



Algorithms for Matrix Multiplication via Sampling and Opportunistic Matrix Multiplication

David G. Harris  

University of Maryland, College Park, MD, USA

Abstract

Karppa & Kaski (2019) proposed a novel type of “broken” or “opportunistic” multiplication algorithm, based on a variant of Strassen’s algorithm, and used this to develop new algorithms for Boolean matrix multiplication, among other tasks. For instance, their algorithm can compute Boolean matrix multiplication in $O(n^{\log_2(6+6/7)} \log n) = O(n^{2.778})$ time. While faster matrix multiplication algorithms exist asymptotically, in practice most such algorithms are infeasible for practical problems.

We describe an alternative way to use the broken matrix multiplication algorithm to approximately compute matrix multiplication, either for real-valued matrices or Boolean matrices. In brief, instead of running multiple iterations of the broken algorithm on the original input matrix, we form a new larger matrix by sampling and run a single iteration of the broken algorithm. Asymptotically, the resulting algorithm has runtime $O(n^{\frac{3 \log 6}{\log 7}} \log n) \leq O(n^{2.763})$, a slight improvement of Karppa-Kaski’s algorithm.

Since the goal is to obtain new practical matrix-multiplication algorithms, these asymptotic runtime bounds are not directly useful. We estimate the runtime for our algorithm for some sample problems which are at the upper limits of practical algorithms; it appears that for these parameters, further optimizations are still needed to make our algorithm competitive.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Matrix multiplication, Boolean matrix multiplication, Strassen’s algorithm-keyword

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.57

Related Version *Full Version:* <https://arxiv.org/abs/2109.13335>

Acknowledgements Thanks for Richard Stong for suggesting the proof of Lemma 10.

1 Introduction

Consider the fundamental computational problem of *matrix multiplication*: given matrices A, B of dimensions $d_1 \times d_3$ and $d_3 \times d_2$ respectively, our goal is to compute the matrix C given by

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

When the matrices are square, we write $n = d_1 = d_2 = d_3$. There is an obvious $O(d_1 d_2 d_3)$ algorithm (the so-called “naive algorithm”), which simply iterates over all values i, j, k . There is a long line of research into a variety of asymptotically faster algorithms. For square matrices, the runtime is given as $n^{\omega+o(1)}$, where ω is the linear algebra constant. Currently, the best bound [2] is $\omega \leq 2.38$, coming from a variant of Coppersmith-Winograd’s algorithm [6]. There are also fast algorithms for rectangular matrix multiplication, although they are less heavily studied [12, 7].

Unfortunately, nearly all the fast matrix multiplications algorithms are completely impractical due to large hidden constants. There are a small handful of algorithms which are efficient *in practice*; by far the most important is Strassen’s algorithms and its variants,



© David G. Harris;

licensed under Creative Commons License CC-BY 4.0

31st Annual European Symposium on Algorithms (ESA 2023).

Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 57; pp. 57:1–57:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

57:2 Matrix Multiplication via Sampling

which have runtime $O(n^{\log_2 7})$. Depending on the matrix shape, a few rectangular algorithms may also be practical [5]. There is an extensive literature on optimizing and implementing Strassen’s algorithm in various computational platforms, see e.g [4, 8, 10].

There is an important special case of *Boolean* matrix multiplication (BMM). Here, the entries of A, B come from the algebra $\{0, 1\}$ with binary operations \vee, \cdot , and our goal is to compute the matrix C given by

$$C_{ij} = \bigvee_k A_{ik} B_{kj}$$

This problem, along with some variants, is a primitive used in algorithms for transitive closures, parsing context-free grammars, and shortest path problems in unweighted graphs, among other applications. Again, the obvious $O(d_1 d_2 d_3)$ naive algorithm can be used. There are a number of other specialized algorithms based on combinatorial optimizations; most recently, [13] described an algorithm with runtime roughly $O(n^3 / \log^4 n)$.

There is a standard reduction from BMM to integer matrix multiplication: compute the matrix product $\tilde{C} = AB$ over the integers, and then set $C_{ij} = 1$ if $\tilde{C}_{ij} \geq 1$. Alternatively, there is a randomized reduction from BMM to matrix multiplication over the finite field $\text{GF}(2)$: each non-zero entry of B is set to zero with probability $1/2$, and then we compute the matrix product $\tilde{C} = AB$ over $\text{GF}(2)$. Then $\tilde{C}_{ij} = 1$ with probability $1/2$ whenever $C_{ij} = 1$, else $\tilde{C}_{ij} = 0$ with probability one. With further $O(\log n)$ repetitions the error probability can be reduced to a negligible level. The advantage of this approach is that a variety of additional optimizations are possible for $\text{GF}(2)$ arithmetic, most importantly, the Four Russians method [3]. These can often be combined with asymptotically fast algorithms such as Strassen. For example, [1] provides an optimized bitsliced implementation which uses Strassen for the high-level iterations.

