captured by $f(x)$, we eventually obtain the desired bound between $\|A\|_{U(k,\ell)}^{k\ell}$ and $\mathbb{E}_{S\in Sc, T\in Tc} \|A[S,T]\|_{U(k,\ell)}^{k\ell}$ by taking some additional expectations (over drawing $y_1, ..., y_\ell \in Y$ and repeating this process with drawing $T \in \mathcal{T}$).

Based on Lemma 8, we can finally satisfy Lemma 6 with Algorithm 4.

Algorithm 4 Deterministic regularity approximation algorithm

```
1: procedure ApproximateNorm($X, Y, A, k, \ell, \alpha$)
2:     Let $\varepsilon = \delta = \alpha^{k\ell}/(2k + 2\ell + 2) = \alpha^{O(k\ell)}$.
3:     Precompute $(\varepsilon, \delta)$-oblivious samplers $\mathcal{S}, \mathcal{T}$ of $X, Y$ respectively
4:     Enumerate each $S \in \mathcal{S}, T \in \mathcal{T}$ and each tuple $(x_1, ..., x_k) \in S^k$, $(y_1, ..., y_\ell) \in T^\ell$ to compute
           $$u_{T,y_1,...,y_\ell} \leftarrow \mathop{\mathbb{E}}_{S\in\mathcal{S}} \mathop{\mathbb{E}}_{x_1,...,x_k\in S} \prod_{\substack{i\in[k]\\ j\in[\ell]}} A(x_i, y_j)$$
5:     Enumerate each $T \in \mathcal{T}$, each $(y_1, ..., y_\ell)$ to compute, for all $x \in X$,
           $$u_x \leftarrow \mathop{\mathbb{E}}_{T\in\mathcal{T}} \mathop{\mathbb{E}}_{y_1,...,y_\ell\in S} [u_{T,y_1,...,y_\ell} \prod_{j\in[\ell]} A(x, y_j)]$$
6:     return $v_x \leftarrow u_x / \deg_A(x)^{\frac{1}{k}}$, for all $x \in X$
7: end procedure
```

The details for correctness and running time of this algorithm can be found in the proof of Lemma 4.3 in [AFK+23]. For correctness, the key idea is that the expectations over drawing $\vec{x}$ and $\vec{y}$ from S and T give us the grid norm $\|A_x[S,T]\|_{U(k,\ell)}^{k\ell}$ up to some scaling involving $\deg_A(x)$, and then we can apply Lemma 8 to bound the deviation from the desired norm $\|A_x\|_{U(k,\ell)}$. Our choice of ε, δ will then yield the desired additive bound in terms of α.

For the runtime, the most expensive operation ends up being computing $u_{T,y_1,\dots,y_\ell}$ — as we might intuitively expect, since this involves enumerating each $S \in \mathcal{S}$ and each $T \in \mathcal{T}$, then enumerating each k-tuple from S and each ℓ-tuple from T, and finally computing the product of $k\ell$ entries. This requires $O(\sum_{S\in\mathcal{S},T\in\mathcal{T}} |S|^k|T|^\ell \cdot k\ell) = n^2\text{poly}(\varepsilon^{-1},\delta^{-1})^{(k+\ell)}$ time by applying properties (a) and (b) of oblivious samplers (Lemma 7). Then, once the overall runtime is bounded by $n^2\text{poly}(\varepsilon^{-1},\delta^{-1})^{(k+\ell)}$, our choice of $\varepsilon = \delta = \alpha^{O(k,\ell)}$ guarantees the desired $n^2 \cdot \alpha^{O(-k\ell(k+\ell))}$ runtime.

4.6 Decomposing a single bipartite graph

We are now ready to apply our key tools of Lemma 4 (MinDegree) and Theorem 4.3 (Sift) as subroutines for our goal in Theorem 4.2: decomposing a single bipartite graph $A \in \{0,1\}^{X\times Y}$ into components that are *both* ε-min-degree and $(\varepsilon, 2, d)$-regular. As a first step, we develop algorithm GoodRect to find a *single* component (subgraph) satisfying these conditions along with additional properties of being at least as dense as the starting graph and sufficiently large. We use the same terminology as [AFK$^+$23] in referring to this as a "good rectangle" (recall the matrix-graph analogy).

