THEORY OF COMPUTING, Volume 8 (2012), pp. 69–94
www.theoryofcomputing.org

SPECIAL ISSUE IN HONOR OF RAJEEV MOTWANI

Regularity Lemmas and Combinatorial Algorithms

Nikhil Bansal   Ryan Williams

Received: May 25, 2010; published: April 1, 2012.

Abstract: We present new combinatorial algorithms for Boolean matrix multiplication (BMM) and preprocessing a graph to answer independent set queries. We give the first asymptotic improvements on combinatorial algorithms for dense BMM in many years, improving on the "Four Russians" $O(n^3/(w\log n))$ bound for machine models with wordsize w. (For a pointer machine, we can set $w = \log n$.) The algorithms utilize notions from Regularity Lemmas for graphs in a novel way.

- We give two randomized combinatorial algorithms for BMM. The first algorithm is essentially a reduction from BMM to the *Triangle Removal Lemma*. The best known bounds for the Triangle Removal Lemma only imply an $O\left((n^3 \log\beta)/(\beta w \log n)\right)$ time algorithm for BMM where $\beta = (\log^\star n)^\delta$ for some $\delta > 0$, but improvements on the Triangle Removal Lemma would yield corresponding runtime improvements. The second algorithm applies the Weak Regularity Lemma of Frieze and Kannan along with several information compression ideas, running in $O\left(n^3(\log\log n)^2/(\log n)^{9/4}\right)$ time with probability exponentially close to 1. When $w \ge \log n$, it can be implemented in $O\left(n^3(\log\log n)/(w\log n)^{7/6}\right)$ time. Our results immediately imply improved combinatorial methods for CFG parsing, detecting triangle-freeness, and transitive closure.

ACM Classification: F.2.2

AMS Classification: 68Q25

Key words and phrases: Boolean matrix multiplication, regularity lemma, combinatorial algorithm, independent set queries

DOI: 10.4086/toc.2012.v008a004

- Using Weak Regularity, we also give an algorithm for answering queries of the form *is $S \subseteq V$ an independent set?* in a graph. Improving on prior work, we show how to randomly preprocess a graph in $O(n^{2+\varepsilon})$ time (for all $\varepsilon > 0$) so that with high probability, all subsequent batches of $\log n$ independent set queries can be answered deterministically in $O\left(n^2(\log\log n)^2/((\log n)^{5/4})\right)$ time. When $w \geq \log n$, w queries can be answered in $O\left(n^2(\log\log n)^2/((\log n)^{7/6})\right)$ time. In addition to its several applications, this problem is interesting in that it is not known how to do better than $O(n^2)$ using "algebraic" methods.

1 Introduction

Szemerédi's Regularity Lemma is one of the most remarkable results of graph theory, having many diverse uses and applications. In computer science, regularity notions have been used extensively in property and parameter testing [4, 5, 15, 48, 12], approximation algorithms [28, 29, 20], and communication complexity [34]. In this paper we show how regularity can lead to faster combinatorial algorithms for basic problems.

Boolean matrix multiplication (BMM) is among the most fundamental problems in computer science. It is a key subroutine in the solution of many other problems such as transitive closure [25], context-free grammar parsing [59], all-pairs path problems [21, 31, 53, 55], and triangle detection [35].

There have been essentially two lines of theoretical research on BMM. Algebraic algorithms, beginning with Strassen's $\tilde{O}(n^{\log_2 7})$ algorithm [56] and ending (so far) with Vassilevska Williams' $\tilde{O}(n^{2.373})$ algorithm [60], reduce the Boolean problem to ring matrix multiplication and give ingenious methods for the ring version by utilizing cancellations. In particular, multiplication-efficient algorithms are found for multiplying small matrices over an arbitrary ring, and these algorithms are applied recursively. There have been huge developments in this direction over the years, with many novel ideas (cf. [45] for an overview of early work, and [19, 18] for a more recent and promising approach). However, these algorithms (including Strassen's) have properties (lack of locality, extra space usage, and leading constants) that may can them less desirable in practice.[1]

The second line of work on Boolean matrix multiplication has studied so-called *combinatorial* algorithms, the subject of the present paper. Combinatorial algorithms for matrix multiplication exploit redundancies that arise from construing matrices as graphs, often invoking word parallelism, lookup tables, and Ramsey-theoretic arguments. Although the term *combinatorial algorithm* has been used in many of the references cited, there is no general criterion for what is "combinatorial" about them: the term is mainly just a way of distinguishing those approaches which are different from the algebraic approach originating with Strassen. Hence for the purposes of this paper, we simply think of a *combinatorial algorithm* as one that does not call an oracle for ring matrix multiplication. These algorithms are considered to be more practical, but fewer advances have been made. All algorithms for the dense case [43, 8, 51, 9, 50, 10, 62] are loosely based on the "Four Russians" approach of Arlazarov, Dinic, Kronrod, and Faradzhev [8] from 1970, which runs in $O(n^3/(w \log n))$ on modern computational models,

[1]For this reason, some practical implementations of Strassen's algorithm switch to standard (or "Four Russians") multiplication when the submatrices are sufficiently small. For more discussion on the (im)practicality of Strassen's algorithm and variants, cf. [40, 17, 2].

where w is the maximum of $\log n$ and the wordsize.[2] Given its importance, we shall briefly describe the approach here. The algorithm partitions the first matrix into $n \times \varepsilon \log n$ submatrices, and the second matrix into $\varepsilon \log n \times n$ submatrices. For each $n \times \varepsilon \log n$ submatrix, we compute a table with $2^{\varepsilon \log n} = n^{\varepsilon}$ entries where each entry is a $n \times 1$ vector corresponding to the union of some subset of columns of the submatrix. With this table one can multiply each $n \times \varepsilon \log n$ and $\varepsilon \log n \times n$ submatrix together in $O(n^2)$ time. An additional w-factor can be saved by storing the n-bit entries in the table as a collection of n/w words, or a log-factor is saved by storing the outputs as a collection of $n/\log n$ pointers to nodes encoding $\log n$ bit strings in a graph, cf. [50, 10, 62]. To date, this is still the fastest known combinatorial algorithm for dense matrices.

Many works (including [1, 21, 40, 52, 47, 41, 16]) have commented on the dearth of better combinatorial algorithms for BMM. As combinatorial algorithms can often be generalized in ways that the algebraic ones cannot (e.g., to work over interesting semirings), the lack of progress does seem to be a bottleneck, even for problems that appear to be more difficult. For instance, the best known algorithm for the general all-pairs shortest paths problem [16] is combinatorial and runs in $O(n^3 \cdot \mathrm{poly}(\log\log n)/\log^2 n)$ time – essentially the same time as Four Russians. Some progress on special cases of BMM has been made: for instance, in the *sparse* case where one matrix has $m \ll n^2$ nonzeros, there is an $O(mn\log(n^2/m)/(w\log n))$ time algorithm [24, 13]. See [44, 52, 41] for a sampling of other partial results. The search for practical and fast Boolean matrix multiplication is still ongoing.

2 Our results

In this paper we present what are arguably the first concrete improvements on combinatorial algorithms for dense BMM since the 70's. Our approach opens a new line of attack on the problem by connecting the complexity of BMM to modern topics in graph theory, such as the Weak Regularity Lemma and the efficiency of Triangle Removal Lemmas.

2.1 Triangle Removal Lemmas and BMM

A Triangle Removal Lemma [49, 33] states that there is a function f satisfying $\lim_{x\to 0} f(x) = 0$ such that for every graph with at most εn^3 triangles, we can efficiently find $f(\varepsilon)n^2$ edges that hit all triangles. This lemma is one of the many deep consequences of Szemerédi's Regularity Lemma [57]. We prove that good removal lemmas imply faster Boolean matrix multiplication. Let w be the wordsize (typically $w = \Theta(\log n)$).

Theorem 2.1. *Suppose there is an $O(T(n,\varepsilon))$ time algorithm that, for every graph $G = (V,E)$ with at most εn^3 triangles, returns a set $S \subseteq E$ with $|S| \le f(\varepsilon)n^2$ such that $G' = (V, E \setminus S)$ is triangle-free. Then there is a randomized algorithm for Boolean matrix multiplication that returns the correct answer with high probability and runs in time*

$$O\left(T(n,\varepsilon) + \frac{f(\varepsilon)n^3\log(1/f(\varepsilon))}{w\log n} + \frac{n^2}{\varepsilon}\cdot\log n + \varepsilon n^3\right).$$

[2]*Historical Note:* The algorithm in [8] was originally stated to run in $O(n^3/\log n)$ time. Similar work of Moon and Moser [43] from 1966 shows that the inverse of a matrix over $GF(2)$ needs exactly $\Theta(n^2/\log n)$ row operations on n-bit vectors, providing an upper and lower bound. On a RAM, their algorithm runs in $O(n^3/(w\log n))$ time.

Unfortunately the best known upper bound for f is $f(\varepsilon) = O(1/(\log^\star 1/\varepsilon)^\delta)$ for some $\delta > 0$ (cf. Section 3.1). For $\varepsilon = 1/\sqrt{n}$, we obtain a very modest runtime improvement over Four Russians. However no major impediment is known (like that proven by Gowers for the full Regularity Lemma [32]) for obtaining a much better f for triangle removal. The best known lower bound on $f(\varepsilon)$ is only $2^{-O(\sqrt{\log(1/\varepsilon)})}$, due to Rusza and Szemerédi [49]. Given a set $S \subseteq [n]$ with no arithmetic progression of length three, Ruzsa and Szemerédi construct a graph G' with $O(n)$ nodes and $O(n|S|)$ edges whose edge set can be partitioned into $n|S|$ edge-disjoint triangles (and there are no other triangles). The best known constructions of such arithmetic progression free sets, due to Behrend [11] and Elkin [23], have $|S| \approx n^{1-\Theta(1/\sqrt{\log n})}$. So, in the case of G' we have

$$\varepsilon = |S|/n^2 = 1/(n2^{\Theta(\sqrt{\log n})}) \quad \text{and} \quad f(\varepsilon) = |S|/n = 1/2^{\Theta(\sqrt{\log n})} \geq 2^{-\Theta(\sqrt{\log(1/\varepsilon)})}.$$

Hence, with improved Triangle Removal Lemmas, Theorem 2.1 could still imply an $n^3/\exp(\Theta(\sqrt{\log n}))$ time bound for BMM.

Recently, Jacob Fox [26] has given a new proof of the Triangle Removal Lemma which improves the function f. However, the algorithm implicit in his proof requires detecting if a given graph partition is "superregular," which we do not know how to do in sub-cubic time (which is necessary for Theorem 2.1 to apply). Turning his proof into an efficient algorithm is an interesting open problem.