It seems difficult to make progress on better practical algorithms for matrix multiplication. In [11], Karppa & Kaski proposed an innovative and novel approach to break this impasse: they described a “broken” or “opportunistic” form of matrix multiplication, which computes the matrix product with some high failure probability. This broken multiplication, in turn, can be computed more efficiently, both asymptotically and practically, by a variant of Strassen’s algorithm. For brevity, we refer to their algorithm as the *KK algorithm*.

By iterating for sufficiently many repetitions, this procedure can be used to solve BMM with high probability. With appropriate choice of parameters, the overall runtime is $O(n^{\log_2(6+6/7)} \log n) \approx n^{2.776}$, a notable improvement over Strassen. See also [10] for further details.

In this paper, we describe an alternative way to use the KK algorithm: instead of executing multiple independent iterations, we combine them all into a single larger randomized broken multiplication. Each term of the original matrix multiplication product gets sampled multiple times into the larger matrix.

For Boolean matrix multiplication, this gives the following crisp result (given here in a slightly simplified form):

► **Theorem 1.** *For square matrices, we can compute BMM using $O(n^{\frac{3 \log 3}{\log 7}} (\log n)^{\frac{\log 6}{\log 7}}) \leq O(n^{2.763})$ bit operations with high probability*

Our algorithm can also be used to obtain a randomized approximation for real-valued matrix multiplication. The precise approximation guarantees depend on the values input matrices. For general matrices, the accuracy of the estimated matrix term \tilde{C}_{ij} depends on the value C_{ij} as well as the “second moment” of matrix, namely

$$\tilde{C}_{ij}^{(2)} = \sum_k A_{ik}^2 B_{kj}^2$$

When the input matrices A, B (and hence C) are non-negative, then note that $\tilde{C}_{ij}^{(2)} \leq C_{ij}^2$; in this case, we have the following crisp statement (again, in simplified form):

► **Theorem 2.** *For non-negative square matrices of dimension n , we can obtain an approximating matrix \tilde{C} using $O(n^{2.763}/\epsilon^2)$ operations with high probability, satisfying $(1 - \epsilon)C_{ij} \leq \tilde{C}_{ij} \leq (1 + \epsilon)C_{ij}$ for all i, j .*

For example, in the setting of Boolean matrix multiplication, we can not only *detect* if there is any value k with $A_{ik} = B_{kj} = 1$, but we can *estimate* the number of such values of k , up to any desired relative accuracy.

The new algorithm is quite simple, and also has a number of advantages from the viewpoint of practical implementations. First, it lends itself easily to rectangular (non-square) input matrices; we will show a more general theorem which describes the runtime scaling for matrices of arbitrary shape. Second, the algorithm itself internally uses square matrices, and is flexible about the precise dimensions used; this avoids a number of tedious issues with padding in matrix algorithms.

We emphasize that this paper is structured somewhat differently from a typical paper in theoretical computer science. Since our goal is to develop a new *practical* algorithm (we already have good theoretical algorithms!), we cannot afford to look only at asymptotic estimates, ignoring constant factors in runtime. Instead, we have developed our formulas and calculations much more precisely to get concrete runtime estimates in addition to the usual asymptotic bounds.

In Section 7, we illustrate with some computational estimates for large problem instances which are at the limit of practicality. Unfortunately, as we will see, the asymptotic behavior does not seem to fully kick in even at such large scales. At the current time, it appears that the new algorithm is unlikely to beat alternative algorithms; further optimizations and improvements may be needed.

We note that [11] also discussed generalizations to other types of broken matrix multiplication tensors. We will restrict our attention of the Strassen-based pseudo-multiplication, for three reasons. First, much of our analysis depends on certain structural properties of Strassen-type tensor; unlike the analysis in [11], which only depended on the gross *number of terms* in the pseudo-multiplication, we need to track the distribution of terms carefully. Second, it is not clear if any other tensors are practical, especially in light of the fact that Strassen’s algorithm appears to be the only truly practical fast multiplication algorithm known. Finally, many of the “generic” amplification techniques for general tensors, which “only” lose constant factors, are too crude for understanding practical implementations.

2 Pseudo-multiplication

To describe our BMM algorithm, we need to discuss the underlying broken multiplication algorithm in depth. We call this procedure “pseudo-multiplication”, to distinguish it from the overall KK algorithm itself. We begin by quoting the algorithmic result of [11].

57:4 Matrix Multiplication via Sampling

► **Theorem 3.** *Consider 2×2 matrices A, B over a ring R . Using 14 additions and 6 multiplications in R , we can compute the following quantities:*

$$\begin{aligned} C_{11} &= A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

This formula differs from the computation of $C = AB$, in that the term C_{11} is missing the summand $A_{11}B_{11}$. This computation is achieved by a variant of a Strassen step, except that one of the 7 multiplications is omitted.