To motivate how we will apply Lemma 4 and Theorem 4.3, we begin by formalizing our earlier claim that the density increment cases in these subroutines do not pose a problem due to a bounded recursion guarantee. The key is that we will execute the GoodRect algorithm *only when* A is sufficiently dense ($\mathbb{E}[A] > 2^{-d}$), because, as suggested by Theorem 4.2's statement and seen in the ideal case of Section 4.3, we will be able to separately handle the sparse case ($\mathbb{E}[A] \le 2^{-d}$). Notice that MinDegree (called with $\gamma = \frac{1}{2}$) and Sift both guarantee a factor $(1 + \varepsilon/2)$ increase over $\mathbb{E}[A]$ in the density increment cases, and furthermore recall that by definition the density can never exceed 1. Then if we recursively call our subroutines on the returned subgraphs whenever a density increment is encountered, the recursion depth will be at most $\log_{1+\varepsilon/2}(1/2^{-d}) = \frac{d}{\lg(1+\varepsilon/2)} = O(d/\varepsilon)$ — we will refer to this as observation $(\star)$ — at which point we will be guaranteed to exit the recursion with our desired ε-min-degree and regularity properties.

We formally state the result that we can efficiently find a good rectangle $X^* \times Y^* \subseteq X \times Y$ in Lemma 9 and present the simple GoodRect algorithm for doing so in Algorithm 5 below.

Lemma 9 (Finding a Good Rectangle: Lemma 5.2 in [AFK$^+$23])**.** *Let $A \in \{0,1\}^{X\times Y}$, let $\varepsilon \in (0,1)$, and $d \ge 1$, and assume $\mathbb{E}[A] \ge 2^{-d}$. There is a deterministic algorithm* GoodRect$(X, Y, A, \varepsilon, d)$ *running in time $n^2 \cdot exp(d^3poly(\varepsilon^{-1})$ (for $n = |X| + |Y|$) that computes $X^* \subseteq X, Y^* \subseteq Y$ s.t. for $A^* = A[X^*, Y^*]$,*

(1) A^* is ε-min degree and $(\varepsilon, 2, d)$-regular

(2) $\mathbb{E}[A^*] \ge \mathbb{E}[A]$

(3) $|X^*||Y^*| \ge \exp(-d^3poly(\varepsilon^{-1}))|X||Y|$

The GoodRect algorithm simply implements the process of calling MinDegree and Sift and recurring if we hit the density increment case in either subroutine, such that we ultimately exit in the alternate ε-min-degree and $(\varepsilon, 2, d)$-regular cases of Lemma 4 and Theorem 4.3 respectively. Observation $(\star)$ guarantees that we do eventually exit the recursion, and we will see this observation's importance in the following proofs of GoodRect's correctness and runtime as well.

Proof of Lemma 9. **Correctness:** Given that GoodRect does terminate $(\star)$, Property 1 is immediate: in the final call to GoodRect, both MinDegree and Sift must not return a density increment (or else GoodRect would recur), so by the cases in Lemma 4 and Theorem 4.3, MinDegree instead returns X' s.t. A' is ε-min-degree and Sift verifies that this A' is $(\varepsilon, 2, d)$-regular.

Property 2 is also straightforward: in a given call to GoodRect with graph A, either we recurse (lines 4 and 7) on a subgraph of A that is at least as dense as A (as we are in the density increment

Algorithm 5 (Algorithm 5.2 in [AFK+23])

```
procedure GoodRect($X, Y, A, \varepsilon, d$)
    Compute $X' \leftarrow$ MinDegree($X, Y, A, \varepsilon, \frac{1}{2}$), inducing $A' \leftarrow A[X', Y]$
    if $\mathbb{E}[A'] \geq (1 + \frac{\varepsilon}{2})\mathbb{E}[A]$ then
        return GoodRect($X', Y, A', \varepsilon, d$)
    end if
    if Sift($X', Y, A', \varepsilon, 2, d$) returns a denser rectangle $X'' \times Y'' \subset X' \times Y'$ then
        return GoodRect($X'', Y'', A[X'', Y''], \varepsilon, d$)
    end if
    return $X', Y$
end procedure
```

of MinDegree or Sift), or we return the subgraph A' returned by MinDegree, for which Lemma 4 guarantees $\mathbb{E}[A'] \geq \mathbb{E}[A]$.

