

Regularity Lemmas and Combinatorial Algorithms

Nikhil Bansal*
IBM T.J. Watson Research Center
Yorktown Heights, NY
nikhil@us.ibm.com

Ryan Williams†
IBM Almaden Research Center
San Jose, CA
ryanw@ias.edu

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 $O((n^3 \log \beta)/(\beta w \log n))$ time algorithm for BMM where $\beta = (\log^* n)^\delta$ 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 $O(n^3(\log \log n)^2/(\log n)^{9/4})$ time with probability exponentially close to 1. When $w \geq \log n$, it can be implemented in $O(n^3(\log \log n)^2/(w \log n)^{7/6})$ time. Our results immediately imply improved combinatorial methods for CFG parsing, detecting triangle-freeness, and transitive closure.
- 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 $O(n^2(\log \log n)^2/((\log n)^{5/4}))$ time. When $w \geq \log n$, w queries can be answered in $O(n^2(\log \log n)^2/((\log n)^{7/6}))$ time. In addition to its nice 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], [6], [11], [41], [12], approximation algorithms [24], [25], [17], and communication complexity [30]. In this paper

†Research performed while the author was a member of the Institute for Advanced Study, Princeton, NJ. Supported by NSF Grant CCF-0832797 (Expeditions in Computing) at IAS, and the Josef Raviv Memorial Fellowship at IBM.

we show how regularity can lead to faster combinatorial algorithms for basic problems.

Boolean matrix multiplication (BMM) is one of the most fundamental problems in computer science. It is a key subroutine in the solution of many other problems such as transitive closure, context-free grammar parsing, all-pairs path problems, and triangle detection. Roughly speaking, there have been two lines of theoretical research on the problem. Algebraic algorithms, beginning with Strassen’s $\tilde{O}(n^{\log_2 7})$ algorithm [46] and ending (so far) with Coppersmith and Winograd’s $\tilde{O}(n^{2.376})$ algorithm [20], 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 finite 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. [39] for an overview of early work, and [18], [19] 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 make them less desirable in practice.¹

The second line of work on matrix multiplication has studied so-called “combinatorial” algorithms. Generally speaking, these algorithms exploit combinatorial redundancies that arise from construing matrices as graphs, often invoking word parallelism and lookup tables.² They are considered more practical, but fewer advances have been made; all algorithms for the dense case [37], [8], [44], [9], [43], [10], [49] are loosely based on the “Four Russians” approach of Arlazarov *et al.* [8] from 1970,³ which runs

¹For 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. [34], [16], [2].

²We would like to give a definition of “combinatorial algorithm”, but this appears elusive. Although the term has been used in many of the cited references, nothing in the literature resembles a definition. For the purposes of this paper, let us think of a “combinatorial algorithm” simply as one that does not call an oracle for ring matrix multiplication.

³*Historical Note:* Similar work of Moon and Moser [37] from 1966 shows that the inverse of a matrix over $GF(2)$ needs exactly $\Theta(n^2/\log n)$ row operations, providing an upper and lower bound. On a RAM, their algorithm runs in $O(n^3/(w \log n))$ time.

in $O(n^3/(w \log n))$ on modern computational models.⁴ To date, this is still the fastest known combinatorial algorithm for dense matrices.

Many works (including [1], [21], [34], [45], [40], [35], [15]) 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 certain 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 is combinatorial and runs in $O(n^3 \text{poly}(\log \log n)/\log^2 n)$ time [15] – 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 [23], [13]. See [38], [45], [35] for a sampling of other partial 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. A Triangle Removal Lemma [42], [28] states that there is a function f satisfying $\lim_{\varepsilon \rightarrow 0} f(\varepsilon) = 0$ 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 [47]. We prove that good removal lemmas imply faster Boolean matrix multiplication. Let w be the wordsize (typically $w = \Theta(\log n)$).

Theorem 1.1 *Suppose there is an $O(T(n))$ time algorithm that, for every graph $G = (V, E)$ with at most εn^3 triangles, returns a set $S \subseteq E$ with $|S| \leq f(\varepsilon)n^2$ 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 $O\left(T(n) + \frac{f(\varepsilon)n^3 \log(1/f(\varepsilon))}{w \log n} + \frac{n^2}{\varepsilon} \cdot \log n + \varepsilon n^3\right)$ time.*

Unfortunately the best known upper bound for f is $f(\varepsilon) = O(1/(\log^* 1/\varepsilon)^\delta)$ for some $\delta > 0$ (cf. Section 2.1). 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 [29]) 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 [42].