Now consider integer matrices A, B of dimension $n = 2^s$. We can identify the integers in the range $[n] = \{0, \dots, n-1\}$ with binary vectors $\{0, 1\}^s$; thus, we may write A_{xy} for vectors $x, y \in \{0, 1\}^s$. We write $z = x \vee y$ where z_i is zero iff $x_i = y_i = 0$, i.e. the disjunction is done coordinate-wise. Also, we let $|x|$ denote the Hamming weight of a binary vector x . By iterating the algorithm of Theorem 3 for s levels, we get the following:

► **Theorem 4.** *Using $7(6^s - 4^s)$ additions and $6^s = n^{\log_2 6}$ multiplications, we can compute the matrix pseudo-product $C = A \boxtimes B$ defined as $C_{xy} = \sum_{x \vee y \vee z = \vec{1}^s} A_{xz}B_{zy}$, where $\vec{1}^s$ denotes the vector of dimension s whose entries are all equal to 1. The computation of C can be performed over any ring.*

Proof. See full paper. ◀

In particular, this implies the following result (which was the only thing shown directly in [11]):

► **Corollary 5.** *The pseudo-product $C = A \boxtimes B$ contains $(7/8)^s$ of the summands in the full matrix product $C = AB$.*

Proof. There are precisely 7^s triples of vectors $x, y, z \in \{0, 1\}^s$ with $x \vee y \vee z = \vec{1}^s$. ◀

Note that [11] includes an additional randomization step, where the entries of the matrices are randomly permuted at each level. We omit this step, since we will later include more extensive randomization in the overall algorithm. (For practical purposes, this shuffling step can be awkward to implement efficiently). For given s , we define $T_s \subseteq \{0, 1\}^s \times \{0, 1\}^s \times \{0, 1\}^s$ to be the set of triples x, y, z with $x \vee y \vee z = \vec{1}^s$.

It is sometimes useful to use Strassen's algorithm, or naive matrix multiplication, for some of the low-level iterations. Let us suppose that we use s steps of pseudo-multiplication, followed by true matrix multiplication on the resulting submatrices of dimension $b_i = d_i/2^s$ (the *base case*). We can view any integer $x \in \{1, \dots, d_i\}$ as equivalent to an ordered pair (x', x'') where $x' \in \{0, 1\}^s$ and $x'' \in \{1, \dots, b_i\}$. The resulting matrix, denoted still by $C = A \boxtimes B$, then satisfies

$$C_{xy} = \sum_{\substack{z=(z', z'') \in \{0, 1\}^s \times \{1, \dots, b\} \\ x' \vee y' \vee z' = \vec{1}^s}} A_{(x', x''), (z', z'')} B_{(z', z'')(y', y'')}$$

The proof is completely analogous to Theorem 4 so we omit it here. For convenience, let us say that, for integers x, y, z , we have $x \vee y \vee z = \vec{1}$ if, when we write $x = (x', x''), y = (y', y''), z = (z', z'')$, we have $x' \vee y' \vee z' = \vec{1}^s$, i.e. $(x', y', z') \in T_s$. When s, d_1, d_2, d_3 are understood, we also define $T^* \subseteq [d_1] \times [d_2] \times [d_3]$ to be the resulting set of triples (x, y, z) with this property. With these conventions, we can define $C = A \boxtimes B$ more compactly as $C_{xy} = \sum_{z:(x,y,z) \in T^*} A_{xz}B_{zy}$.

If the matrices are square, then the base case would also likely be square $b = b_1 = b_2 = b_3$. From a theoretical point of view, b may be viewed as a constant. When working in the field $\text{GF}(2)$, the base case multiplication would likely involve switching to a Four Russians method and/or bitsliced operations. These optimizations are very powerful in practice, possibly more important than the asymptotic gains from a fast matrix multiplication algorithm. Consequently, in practical implementations, b may be very large; for instance, in the implementation of [1], they choose $b \approx 2^{11}$ (roughly matching the L2 cache size).

3 Algorithm overview

Consider matrices A, B of dimension $d_1 \times d_3$ and $d_3 \times d_2$; here the matrices are either real-valued or Boolean-valued, and our goal is to approximate the product $C = AB$. The algorithms we use are very similar for these two cases: we choose some parameter s and base sizes b_1, b_2, b_3 . For $\ell = 1, 2, 3$ we define $m_\ell = 2^s b_\ell$, and draw random functions $f_\ell : [m_\ell] \rightarrow [d_\ell]$. We then form matrices \bar{A}, \bar{B} of dimension $m_1 \times m_3, m_3 \times m_2$ by sampling from matrices A, B according to the function f , i.e. we will have

$$\bar{A}_{xz} = A_{f_1(x)f_3(z)}, \quad \bar{B}_{zy} = B_{f_3(z)f_2(y)}$$

(the actual matrices \bar{A}, \bar{B} will have some additional scaling factors)

We then compute the pseudo-product $\bar{C} = \bar{A} \boxtimes \bar{B}$, with the given base case sizes, and produce an estimated matrix \tilde{C} , where, for each entry ij , the estimate \tilde{C}_{ij} is derived by aggregating the entries \bar{C}_{xy} where $x \in f_1^{-1}(i), y \in f_2^{-1}(j)$.