It remains to lower bound the size $|X^*||Y^*|$ to prove Property 3. Intuitively, the concern is that size of the the (sub)graph A we are searching decreases in each recursive call to GoodRect, but we have the following three guarantees:

a. The recursion depth is upper bounded by $D := \log_{1+\varepsilon/2}(2^d)$ $(\star)$.

b. In the final GoodRect(X, Y, A, ε, d) call to prior to termination, we return X', Y s.t. $|X'||Y| \geq \lfloor \frac{1}{2}|X| \rfloor |Y| \geq \frac{1}{3}|X||Y|$, since MinDegree returns X' s.t. $|X'| \geq \lfloor \frac{1}{2}|X| \rfloor$ and Y does not change. Let $F := \frac{1}{3}$ be this lower bound on the final size factor.

c. The input size does not decrease too much in any given recursive call. In particular, MinDegree still returns X' s.t. $|X'| \geq \lfloor \frac{1}{2}|X| \rfloor$, and in the density increment case Sift returns X', Y' s.t. $|X'||Y'| \geq \frac{\varepsilon}{16}\mathbb{E}[A]^{2d}|X||Y| \leq \lfloor \frac{1}{2}|X| \rfloor |Y|$ (Theorem 4.3). Thus the input to each recursive call is at least a $\frac{\varepsilon}{16}\mathbb{E}[A]^{k\ell}$-fraction of the parent call's input graph size (measured by $|X||Y|$). Let $C := \frac{\varepsilon}{16}\mathbb{E}[A]^{k\ell}$ be this lower bound on the recursive size factor.

Combining these three guarantees, it follows that our final good rectangle size $|X^*||Y^*|$ must be at least $F \cdot C^D |X||Y| = \frac{1}{3} \cdot (\frac{\varepsilon}{16}\mathbb{E}[A]^{2d})^{\log_{1+\varepsilon/2}(2^d)} \geq \frac{1}{3} \cdot (\frac{\varepsilon}{16} 2^{-2d^2})^{\frac{d}{\lg(1+\varepsilon/2)}} |X||Y| = \exp(-d^3 \text{poly}(\varepsilon^{-1}))|X||Y|$. For the first inequality we use the fact that $\mathbb{E}[A] \geq 2^{-d}$, and for the second we use $\lg(1 + a) \geq a$ for any $a \in (0, 1)$. This completes the proof of Property 3.

Runtime: By Lemma 4 a call to MinDegree takes $O(|X||Y|) = O(n^2)$ time, and by Theorem 4.3 a call to Sift takes $n^2 \cdot (\varepsilon\mathbb{E}[a])^{-O(d^2)} = n^2 \cdot \exp(-d^3 \text{poly}(\varepsilon^{-1}))$ time (where we are substituting $k = 2$ and $\ell = d$, and again noting $\mathbb{E}[A] \geq 2^{-d}$). So each execution of a call to GoodRect takes $n^2 \cdot \exp(-d^3 \text{poly}(\varepsilon^{-1}))$ (i.e. the Sift call dominates). Since the recursion depth is $O(d/\varepsilon)$ $(\star)$, the total running time for GoodRect is $O(d/\varepsilon) * n^2 \cdot \exp(-d^3 \text{poly}(\varepsilon^{-1}))$, which is asymptotically just $n^2 \cdot \exp(-d^3 \text{poly}(\varepsilon^{-1}))$. ■

We now proceed to prove Theorem 4.2. The promised decomposition algorithm is Algorithm 6.

Algorithm 6 (Algorithm 5.3 in [AFK+23])

```
procedure ADecomposition($X, Y, A, \varepsilon, d$)
    if $\mathbb{E}[A] \leq 2^{-d}$ then
        return $\{(X, Y, A)\}$
    end if
    Compute $X^*, Y^* \leftarrow$ GoodRect($X, Y, A, \varepsilon, d$)
    return $\{(X^*, Y^*, A[X^*, Y^*])\} \cup$ ADecomposition($X, Y, A - A[X^*, Y^*], \varepsilon, d$)
end procedure
```

Given that we've already developed the GoodRect subroutine, the algorithm is an extremely simple recursion: For the base case, if the input graph A is sparse, then we simply return the graph as is. Otherwise, we

(1) find a good rectangle (as described in Lemma 9)

(2) delete the edges in the subgraph induced by the good rectangle from the graph

(3) add this subgraph to the list of tuples we return

(4) recurse on the remainder of the graph (we will argue that the recursion depth is bounded later in this section).

We claim that this algorithm satisfies all the correctness and runtime properties specified in Theorem 4.2.