2.2 Weak Regularity and BMM

Our second algorithm for BMM gives a more concrete improvement, relying on the Weak Regularity Lemma of Frieze and Kannan [28, 29] along with several other combinatorial ideas.

Theorem 2.2. *There is a randomized combinatorial algorithm for Boolean matrix multiplication that runs in $\hat{O}(n^3/(\log^{2.25} n))$ (worst-case) time on a pointer machine, and computes the product with high probability.*[3] *More precisely, for any $n \times n$ Boolean matrices A and B, the algorithm computes their Boolean product with probability $1 - \exp(-n)$, and takes time $O(n^3(\log\log n)^2/(\log^{2.25} n))$. On a RAM with wordsize $w \geq \log n$, the algorithm can be implemented in $O(n^3(\log\log n)/(w\log^{7/6} n))$ time.*

These new algorithms are interesting not so much for their quantitative improvements, but because they show *some* further improvement. Some researchers believed that $O(n^3/(w\log n))$ would be the end of the line for algorithms not based on algebraic methods. This belief was quantified by Angluin [7] and Savage [51], who proved in the mid 70's that for a straight-line program model which includes Four Russians, $\Omega(n^3/(w\log n))$ operations are indeed required.[4]

2.3 Preprocessing for fast independent set queries

Finally, we show how our approach can improve the solution of problems that seem beyond the reach of algebraic methods, and give a partial derandomization of some applications of BMM. In the *independent*

[3]The $\hat{O}$ notation suppresses $\text{poly}(\log\log n)$ factors.

[4]More precisely, they proved that Boolean matrix multiplication requires $\Theta(n^2/\log n)$ bitwise OR operations on n-bit vectors, in a straight-line program model where each line is a bitwise OR of some subset of vectors in the matrices and a subset of previous lines in the program, and each row of the matrix product appears as the result of *some* line of the program.

set query problem, we wish to maintain a data structure (with polynomial preprocessing time and space) that can quickly answer if a subset $S \subseteq V$ is independent. It is not known how to solve this problem faster than $O(n^2)$ using "Strassenesque" methods. Previously it was known that one could answer one independent set query in $O(n^2/\log^2 n)$ [62] (or $O(n^2/(w\log n))$ with wordsize w). We show the following result.

Theorem 2.3. *For all $\varepsilon \in (0,1/2)$, we can preprocess a graph G in $O(n^{2+\varepsilon})$ time such that with high probability, all subsequent batches of $\log n$ independent set queries on G can be answered deterministically in $O(n^2(\log\log n)^2/(\varepsilon(\log n)^{5/4}))$ time. On the word RAM with $w \geq \log n$, we can answer w independent set queries in $O(n^2(\log\log n)/(\varepsilon(\log n)^{7/6}))$ time.*

That is, the $O(n^{2+\varepsilon})$ preprocessing is randomized, but the algorithm which answers batches of queries is deterministic, and these answers will always be correct with high probability. Recent work has shown that the preprocessing can be made deterministic as well (cf. the conclusion of this paper). The independent set query problem of Theorem 2.3 has several interesting applications; the last three were communicated to us by Avrim Blum [14].

1. **Triangle Detetection in Graphs.** The query algorithm immediately implies a triangle detection algorithm that runs in $O(n^3(\log\log n)/(\log n)^{9/4})$ time, or $O(n^3(\log\log n)/(w(\log n)^{7/6}))$ time. (A graph is triangle-free if and only if all vertex neighborhoods are independent sets.)

2. **Partial Match Retrieval.** The query problem can also model a special case of partial match retrieval. Let $\Sigma = \{\sigma_1,\ldots,\sigma_k\}$, and let $\star \notin \Sigma$. Imagine we are given a collection of m vectors $v_1,\ldots,v_m$ of length n over $\Sigma \cup \{\star\}$ such that every v_j has only two components from Σ (the rest of the components are all $\star$'s). A series of vectors $q \in \Sigma^n$ arrive one at a time, and we want to determine if q "matches" some v_j, i. e., there is a j such that for all $i = 1,\ldots,n$, $v_j[i] = q[i]$ whenever $v_j[i] \neq \{\star\}$. To formulate this problem as an independent set query problem, make a graph with kn nodes in equal-sized parts $V_1,\ldots,V_k$. Put the edge $(i,i') \in V_a \times V_b$ iff there is a vector v_ℓ in the collection such that $v_\ell[i] = \sigma_a$ and $v_\ell[i'] = \sigma_b$. A query vector q corresponds to asking if $S_q = \bigcup_{\ell=1}^{k}\{i \in V_\ell \mid q[j] = \sigma_\ell\}$ is an independent set in the graph.

3. **Preprocessing 2-CNF Formulas.** We can also a preprocess 2-CNF formula F on n variables, in order to quickly evaluate F on arbitrary assignments. Make a graph with $2n$ nodes, one for each possible literal in F. For each clause $(\ell_i \vee \ell_j)$ in F, put an edge between nodes $\neg\ell_i$ and $\neg\ell_j$ in the graph. Now given a variable assignment $A : \{0,1\}^n \to \{0,1\}$, observe that the set $S_A = \{x \mid A(\ell) = 1\} \cup \{\neg x \mid A(x) = 0\}$ is independent if and only if A satisfies F.

4. **Answering 3-SUM Queries.** Independent set queries can solve a query version of the well-known *3-SUM problem* [30]. The 3-SUM problem asks: given two sets A and B of n elements each, are there two elements in A that add up to some element in B? The assumption that 3-SUM cannot be solved much faster than the trivial $O(n^2)$ bound has been used to show hardness for many computational geometry problems [30], as well as lower bounds on data structures [46].

   A natural query version of the problem is: given two sets A and B of n integers each, preprocess them so that for any query set $S \subseteq A$, one can quickly answer whether two elements in S sum to an element in B. Make a graph with a node for each integer in A, and an edge between two integers in A if their sum is an element in B: this gives exactly the independent set query problem.

3 Preliminaries

The *Boolean semiring* is the semiring on $\{0,1\}$ with OR as addition and AND as multiplication. For Boolean matrices A and B, $A \vee B$ is the componentwise OR of A and B, $A \wedge B$ is the componentwise AND, and $A \star B$ is the (Boolean) matrix product over the Boolean semiring. When it is clear from the context, sometimes we omit the $\star$ and write AB for the product.

Since the running times of our algorithms involve polylogarithmic terms, we must make the computational model precise. Unless otherwise specified, we assume a standard word RAM with wordsize w. That is, accessing a memory location takes $O(1)$ time, and we can perform simple operations (such as addition, componentwise AND and XOR, but not multiplication) on w-bit numbers in $O(1)$ time. Typically, speedups in combinatorial algorithms come from either exploiting some combinatorial substructure, by preprocessing and doing table lookups, or by some "word tricks" which utilize the bit-level parallelism of the machine model. In our results, we explicitly state the dependence of the word size, denoted by w. The reader may assume $w = \Theta(\log n)$ for convenience. In fact all algorithms in this paper can be implemented on a pointer machine under this constraint.

We now describe some of the tools we need.

3.1 Regularity

Let $G = (V,E)$ be a graph and let $S,T \subseteq V$ be disjoint. Define $e(S,T) = \{(u,v) \in E \mid u \in S, v \in T\}$. The *density of* (S,T) is $d(S,T) = e(S,T)/(|S||T|)$. Thus $d(S,T)$ is the probability that a random pair of vertices, one from S and one from T, have an edge between them. For $\varepsilon > 0$, the pair (S,T) is ε*-regular* if over all $S' \subseteq S$ and $T' \subseteq T$ with $|S'| \geq \varepsilon|S|$ and $|T'| \geq \varepsilon|T|$, we have $|d(S',T') - d(S,T)| \leq \varepsilon$. That is, the density of all sufficiently large subsets of (S,T) is approximately $d(S,T)$.

Definition 3.1. A partition $\{V_1,\ldots,V_k\}$ of V is an ε*-regular partition of* G if

- for all i, $|V_i| \leq \varepsilon|V|$,
- for all i,j, $\big||V_i| - |V_j|\big| \leq 1$, and
- all but at most εk^2 of the pairs (V_i,V_j) are ε-regular.

Szemerédi's celebrated theorem [57] states that in *every sufficiently large graph* and *every* ε, an ε-regular partition exists.

Lemma 3.2 (Regularity Lemma)**.** *For all $\varepsilon > 0$, there is a $K(\varepsilon)$ such that every G has an ε-regular partition where the number of parts k is at most $K(\varepsilon)$.*

We need to compute such a partition in less than cubic time, in order to perform faster matrix multiplication. There exist several polynomial time constructions of ε-regular partitions [3, 27, 29, 37].

The fastest deterministic algorithm runs in $O(K'(\varepsilon)n^2)$ time (for some $K'(\varepsilon)$ related to $K(\varepsilon)$) and is due to Kohayakawa, Rödl, and Thoma [37].[5]

Theorem 3.3 (Kohayakawa-Rödl-Thoma [37])**.** *There is an algorithm that, on input $\varepsilon > 0$ and graph G on n nodes, outputs an ε-regular partition with $K'(\varepsilon)$ parts and runs in $O(20/(\varepsilon')^5(n^2 + K'(\varepsilon)n))$ time. $K'(\varepsilon)$ is a tower of at most $20/(\varepsilon')^5$ twos where $\varepsilon' = (\varepsilon^{20}/10^{24})$.*

Let us give a few more details on how the above algorithm is obtained. The above theorem is essentially Corollary 1.6 in Section 3.2 of [37], however we have explicitly spelled out the dependency between ε', K', and ε. Theorem 1.5 in [37] shows that in $O(n^2)$ time, we can either verify ε-regularity or obtain a *witness* for ε'-irregularity (with ε' as above). Here, a witness is simply a pair of subsets of vertices for which the ε'-regularity condition fails to hold. Lemma 3.6 in Section 3.2 of [37] shows how to take proofs of ε'-irregularity for a partition and refine the partition in linear time, so that the index (a quantity that always lies in $[0,1]$) of the partition increases by $(\varepsilon')^5/20$. Thus, in at most $20/(\varepsilon')^5$ iterations of partition refinement (each refinement taking $O(K'(\varepsilon)n)$ time) we can arrive at an ε-regular partition.

We also need the Triangle Removal Lemma, first stated by Ruzsa and Szemerédi [49]. In one formulation, the lemma says there is a function f such that $f(\varepsilon) \to 0$ as $\varepsilon \to 0$, and for every graph with at most εn^3 triangles, at most $f(\varepsilon)n^2$ edges need to be removed to make the graph triangle-free. We use a version stated by Green ([33], Proposition 1.3); for completeness, we give a full proof.