Our second algorithm for BMM gives a more concrete improvement, relying on the Weak Regularity Lemma of Frieze and Kannan [24], [25] along with several other combinatorial ideas.

Theorem 1.2 *There is a combinatorial algorithm for Boolean matrix multiplication in $\hat{O}(n^3/(\log^{2.25} n))$ (worst-case) expected time on a pointer machine.⁵ More precisely, for any $n \times n$ Boolean matrices A and B , the algorithm computes their Boolean product with probability exponentially close to 1, 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 [44], 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.⁶

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 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 $O(n^2/\log^2 n)$ [49] (or $O(n^2/(w \log n))$ with wordsize w).

Theorem 1.3 *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. The independent set query problem has several

⁵The \hat{O} notation suppresses $\text{poly}(\log \log n)$ factors.

⁶More 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.

⁴The algorithm in [8] was originally stated to run in $O(n^3/\log n)$ time. At a high level, it partitions the first matrix into $n \times \varepsilon \log n$ submatrices, and the second matrix into $\varepsilon \log n \times n$ submatrices. Each $n \times \varepsilon \log n$ submatrix is treated as a function from $\varepsilon \log n$ bits to n bits; this function is stored in a table for direct access. Each table has n^ε entries, and n bits in each entry. 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 outputs of the function 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. [43], [10], [49].

interesting applications. For one, Theorem 1.3 immediately implies a faster triangle detection algorithm. (A graph is triangle-free if and only if all vertex neighborhoods are independent sets.) The query problem can also model a special case of partial match retrieval, and the problem of preprocessing a 2-CNF formula F in order to evaluate F on arbitrary assignments a quickly [14]. It can also be used to solve a query version of the well-known 3-SUM problem [27]. 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 can't be solved much faster than the trivial $O(n^2)$ bound is used to show hardness for computational geometry problems [27]. We can define a query version as: given two sets A and B of n elements 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 . If we make a graph with a node for each element in A , and an edge between two elements in A if their sum is an element in B , this gives exactly the independent set query problem [14].

2. 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 $A * B$ is the (Boolean) matrix product over the Boolean semiring.

Since the running times of our algorithms involve poly-logarithmic 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. 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.

2.1. 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 2.1 A partition $\{V_1, \dots, V_k\}$ of V is an ε -regular partition of G if $|V_i| \leq \varepsilon|V|$ for all i , $||V_i| - |V_j|| \leq 1$ for

all i, j , and all but at most εk^2 of the pairs (V_i, V_j) are ε -regular.

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

Lemma 2.1 (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], [26], [25], [32]. 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 [32].

Theorem 2.1 (Kohayakawa-Rödl-Thoma [32], Cor. 1.6) There is an algorithm that, on input $\varepsilon > 0$ and graph G on n nodes, outputs an ε -regular partition in $K'(\varepsilon)$ 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 $\varepsilon' = (\varepsilon^{20}/10^{24})$.

We also need the Triangle Removal Lemma, first stated by Ruzsa and Szemerédi [42]. In one formulation, the lemma says there is a function f such that $f(\varepsilon) \rightarrow 0$ as $\varepsilon \rightarrow 0$, 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 ([28], Proposition 1.3).

Lemma 2.2 (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 $4\varepsilon k^{-3} > \delta$. Then there is a set of at most $O(\varepsilon n^2)$ edges such that their removal makes G triangle-free.

In particular, let $\{V_1, \dots, V_k\}$ 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.

Notice that the lemma gives an effective way of discovering the 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 $c/(\log^* 1/\varepsilon)^\delta$ for some constants $c > 1$ and $\delta > 0$. It is of great interest to prove a triangle removal lemma with much smaller $f(\varepsilon)$.

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 [25]. There are also other variants known, for example [31], [4], [22]. We refer the reader to the survey [33]. Frieze and Kannan defined the following notion of a pseudoregular partition.