Proof. **Correctness of Properties 1 and 2.** These follow immediately from the properties of the return value of GoodRect, see Lemma 9.

Conceptually, this is already the most important part of the algorithm: we've successfully partitioned the graph into components that are either sparse, or regular in the sense of Theorem 4.1.

Thus, our only remaining task is to prove that properties (3) and (4) are met – that is, that the total size of the returned vertex parts is not too large, and that the recursion depth is suitably bounded.

Correctness of Property 3. We will prove the correctness of property (3) by induction.

The base case is when the algorithm is called on an input A that is sparse (that is, $\mathbb{E}[A] \leq 2^{-d}$). In this case, we return the trivial partition X, Y, which clearly satisfies $|X||Y| \leq (d+2)|X||Y|$.

For the inductive step, we will show the stronger statement that for any input X, Y, A, ε, d such that $\mathbb{E}[A] > 2^{-d}$, we have that the output of ADecomposition satisfies

$$\sum_{\ell=1}^{L} |X_\ell||Y_\ell| \leq (d + 2 - \log(\mathbb{E}[A]^{-1}))|X||Y|.$$

Thus, fix A with $\mathbb{E}[A] > 2^{-d}$. Note that in this case $\log(\mathbb{E}[A]^{-1}) \leq d \log 2$ and hence $d - \log(\mathbb{E}[A]^{-1}) \geq 0$.

Let $A^* = A[X^*, Y^*]$ be the subgraph induced by the vertex sets returned by GoodRect$(X, Y, A, \varepsilon, d)$.

There are two cases to consider. If $\mathbb{E}[A - A^*] \leq 2^{-d}$ (that is, the very next recursive call is the base case), then ADecomposition$(X, Y, A, \varepsilon, d)$ returns $\{(X^*, Y^*, A^*), (X, Y, A - A^*)\}$. We verify that

$$|X^*||Y^*| + |X||Y| \leq 2|X||Y| \leq (d + 2 - \log(\mathbb{E}[A]^{-1})|X||Y|.$$

The first inequality follows since X^*, Y^* are subsets of X, Y and the second inequality follows from the observation that $d - \log(\mathbb{E}[A]^{-1}) \geq 0$ above.

We now move on to the second case. Suppose that the call to ADecomposition$(X, Y, A - A^*, \varepsilon, d)$ returns a set of tuples $\{(X_\ell, Y_\ell, A_\ell)\}_{\ell=1}^{L}$. We need to prove that

$$|X^*||Y^*| + \sum_{\ell=1}^{L} |X_\ell||Y_\ell| \leq (d + 2 - \log(\mathbb{E}[A]^{-1}))|X||Y|.$$

We can now use the induction hypothesis to claim that

$$|X^*||Y^*| + \sum_{\ell=1}^{L} |X_\ell||Y_\ell| \leq |X^*||Y^*| + (d + 2 - \log(\mathbb{E}[A - A^*]^{-1}))|X||Y|.$$

Comparing the two inequalities above, by rearranging, it suffices to prove that

$$\frac{|X^*||Y^*|}{|X||Y|} \leq \log\left(\frac{\mathbb{E}[A]}{\mathbb{E}[A - A^*]}\right).$$

Note that by our definition of density, the number of edges that exist in $A - A^*$ is

$$|X||Y|\mathbb{E}[A] - |X^*||Y^*|\mathbb{E}[A^*].$$

Since the vertex parts of $A - A^*$ still have sizes X, Y, edge density of $A - A^*$ is

$$\mathbb{E}[A - A^*] = \frac{|X||Y|\mathbb{E}[A] - |X^*||Y^*|\mathbb{E}[A^*]}{|X||Y|}.$$

Plugging this value into our inequality above yields that it suffices to prove

$$\frac{|X^*||Y^*|}{|X||Y|} \leq -\log\left(1 - \frac{|X^*||Y^*|}{|X||Y|}\frac{\mathbb{E}[A^*]}{\mathbb{E}[A]}\right).$$

Now we use the crucial fact that by Lemma 9, $\mathbb{E}[A^*]/\mathbb{E}[A] \geq 1$. Let $p = |X^*||Y^*|/(|X||Y|)$ and let $q = \mathbb{E}[A^*]/\mathbb{E}[A]$. Then the inequality above is equivalent to

$$-p \geq \log(1 - pq) \iff e^{-p} \geq 1 - pq.$$

Of course, this last inequality holds since $q \geq 1$ and for all real p we have $e^{-p} \geq 1 - p$.