Lemma 3.4 (Triangle Removal Lemma)**.** *Suppose G has at most δn^3 triangles. Let $k = K(\varepsilon)$ be the number of parts in some ε-regular partition of G, where $4\varepsilon k^{-3} > \delta$ and ε is sufficiently small. Then there is a set of at most $4\varepsilon^{1/3}n^2$ edges such that their removal makes G triangle-free.*[6]

In particular, let $\{V_1, \ldots, V_k\}$ be an ε-regular partition of G. By removing all edges in pairs (V_i, V_i), the pairs (V_i, V_j) with density less than $2\varepsilon^{1/3}$, and all non-regular pairs, G becomes triangle-free.

Proof. Let $G' = (V, E')$ be the graph obtained by removing all edges from the pairs (V_i, V_j) that have density less than $2\varepsilon^{1/3}$ or are non-regular, and remove all edges from the pairs (V_i, V_i). As each low-density pair has at most $2\varepsilon^{1/3}(n/k)^2$ edges, there are at most εk^2 non-regular pairs, and the number of edges with both vertices in the same part is at most $k \cdot (n/k)^2$, the number of edges removed is at most

$$k^2 \cdot 2\varepsilon^{1/3}(n/k)^2 + \varepsilon k^2 \cdot (n/k)^2 + n^2/k \leq 4\varepsilon^{1/3}n^2\,,$$

for sufficiently small $\varepsilon > 0$ (note that $k \gg \mathrm{poly}(1/\varepsilon)$).

We now show that G' is triangle-free. Suppose there is a triangle (u, v, w) with $u \in V_i$, $v \in V_j$, and $w \in V_k$ for distinct indices i, j, k. By construction, $|V_i|$, $|V_j|$ and $|V_k|$ are all at least n/k, the density of all pairs of edges is at least $2\varepsilon^{1/3}$, and all pairs are ε-regular. We claim that the number of triangles between the parts V_i, V_j, and V_k is at least δn^3, contradicting our hypothesis on G.

[5] The paper [37] claims that [28, 29] give an algorithm for constructing a regular partition that runs in *linear* time, but we are unsure of this claim. The algorithm given in Frieze-Kannan [29] seems to require that we can verify regularity in linear time without giving an algorithm for this verification.

[6] In our later algorithms, we will choose ε to depend on n.

WLOG we may assume $|V_i| = |V_j| = |V_k|$; let their cardinality be $t \geq n/k$. Let $b = 2\varepsilon^{1/3} - \varepsilon$. First we claim there are less than εt nodes in V_i which have less than bt neighbors in V_j. If not, then there would be a $V_i' \subset V_i$ with $|V_i'| = \varepsilon t$ such that

$$d(V_i', V_j) < \frac{\varepsilon b t^2}{\varepsilon t \cdot t} = b = 2\varepsilon^{1/3} - \varepsilon\,,$$

yet $d(V_i, V_j) \geq 2\varepsilon^{1/3}$, contradicting ε-regularity. Analogously, there are less than εt nodes in V_i with less than bt neighbors in V_k. Hence there are at least $(1-2\varepsilon)t$ nodes in V_i that have at least bt neighbors in V_j and bt neighbors in V_k. Let $v \in V_i$ be such a node and let S_j and S_k be the neighbors of v in V_i and V_k, respectively. By ε-regularity, the number of edges between S_j and S_k is at least $(2\varepsilon^{1/3} - \varepsilon)(bt)^2 = b^3t^2$. So at least $(1-2\varepsilon)t$ nodes in V_i participate in at least b^3t^2 triangles, hence the number of triangles among V_i, V_j, V_k is at least $T = (1-2\varepsilon)t \cdot b^3 t^2$. For small enough $\varepsilon > 0$, we have $2\varepsilon^{1/3} - \varepsilon > (5/3)\varepsilon^{1/3}$ and $4.5\varepsilon - 9\varepsilon^2 > 4\varepsilon$, so

$$T = (1-2\varepsilon)b^3t^3 > (1-2\varepsilon)(5/3)^3\varepsilon t^3 > 4.5\varepsilon t^3 - 9\varepsilon^2 t^3 > 4\varepsilon(n/k)^3 > \delta n^3,$$

contradicting our assumption that there were less than δn^3 triangles. □

Notice that the lemma gives an efficient way of discovering which edges to remove, when combined with an algorithmic Regularity Lemma. However the above proof yields only a very weak bound on $f(\varepsilon)$, of the form $c/(\log^\star 1/\varepsilon)^\delta$ for some constants $c > 1$ and $\delta > 0$. It is of great interest to prove a triangle removal lemma with much smaller $f(\varepsilon)$. A step in this direction is the result result of Fox [26] mentioned earlier.

There are also other (weaker) notions of regularity that suffice for certain applications, where the dependence on ε is much better. We discuss below a variant due to Frieze and Kannan [29]. There are also other variants known, for example [36, 4, 22]. We refer the reader to the survey [38]. Frieze and Kannan defined the following notion of a pseudoregular partition.

Definition 3.5 (ε-pseudoregular partition)**.** Let $\mathcal{P} = V_1, \ldots, V_k$ be a partition of V, and let d_{ij} be the density of (V_i, V_j). For a subset $S \subseteq V$, and $i = 1, \ldots, k$, let $S_i = S \cap V_i$. The partition $\mathcal{P}$ is *ε-pseudoregular* if the following relation holds for all disjoint subsets S, T of V:

$$\left| e(S,T) - \sum_{i,j=1}^{k} d_{ij}|S_i||T_j| \right| \leq \varepsilon n^2.$$

A partition is *equitable* if for all i, j, $\big||V_i| - |V_j|\big| \leq 1$.

Theorem 3.6 (Frieze-Kannan [29], Theorem 2 and Section 5.1)**.** *For all $\varepsilon \geq 0$, an equitable ε-pseudoregular partition of an n node graph with at most* $\min\{n, 2^{4\lceil 64/(3\varepsilon^2)\rceil}\}$ *parts can be constructed in*

$$O\left(2^{O(1/\varepsilon^2)} \frac{n^2}{\varepsilon^2 \delta^3}\right)$$

time with a randomized algorithm that succeeds with probability at least $1 - \delta$.

The runtime bound above is a little tighter than what Frieze and Kannan claim, but an inspection of their algorithm shows that this bound is achieved. Note that Lovász and Szegedy [42] have proven that for any ε-pseudoregular partition, the number of parts must be at least $(1/4) \cdot 2^{1/(8\varepsilon)}$.

3.2 Preprocessing Boolean matrices for sparse operations

Our algorithms exploit regularity to reduce dense BMM to a collection of somewhat sparse matrix multiplications. To this end, we need results on preprocessing matrices to speed up computations on sparse inputs. The first deals with multiplication of an arbitrary matrix with a sparse vector, and the second deals with multiplication of a sparse matrix with another (arbitrary) matrix.

Theorem 3.7 (Blelloch-Vassilevska-Williams [13])**.** *Let B be a $n \times n$ Boolean matrix and let w be the wordsize. Let $\kappa \geq 1$ and $\ell > \kappa$ be integer parameters. There is a data structure that can be constructed with $O((n^2\kappa/\ell) \cdot \sum_{b=1}^{\kappa} \binom{\ell}{b})$ preprocessing time, so that for any Boolean vector v, the product $B \star v$ can be computed in*

$$O\left(n \log n + \frac{n^2}{\ell w} + \frac{nt}{\kappa w}\right)$$

time, where t is the number of nonzeros in v.

This result is typically applied as follows. Fix a value of t to be the number of nonzeros we expect in a typical vector v. Choose ℓ and κ such that $n/\ell \approx t/\kappa$, and $\sum_{b=1}^{\kappa} \binom{\ell}{b} = n^{\delta}$ for some $\delta > 0$. One such choice is $\kappa = \delta \ln(n)/\ln(en/t)$ and $\ell = \kappa \cdot en/t$ in which case we obtain:

Theorem 3.8. *Let B be a $n \times n$ Boolean matrix. There is a data structure that can be constructed with $\tilde{O}(n^{2+\delta})$ preprocessing time, so that for any Boolean vector v, the product $B \star v$ can be computed in*

$$O\left(n \log n + \frac{nt \ln(en/t)}{\delta w \ln n}\right)$$

time, where t is the number of nonzeros in v.

We should remark that we do not explicitly apply the above theorem, but the idea (of preprocessing for sparse vectors) is used liberally in this paper.

The following result is useful for multiplying a sparse matrix with another arbitrary matrix.

Theorem 3.9. *There is an $O(mn \log(n^2/m)/(w \log n))$ time algorithm for computing $A \star B$, for every $n \times n$ A and B, where A has m nonzeros and B is arbitrary.*

This result follows in a straightforward manner by combining the two lemmas below. The first is a graph compression method due to Feder and Motwani.

Lemma 3.10 (From Feder-Motwani [24], Theorem 3.3)**.** *Let $\delta \in (0,1)$ be constant. We can write any $n \times n$ Boolean matrix A with m nonzeros as $A = (C \star D) \vee E$ where C, D are $n \times m/n^{1-\delta}$, $m/n^{1-\delta} \times n$, respectively, both with at most $m(\log n^2/m)/(\delta \log n)$ nonzeros, and E is $n \times n$ and has at most $n^{2-\delta}$ nonzeros. Furthermore, finding C, D, E takes $O(mn^{\delta} \log^2 n)$ time.*

Since the lemma is not stated explicitly in [24], let us sketch the proof for completeness. Using algorithmic Ramsey theoretic arguments (i. e., finding large bipartite cliques in a dense enough graph), Feder and Motwani show that for every bipartite graph G on $2n$ nodes (with n nodes each on left and right) and $m > n^{2-\delta}$ edges, its edge set can be decomposed into $m/n^{1-\delta}$ edge-disjoint bipartite cliques, where the total sum of vertices over all bipartite cliques (a vertex appearing in K cliques is counted K times) is at most $m(\log n^2/m)/(\delta \log n)$. Every A can be written in the form $(C \star D) \vee E$, by having the columns of C (and rows of D) correspond to the bipartite cliques. Set $C[i,k] = 1$ iff the ith node of the LHS of G is in the kth bipartite clique, and similarly set D for the nodes on the RHS of G. Note that E is provided just in case A turns out to be sparse.

We also need the following simple folklore result. It is stated in terms of wordsize w, but it can easily be implemented on other models such as pointer machines with $w = \log n$.