Definition 2.2 (ε -pseudoregular partition) Let $\mathcal{P} = V_1, \dots, V_k$ be a partition of V , and let d_{ij} be the

density of (V_i, V_j) . For a subset $S \subseteq V$, and $i = 1, \dots, k$, 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 , $||V_i| - |V_j|| \leq 1$.

Theorem 2.2 (Frieze-Kannan [25], Thm 2 and Sec 5.1) For all $\varepsilon \geq 0$, an equitable ε -pseudoregular partition of an n node graph with at most $\min\{n, 2^{\lceil 64/(3\varepsilon^2) \rceil}\}$ parts can be constructed in $O(2^{O(1/\varepsilon^2)} \frac{n^2}{\varepsilon^2 \delta^3})$ 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 Lovasz and Szegedy [36] have proven that for any ε -pseudoregular partition, the number of parts must be at least $1/4 \cdot 2^{1/(8\varepsilon)}$.

2.2. 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 2.3 (Blleloch-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 $O(n^2 \kappa / \ell \cdot \sum_{b=1}^{\kappa} \binom{\ell}{\kappa})$ preprocessing time, so that for any Boolean vector v , the product Bv can be computed in $O(n \log n + \frac{n^2}{\ell w} + \frac{nt}{\kappa w})$ 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 $n/\ell = t/\kappa$, and $\sum_{b=1}^{\kappa} \binom{\ell}{\kappa} = n^\delta$ for some $\delta > 0$. Letting $\kappa = \delta \ln(n) / \ln(en/t)$ and $\ell = \kappa \cdot en/t$ we obtain:

Theorem 2.4 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 Bv can be computed in $O(n \log n + \frac{nt \ln(en/t)}{\delta w \ln n})$ 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 2.5 There is an $O(mn \log(n^2/m)/(w \log n))$ time algorithm for computing $A * B$, 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 2.3 (From Feder-Motwani [23], Thm 3.3) Let $\delta \in (0, 1)$ be constant. We can write any $n \times n$ Boolean matrix A with m nonzeros as $A = (C * D) \vee E$ 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 [23], let us sketch the proof for completeness. Using Ramsey theoretic arguments, 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 $(C * D) \vee E$, by having the columns of C (and rows of D) correspond to the bipartite cliques. Set $C[i, k] = 1$ iff the i th node of the LHS of G is in the k th bipartite clique, and similarly set D for the nodes on the RHS of G . Note 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 2.4 (Folklore) There is an $O(mn/w + pq + pn)$ time algorithm for computing $A * B$, for every $p \times q$ A and $q \times n$ 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 j th row of B and C_i be the i th 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. ■

3. COMBINATORIAL BOOLEAN MATRIX MULTIPLICATION VIA TRIANGLE REMOVAL

In this section, we prove Theorem 1.1. That is, we show 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 graph⁷. 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.

Let \mathcal{C} be a collection of sets over a universe U . A set $R \subseteq U$ is an ε -net for \mathcal{C} if for all $S \in \mathcal{C}$ with $|S| \geq \varepsilon|U|$, $R \cap S \neq \emptyset$. The following lemma is well known.

Lemma 3.1 *Let \mathcal{C} be a collection of sets over a universe U . A random sample $R \subseteq U$ of size $\frac{3 \ln |\mathcal{C}|}{\varepsilon}$ is an ε -net with probability at least $1 - |\mathcal{C}|^{-2}$.*

We now describe our algorithm for BMM.

Algorithm: Let A and B be $n \times n$ matrices. We want $D = A * B$, i.e. $D[i, j] = (\bigvee_{k=1}^n A[i, k] \wedge B[k, j])$.

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) , $1 \leq i, j \leq n$, 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 C is dominated by the desired D , in that $C[i, j] \leq D[i, j]$ for all i, j . Let $S_{i,j}$ be the set of witnesses for (i, j) . By Lemma 3.1, R is an ε -net for the collection $\{S_{i,j}\}$ with probability at least $1 - 1/n^4$. Hence we may assume $C[i, j] = D[i, j] = 1$ for every (i, j) with at least εn witnesses.