Correctness of Property 4. Note that we add exactly one tuple to our decomposition for each call of the ADecomposition subroutine, and thus the recursion depth is exactly L. Thus, it suffices to bound the number of recursive calls until the density of the graph drops below 2^{-d} (when we reach the base case and return).

Suppose Algorithm 6 is called with a graph A as input. By the formula for $\mathbb{E}[A - A^*]$ in the proof of correctness of Property 3 above, we have that

$$\mathbb{E}[A - A^*] = \mathbb{E}[A] - \frac{|X^*||Y^*|}{|X||Y|}\mathbb{E}[A^*] \leq \left(1 - \frac{|X^*||Y^*|}{|X||Y|}\right)\mathbb{E}[A],$$

where the inequality follows from $\mathbb{E}[A^*] \geq \mathbb{E}[A]$ (Lemma 9). Therefore, in each iteration, the density of the matrix passed as input to the algorithm decreases by a factor of $1 - \frac{|X^*||Y^*|}{|X||Y|}$.

We are in good shape because By Lemma 9, we can lower bound the cardinalities of $|X^*||Y^*|$. In particular, we have that

$$\frac{|X^*||Y^*|}{|X||Y|} \geq \exp\bigl(-d^3\text{poly}(\varepsilon^{-1})\bigr) = r.$$

It is easy to see (using the inequality $(1 - \alpha)^{1/\alpha} \leq e^{-1}$ for $\alpha \in (0, 1)$, for example), that after $d/r = \exp\bigl(-d^3\text{poly}(\varepsilon^{-1})\bigr)$ iterations, the density of the graph passed as input will have dropped below 2^{-d}, after which we return. This finishes the proof of the bound on the recursion depth.

Correctness of Runtime. Since the X, Y, ε, d inputs to the algorithm remain constant throughout execution, and we can compute $\mathbb{E}[A]$ in $O(n^2)$ time (which is dominated by the runtime of GoodRect), the runtime of the algorithm is bounded by the runtime of a single GoodRect call when fed a graph of size n as input, multiplied by L (the recursion depth). By Lemma 9 and the bound on L from Property 4 above, this quantity is asymptotically $n^2 \exp\bigl(d^3\text{poly}(\varepsilon^{-1})\bigr)$, as desired. ■

4.7 The full decomposition

Having established Theorem 4.2, we now turn to briefly discussing the *full* simultaneous decomposition of a tripartite graph. For completeness, we include the result here:

Theorem 4.4 (Theorem 3.1 in [AFK$^+$23])**.** *Let $A \in \{0,1\}^{X\times Y}$, $B \in \{0,1\}^{Y\times Z}$, with $\varepsilon \in (0,1)$ and $d \geq 1$. There is an algorithm* ABDecomposition$(X, Y, Z, A, B, \varepsilon, d)$ *that computes a collection of tuples $\{(X_k, Y_k, Z_k, A_k, B_k)\}_{k=1}^{K}$, where $X_k \subseteq X$, $Y_k \subseteq Y$, $Z_k \subseteq Z$, $A_k \in \{0,1\}^{X_k\times Y_k}$, $B_k \in \{0,1\}^{Y_k\times Z_k}$ such that*

(1) $AB = \sum_{k=1}^{K} A_k B_k$.

(2) For all $k \in [K]$, either

(i) $\mathbb{E}[A_k] \leq 2^{-d}$ or $\mathbb{E}[B_k] \leq 2^{-d}$, or

(ii) A_k and B_k^T are both $(\varepsilon, 2, d)$-regular and ε-min-degree.

(3) $\sum_{k=1}^{K} |X_k||Y_k||Z_k| \leq 2(d+2)^2|X||Y||Z|$.

(4) $K \leq \exp(d^7 \mathrm{poly}(\varepsilon^{-1}))$.

Moreover, the algorithm is deterministic and runs in time $n^2\exp(d^7\mathrm{poly}(\varepsilon^{-1}))$ (where here $n = |X| + |Y| + |Z|$).

As promised, the decomposition is in a very similar spirit to Theorem 4.2: this time we decompose the *pair* of bipartite graphs A, B into disjoint pieces A_k, B_k such that – for each k – either one of these pieces is sparse, or together they satisfy the conditions of Theorem 4.1.