The runtime for this process will be $O(6^s)$ field operations, and the memory required is $O(4^s)$ to store the matrices $\bar{A}, \bar{B}, \bar{C}$ (assuming b is constant). For any triple i, j, k in the matrix product $A_{ik}B_{kj}$, there are roughly $\frac{7^s b_1 b_2 b_3}{d_1 d_2 d_3}$ entries $u = (x, y, z) \in T^*$ which get mapped to i, j, k , i.e. $f_1(x) = i, f_2(y) = j, f_3(z) = k$. Thus, as long as $7^s b_1 b_2 b_3 \gg d_1 d_2 d_3$, we should expect that the sampled matrix “covers” the original matrix, and so we get accurate answers. However, to analyze it formally, we need to take account of some potentially problematic dependencies between entries in the matrix. This was not an issue encountered in the original analysis [11], which only worked expectation-wise.

There are a number of other details to work out for the algorithms. For the real-valued case, we need certain weighting factors to account for the fact that some entries of the product $\bar{A} \boxtimes \bar{B}$ are over-represented in the pseudo-multiplication. For the Boolean case, we need to have an additional randomization to handle the Boolean-to- $\text{GF}(2)$ reduction. We provide the details next.

Real-valued algorithm. Having chosen the parameter s and the random functions f , we will choose scaling matrices G^A, G^B, G^C of dimensions $m_1 \times m_3, m_3 \times m_2, m_1 \times m_2$ respectively (their role will be discussed shortly). We then define

$$\bar{A}_{xz} = G_{xz}^A A_{f_1(x)f_3(z)}, \quad \bar{B}_{zy} = G_{zy}^B B_{f_3(z)f_2(y)}$$

and, after computing the pseudo-product $\bar{C} = \bar{A} \boxtimes \bar{B}$ over \mathbb{R} , we estimate C by

$$\tilde{C}_{ij} = \sum_{x \in f_1^{-1}(i), y \in f_2^{-1}(j)} G_{xy}^C \bar{C}_{xy}$$

Importantly, for values $x = (x', x'') \in \{0, 1\}^s \times [b_1], z = (z', z'') \in \{0, 1\}^s \times [b_2]$, the value of each term G_{xz}^A should only depend x', z' , in particular, it should depend on the Hamming weights of $|x'|, |z'|, |x' \vee z'|$. A similar criterion should hold for G^B, G^C matrices. Thus, with

57:6 Matrix Multiplication via Sampling

a slight abuse of notation, we write $G_{x',z'}^A$ and similarly $G_{z',y'}^B, G_{x',y'}^C$. Thus, the matrices G really only have $O(s^3)$ degrees of freedom; we discuss later how to choose their values (both for asymptotic analysis and practical parameters).

Boolean algorithm. In this case, after drawing random functions f , we will also draw a uniformly random $m_3 \times m_2$ binary matrix D . We then form matrices \bar{A}, \bar{B} by setting:

$$\bar{A}_{xz} = A_{f_1(x)f_3(z)}, \quad \bar{B}_{zy} = B_{f_3(z)f_2(y)} D_{zy}$$

We then compute the pseudo-product $\tilde{C} = \bar{A} \boxtimes \bar{B}$ over the finite field $\text{GF}(2)$, with the given base case sizes, and produce an estimated matrix \tilde{C} by $\tilde{C}_{ij} = \bigvee_{x \in f_1^{-1}(i), y \in f_2^{-1}(j)} \tilde{C}_{xy}$.

Analysis overview. We will analyze the algorithms in three main steps. First, we derive some non-asymptotic bounds on the accuracy, in terms of parameters which can be computed explicitly as functions of s . Next, we use these to derive asymptotic bounds on the algorithm complexity. While illuminating as to the basic algorithm shape, these asymptotic bounds are not optimized and not of direct relevance for practical-scale computations. So we finish with some cases studies on very large problem parameters pushing the limits of practical computations, with concrete bounds.

Throughout, we define parameters

$$\psi_1 = d_1 + d_2 + d_3, \psi_2 = d_1 d_2 + d_2 d_3 + d_1 d_3, \psi_3 = d_1 d_2 d_3$$

Also, for each $\ell = 1, 2, 3$, we define $q_\ell = b_\ell / d_\ell$.