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 \bar{C} , the complement of C . Observe $(i, k, j) \in (V_1, V_2, V_3)$ 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, we can find in time $O(T(n))$ a

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

set of edges F where $|F| \leq f(\varepsilon)n^2$ 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 \bar{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_C) denote the set of triangles with an edge in A_F (resp. B_F and \bar{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 $C_1 = A_F * B$ and $C_2 = A * B_F$, respectively. As A_F and B_F are sparse, by Theorem 2.5, these products can be computed in $O(|F| \log(n^2/|F|)/(w \log n))$ time. The 1-entries of $\bar{C} \wedge C_1$ (resp. $\bar{C} \wedge C_2$) 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 $C' = C \vee (C_1 \wedge \bar{C}) \vee (C_2 \wedge \bar{C})$, and update \bar{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 \bar{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 3.1 *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 $O(\delta n^3 \log(1/\delta)/(w \log n) + n^2 t)$ time.*

Setting $\delta = f(\varepsilon)$ and $t = \varepsilon n$, Theorem 3.1 implies the desired time bound in Theorem 1.1. The idea of the proof of Theorem 3.1 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 3.1: 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'_3 is obtained by partitioning the nodes of V_3 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'_3 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 $\ell = \gamma(\log n)/(\delta \log(1/\delta))$. For each group, we replace it by

n^γ nodes, one corresponding to each subset of size up to $\kappa = \gamma(\log n)/(\log(1/\delta))$. Simple combinatorics show this is possible, and that 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 $O(\log^2 n)$.

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, \dots, s_t each of size at most κ . Put edges from v to s_1, \dots, 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 $O(n^2 t)$. 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: Finally we show how to replace a log-speedup by a w -speedup with wordsize w . In the above, each node in V_1 and V'_3 has $n/(\gamma \log n)$ edges to nodes in V'_2 , and these edges specify an n -bit vector. The idea is to simply replace these edge sets to V'_2 with ordered sets of n/w words, each holding a w -bit string. Each $v \in V_1$ now points to a collection S_v of n/w words. Each node x in V'_3 also points to a collection T_x of n/w words, and an array of n/w pointers, each of which point to an appropriate list of

edges in (V_2, V_3) analogous to the above construction. Now for every v in V_1 , the i th word q from S_v for $i = 1, \dots, n/w$, and every neighbor $x \in V'_3$ to v , we look up the i th 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 the set corresponding to x . ■

Remark: 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 1.1:

Corollary 3.1 *There is a $\delta > 0$ and a randomized algorithm for Boolean matrix multiplication that works with high probability and runs in $O\left(\frac{n^3 \log(\log^* n)}{w(\log n)(\log^* n)^\delta}\right)$ time.*

Proof: Let $\varepsilon = 1/\sqrt{n}$. By the usual proof of the triangle removal lemma (via the Regularity Lemma), it suffices to set $f(\varepsilon) = 1/(\log^* 1/\varepsilon)^\delta$ in Theorem 1.1 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.

4. FASTER BOOLEAN MATRIX MULTIPLICATION VIA WEAK REGULARITY

We first state a useful lemma, inspired by Theorem 2.3. It uses a similar technique to the proof of Theorem 3.1.

Theorem 4.1 *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 $O(n^2/\ell^2 \cdot (\sum_{b=1}^{\kappa} \binom{\ell}{b})^2)$ time, so that for any $u, v \in \{0, 1\}^n$, the product $u^T B v$ over the Boolean semiring can be computed in $O(n\ell + (\frac{n}{\ell} + \frac{t_u}{\kappa})(\frac{n}{\ell} + \frac{t_v}{\kappa}))$ 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 $O(n\ell + \frac{n}{w} \cdot (\frac{n}{\ell} + \frac{\min(t_u, t_v)}{\kappa}))$ time.

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

$$O\left(\left(\frac{n}{\log^2 n} + \frac{t_u \log \log n}{\log n}\right) \left(\frac{n}{\log^2 n} + \frac{t_v \log \log n}{\log n}\right)\right) \quad (1)$$

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

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

Proof of Theorem 4.1: As in the proof of Theorem 3.1, 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 nonzeros. Clearly $|U'| = |V'| = O(n/\ell \cdot \sum_{b=1}^{\kappa} \binom{\ell}{b})$.

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 $u^T B v$ 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 nonzeros 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 nonzeros. 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 at most $O(n\ell)$ 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 $u^T B v = 1$ iff there is some $T_{u'}[v']$ that equals 1. It is easily seen that this procedure satisfies the desired running time bounds.