Lemma 3.11 (Folklore)**.** *There is an $O(mn/w + pq + pn)$ time algorithm for computing $A \star B$, for every $p \times q$ matrix A and $q \times n$ matrix B where A has m nonzeros and B is arbitrary.*

Proof. We assume the nonzeros of A are stored in a list structure; if not we construct this in $O(pq)$ time. Let B_j be the jth row of B and C_i be the ith row of C in the following. We start with an output matrix C that is initially zero. For each nonzero entry (i, j) of A, update C_i to be the OR of B_j and C_i. Each update takes only $O(n/w)$ time. It is easy to verify that the resulting C is the matrix product. $\square$

4 Combinatorial Boolean matrix multiplication via triangle removal

In this section, we prove Theorem 2.1. That is, we show that a more efficient Triangle Removal Lemma implies more efficient Boolean matrix multiplication. Let A and B be the matrices whose product D we wish to compute. The key idea is to split the task into two cases. First, we use simple random sampling to determine the entries in the product that have many witnesses (where k is a *witness for* (i, j) if $A[i,k] = B[k, j] = 1$). To compute the entries with few witnesses, we set up a tripartite graph corresponding to the remaining undetermined entries of the matrix product, and argue that it has few triangles. (Each triangle corresponds to a specific witness for a specific entry in D that is still undetermined.) By a Triangle Removal Lemma, a sparse number of edges hit all the triangles in this graph.[7] Using three carefully designed sparse matrix products (which only require one of the matrices to be sparse), we can recover all those entries $D[i, j] = 1$ which have few witnesses.

We now describe our algorithm for BMM.

Algorithm: Let A and B be $n \times n$ matrices. We wish to compute $D = A \star B$, i. e.,

$$D[i, j] = \left(\bigvee_{k=1}^{n} A[i,k] \wedge B[k, j] \right).$$

Random sampling for pairs with many witnesses. First, we detect the pairs (i, j) with at least εn witnesses. Construct a $n \times n$ matrix C as follows. Pick a sample R of $(6 \log n)/\varepsilon$ elements from $[n]$. For each (i, j), $1 \le i, j \le n$, check if there is a $k \in R$ that is a witness for (i, j) in the product. If yes, set $C[i, j] = 1$, otherwise $C[i, j] = 0$. Clearly, this takes at most $O((n^2 \log n)/\varepsilon)$ time. Note that C is dominated by the desired D, in that $C[i, j] \le D[i, j]$ for all i, j. If (i, j) has at least εn witnesses, then some witness lies in R with probability at least $1 - 1/n^6$. Thus with probability at least $1 - 1/n^4$, $C[i, j] = D[i, j] = 1$ for every (i, j) with at least εn witnesses.

[7]Note that the triangle removal lemma may also return edges that do not lie in any triangle.

Triangle removal for pairs with few witnesses. It suffices to determine those (i, j) such that $C[i, j] = 0$ and $D[i, j] = 1$. We shall exploit the fact that such pairs do not have many witnesses. Make a tripartite graph H with vertex sets V_1, V_2, V_3, each with n nodes indexed by $1,\dots,n$. Define edges as follows:

- Put an edge $(i,k) \in (V_1,V_2)$ if and only if $A[i,k] = 1$.
- Put an edge $(k, j) \in (V_2,V_3)$ if and only if $B[k, j] = 1$.
- Put an edge $(i, j) \in (V_1,V_3)$ if and only if $C[i, j] = 0$.

That is, edges from V_1 to V_3 are given by $\overline{C}$, the complement of C. Observe that $(i,k, j) \in (V_1,V_2,V_3)$ is a triangle if and only if k is a witness for (i, j) and $C[i, j] = 0$. Thus our goal is to find the pairs $(i, j) \in (V_1,V_3)$ that are in triangles of H.

Since every $(i, j) \in (V_1,V_3)$ has at most εn witnesses, there are at most εn^3 triangles in H. Applying the promised Triangle Removal Lemma (in Theorem 2.1), we can find in time $O(T(n))$ a set of edges F where $|F| \le f(\varepsilon)n^2$ and each triangle must use an edge in F. Hence it suffices to compute those edges $(i, j) \in (V_1,V_3)$ that participate in a triangle with an edge in F.

Define $A_F[i, j] = 1$ if and only if $A[i, j] = 1$ and $(i, j) \in F$. Similarly define B_F and $\overline{C}_F$. Every triangle of H passes through at least one edge from one of these three matrices. Let T_A (resp. T_B and $T_{\overline{C}}$) denote the set of triangles with an edge in A_F (resp. B_F and $\overline{C}_F$). Note that we do not know these triangles.

We can determine the edges $(i, j) \in (V_1,V_3)$ that are in some triangle in T_A or T_B directly by computing $C_1 = A_F \star B$ and $C_2 = A \star B_F$, respectively. As A_F and B_F are sparse, by Theorem 3.9, these products can be computed in $O(|F|\log(n^2/|F|)/(w\log n))$ time. The 1-entries of $\overline{C} \wedge C_1$ (resp. $\overline{C} \wedge C_2$) participate in a triangle in T_A (resp. T_B). This determines the edges in (V_1,V_3) participating in triangles from $T_A \cup T_B$.

Set $C = C \vee (C_1 \wedge \overline{C}) \vee (C_2 \wedge \overline{C})$, and update $\overline{C}$ and the edges in (V_1,V_3) accordingly. The only remaining edges in (V_1,V_3) that could be involved in a triangle are those corresponding to 1-entries in $\overline{C}_F$. We now need to determine which of these actually lie in a triangle.

Our remaining problem is the following: we have a tripartite graph on vertex set (V_1,V_2,V_3) with at most $f(\varepsilon)n^2$ edges between V_1 and V_3, and each such edge lies in at most εn triangles. We wish to determine the edges in (V_1,V_3) that participate in triangles. This problem is solved by the following theorem.

Theorem 4.1 (Reporting Edges in Triangles)**.** *Let G be a tripartite graph on vertex set (V_1,V_2,V_3) such that there are at most δn^2 edges in (V_1,V_3), and every edge of (V_1,V_3) is in at most t triangles. Then the set of edges in (V_1,V_3) that participate in triangles can be computed in $O(\delta n^3\log(1/\delta)/(w\log n)+n^2t)$ time.*

Setting $\delta = f(\varepsilon)$ and $t = \varepsilon n$, Theorem 4.1 implies the desired time bound in Theorem 2.1. The idea of the proof of Theorem 4.1 is to work with a new tripartite graph where the vertices have asymptotically smaller degrees, at the cost of adding slightly more nodes. This is achieved by having some nodes in our new graph correspond to *small subsets of nodes* in the original tripartite graph.

Proof of Theorem 4.1. We first describe how to do the computation on a pointer machine with $w = \log n$, then describe how to modify it to work for the word RAM.

Graph Construction We start by defining a new tripartite graph G' on vertex set (V_1, V_2', V_3'). Let $\gamma < 1/2$. V_2' is obtained by partitioning the nodes of V_2 into $n/(\gamma \log n)$ groups of size $\gamma \log n$ each. For each group, we replace it by $2^{\gamma \log n} = n^\gamma$ nodes, one corresponding to each subset of nodes in that group. Thus V_2' has $n^{1+\gamma}/(\gamma \log n)$ nodes.

V_3' is also constructed out of subsets of nodes. We form n/ℓ groups each consisting of ℓ nodes in V_3, where $\ell = \gamma(\log n)/(\delta \log(e/\delta))$. For each group, we replace it by $\binom{\ell}{k} \leq (el/k)^k \leq n^\gamma$ nodes, one corresponding to each subset of size up to $\kappa = \gamma(\log n)/(\log(e/\delta))$. So V_3' has $O(n^{1+\gamma}/\ell)$ nodes.

Edges in (V_2', V_3')*:* Put an edge between u in V_2' and x in V_3' if there is an edge (i,j) in (V_2, V_3) such that i lies in the set corresponding to u, and j lies in the set corresponding to x. For each such edge (u,x), we make a list of all edges $(i,j) \in (V_2, V_3)$ corresponding to it. Observe the list for a single edge has size at most $\gamma \log n \cdot \ell = O(\log^2 n)$.

Edges in (V_1, V_2')*:* The edges from $v \in V_1$ to V_2' are defined as follows. For each group in V_2 consider the neighbors of v in that group. Put an edge from v to the node in V_2' corresponding to this subset. Each v has at most $n/(\gamma \log n)$ edges to nodes in V_2'.

Edges in (V_1, V_3')*:* Let $v \in V_1$. For each group g of ℓ nodes in V_3, let $N_{v,g}$ be the set of neighbors of v in g. Let $d_{v,g} = |N_{v,g}|$. Partition $N_{v,g}$ arbitrarily into $t = \lceil d_{v,g}/\kappa \rceil$ subsets $s_1, \ldots, s_t$ each of size at most κ. Put edges from v to $s_1, \ldots, s_t$ in V_3'. The number of these edges from v is at most $\sum_g \lceil d_{v,g}/\kappa \rceil \leq n/\ell + d_v/\kappa$, where d_v is the number of edges from v to V_3. Since $\sum_v d_v \leq \delta n^2$, the total number of edges from V_1 to V_3' is $O(\delta \log(1/\delta) n^2/(\gamma \log n))$.

Final Algorithm For each vertex $v \in V_1$, iterate over each pair of v's neighbors $u \in V_2'$ and $x \in V_3'$. If (u,x) is an edge in G', output the list of edges (i,j) in (V_2, V_3) corresponding to (u,x), otherwise continue to the next pair. From these outputs we can easily determine the edges (v,j) in (V_1, V_3) that are in triangles: (v,j) is in a triangle if and only if node j in V_3 is output as an end point of some edge $(i,j) \in (V_2, V_3)$ during the loop for v in V_1.

Running Time: The graph construction takes at most $O(n^{2+2\gamma})$. In the final algorithm, the total number of pairs (u,w) in (V_2', V_3') that are examined is at most

$$(n/\log n) \cdot O(\delta n^2 (\log 1/\delta)/\log n)) \leq O(\delta \log(1/\delta) n^3/\log^2 n).$$

We claim that the time used to output the lists of edges is at most $O(n^2 t)$ time. A node j from V_3 is on an output list during the loop for v in V_1 if and only if (v,j) is an edge in a triangle, with some node in V_2 that has a 1 in the node i in V_2'. Since each edge from (V_1, V_3) in a triangle is guaranteed to have at most t witnesses in V_2, the node j is output at most t times over the loop for v in V_1. Hence the length of all lists output during the loop for v is at most nt, and the total time for output is at most $O(n^2 t)$.

Modification for w-word RAM: We now show how to replace a log-speedup by a w-speedup with wordsize w. We form V_3' as above and consider the graph on (V_1, V_2, V_3') (note the V_2 instead of V_2' previously). Recall that a node $x \in V_3'$ corresponds to a subset $V_x \subset V_3$ of at most κ vertices (from some group of size ℓ). An edge $(v,x) \in (V_1, V_3')$ implies that v is adjacent to every vertex in V_x. For each $v \in V_1$, let S_v be the n-bit indicator vector corresponding to neighbors of v in V_2. For each $x \in V_3'$, let T_x denote the n-bit indicator vector corresponding to the union of neighbors of vertices V_x in V_2, i. e., $T_x[i] = 1$ iff some $v \in V_x$ is adjacent to $i \in V_2$. The vectors S_v and T_x are stored as n/w words. With each bit $T_x[i]$ of T_x such that

$T_x[i] = 1$, we store a pointer to a list of vertices $v \in V_x$ that are adjacent to $i \in V_2$. It is easily checked that the overall space usage is bounded by $O(n \cdot |V_3'| \cdot \kappa) = \tilde{O}(n^{2+\gamma})$.