Unfortunately, as the authors in [AFK$^+$23] themselves point out, achieving this full decomposition that satisfies the properties we want within the desired runtime is quite a bit more intricate than the decomposition of the single graph that we finished reviewing in Section 4.6, so we omit the details of the corresponding proofs here.

Very roughly, whereas in the single-graph decomposition we repeatedly found "good *rectangles*" (subgraphs of A induced by $X^* \subseteq X$ and $Y^* \subseteq Y$), our goal for the simultaneous decomposition of A and B is to find (again in the language of the paper) a good *cube* $Y^*, Z^*, \{(X_\ell, Y_\ell, A_\ell)\}_{\ell=1}^{L}$. We now describe our end goal in more detail.

Start with the graph B. We would like to find a good rectangle $Y^* \times Z^*$ in B that works well with a regularity decomposition of A.

We know from Theorem 4.2 that we can compute a decomposition $(X_\ell, Y_\ell, A_\ell)_{\ell=1}^{L}$ of $A[X, Y^*]$ such that each edge part is either sparse or regular.

However, this is not quite enough. Even though $Y^* \times Z^*$ gives a good rectangle in B, in order to set $B_k = B[Y_\ell, Z^*]$ (for the decomposition required by Theorem 4.4) we need this induced subgraph of $Y \times Z^*$ in B to be $(\varepsilon, 2, d)$-regular and ε-min-degree, which is not automatically guaranteed by $Y^* \times Z^*$ yielding a good rectangle (a general good rectangle $Y^* \times Z^*$ is completely oblivious to the decomposition of A).

Thus, it turns out that repeatedly computing such a good cube (and removing it to recurse on the remaining graph in a very similar way to Algorithm 6), we need an additional level of recursion that worsens the dependence on d, which goes from $\exp(d^3)$ to $\exp(d^7)$. For all the details that we overlook here, see Sections 3.3 and 5.2 of [AFK$^+$23].

4.8 A state-of-the-art combinatorial algorithm for triangle detection

With Theorem 4.4, we are now ready to prove the main result of [AFK$^+$23]: the existence of an efficient combinatorial algorithm for triangle detection. The main idea of the algorithm is to use Theorem 4.4 to reduce a graph to a collection of regular or sparse parts, at which point we can use the ideas of Section 4.3 to determine whether or not each part contains a triangle.

Theorem 4.5 (Triangle detection, Theorem 6.2 in [AFK⁺23])**.** *There is a deterministic combinatorial algorithm detecting whether or not a graph contains a triangle in time* $n^3/2^{\Omega(\sqrt[7]{\log n})}$.

Proof. Let $\varepsilon = \frac{1}{160}$ (so that Theorem 4.1 applies), and let $d \geq \frac{2}{\varepsilon}$ be a parameter to be determined later. Without loss of generality, we can assume that the input is a tripartite graph (X, Y, Z, A, B, C) due to Lemma 1. The triangle detection algorithm on this graph is as follows. Use Theorem 4.4 to obtain the regularity decomposition $\left\{(X_k, Y_k, Z_k, A_k, B_k\right\}_{k=1}^{K}$, and let $C_k = C[X_k, Z_k]$. Then for each part (A_k, B_k, C_k), check whether or not it contains a triangle:

a. if $\mathbb{E}[A] \leq 2^{-d}$ or $\mathbb{E}[B] \leq 2^{-d}$ or $\mathbb{E}[C] \leq 2^{-\varepsilon d/2}$, then search in $(X_k, Y_k, Z_k, A_k, B_k, C_k)$ for a triangle using the brute force method, and return "yes" if we find one.

b. otherwise, return "yes."

If we do not return "yes" in any part, then return "no."

Correctness: First, observe that property (1) of 4.4, which asserts $AB = \sum_{k=1}^{K} A_k B_k$, implies that an edge in AB is present if that edge is present in some $A_k B_k$ for $k \in [K]$. It follows that a triangle in (A, B, C) is present iff it is present in (A_k, B_k, C_k) for some $k \in [K]$, so it remains to show that the above procedure for triangle detection in a part (A_k, B_k, C_k) is correct.