We now consider the word RAM model. We will have two (analogous) data structures depending on whether $t_u \leq t_v$ or not. Suppose $t_u \leq t_v$. As previously, 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 at most $n/\ell + t_u/\kappa$ relevant vectors $T_{u'}$ and hence the product $u^T B v$ can be computed in time $O((n/\ell + t_u/\kappa)(n/w))$. ■

Theorem 4.2 *There is a combinatorial algorithm that, given any two Boolean $n \times n$ matrices A and B , computes $A * B$ 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 Theorem 1.1 (the BMM algorithm using triangle removal), while applying Theorem 4.1, Theorem 3.1, and Weak Regularity. We first describe the algorithm for pointer machines.

Random Sampling: As in Theorem 1.1, by taking a random sample of \sqrt{n} indices from $[n]$, we can determine those pairs (i, j) such that $(A * B)[i, j] = 1$ 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 1.1, 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 .

Preprocessing: Compute an ε -pseudoregular partition $\{W_1, \dots, W_k\}$ of the bipartite subgraph (V_1, V_3) , with $\varepsilon = \frac{1}{\alpha \sqrt{\log n}}$ for an $\alpha > 0$. By Theorem 2.2 this partition can be found in $2^{O(\alpha^2 \log n)}$ time. Set α to make the runtime $(n^{2.5})$. Recall d_{ij} is the density of the pair (W_i, W_j) .

Sparse Pairs: Let F be the set of all edges in (V_1, V_3) that lie in some pair (W_i, W_j) , where $d_{ij} \leq \sqrt{\varepsilon}$. Note $|F| \leq \sqrt{\varepsilon} n^2$. Apply the algorithm of Theorem 3.1 to determine the subset of edges in F that participate in triangles. Remove the edges of F from G .

Dense Pairs: For all pairs (W_i, W_j) with $d_{ij} > \sqrt{\varepsilon}$, build the data structure of Theorem 4.1 for the submatrix corresponding to the pair, 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 $T_j(v) = N(v) \cap W_j$. Compute all pairs of nodes in $S_i(v) \times T_j(v)$ that form a triangle with v , using the query algorithm of Theorem 4.1.

Analysis: Clearly, the random sampling step takes $O(n^{2.75})$ time. Consider the sparse pairs step. Recall $|F| \leq \sqrt{\varepsilon} n^2$ 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 in Theorem 3.1 takes at most $O(\sqrt{\varepsilon} n^3 \log(1/\varepsilon)/\log^2 n) \leq O(n^3 \log \log n / \log^{2.25} n)$ time.

Now we bound the runtime of the dense pairs step. First note that the preprocessing of Theorem 4.1 takes only $O\left(\frac{n^2}{\log^2 n} \cdot \left(\frac{\log^2 n}{\log n/(5 \log \log n)}\right)^2\right) \leq O(n^{2+4/5})$ time overall.

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

$$\sum_{v \in V_2} e(N(v) \cap V_1, N(v) \cap V_3) \leq n^{2.75}. \quad (3)$$

Since $\{W_i\}$ is ε -pseudoregular, (3) implies

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

for large n . Summing over densities $d_{i,j} \geq \sqrt{\varepsilon}$, we obtain

$$\sum_{v \in V_2} \sum_{i,j: d_{i,j} \geq \sqrt{\varepsilon}} |S_i(v)| |T_j(v)| \leq 2\sqrt{\varepsilon} n^3 \leq \frac{2n^3}{\log^{.25} n}. \quad (4)$$

Applying expression (1), the time taken by all queries on the data structure of Theorem 4.1 for a fixed pair (W_i, W_j) is at most

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

Expanding the products and applying (4), it is easy to see that the total runtime is upper bounded by

$$\sum_{v \in V_2} \sum_{i,j: d_{i,j} \geq \sqrt{\varepsilon}} \frac{|S_i(v)| |T_j(v)| (\log \log n)^2}{w \log n} \leq \frac{2n^3 (\log \log n)^2}{w \log^{1.25} n}.$$

Finally, the random sampling step ensures that the number of witnesses is at most $n^{.75}$ for every edge, so the output cost in the algorithm is at most $O(n^{2.75})$.