The algorithms works as follows: Now for every v in V_1 and every neighbor $x \in V_3'$ of v, for $i = 1, \ldots, n/w$, we look up the ith word q from S_v and the ith word q' in the T_x, and compute $q \wedge q'$. If this is nonzero, then each bit location b where $q \wedge q'$ has a 1 means that the node corresponding to b forms a triangle with v and some vertex in V_x. For each such bit (say at location i), the list associated to it consists of precisely the nodes in V_x forming a triangle with $i \in V_2$ and v.

The running time follows as there are $O(\delta n^2(\log 1/\delta)/\log n))$ edges in (V_1, V_3'), and hence there we look up at most $O(n(\delta n^2(\log 1/\delta)/\log n)(n/w)$ words q, q'. Since the total number of triangles is $n^2 t$, the total time spent in processing whenever $q \wedge q' = 1$ is $O(n^2 t)$. □

Remark 4.2. Note that we only use randomness in the BMM algorithm to determine the pairs (i, j) that have many witnesses. Moreover, by choosing a larger sample R in the random sampling step (notice we have a lot of slack in the running time of the random sampling step), the probability of failure can be made exponentially small.

Using the best known bounds for triangle removal, we obtain the following corollary to Theorem 2.1:

Corollary 4.3. *There is a $\delta > 0$ and a randomized algorithm for Boolean matrix multiplication that works with high probability and runs in*

$$O\left(\frac{n^3 \log(\log^\star n)}{w(\log n)(\log^\star n)^\delta}\right)$$

time.

Proof. Let $\varepsilon = 1/\sqrt{n}$. By the usual proof of the triangle removal lemma (via the Regularity Lemma), it suffices to set $f(\varepsilon) = 1/(\log^\star 1/\varepsilon)^\delta$ in Theorem 2.1 for a constant $\delta > 0$. □

It is our hope that further work on triangle removal may improve the dependency of f. In the next section, we show how to combine the Weak Regularity Lemma along with the above ideas to construct a faster algorithm for BMM.

5 Faster Boolean matrix multiplication via Weak Regularity

We first state a useful lemma for processing a boolean matrix B to compute the product $u^T Bv$ quickly given any boolean vectors u and v. Viewing B as an incidence matrix of a bipartite graph, this corresponds to determining whether there is some edge $e \in (U, V)$ where U and V are sets corresponding to characteristic vectors u and v. This lemma is inspired by Theorem 3.7 and uses a similar technique to our algorithm for reporting the edges that appear in triangles (Theorem 4.1).

Theorem 5.1 (Preprocessing for Bilinear Forms). *Let B be an $n \times n$ Boolean matrix. Let $\kappa \geq 1$ and $\ell \geq \kappa$ be integer parameters. For the pointer machine, there is a data structure that can be built in*

$$O\left(n^2/\ell^2 \cdot \left(\sum_{b=1}^{\kappa} \binom{\ell}{b}\right)^2\right)$$

time, so that for any $u, v \in \{0,1\}^n$, the product $u^T Bv$ over the Boolean semiring can be computed in

$$O\left(n\ell + \left(\frac{n}{\ell} + \frac{t_u}{\kappa}\right)\left(\frac{n}{\ell} + \frac{t_v}{\kappa}\right)\right)$$

time, where t_u and t_v are the number of nonzeros in u and v, respectively. Moreover, the data structure can output the list of pairs (i, j) such that $u_i B[i, j] v_j = 1$ in $O(p)$ additional time, where p is the number of such pairs.

On the word RAM with $w \geq \log n$, the same can be achieved in

$$O\left(n\ell + \frac{n}{w} \cdot \left(\frac{n}{\ell} + \frac{\min(t_u, t_v)}{\kappa}\right)\right)$$

time.

For our applications, we shall set $\ell = \log^2 n$ and $\kappa = \log n/(5 \log\log n)$. Then the preprocessing is $n^{3-\Omega(1)}$, $u^T Bv$ can be computed in time

$$O\left(\left(\frac{n}{\log^2 n} + \frac{t_u \log\log n}{\log n}\right)\left(\frac{n}{\log^2 n} + \frac{t_v \log\log n}{\log n}\right)\right) \tag{5.1}$$

on a pointer machine, and it can be computed on RAMs with large wordsize w in time

$$O\left(\frac{n^2}{w \log^2 n} + \frac{n \min(t_u, t_v) \log\log n}{w \log n}\right). \tag{5.2}$$

Proof of Theorem 5.1. As in the proof of Theorem 4.1, we first describe how to implement the algorithm on a pointer machine, then show how it may be adapted. We view B as a bipartite graph $G = (U, V, E)$ in the natural way, where $U = V = [n]$ and $(i, j) \in E$ iff $B[i, j] = 1$. We group vertices in U and V into $\lceil n/\ell \rceil$ groups, each of size at most ℓ. For each group g, we introduce a new vertex for every subset of up to κ vertices in that group. Let U' and V' be the vertices obtained. We view the nodes of U' and V' also as vectors of length ℓ with up to κ non-zeros. Clearly

$$|U'| = |V'| = O\left((n/\ell)\left(\sum_{b=1}^{\kappa} \binom{\ell}{b}\right)\right).$$

For every vertex $u' \in U'$, we store a table $T_{u'}$ of size $|V'|$. The v'-th entry of $T_{u'}$ is 1 iff there is an $i \in U$ in the set corresponding to u', and a $j \in V$ in the set corresponding to v', such that $B[i, j] = 1$. Each (i, j) is said to be a *witness to* $T_{u'}[v'] = 1$. In the output version of the data structure, we associate a list $L_{v'}$ with every nonzero entry v' in the table $T_{u'}$ which contains those (i, j) pairs which are witnesses to $T_{u'}[v'] = 1$. Note that $|L_{v'}| \leq O(\kappa^2)$.

Given query vectors u and v, we compute $u^T Bv$ and those (i, j) satisfying $u_i B[i, j] v_j = 1$ as follows. Let u_g be the restriction of the vector u to group g of U. Note $|u_g| \leq \ell$. Let $t(u, g)$ denote the number of non-zeros in u_g. Express u_g as a Boolean sum of at most $\lceil t(u, g)/\kappa \rceil$ vectors (nodes) from U'; this can be done since each vector in U' has up to κ non-zeros. Do this over all groups g of U. Now u can be represented as a Boolean sum of at most $n/\ell + t_u/\kappa$ vectors from U'. We repeat a similar procedure for v

over all groups g of V, obtaining a representation of v as a sum of at most $n/\ell + t_v/\kappa$ vectors from V'. These representations can be determined in $O(n\ell)$ time.

Let $S_u \subseteq U'$ be the subset of vectors representing u, and $S_v \subseteq V'$ be the vectors for v. For all $u' \in S_u$ and $v' \in S_v$, look up $T_{u'}[v']$; if it is 1, output the list $L_{v'}$. Observe $u^T Bv = 1$ iff there is some $T_{u'}[v']$ that equals 1. It is easily seen that this procedure satisfies the desired running time bounds.

Finally, we consider how to implement the above on the word RAM model. We shall have two (analogous) data structures depending on whether $t_u \le t_v$ or not.

Suppose $t_u \le t_v$ (the other situation is analogous). As previously in Theorem 4.1, we form the graph U' with vertices corresponding to subsets of up to κ nonzeros within a vector of size ℓ. With each such vertex $u' \in U'$ we associate an n-bit vector $T_{u'}$ (which is stored as an n/w-word vector), obtained by taking the union of the rows of B corresponding to u'. Now, since v can also be stored as an n/w-word vector, the product $T_{u'} \cdot v$ can be performed in n/w time. For a given u there are at most $n/\ell + t_u/\kappa$ relevant vectors $T_{u'}$ and hence the product $u^T Bv$ can be computed in time $O((n/\ell + t_u/\kappa)(n/w))$. □

Theorem 5.2. *There is a combinatorial algorithm that, given any two Boolean $n \times n$ matrices A and B, computes $A \star B$ correctly with probability exponentially close to 1, in $O(n^3(\log\log n)^2/(\log^{2.25} n))$ time on a pointer machine, and $O(n^3(\log\log n)/(w\log^{7/6} n))$ time on a word RAM.*

Proof. The algorithm builds on the ideas in Theorem 2.1 (the BMM algorithm using triangle removal), while applying the bilinear form preprocessing of Theorem 5.1, the algorithm for reporting edges in triangles (Theorem 4.1), and Weak Regularity. We first describe the algorithm for pointer machines.

Algorithm As in Theorem 2.1, by taking a random sample of $\sqrt{n}$ indices from $[n]$, we can determine those pairs (i, j) such that $(A \star B)[i, j] = 1$ where there are at least $n^{3/4}$ witnesses to this fact. This takes $O(n^{2.5})$ time and succeeds with probability $1 - \exp(-n^{\Omega(1)})$.

Next we construct a tripartite graph $G = (V_1,V_2,V_3,E)$ exactly as in Theorem 2.1, and just as before our goal is to determine all edges $(i, j) \in (V_1,V_3)$ that form at least one triangle with some vertex in V_2.

Compute an ε-pseudoregular partition $\{W_1, \dots, W_k\}$ of the bipartite subgraph (V_1,V_3), with $\varepsilon = 1/(\alpha\sqrt{\log n})$ for an $\alpha > 0$. By Theorem 3.6 this partition can be found in $2^{O(\alpha^2 \log n)}$ time. Set α to make the runtime $O(n^{2.5})$. Recall d_{ij} is the density of the pair (W_i,W_j). The preprocessing stores two data structures, one for pairs with "low" density and one for pairs with "high" density.

1. (Low Density Pairs) Let F be the set of all edges in (V_1,V_3) that lie in some pair (W_i,W_j), where $d_{ij} \le \sqrt{\varepsilon}$. Note $|F| \le \sqrt{\varepsilon} n^2$. Apply the algorithm of Theorem 4.1 to determine the subset of edges in F that participate in triangles. Remove the edges of F from G.

2. (High Density Pairs) For all pairs (W_i,W_j) with $d_{ij} > \sqrt{\varepsilon}$, build the data structure for computing bilinear forms (Theorem 5.1) for the submatrix A_{ij} corresponding to the graph induced by (W_i,W_j), with $\ell = \log^2 n$ and $\kappa = \log n/(5\log\log n)$.