4 Analysis: the real-valued case

Consider some entry i, j ; we can see that

$$\tilde{C}_{ij} = \sum_k \sum_{(x,y,z) \in T^*} G_{xz}^A G_{yz}^B G_{xy}^C [f_1(x) = i][f_2(y) = j][f_3(z) = k] A_{ik} B_{kj}$$

where we use Iverson notation for any boolean predicate here and throughout the paper, i.e. $[E] = 1$ if E holds, otherwise $[E] = 0$. Our strategy to analyze \tilde{C}_{ij} is to compute the mean and variance. We will show that $\mathbb{E}[\tilde{C}_{ij}] \propto C_{ij}$ (with known proportionality constants). Thus, by rescaling the matrix entries G as needed, we can obtain an unbiased estimate of C_{ij} . Likewise, the variance of \tilde{C}_{ij} will depend on C_{ij} as well as its ‘‘second moment’’

$$C_{ij}^{(2)} = \sum_k A_{ik}^2 B_{kj}^2$$

To simplify the notation, let us write

$$G_{xyz} := G_{xz}^A G_{yz}^B G_{xy}^C$$

for a triple $(x, y, z) \in T$. We have the following main estimate:

► **Lemma 6.** Define the sums r_0, \dots, r_7 by:

$$\begin{aligned} r_0 &= \sum_{(x,y,z) \in T_s} G_{xyz} & r_1 &= \sum_{x \in \{0,1\}^s} \left(\sum_{y,z:(x,y,z) \in T_s} G_{xyz} \right)^2 \\ r_2 &= \sum_{y \in \{0,1\}^s} \left(\sum_{x,z:(x,y,z) \in T_s} G_{xyz} \right)^2, & r_3 &= \sum_{z \in \{0,1\}^s} \left(\sum_{x,y:(x,y,z) \in T_s} G_{xyz} \right)^2 \\ r_4 &= \sum_{x,y \in \{0,1\}^s} \left(\sum_{z:(x,y,z) \in T_s} G_{xyz} \right)^2, & r_5 &= \sum_{x,z \in \{0,1\}^s} \left(\sum_{y:(x,y,z) \in T_s} G_{xyz} \right)^2 \\ r_6 &= \sum_{y,z \in \{0,1\}^s} \left(\sum_{x:(x,y,z) \in T_s} G_{xyz} \right)^2, & r_7 &= \sum_{(x,y,z) \in T_s} G_{xyz}^2 \end{aligned}$$

Then, for any i, j , there holds $\mathbb{E}[\tilde{C}_{ij}] = q_1 q_2 q_3 r_0 C_{ij}$ and

$$\mathbb{V}[\tilde{C}_{ij}] \leq q_1 q_2 q_3^2 (q_2 r_1 + q_1 r_2 + r_4) C_{ij}^2 + q_1 q_2 q_3 (q_1 q_2 r_3 + q_2 r_5 + q_1 r_6 + r_7) C_{ij}^{(2)}$$

Proof. See full paper. ◀

For non-negative matrices, the statistic \tilde{C}_{ij} can be used to estimate the value C_{ij} up to some relative accuracy. We summarize this as follows:

► **Corollary 7.** If matrices A, B are non-negative, then

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \leq \frac{(q_3 q_2 r_1 + q_3 q_1 r_2 + q_1 q_2 r_3) + (q_3 r_4 + q_2 r_5 + q_1 r_6) + r_7}{q_1 q_2 q_3 r_0^2}$$

Proof. We know $C_{ij} \geq 0$ and $C_{ij}^{(2)} \leq C_{ij}^2$ for each entry i, j . ◀

5 Analysis: the Boolean case

In the Boolean case, it does not make sense to talk about how close the matrices entries \tilde{C}_{ij} are to the true value C_{ij} . Instead, we will argue that, with high probability, we have $\tilde{C}_{ij} = C_{ij}$ exactly. It is clear that if $C_{ij} = 0$, then $\tilde{C}_{ij} = 0$ with probability one. Thus, consider some pair i, j with $C_{ij} = 1$ and some witness value k with $A_{ik} = B_{kj} = 1$. We want to find a triple of values $(x, y, z) \in T^*$ such that $f_1(x) = i, f_2(y) = j, f_3(z) = k$. If such a triple exists, then the randomization involved in the D matrix should ensure that $\tilde{C}_{xy} = 1$ with probability $1/2$, in which case $\tilde{C}_{ij} = 1$ as well.

To show the existence of such a triple (x, y, z) , we will use an important general probabilistic inequality of Janson [9]. We summarize it in the following form:

► **Theorem 8** ([9]). Suppose that X_1, \dots, X_N are independent Bernoulli variables, and suppose \mathcal{E} is a collection of monomial events over ground set $[N]$, i.e. each event $E \in \mathcal{E}$ is a conjunction of the form $\bigwedge_{r \in R_E} X_r$ for a given subset $R_E \subseteq [n]$. For events $E, E' \in \mathcal{E}$, we write $E \sim E'$ if $R_E \cap R_{E'} \neq \emptyset$, and we write \bar{E} for the complement of E (i.e. that event E does not hold.)