Modification for the word RAM: We apply the same algorithm as above, except we run the sparse pairs step for pairs (W_i, W_j) with density $d_{ij} \leq \varepsilon^{1/3}$ (instead of $\sqrt{\varepsilon}$ as before). For the pairs (W_i, W_j) with $d_{ij} > \varepsilon^{1/3}$, construct the data structure of Theorem 4.1 for the word RAM.

As in the analysis above, the preprocessing step of Theorem 3.1 has running time $O(\varepsilon^{1/3} n^3 \log(1/\varepsilon) / (w \log n)) \leq O(n^3 \log \log n / (w \log^{7/6} n))$ time. Now consider the time due to queries on the data structure of Theorem 4.1. Using an argument identical to that used to obtain (4), the time is

$$\sum_{v \in V_2} \sum_{i,j: d_{i,j} \geq \varepsilon^{1/3}} |S_i(v)| |T_j(v)| \leq 2\varepsilon^{2/3} n^3 \leq \frac{2n^3}{\log^{1/3} n}. \quad (5)$$

Applying expression (2), the total running time is

$$\begin{aligned} & \sum_{v \in V_2} \sum_{i,j} \left(\frac{\left(\frac{n}{k}\right)^2}{w \log^2 \frac{n}{k}} + \frac{\frac{n}{k} \cdot \min(|S_i(v)|, |T_j(v)|) \log \log \frac{n}{k}}{w \log(n/k)} \right) \\ & \leq \frac{n^3}{w \log^2 \frac{n}{k}} + \sum_{v \in V_2} \sum_{i,j} \frac{\frac{n}{k} \cdot \min(|S_i(v)|, |T_j(v)|) \log \log \frac{n}{k}}{w \log \frac{n}{k}}. \end{aligned} \quad (6)$$

To bound the second term, observe that

$$\begin{aligned} & \sum_{v \in V_2} \sum_{i,j} \min(|S_i(v)|, |T_j(v)|) \\ & \leq \sum_{v \in V_2} \sum_{i,j} (|S_i(v)| \cdot |T_j(v)|)^{1/2} \\ & \leq k\sqrt{n} \cdot \sqrt{\sum_{v \in V_2} \sum_{i,j} |S_i(v)| |T_j(v)|}, \end{aligned}$$

by Cauchy-Schwarz. By (5), this is at most $2kn^2 / \log^{1/6} n$. Thus (6) can be upper bounded by $O(n^3 \log \log n / (w \log^{7/6} n))$ as desired. \blacksquare

5. 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 5.1 *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 5.1: Due to space constraints we only describe the algorithm on the pointer machine. The algorithm can be extended to the word RAM by using a modification identical to that in Theorem 4.2. 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 2.2. (Note 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 $k \leq 2^{O(1/\varepsilon^2)}$.

Let A_{ij} be the submatrix corresponding to the graph (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 $d_{ij} \leq \Delta$, apply graph compression (Theorem 2.5) to preprocess A_{ij} in time $mn^\delta \log^2 n$, so that A_{ij} can be multiplied by any $n/k \times \log n$ matrix B in time $O(m \log((n/k)^2/m) / \log(n/k))$, where m is the number of nonzeros in A_{ij} . (Note $m \leq \Delta(n/k)^2$.)
- 2) If $d_{ij} > \Delta$, apply the preprocessing of Theorem 4.1 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 $1 \leq i, j \leq k$, let $S_i^p = S^p \cap V_i$ and $T_j^p = T^p \cap W_j$.

- 1) Compute $Q^p = \sum_{i,j=1}^k d_{ij} |S_i^p| |T_j^p|$ for all $p = 1, \dots, \log n$. If $Q^p > \varepsilon n^2$, then output $u_p^T A v_p = 1$.
- 2) Let $I = \{p : Q^p \leq \varepsilon n^2\}$. Note $|I| \leq \log n$. 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 algorithm of Theorem 4.1 to compute $e_{ij}^p = (S_i^p)^T A_{ij} T_j^p$ for each $p \in I$.
 - For all (i, j) with $d_{ij} \leq \Delta$, form an $\frac{n}{k} \times |I|$ matrix B_j with columns T_j^p over all $p \in I$. Compute $C_{ij} = A_{ij} * B_j$ 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 2.2, we can choose ε so that the ε -pseudoregular partition is constructed in $O(n^{2+\delta})$ time. By Theorems 2.5 and 4.1, 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. Note step 1 of the query algorithm works by 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 $\sum_{i,j} d_{ij} |S_i^p| |T_j^p| \leq \varepsilon n^2$ for each $p \in I$, we have

$$\sum_{i,j:d_{ij} \geq \Delta} |S_i^p| |T_j^p| \leq \frac{\varepsilon n^2}{\Delta} = \sqrt{\varepsilon} n^2. \quad (7)$$