Then for each vertex $v \in V_2$, let $S_i(v) = N(v) \cap W_i$, and $T_j(v) = N(v) \cap W_j$. Whenever (i, j) is a high density pair, compute all pairs of nodes in $S_i(v) \times T_j(v)$ that form a triangle with v, using the bilinear form query algorithm of Theorem 5.1.

Analysis Clearly, the random sampling step takes $O(n^{2.75})$ time. Consider the low density pairs step. Recall $|F| \le \sqrt{\varepsilon} n^2$ and every edge in (V_1, V_3) is in at most $n^{3/4}$ triangles. Moreover, the function $f(\delta) = \delta \log(1/\delta)$ is increasing for small δ (e.g., over $[0, 1/4]$). Hence the algorithm that reports all edges appearing in triangles (from Theorem 4.1) takes at most

$$O(\sqrt{\varepsilon} n^3 \log(1/\varepsilon)/\log^2 n) \le O(n^3 \log\log n/\log^{2.25} n)$$

time.

Now we bound the runtime of the high density pairs step. First note that the preprocessing for bilinear forms (Theorem 5.1) takes only

$$O\left(\frac{n^2}{\log^2 n} \cdot \left(\frac{\log^2 n}{\log n/(5\log\log n)}\right)^2\right) \le O\left(\frac{n^2}{\log^2 n} \cdot (\log^2 n)^{2\log n/(5\log\log n)}\right) = O(n^{2+4/5})$$

time overall.

Let $e(S,T)$ denote the number of edges between subsets S and T. Since there are $O(n^{2.75})$ triangles, we have that

$$\sum_{v \in V_2} e(N(v) \cap V_1, N(v) \cap V_3) \le n^{2.75}. \tag{5.3}$$

Since $\{W_i\}$ is ε-pseudoregular partition of (V_1, V_3), by Definition 3.5, for any vertex $v \in V_2$ we have that

$$\sum_{i,j} d_{ij} |S_i(v)||T_j(v)| - e(N(v) \cap V_1, N(v) \cap V_3) \le \varepsilon n^2.$$

Summing up over all vertices v, together with (5.3) implies that

$$\sum_{v \in V_2} \sum_{i,j} d_{ij} |S_i(v)||T_j(v)| \le \varepsilon n^3 + O(n^{2.75}) \le 2\varepsilon n^3$$

for large n. Summing over densities $d_{ij} \ge \sqrt{\varepsilon}$, we obtain

$$\sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \sqrt{\varepsilon}} |S_i(v)||T_j(v)| \le 2\sqrt{\varepsilon} n^3 \le \frac{2n^3}{\log^{0.25} n}. \tag{5.4}$$

Applying expression (5.1), the time taken by all queries on the data structure for bilinear forms (Theorem 5.1) for a fixed pair (W_i, W_j) is at most

$$\sum_{v \in V_2} \left(\frac{(n/k)}{\log^2 \frac{n}{k}} + \frac{|S_i(v)| \log\log \frac{n}{k}}{\log \frac{n}{k}}\right) \left(\frac{(n/k)}{\log^2 \frac{n}{k}} + \frac{|T_j(v)| \log\log \frac{n}{k}}{\log \frac{n}{k}}\right).$$

Expanding the products, summing up over the pairs (i,j) with $d_{ij} \ge \sqrt{\varepsilon}$, the total running time is bounded by

$$\sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \sqrt{\varepsilon}} \frac{|S_i(v)||T_j(v)|(\log\log n)^2}{\log^2(n/k)}$$

plus other terms with a total contribution of at most $O(n^3 \log\log(n/k)/\log^3(n/k))$. Thus by (5.4), the total runtime is upper bounded by $O(n^3(\log\log n)^2/\log^{2.25} n)$. Finally, the random sampling step ensures that the number of witnesses is at most $n^{0.75}$ for every edge, so the output cost in the algorithm is at most $O(n^{2.75})$.

Modification for the word RAM. To exploit a model with a larger wordsize, we apply the same algorithm as above, except we run the low density pairs step for pairs (W_i, W_j) with density $d_{ij} \le \varepsilon^{1/3}$ (instead of $\sqrt{\varepsilon}$). For the pairs (W_i, W_j) with $d_{ij} > \varepsilon^{1/3}$, construct the data structure for bilinear forms (Theorem 5.1) for the word RAM.

First we consider pairs (i, j) for which $d_{ij} < \varepsilon^{1/3}$. For these pairs, as above, we apply the processing step in Theorem 4.1 for reporting the edges appearing in triangles. This has a running time

$$O(\varepsilon^{1/3} n^3 \log(1/\varepsilon)/(w \log n)) \le O(n^3 \log\log n/(w \log^{7/6} n)).$$

Pairs (i, j) for which $d_{ij} > \varepsilon^{1/3}$, we use the data structure of Theorem 5.1 to answer the bilinear queries. By (5.2), the total running time for such pairs is

$$\begin{aligned} &\sum_{v \in V_2} \sum_{i,j: d_{ij} > \varepsilon^{1/3}} \left(\frac{\left(\frac{n}{k}\right)^2}{w \log^2 \frac{n}{k}} + \frac{\frac{n}{k} \cdot \min(|S_i(v)|, |T_j(v)|) \log\log \frac{n}{k}}{w \log(n/k)} \right) \\ \le\ & \frac{n^3}{w \log^2 \frac{n}{k}} + \sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \varepsilon^{1/3}} \frac{\frac{n}{k} \cdot \min(|S_i(v)|, |T_j(v)|) \log\log \frac{n}{k}}{w \log \frac{n}{k}}. \end{aligned} \tag{5.5}$$

To bound the second term above, observe that

$$\begin{aligned} &\sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \varepsilon^{1/3}} \min(|S_i(v)|, |T_j(v)|) \\ \le\ & \sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \varepsilon^{1/3}} (|S_i(v)| \cdot |T_j(v)|)^{1/2} \\ \le\ & k\sqrt{n} \cdot \sqrt{\sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \varepsilon^{1/3}} |S_i(v)||T_j(v)|} \\ \le\ & \frac{2kn^2 \log\log n}{(\log^{1/6} n)}. \end{aligned} \tag{5.6}$$

Here the second inequality follows by Cauchy-Schwarz and the last inequality follows as

$$\sum_{v \in V_2} \sum_{i,j: d_{ij} \ge \varepsilon^{1/3}} |S_i(v)||T_j(v)| \le 2\varepsilon^{2/3} n^3 \le \frac{2n^3}{\log^{1/3} n}$$

using an argument identical to that used to obtain (5.4).

By (5.6) the total running time in (5.5) is $O(n^3 \log\log n/(w \log^{7/6} n))$ as desired. □

6 Independent set queries via Weak Regularity

We consider the following *independent set query* problem. We want to preprocess an n-node graph in polynomial time and space, so that given any $S_1, \dots, S_w \subseteq V$, we can determine in $n^2/f(n)$ time which of $S_1, \dots, S_w$ are independent sets. Using such a subroutine, we can easily determine in $n^3/(wf(n))$ time if a graph has a triangle (provided the preprocessing itself can be done in $O(n^3/(wf(n)))$ time), by executing the subroutine on collections of sets corresponding to the neighborhoods of each vertex.

The independent set query problem is equivalent to: preprocess a Boolean matrix A so that w queries of the form "$v_j^T A v_j = 0$?" can be computed in $n^2/f(n)$ time, where the products are over the Boolean semiring. We shall solve a more general problem: preprocess A to answer w queries of the form "$u^T A v = 0$?", for arbitrary $u, v \in \{0,1\}^n$.

Our method employs weak regularity along with other combinatorial ideas seen earlier in the paper.

Theorem 6.1. *For all $\delta \in (0, 1/2)$, every $n \times n$ Boolean matrix A can be preprocessed in $O(n^{2+\delta})$ time such that given arbitrary Boolean vectors $u_1, \dots, u_{\log n}$ and $v_1, \dots, v_{\log n}$, we can determine if $u_p^T A v_p = 0$, for all $p = 1, \dots, \log n$ in*

$$O\left(\frac{n^2(\log\log n)^2}{\delta(\log n)^{5/4}}\right)$$

time on a pointer machine.

On the word RAM we can determine if $u_p^T A v_p = 0$, for all $p = 1, \dots, w$ in time

$$O\left(\frac{n^2(\log\log n)}{\delta(\log n)^{7/6}}\right)$$

where w is the wordsize.

Proof of Theorem 6.1. We describe the algorithm on the pointer machine; it can be extended to the word RAM by a modification identical to that in Theorem 5.2. We start with the preprocessing.

Preprocessing Interpret A as a bipartite graph in the natural way. Compute a ε-pseudoregular partition of the bipartite $A = (V, W, E)$ with $\varepsilon = \Theta(1/\sqrt{\log n})$, using Theorem 3.6. (Note that this is the only randomized part of the algorithm.) Let $V_1, V_2, \dots, V_k$ be the parts of V and let $W_1, \dots, W_k$ be the parts of W, where $k \le 2^{O(1/\varepsilon^2)}$.

Let A_{ij} be the submatrix of A corresponding to the subgraph induced by the pair (V_i, W_j). Let d_{ij} be the density of (V_i, W_j). Let $\Delta = \sqrt{\varepsilon}$.

For each of the k^2 submatrices A_{ij}, do the following:

1. If $d_{ij} \le \Delta$, apply the graph compression of Theorem 3.10 to preprocess A_{ij} in time $mn^\delta \log^2 n$, so that (using Lemma 3.11) the submatrix A_{ij} can be multiplied by any $n/k \times \log n$ matrix B in time
$$O\left(\frac{m\log((n/k)^2/m)}{\log(n/k)}\right),$$
where m is the number of nonzeros in A_{ij}. (Note that $m \le \Delta(n/k)^2$.)

2. If $d_{ij} > \Delta$, apply the bilinear form preprocessing of Theorem 5.1 to A_{ij} with $\ell = \log^2 n$ and $\kappa = \delta \log n/(5 \log\log n)$.

Query Algorithm Given Boolean vectors u_p and v_p for $p = 1,\dots,\log n$, let $S^p \subseteq [n]$ be the subset corresponding to u_p and $T^p \subseteq [n]$ be the subset corresponding to v_p. For $1 \le i, j \le k$, let $S_i^p = S^p \cap V_i$ and $T_j^p = T^p \cap W_j$.