If we define

$$\kappa = \sum_{E \in \mathcal{E}} \mathbb{E} \left[\frac{[E]}{\sum_{E' \sim E} [E']} \right]$$

then we have

$$\Pr \left(\bigwedge_{E \in \mathcal{E}} \bar{E} \right) \leq e^{-\kappa}$$

57:8 Matrix Multiplication via Sampling

We use it to get the following main result:

► **Theorem 9.** *Define values*

$$q'_1 = 1 - (1 - 1/d_1)^{b_1}, q'_2 = 1 - (1 - 0.5/d_2)^{b_2}, q'_3 = 1 - (1 - 1/d_3)^{b_3}$$

Consider independent Bernoulli variables $I_{\ell,x}$ for $\ell = 1, 2, 3$ and $x \in \{0, 1\}^s$, wherein each $I_{\ell,x}$ has expected value q'_ℓ . For each tuple $u = (x, y, z) \in T_s$, define associated event E_u to be the conjunction $I_{1,x} = I_{2,y} = I_{3,z} = 1$.

Then, for any pair i, j with $C_{ij} = 1$, there holds $\Pr(\tilde{C}_{ij} = 0) \leq \Pr(\bigwedge_{u \in T_s} \bar{E}_u)$.

Proof. We divide the analysis into two parts. first, we consider the random functions f_ℓ , and then consider the random matrix D . Let us fix some arbitrary value k with $A_{ik} = B_{kj} = 1$. Suppose that we chosen the functions f_ℓ , and let P denote the set of triples $(x, y, z) \in T^*$ with $f_1(x) = i, f_2(y) = j, f_3(z) = k$. From this, let S denote the number of *distinct* values y encountered, i.e. $S = |\{y : (x, y, z) \in P\}|$.

We claim that, having fixed the functions f_ℓ , the probability that $C_{ij} = 0$ is at most 2^{-S} . For, suppose that P contains triples $(x_1, y_1, z_1), \dots, (x_S, y_S, z_S)$ where y_1, \dots, y_S are distinct. In forming each entry $\tilde{C}_{x_\ell y_\ell}$, we are adding in $D_{z_\ell y_\ell} A_{ik} B_{kj} = D_{z_\ell y_\ell}$ as well as potentially other entries $D_{z' y_\ell}$. These are all independent random bits, and hence their sum over GF(2) is equal to 1 with probability precisely 1/2; in this case we would have $\tilde{C}_{x_\ell y_\ell} = 1$ and hence $\tilde{C}_{ij} = 1$. Furthermore, since y_1, \dots, y_S are distinct, all such events are independent.

We also define independent Bernoulli random variables $I'_{1,x}, I'_{2,y}, I'_{3,z}$ to be the events that, respectively, $f_1(x) = i$, or $f_2(y) = j$, or $f_3(z) = k$; these have means $\frac{1}{d_1}, \frac{1}{d_2}, \frac{1}{d_3}$ respectively. Note that P , and hence S , is determined by these variables. Integrating over them, we have

$$\Pr(\tilde{C}_{ij} = 0) \leq \mathbb{E}[2^{-S}]$$

For each $x \in \{0, 1\}^s$, we define events

$$I_{1,x} = \bigvee_{x'} I'_{1,(x,x')}, \quad I_{2,y} = \bigvee_{y'} I'_{2,(y,y')} J_{(y,y')}, \quad I_{3,z} = \bigvee_{z'} I'_{3,(z,z')},$$

where x', y', z' range over $[b_1], [b_2], [b_3]$ respectively, and where $J_{(y,y')}$ are independent Bernoulli-1/2 random variables. Observe that $I_{1,x}, I_{2,y}, I_{3,z}$ are all independent Bernoulli random variables with means q'_1, q'_2, q'_3 respectively. Now consider the probability of the event \bar{E}_u . Suppose we reveal random variables $I'_{1,x}, I'_{2,y}, I'_{3,z}$. Conditional on these values, the only way for $\bigwedge_u \bar{E}_u$ to occur is that $J_y = 0$ holds for all tuples $u = (x, y, z) \in P$. This has probability precisely 2^{-S} . Integrating over variables I' , we get:

$$\Pr\left(\bigwedge_u \bar{E}_u\right) = \mathbb{E}[2^{-S}]$$

Putting the inequalities together gives the claimed result

$$\Pr(\tilde{C}_{ij} = 0) \leq \mathbb{E}[2^{-S}] = \Pr\left(\bigwedge_u \bar{E}_u\right) \quad \blacktriangleleft$$

Observe that events E_u of Theorem 9 are precisely the types of monomial events covered by Theorem 8; this will be used for asymptotic and concrete bounds later.

6 Asymptotic analysis

The asymptotic analysis of both the real-valued and Boolean use similar arguments. For this section, we let $b_1 = b_2 = b_3 = 1$, since the base cases should only affect analysis by constant factors.