The proof of correctness for triangle detection in each part is identical to the argument presented in Section 4.3; for completeness we repeat it here. Case (a) of the procedure evidently yields the correct result since it simply does an exhaustive brute-force search, so consider case (b). Since $\mathbb{E}[A], \mathbb{E}[B] > 2^{-d}$, it must be that A_k and $B_k^\top$ are both $(\varepsilon, 2, d)$-regular and ε-min-degree by property (2) of Theorem 4.4. Then Theorem 4.1 guarantees that $A_k \circ B_k$ is $(\mathbb{E}[A_k]\mathbb{E}[B_k], 80\varepsilon, 2^{-\varepsilon d/2})$-uniform. By definition of uniformity

$$\mathbb{P}_{(x,z)\in X\times Z}\left[(A_k \circ B_k)(x, z) \in [(1 - 80\varepsilon)\mathbb{E}[A_k]\mathbb{E}[B_k], (1 + 80\varepsilon)\mathbb{E}[A_k]\mathbb{E}[B_k]\right] \geq 1 - 2^{-\varepsilon d/2},$$

so in particular with probability at least $1 - 2^{-\varepsilon d/2}$ we have $(A_k \circ B_k)(x, z) > 0$ (since $1 - 80\varepsilon = 1 - \frac{80}{160} > 0$), or equivalently $(A_k B_k)(x, z) = 1$. Meanwhile $\mathbb{E}[C_k] > 2^{-\varepsilon d/2}$, as otherwise we would be in the first case. By the Pigeonhole Principle, $A_k B_k$ and C_k must have a nonzero entry in common, so there indeed exists a triangle in (A_k, B_k, C_k). Therefore case (b) correctly returns "yes."

Running time: There are two phases to the algorithm: a precomputation phase, where the decomposition promised by Theorem 4.4 is computed, and the triangle detection phase, where we check whether each part of the decomposition contains a triangle. The runtime of the precomputation phase is $n^2 \cdot \exp\bigl(d^7 \operatorname{poly}(\varepsilon^{-1})\bigr)$. The runtime of the brute force search in the sparse case for a given part (X_k, Y_k, Z_k) is at most $|X_k||Y_k||Z_k|/2^{-\varepsilon d/2}$, so using property (3) of Theorem 4.4 the total runtime of these searches is bounded by

$$\sum_{k=1}^{K} \frac{|X_k||Y_k||Z_k|}{2^{\Omega(d)}} \leq \frac{|X||Y||Z| \cdot 2(d+2)^2}{2^{\Omega(d)}} = \frac{|X||Y||Z|}{2^{\Omega(d)}}.$$

To optimize d, we must choose d small enough so that the precomputation running time $n^2 \cdot \exp\bigl(d^7 \operatorname{poly}(\varepsilon^{-1})\bigr) = n^2 \cdot 2^{O(d^7)}$ is subcubic, but as large as possible to minimize the sparse search running time $|X||Y||Z|/2^{\Omega(d)} \leq n^3/2^{\Omega(d)}$. Choosing $d = \Theta(\sqrt[7]{\log n})$ with a sufficiently small constant factor such that $n^2 \cdot 2^{O(d^7)}$ is truly subcubic (e.g. $O(n^{2.1})$), the search running time dominates, and hence the total running time is $n^3/2^{\Omega(\sqrt[7]{\log n})}$, which suffices for the proof. ■

Theorem 4.5 and Corollary 2.1.1 allow us to connect this result back to BMM: together, they directly imply Theorem 1.1, giving us the desired super-poly-logarithmic saving over the naïve $O(n^3)$ algorithm. Note that Theorem 4.5 is robust, as far as BMM is concerned, in the sense that we prove more powerful results than we actually need. For instance, Theorem 4.5 is proved in generality for any graph, but the graph representation that is useful for BMM has some special properties, namely, that $\mathbb{E}[C] = 1$, so in fact the sparse case that $\mathbb{E}[C_k] \leq 2^{-\varepsilon d/2}$ is not even possible. Additionally, property (3) proves the bound $\sum_{k=1}^{K} |X_k||Y_k||Z_k| \leq O(d^2) \cdot |X||Y||Z|$, but it would also have sufficed to show any bound with a poly(d) factor, since it is absorbed into a $2^{-\Omega(d)}$ factor in the final running time.