Analogously to Theorem 4.2, the total runtime over all $p \in I$ and pairs (i, j) with $d_{ij} > \Delta$ is on the order of

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

The inequality (7), the fact that $|I| \leq \log n$, and our choice of ε imply that the above is $O(n^3 (\log \log n)^2 / \log^{5/4} n)$.

Now we consider the pairs (i, j) with $d_{ij} \leq \Delta$. By Theorem 2.5, computing the product $C_{ij} = A_{ij} B_j$ for all $p \in I$ (at once) takes

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

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 . ■

6. 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. Secondly, Trevisan [48] 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. Furthermore, there may be similar algorithms for matrix products over other structures such as finite fields or the $(\min, +)$ -semiring. These algorithms would presumably apply removal lemmas from additive combinatorics.

We end with an open question that we believe can be answered positively. Is there a proof of the Weak Regularity Lemma that achieves $\varepsilon = \Theta(1/\sqrt{\log n})$ with a *deterministic* algorithm that runs in $O(n^{2.9})$ time? Known deterministic algorithms (such as Alon and Naor [5]) require approximately solving an SDP for the cut-norm, which is not known to be solvable in this running time. Such an algorithm would be sufficient to give a deterministic triangle detection algorithm that beats Four Russians, by our results.

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.

REFERENCES

- [1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.* 28(4):1167–1181, 1999. Preliminary version in SODA’96.
- [2] M. Albrecht, G. Bard, and W. Hart. Efficient Multiplication of Dense Matrices over $GF(2)$. *ACM Transactions on Mathematical Software*, to appear.
- [3] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster. The algorithmic aspects of the regularity lemma. *J. Algorithms* 16(1):80–109, 1994. Preliminary version in FOCS’92.
- [4] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica* 20(4):451–476, 2000. Preliminary version in FOCS’99.
- [5] N. Alon and A. Naor. Approximating the cut-norm via Grothendieck’s inequality. *SIAM J. Computing* 35:787–803, 2006. Preliminary version in STOC’04.