We want to determine the number of triples $(x, y, z) \in T_s$ with $f_1(x) = i, f_2(y) = j, f_3(z) = k$ for values i, j, k ; the relevant quantities (r_0, \dots, r_7 for the real-valued case and κ for the Boolean case) are in effect counting these triples. Our basic strategy is to restrict the analysis to “typical” triples, specifically, let us define $H \subseteq T_s$ to be the set of triples $(x, y, z) \in T$ satisfying the following conditions:

1. $|x| \leq 4s/7$ and $|y| \leq 4s/7$ and $|z| \leq 4s/7$
2. $|x \vee y| \leq 6s/7$ and $|x \vee z| \leq 6s/7$ and $|y \vee z| \leq 6s/7$

► **Lemma 10.** *There holds $|H| \geq \Omega(7^s)$.*

Proof. There are precisely 7^s triples (x, y, z) in T_s . Let us consider the probability space which is the uniform distribution on T_s ; equivalently, for each coordinate i , the values (x_i, y_i, z_i) are chosen uniformly at random among the 7 non-zero values. We need to show that the conditions on the densities of $x, y, z, x \vee y, x \vee z, y \vee z$ are satisfied with constant probability.

For each coordinate i , consider the 6-dimensional random vector $V_i = (x_i, y_i, z_i, x_i \vee y_i, x_i \vee z_i, y_i \vee z_i)$ and let $V = \sum_i V_i$. We need to show that $V - \beta s$ has non-positive entries where $\beta = (4/7, 4/7, 4/7, 6/7, 6/7, 6/7)$. For this, we use the multidimensional Central Limit Theorem. Since the variables V_i are i.i.d. and each has mean β , the scaled value

$$\sqrt{s}(V/s - \beta)$$

approaches to a 6-dimensional normal distribution $N(0, \Sigma)$ with covariance matrix given by

$$\Sigma = \frac{1}{49} \begin{bmatrix} 12 & -2 & -2 & 4 & 4 & -3 \\ -2 & 12 & -2 & 4 & -3 & 4 \\ -2 & -2 & 12 & -3 & 4 & 4 \\ 4 & 4 & -3 & 6 & -1 & -1 \\ 4 & -3 & 4 & -1 & 6 & -1 \\ -3 & 4 & 4 & -1 & -1 & 6 \end{bmatrix}$$

as $s \rightarrow \infty$.

Note that Σ is non-singular. Hence, for the corresponding distribution $N(0, \Sigma)$ there is positive probability that all six coordinates are negative. For large enough s , the probability that $V - \beta s$ has negative entries converges; in particular, it approaches to some constant value as $s \rightarrow \infty$. (The lemma holds vacuously when $s = O(1)$.) ◀

We will define two constants which show up repeatedly in the analysis:

$$\alpha_1 = 14 \cdot 2^{1/7} \cdot 3^{3/7} \approx 24.75$$

$$\alpha_2 = 7 \cdot 2^{6/7} \approx 12.68$$

► **Lemma 11.** *For $i = 1, 2$, there are $O(\frac{\alpha_i^s}{\sqrt{s}})$ pairs $(x, y, z), (x', y', z') \in H$ with $[x = x'] + [y = y'] + [z = z'] \geq i$.*

Proof. For $i = 1$, we will count the pairs $(x, y, z), (x', y', z') \in H$ with $x = x'$; the cases with the other two coordinates are completely symmetric. Let us suppose that, among the s entries of x , there are ℓ coordinated equal to one; since $(x, y, z) \in H$ we must have $\ell \leq 4s/7$.

57:10 Matrix Multiplication via Sampling

For each coordinate i with $x_i = x'_i = 1$, the values y_i, y'_i, z_i, z'_i can be arbitrary (giving 16^ℓ choices); for each coordinate i with $x_i = x'_i = 0$, there are three non-zero choices for (y_i, z_i) and three non-zero choices for (y'_i, z'_i) . Overall, there are $16^\ell 9^{s-\ell}$ choices for the vectors y, z, y', z' . Summing over ℓ , we get that the total contribution is

$$\sum_{\ell=0}^{\lfloor 4s/7 \rfloor} \binom{s}{\ell} 16^\ell 9^{s-\ell}$$

Let $h(\ell) = \binom{s}{\ell} 16^\ell 9^{s-\ell}$ be the summand corresponding to ℓ ; we can observe that $h(\ell)/h(\ell-1) = \frac{16(s+1-\ell)}{9}$. In particular, for $\ell \leq 4s/7$, this ratio is at least $4/3$. Hence, the entire sum $\sum_{\ell=0}^{\lfloor 4s/7 \rfloor} h(\ell)$ is within a constant factor of $h(4s/7)$. In turn, by Stirling's formula, we can calculate $h(4s/7) \leq O(\alpha_1^s/\sqrt{s})$.