5 Conclusion

In this paper, we've reviewed recent advances in combinatorial BMM algorithms, focusing on those that exploit regularity decomposition theorems to achieve subcubic runtimes. The remarkable quasi-polynomial improvement in [AFK$^+$23] is based on employing a measure of regularity ((ε, k, ℓ)-regularity discussed in Section 4.2)) that is in some sense "just right": it allows for a decomposition of an arbitrary graph into pieces that are pseudo-random enough to make the Triangle Detection problem easy, but not into so many pieces that efficiency is excessively sacrificed. In this way, the authors manage to surpass the improvements achieved in [BW09], whose use of Szemerédi regularity and Frieze-Kannan's weak regularity was not quite enough to go beyond polylog(n) savings.

Of course, despite this remarkable result, Conjecture 1.1 still stands. Given the potential implications of improved combinatorial BMM algorithms to well-known lower bounds in fine-grained complexity [5], this result should motivate further research into the problem. In particular, as Prof. Yu pointed out in a 2018 publication of his algorithm ([Yu18]) and we note in Section 2.1, the research community has not established a formal, widely agreed upon definition of what exactly counts as a "combinatorial" algorithm. The lower-bounds paper by Das et al. ([DKS18]) was a step forward in attempting to remedy this situation, but as [AFK$^+$23] points out, the algorithmic models they provide (and the associated lower bounds they derive) are not expansive enough to cover any of the state-of-the-art algorithms proposed since 2009 ([BW09], [Cha14], [Yu15], and [AFK$^+$23]) as combinatorial. Rigorously defining this concept in a way that encompasses these recent advances would go a long way to improving our understanding of the limitations of combinatorial techniques.

[5]e.g. for All-Pairs Shortest-Paths, Online Matrix-Vector Multiplication, finding hypercliques in uniform hypergraphs, etc. See the discussion and associated citations in Section 1.1 of [AFK$^+$23] for a detailed discussion.

References

[ABH10] Martin Albrecht, Gregory Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over gf(2). *ACM Transactions on Mathematical Software*, 37(1):1–14, January 2010.

[ADKz70] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. On economical construction of the transitive closure of an oriented graph. *Soviet Mathematics Doklady*, 11:1209–1210, 1970. URL: `http://cr.yp.to/bib/entries.html#1970/arlazarov`.

[AFK$^+$23] Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms, 2023.

[Ang76] Dana Angluin. The four russians' algorithm for boolean matrix multiplication is optimal in its class. *ACM SIGACT News*, 8(1):29–33, 1976.

[BKM95] Julien Basch, Sanjeev Khanna, and Rajeev Motwani. On diameter verification and boolean matrix multiplication. Technical report, Stanford, CA, USA, 1995.

[BW09] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 745–754, 2009.

[Cha14] Timothy M Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 212–217. SIAM, 2014.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms /*. MIT Press,, Cambridge, Mass. :, 3rd ed. edition, 2009.

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[DKS18] Debarati Das, Michal Koucký, and Michael E. Saks. Lower bounds for combinatorial algorithms for boolean matrix multiplication. *ArXiv*, abs/1801.05202, 2018.

[FK99] Alan Frieze and Ravindran Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19:175–220, 02 1999.

[FM71] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 129–131, 1971.

[Fur70] ME Furman. Application of a method of rapid multiplication of matrices to the problem of finding the transitive closure of a graph. In *Doklady Akademii Nauk*, volume 194, pages 524–524. Russian Academy of Sciences, 1970.

[GM97] Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.

[GM17] Joshua A. Grochow and Cristopher Moore. Designing strassen's algorithm, 2017.

[GW94] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties (preliminary version) a quality-size trade-off for hashing. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 574–584, 1994.

[HSHvdG16] Jianyu Huang, Tyler M. Smith, Greg M. Henry, and Robert A. van de Geijn. Implementing strassen's algorithm with blis, 2016.

[IR77] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, page 1–10, New York, NY, USA, 1977. Association for Computing Machinery.

[KLM23] Zander Kelley, Shachar Lovett, and Raghu Meka. Explicit separations between randomized and deterministic number-on-forehead communication. *Electron. Colloquium Comput. Complex.*, TR23, 2023.

[KSSS02] János Komlós, Ali Shokoufandeh, Miklós Simonovits, and Endre Szemerédi. *The Regularity Lemma and Its Applications in Graph Theory*, pages 84–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Lee01] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication, 2001.

[Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969.

[Su21] Jessica Su. Lecture 2: Boolean matrix multiplication (bmm), September 2021.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 645–654, 2010.

[WXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega, 2023.

[Yu15] Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 1094–1105, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Yu18] Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Information and Computation*, 261:240–247, 2018. ICALP 2015.