1. Compute the estimate $Q^p = \sum_{i,j=1}^k d_{ij}|S_i^p||T_j^p|$ for all $p = 1,\dots,\log n$. If $Q^p > \varepsilon n^2$, then output $u_p^T A v_p = 1$.
2. Let $I = \{p : Q^p \le \varepsilon n^2\}$. Note that $|I| \le \log n$. We determine $u_p^T A v_p$ for each $p \in I$ as follows:
   - For all (i, j) with $d_{ij} > \Delta$, apply the bilinear form algorithm of Theorem 5.1 to compute $e_{ij}^p = (S_i^p)^T A_{ij} T_j^p$ for each $p \in I$.
   - For all (i, j) with $d_{ij} \le \Delta$, form an $\frac{n}{k} \times |I|$ matrix B_j with columns T_j^p over all $p \in I$. Compute $C_{ij} = A_{ij} \star B_j$ using the A_{ij} from preprocessing step 1. For each $p \in I$, compute the (Boolean) dot product $e_{ij}^p = (S_i^p)^T \cdot C_{ij}^p$, where C_{ij}^p is the p-th column of C_{ij}.
   - For each $p \in I$, return $u_p^T A v_p = \bigvee_{i,j} e_{ij}^p$.

Analysis. We first consider the preprocessing time. By Theorem 3.6, we can choose ε so that the ε-pseudoregular partition is constructed in $O(n^{2+\delta})$ time. By Theorems 3.9 and 5.1, the preprocessing for matrices A_{ij} takes at most $O(k^2(n/k)^{2+\delta})$ time for some $\delta < 1/2$. Thus, the total time is at most $O(n^{2+\delta})$.

We now analyze the query algorithm. Notice that step 1 of the query algorithm works due to ε-pseudoregularity: if $Q^p > \varepsilon n^2$ then the number of edges between S^p and T^p in A is greater than 0. Computing all Q^p takes time at most $O(k^2 n \log n)$.

Consider the second step. As $\sum_{i,j} d_{ij}|S_i^p||T_j^p| \le \varepsilon n^2$ for each $p \in I$, we have

$$\sum_{i,j:d_{ij}\ge\Delta} |S_i^p||T_j^p| \le \frac{\varepsilon n^2}{\Delta} = \sqrt{\varepsilon} n^2 . \tag{6.1}$$

Analogously to Theorem 5.2, the total runtime over all $p \in I$ and pairs (i, j) with $d_{ij} > \Delta$ is at most

$$\begin{aligned} &\sum_{p\in I}\sum_{i,j:d_{ij}>\Delta} \left(\frac{n/k}{\log^2 \frac{n}{k}} + \frac{|S_i^p| \log\log \frac{n}{k}}{\log \frac{n}{k}} \right) \cdot \left(\frac{n/k}{\log^2 \frac{n}{k}} + \frac{|T_j^p| \log\log \frac{n}{k}}{\log \frac{n}{k}} \right) \\ \le\ & O\left(\frac{n^2 \log\log n}{\log^3 n} + \sum_{p\in I}\sum_{i,j:d_{ij}>\Delta} \frac{|S_i^p||T_j^p|(\log\log n)^2}{\log^2 n} \right) . \end{aligned} \tag{6.2}$$

The inequality (6.1), the fact that $|I| \le \log n$, and our choice of ε imply that (6.2) is at most

$$O(n^3 (\log\log n)^2 / \log^{5/4} n) .$$

Now we consider the pairs (i, j) with $d_{ij} \le \Delta$. By Theorem 3.9, computing the product $C_{ij} = A_{ij} B_j$ for all $p \in I$ (at once) takes

$$O\left(\frac{\Delta \left(\frac{n}{k}\right)^2 \log(1/\Delta)}{\log(n/k)} \right)$$

time. Summing over all relevant pairs (i,j) (there are at most k^2), this is $O(n^2(\log\log n)/\log^{5/4} n)$ by our choice of Δ.

The extension to the word RAM model follows by a modification identical to that in Theorem 5.2. □

7 Conclusion

We have shown how regularity concepts can be applied to yield faster combinatorial algorithms for fundamental graph problems. These results hint at an alternative line of research on Boolean matrix multiplication that has been unexplored. It is likely that the connections are deeper than we know; let us give a few reasons why we believe this.

First, we applied generic tools that are probably stronger than necessary, so it should be profitable to search for regularity concepts that are designed with matrix multiplication in mind (see the last paragraph for more details). Secondly, Trevisan [58] has promoted the question of whether or not the Triangle Removal Lemma requires the full Regularity Lemma. Our work gives a rather new motivation for this question, and opens up the possibility that BMM may be related to other combinatorial problems as well. As mentioned earlier, Jacob Fox [26] has recently proved a sharper Triangle Removal Lemma, but it remains to be seen if his argument can be applied to solve BMM faster.

Another interesting direction is to explore different algebraic structures. There may be similar algorithms for matrix products over finite fields or the $(\min,+)$-semiring. These algorithms would presumably apply different removal lemmas, but such lemmas are known to exist. For instance, Shapira [54] (and independently, Král, Serra, and Vena [39]) recently proved the following, generalizing a result of Green [33]. Let $Ax = b$ be a set of linear equations over a finite field F, with n variables and m equations. If $S \subseteq F$ has the property that there are only $o(|F|^{n-m})$ solutions in S^n to $Ax = b$, then $o(|F|)$ elements can be removed from S so that the resulting S^n has no solutions to $Ax = b$. In light of our work, results such as this are possible tools for finite field linear algebra with combinatorial algorithms.

Finally, Vassilevska Williams and Williams [61] have recently shown how to derandomize the algorithms of this paper. By exploiting the structure of BMM itself, one can show that any *polynomial time* algorithm for computing a Weak Regularity partition can be applied to solve BMM faster; hence the deterministic algorithm of Alon and Naor [6] suffices. In fact, the paper [61] proves that an $n^{3-\varepsilon}$ time algorithm for BMM follows from an $n^{3-3\varepsilon}$ time algorithm for triangle detection in n-node graphs, and stronger relations hold for $n^3/\text{poly}(\log n)$ runtimes. It is likely that faster triangle detection does not require a partitioning as "strong" as Weak Regularity. We should not have to preprocess a graph so that *all* independent set queries are fast; we only have to preprocess so that a given collection of n independent set queries are fast. That is, triangle detection only requires that we query the n neighborhoods of the n vertices in the graph. We believe that further progress on alternative algorithms for BMM can be made by continuing to study the problem in a graph-theoretic way.

Acknowledgements

We thank Avrim Blum for suggesting the independent set query problem, which led us to this work. We also thank the anonymous referees and the program committee for helpful comments. This work was conducted while N.B. worked at IBM Watson, and R.W. worked at the Institute for Advanced Study and IBM Almaden. R.W. was supported by NSF Grant CCF-0832797 (Expeditions in Computing) at the Institute for Advanced Study, Princeton, NJ, and the Josef Raviv Memorial Fellowship at IBM.

References

[1] DONALD AINGWORTH, CHANDRA CHEKURI, PIOTR INDYK, AND RAJEEV MOTWANI: Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. Preliminary version in SODA'96. [doi:10.1137/S0097539796303421] 71

[2] MARTIN ALBRECHT, GREGORY BARD, AND WILLIAM HART: Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Trans. Math. Softw.*, 37(1), 2010. [doi:10.1145/1644001.1644010] 70

[3] NOGA ALON, RICHARD A. DUKE, HANNO LEFMANN, VOJTECH RÖDL, AND RAPHAEL YUSTER: The algorithmic aspects of the regularity lemma. *J. Algorithms*, 16(1):80–109, 1994. Preliminary version in FOCS'92. [doi:10.1006/jagm.1994.1005] 74

[4] NOGA ALON, ELDAR FISCHER, MICHAEL KRIVELEVICH, AND MARIO SZEGEDY: Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. Preliminary version in FOCS'99. [doi:10.1007/s004930070001] 70, 76

[5] NOGA ALON, ELDAR FISCHER, ILAN NEWMAN, AND ASAF SHAPIRA: A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM J. Comput.*, 39(1):143–167, 2009. Preliminary version in STOC'06. [doi:10.1137/060667177] 70

[6] NOGA ALON AND ASSAF NAOR: Approximating the cut-norm via Grothendieck's inequality. *SIAM J. Comput.*, 35(4):787–803, 2006. Preliminary version in STOC'04. [doi:10.1137/S0097539704441629] 88

[7] DANA ANGLUIN: The four Russians' algorithm for Boolean matrix multiplication is optimal for its class. *SIGACT News*, 1:29–33, 1976. [doi:10.1145/1008591.1008593] 72

[8] V. Z. ARLAZAROV, E. A. DINIC, M. A. KRONROD, AND I. A. FARADZHEV: On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, 11(5):1209–1210, 1970. 70, 71

[9] MICHAEL D. ATKINSON AND NICOLA SANTORO: A practical algorithm for Boolean matrix multiplication. *Inf. Process. Lett.*, 29:37–38, 1988. [doi:10.1016/0020-0190(88)90130-5] 70

[10] JULIEN BASCH, SANJEEV KHANNA, AND RAJEEV MOTWANI: On diameter verification and Boolean matrix multiplication, 1995. Technical Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University. 70, 71

[11] F. A. BEHREND: On sets of integers which contain no three terms in arithmetic progression. *Proc. Nat. Acad. Sci.*, 32(12):331–332, 1946. 72

[12] ARNAB BHATTACHARYYA, VICTOR CHEN, MADHU SUDAN, AND NING XIE: Testing linear-invariant non-linear properties. *Theory of Computing*, 7:75–99, 2011. Preliminary version in STACS'06. [doi:10.4086/toc.2011.v007a006] 70

[13] GUY E. BLELLOCH, VIRGINIA VASSILEVSKA, AND RYAN WILLIAMS: A new combinatorial approach for sparse graph problems. In *Proc. 35th Internat. Colloq. on Automata, Languages and Programming (ICALP'08)*, pp. 108–120, 2008. [doi:10.1007/978-3-540-70575-8_10] 71, 77

[14] AVRIM BLUM: 2009. Personal communication. 73

[15] CHRISTIAN BORGS, JENNIFER T. CHAYES, LÁSZLÓ LOVÁSZ, VERA T. SÓS, BALÁZS SZEGEDY, AND KATALIN VESZTERGOMBI: Graph limits and parameter testing. In *Proc. 38th STOC*, pp. 261–270. ACM Press, 2006. [doi:10.1145/1132516.1132556] 70

[16] TIMOTHY M. CHAN: More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. 39th STOC*, pp. 590–598. ACM Press, 2007. [doi:10.1145/1250790.1250877] 71

[17] SIDDHARTHA CHATTERJEE, ALVIN R. LEBECK, PRAVEEN K. PATNALA, AND MITHUNA THOTTETHODI: Recursive array layouts and fast matrix multiplication. *IEEE Trans. on Parallel and Distributed Systems*, 13:1105–1123, 2002. [doi:10.1109/TPDS.2002.1058095] 70