- [6] N. Alon, E. Fischer, I. Newman, and A. Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. *Proc. of STOC*, 251–260, 2006.
- [7] D. Angluin. The four Russians' algorithm for Boolean matrix multiplication is optimal for its class. *SIGACT News*, 29–33, Jan-Mar 1976.
- [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. *Doklady Akademii Nauk SSSR* 194:487–488, 1970. In English: *Soviet Mathematics Doklady* 11(5):1209–1210, 1970.
- [9] M.D. Atkinson and N. Santoro. A practical algorithm for Boolean matrix multiplication. *IPL* 29:37–38, 1988.
- [10] J. Basch, S. Khanna, and R. Motwani. On Diameter Verification and Boolean Matrix Multiplication. Technical Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University, 1995.
- [11] C. Borgs, J. Chayes, L. Lovász, V. T. Sós, B. Szegedy, and K. Vesztegombi. Graph limits and parameter testing. *Proc. of STOC*, 261–270, 2006.
- [12] A. Bhattacharyya, V. Chen, M. Sudan, and N. Xie. Testing Linear-Invariant Non-Linear Properties. *Proc. of STACS*, 135–146, 2009.
- [13] G. Blelloch, V. Vassilevska, and R. Williams. A new combinatorial approach for sparse graph problems. *Proc. of ICALP Vol. 1*, 108–120, 2008.
- [14] A. Blum. Personal communication, 2009.
- [15] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *Proc. of STOC*, 590–598, 2007.
- [16] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast matrix multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 13(11):1105–1123, 2002.
- [17] A. Coja-Oghlan, C. Cooper, and A. M. Frieze. An efficient sparse regularity concept. *Proc. of SODA*, 207–216, 2009.
- [18] H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. *Proc. of FOCS*, 438–449, 2003.
- [19] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. *Proc. of FOCS*, 379–388, 2005.
- [20] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990. Preliminary version in STOC'87.
- [21] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM J. Comput.* 29(5):1740–1759, 2000. Preliminary version in FOCS'96.
- [22] R. A. Duke, H. Lefmann, and V. Rödl. A fast approximation algorithm for computing the frequencies of subgraphs in a given graph. *SIAM J. Computing* 24(3):598–620, 1995.
- [23] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.* 51(2):261–272, 1995. Preliminary version in STOC'91.
- [24] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. *Proc. of FOCS*, 12–20, 1996.
- [25] A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica* 19(2):175–220, 1999.
- [26] A. Frieze and R. Kannan. A simple algorithm for constructing Szemerédi's regularity partition. *Electr. J. Comb.* 6, 1999.
- [27] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.* 5: 165–185, 1995.
- [28] B. Green. A Szemerédi-type regularity lemma in abelian groups. *Geom. and Funct. Anal.* 15(2):340–376, 2005.
- [29] W. T. Gowers. Lower bounds of tower type for Szemerédi's uniformity lemma. *Geom. and Funct. Anal.* 7(2), 322–337, 1997.
- [30] A. Hajnal, W. Maass, and G. Turán. On the communication complexity of graph properties. *Proc. of STOC*, 186–191, 1988.
- [31] Y. Kohayakawa. Szemerédi's regularity lemma for sparse graphs. *Found. of Computational Mathem.*, 216–230, 1997.
- [32] Y. Kohayakawa, V. Rödl, and L. Thoma. An optimal algorithm for checking regularity. *SIAM J. Comput.* 32(5):1210–1235, 2003.
- [33] J. Komlós and M. Simonovits. Szemerédi's Regularity Lemma and its applications in graph theory. In *Combinatorics, Paul Erdos is Eighty*, (D. Miklos et. al, eds.), Bolyai Society Mathematical Studies 2:295–352, 1996.
- [34] L. Lee. Fast context-free grammar parsing requires fast Boolean matrix multiplication. *J. ACM* 49(1):1–15, 2002.
- [35] A. Lingas. A geometric approach to Boolean matrix multiplication. *Proc. of ISAAC*, Springer LNCS 2518, 501–510, 2002.
- [36] L. Lovasz and B. Szegedy. Szemerédi's theorem for the analyst. *Geom. and Funct. Anal.* 17:252–270, 2007.
- [37] J. W. Moon and L. Moser. A Matrix Reduction Problem. *Mathematics of Computation* 20(94):328–330, 1966.
- [38] P. E. O'Neil and E. J. O'Neil. A fast expected time algorithm for Boolean matrix multiplication and transitive closure matrices. *Information and Control* 22(2):132–138, 1973.
- [39] V. I. Pan. *How to multiply matrices faster*. Springer-Verlag LNCS 179, 1984.
- [40] L. Roditty and U. Zwick. On Dynamic Shortest Paths Problems. *Proc. of ESA*, 580–591, 2004.
- [41] V. Rödl and M. Schacht. Property testing in hypergraphs and the removal lemma. *Proc. of STOC*, 488–495, 2007.
- [42] I. Z. Ruzsa and E. Szemerédi. Triple systems with no six points carrying three triangles. *Colloquia Mathematica Societatis János Bolyai* 18:939–945, 1978.
- [43] W. Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control* 67(1-3):12–22, 1985. Preliminary version in MFCS'84.
- [44] J. E. Savage. An algorithm for the computation of linear forms. *SIAM J. Comput.* 3(2):150–158, 1974.
- [45] C.-P. Schnorr and C. R. Subramanian. Almost optimal (on the average) combinatorial algorithms for Boolean matrix product witnesses, computing the diameter. *Proc. of RANDOM-APPROX*, Springer LNCS 1518, 218–231, 1998.
- [46] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik* 13:354–356, 1969.
- [47] E. Szemerédi. Regular partitions of graphs. *Proc. Colloque Inter. CNRS (J. C. Bermond, J. C. Fournier, M. Las Vergnas and D. Sotteau, eds.)*, 399–401, 1978.
- [48] L. Trevisan. Additive Combinatorics and Theoretical Computer Science. *SIGACT News Complexity Column* 63, 2009.
- [49] R. Williams. Matrix-vector multiplication in subquadratic time (some preprocessing required). *Proc. of SODA*, 995–1001, 2007.