[18] HENRY COHN, ROBERT D. KLEINBERG, BALÁZS SZEGEDY, AND CHRISTOPHER UMANS: Group-theoretic algorithms for matrix multiplication. In *Proc. 46th FOCS*, pp. 379–388. IEEE Comp. Soc. Press, 2005. [doi:10.1109/SFCS.2005.39] 70

[19] HENRY COHN AND CHRISTOPHER UMANS: A group-theoretic approach to fast matrix multiplication. In *Proc. 44th FOCS*, pp. 438–449. IEEE Comp. Soc. Press, 2003. [doi:10.1109/SFCS.2003.1238217] 70

[20] AMIN COJA-OGHLAN, COLIN COOPER, AND ALAN M. FRIEZE: An efficient sparse regularity concept. *SIAM J. Discrete Math.*, 23(4):2000–2034, 2010. [doi:10.1137/080730160] 70

[21] DORIT DOR, SHAY HALPERIN, AND URI ZWICK: All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000. Preliminary version in FOCS'96. [doi:10.1137/S0097539797327908] 70, 71

[22] RICHARD A. DUKE, HANNO LEFMANN, AND VOJTECH RÖDL: A fast approximation algorithm for computing the frequencies of subgraphs in a given graph. *SIAM J. Comput.*, 24(3):598–620, 1995. [doi:10.1137/S0097539793247634] 76

[23] MICHAEL ELKIN: An improved construction of progression-free sets. In *Proc. 21st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'10)*, pp. 886–905, 2010. 72

[24] TOMÁS FEDER AND RAJEEV MOTWANI: Clique partitions, graph compression and speeding-up algorithms. *J. Comput. System Sci.*, 51(2):261–272, 1995. Preliminary version in STOC'91. [doi:10.1006/jcss.1995.1065] 71, 77

[25] MICHAEL J. FISCHER AND ALBERT R. MEYER: Boolean matrix multiplication and transitive closure. In *Proc. 12th FOCS (SWAT'71)*, pp. 129–131. IEEE Comp. Soc. Press, 1971. [doi:10.1109/SWAT.1971.4] 70

[26] JACOB FOX: A new proof of the graph removal lemma. *Ann. of Math.*, 174(1):561–579, 2011. [doi:10.4007/annals.2011.174.1.17] 72, 76, 88

[27] ALAN FRIEZE AND RAVI KANNAN: A simple algorithm for constructing Szemerédi's regularity partition. *Electr. J. Comb.*, 6, 1999. 74

[28] ALAN M. FRIEZE AND RAVI KANNAN: The Regularity Lemma and approximation schemes for dense problems. In *Proc. 37th FOCS*, pp. 12–20. IEEE Comp. Soc. Press, 1996. [doi:10.1109/SFCS.1996.548459] 70, 72, 75

[29] ALAN M. FRIEZE AND RAVI KANNAN: Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999. [doi:10.1007/s004930050052] 70, 72, 74, 75, 76

[30] ANKA GAJENTAAN AND MARK H. OVERMARS: On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry*, 5:165–185, 1995. [doi:10.1016/0925-7721(95)00022-2] 73

[31] ZVI GALIL AND ODED MARGALIT: All pairs shortest distances for graphs with small integer length edges. *Inform. and Comput.*, 134:103–139, 1997. [doi:10.1006/inco.1997.2620] 70

[32] W. T. GOWERS: Lower bounds of tower type for Szemerédi's uniformity lemma. *Geom. and Funct. Anal.*, 7:322–337, 1997. [doi:10.1007/PL00001621] 72

[33] BEN GREEN: A Szemerédi-type regularity lemma in abelian groups. *Geom. and Funct. Anal.*, 15:340–376, 2005. [doi:10.1007/s00039-005-0509-8] 71, 75, 88

[34] ANDRÁS HAJNAL, WOLFGANG MAASS, AND GYÖRGY TURÁN: On the communication complexity of graph properties. In *Proc. 20th STOC*, pp. 186–191. ACM Press, 1988. [doi:10.1145/62212.62228] 70

[35] ALON ITAI AND MICHAEL RODEH: Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. [doi:10.1137/0207033] 70

[36] Y. KOHAYAKAWA: Szemerédi's regularity lemma for sparse graphs. In F. CUCKER AND M. SHUB, editors, *Foundations of Computational Mathematics*, pp. 216–230. Springer, 1997. 76

[37] YOSHIHARU KOHAYAKAWA, VOJTECH RÖDL, AND LUBOS THOMA: An optimal algorithm for checking regularity. *SIAM J. Comput.*, 32(5):1210–1235, 2003. [doi:10.1137/S0097539702408223] 74, 75

[38] JÁNOS KOMLÓS AND MIKLÓS SIMONOVITS: Szemerédi's Regularity Lemma and its applications in graph theory. In *Combinatorics, Paul Erdős is Eighty, (D. Miklós et. al, eds.), Bolyai Society Mathematical Studies*, volume 2, pp. 295–352, 1996. 76

[39] DANIEL KRÁL, ORIOL SERRA, AND LLUÍS VENA: A removal lemma for systems of linear equations over finite fields. *Israel J. Math.*, 187(1):193–207, 2012. [doi:10.1007/s11856-011-0080-y] 88

[40] LILLIAN LEE: Fast context-free grammar parsing requires fast Boolean matrix multiplication. *J. ACM*, 49:1–15, 2002. [doi:10.1145/505241.505242] 70, 71

[41] ANDRZEJ LINGAS: A geometric approach to Boolean matrix multiplication. In *Proc. 13th Int. Symp. on Algorithms and Computation (ISAAC'02)*, pp. 501–510, 2002. [doi:10.1007/3-540-36136-7_44] 71

[42] LÁSZLÓ LOVÁSZ AND BALÁZS SZEGEDY: Szemerédi's theorem for the analyst. *Geom. and Funct. Anal.*, 17(1):252–270, 2007. [doi:10.1007/s00039-007-0599-6] 76

[43] J. W. MOON AND L. MOSER: A matrix reduction problem. *Mathematics of Computation*, 20:328–330, 1966. [doi:10.1090/S0025-5718-66-99935-2] 70, 71

[44] PATRICK E. O'NEIL AND ELIZABETH J. O'NEIL: A fast expected time algorithm for Boolean matrix multiplication and transitive closure matrices. *Inf. Control*, 22:132–138, 1973. [doi:10.1016/S0019-9958(73)90228-3] 71

[45] VICTOR Y. PAN: *How to Multiply Matrices Faster*. Volume 179 of *Lecture Notes in Computer Science*. Springer, 1984. 70

[46] MIHAI PATRASCU: Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd STOC*, pp. 603–610. ACM Press, 2010. [doi:10.1145/1806689.1806772] 73

[47] LIAM RODITTY AND URI ZWICK: On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2010. Preliminary version in 12th Europ. Symp. Algor. (ESA'04). [doi:10.1007/s00453-010-9401-5] 71

[48] VOJTĚCH RÖDL AND MATHIAS SCHACHT: Property testing in hypergraphs and the removal lemma. In *Proc. 39th STOC*, pp. 488–495. ACM Press, 2007. [doi:10.1145/1250790.1250862] 70

[49] I. Z. RUZSA AND E. SZEMERÉDI: Triple systems with no six points carrying three triangles. *Colloquia Mathematica Societatis János Bolyai*, 18:939–945, 1978. 71, 72, 75

[50] WOJCIECH RYTTER: Fast recognition of pushdown automaton and context-free languages. *Inf. Control*, 67:12–22, 1985. Preliminary version in MFCS'84. [doi:10.1016/S0019-9958(85)80024-3] 70, 71

[51] JOHN E. SAVAGE: An algorithm for the computation of linear forms. *SIAM J. Comput.*, 3:150–158, 1974. [doi:10.1137/0203011] 70, 72

[52] C.-P. SCHNORR AND C. R. SUBRAMANIAN: Almost optimal (on the average) combinatorial algorithms for Boolean matrix product witnesses, computing the diameter. In *Proc. 2nd Intern. Workshop Random. and Approx. Tech. Comput. Sci. (RANDOM'98)*, pp. 218–231, 1998. [doi:10.1007/3-540-49543-6_18] 71

[53] RAIMUND SEIDEL: On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. System Sci.*, 51(3):400–403, 1995. [doi:10.1006/jcss.1995.1078] 70

[54] ASAF SHAPIRA: Green's conjecture and testing linear-invariant properties. In *Proc. 41st STOC*, pp. 159–166. ACM Press, 2009. [doi:10.1145/1536414.1536438] 88

[55] AVI SHOSHAN AND URI ZWICK: All pairs shortest paths in undirected graphs with integer weights. In *Proc. 40th FOCS*, pp. 605–614. IEEE Comp. Soc. Press, 1999. [doi:10.1109/SFFCS.1999.814635] 70

[56] VOLKER STRASSEN: Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. [doi:10.1007/BF02165411] 70

[57] ENDRE SZEMERÉDI: Regular partitions of graphs. In *Proc. Colloque Inter. CNRS (J. C. Bermond, J. C. Fournier, M. Las Vergnas and D. Sotteau, eds.)*, pp. 399–401, 1978. 71, 74

[58] LUCA TREVISAN: Additive combinatorics and theoretical computer science. In *SIGACT News Complexity Column 63*, 2009. 88

[59] LESLIE G. VALIANT: General context-free recognition in less than cubic time. *J. Comput. System Sci.*, 10(2):308–315, 1975. [doi:10.1016/S0022-0000(75)80046-8] 70

[60] VIRGINIA VASSILEVSKA WILLIAMS: Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th STOC*, 2012. To appear. 70

[61] VIRGINIA VASSILEVSKA WILLIAMS AND RYAN WILLIAMS: Subcubic equivalences between path, matrix, and triangle problems. In *Proc. 51st FOCS*, pp. 645–654. IEEE Comp. Soc. Press, 2010. [doi:10.1109/FOCS.2010.67] 88

[62] RYAN WILLIAMS: Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In *Proc. 18th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'07)*, pp. 995–1001. ACM Press, 2007. 70, 71, 73

AUTHORS

Nikhil Bansal
Eindhoven University of Technology
Eindhoven, Netherlands
n.bansal@tue.nl

Ryan Williams
Stanford University
Stanford, CA USA
rrwilliams@gmail.com

ABOUT THE AUTHORS

NIKHIL BANSAL graduated from CMU in 2003. His advisor was Avrim Blum. His research has focused on approximation and online algorithms for scheduling and other optimization problems.

RYAN WILLIAMS graduated from CMU in 2007. His advisor was Manuel Blum, who tried for years to get Ryan to help him formulate a nontrivial theory of machine consciousness. Failing miserably, Ryan turned to easier problems. Ryan is from rural Alabama, where squirrel meat is a delicacy and farming is not an exotic profession.