For $i = 2$, we count the pairs $(x, y, z), (x', y', z') \in H$ with $x = x', y = y'$; the cases with the other two coordinates are again completely symmetric. Let us suppose that, among the s entries of x, y , there are ℓ coordinates with $x_i \vee y_i = 1$; since $(x, y, z) \in H$ we must have $\ell \leq 6s/7$. For each such coordinate i , there are 4 choices for z, z' and 3 choices for x, y ; for each coordinate with $x_i \vee y_i = x'_i \vee y'_i = 0$, we must have $z_i = z'_i = 1$. Summing over ℓ , the total contribution is

$$\sum_{\ell=0}^{\lfloor 6s/7 \rfloor} \binom{s}{\ell} 3^\ell 4^\ell$$

Let $h(\ell) = \binom{s}{\ell} 12^\ell$ be the summand corresponding to value ℓ ; note that $h(\ell)/h(\ell-1) = \frac{12(s+1-\ell)}{\ell}$. For $\ell \leq 6s/7$, this ratio is at least 2, and hence the entire sum is within a constant factor of $h(6s/7)$. In turn, by Stirling's formula, we get $h(6s/7) \leq O(\alpha^2 s/\sqrt{s})$. \blacktriangleleft

► **Proposition 12.** *Suppose we define matrices G^A, G^B, G^C by*

$$\begin{aligned} G_{xz}^A &= [|x| \leq 4s/7][|z| \leq 4s/7][|x \vee z| \leq 6s/7] \\ G_{yz}^B &= [|y| \leq 4s/7][|z| \leq 4s/7][|y \vee z| \leq 6s/7] \\ G_{xy}^C &= [|x| \leq 4s/7][|y| \leq 4s/7][|x \vee y| \leq 6s/7] \end{aligned}$$

Then the real-valued algorithm satisfies

$$\begin{aligned} \mathbb{E}[\tilde{C}_{ij}] &= \nu C_{ij} 7^s / \psi_3 \quad \text{for a known proportionality constant } \nu \\ \mathbb{V}[\tilde{C}_{ij}] &= O\left(q_1 q_2 q_3^2 (q_1 + q_2) \alpha_1^s / \sqrt{s} + \alpha_2^s / \sqrt{s}\right) C_{ij}^2 + O\left(q_1 q_2 q_3 (q_1 q_2 \alpha_1^s / \sqrt{s} + (q_1 + q_2) \alpha_2^s / \sqrt{s} + 7^s)\right) C_{ij}^{(2)} \end{aligned}$$

Proof. With this definition of the matrices G^A, G^B, G^C , we have $G_{xyz} = [(x, y, z) \in H]$. Now let us compute the sums r_0, \dots, r_7 . For r_0 , we have $r_0 = \sum_{(x,y,z) \in T_s} G_{xyz} = \sum_{(x,y,z) \in H} 1 = |H|$; by Lemma 10 this is $\Omega(7^s)$.

For r_1 , we compute

$$r_1 = \sum_{x \in 0, 1^s} \left(\sum_{y, z: (x,y,z) \in T_s} G_{xyz} \right)^2 = \sum_{x \in 0, 1^s} \left(\sum_{y, z: (x,y,z) \in H} 1 \right)^2 = \sum_{\substack{(x,y,z) \in H, (x',y',z') \in H \\ x=x'}} 1$$

By Lemma 11, this is $O(\alpha_1^s/\sqrt{s})$. By completely analogous reasoning, we have $r_2, r_3 \leq O(\alpha_1^s/\sqrt{s})$ and $r_4, r_5, r_6 \leq O(\alpha_2^s/\sqrt{s})$. For r_7 , we have

$$r_7 = \sum_{(x,y,z) \in T_s} G_{xyz}^2 = \sum_{(x,y,z) \in H} 1 = |H| \leq |T_s| \leq 7^s$$

Now we directly apply Lemma 6. \blacktriangleleft

By collecting terms in Proposition 12, and noting that $q_i = 1/d_i$ for $i = 1, 2, 3$, we immediately get the following corollary:

► **Corollary 13.** *Suppose that $C_{ij}^{(2)} \leq C_{ij}^2$, and that $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ and $(\alpha_2/7)^s/\sqrt{s} \leq O(\psi_3/\psi_2)$. Then*

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \leq O(\psi_3/7^s)$$

► **Theorem 14.** *Suppose that matrices A, B are non-negative. Let $\epsilon \in (0, 1)$, and define parameters $\gamma = \psi_3/\epsilon^2$ and $\beta_1 = \psi_3/\psi_1$ and $\beta_2 = \psi_3/\psi_2$. With appropriate choice of parameters, we can obtain an estimated matrix \tilde{C} , such that, for any entry i, j , there holds*

$$\Pr(\tilde{C}_{ij} \in [(1 - \epsilon)C_{ij}, (1 + \epsilon)C_{ij}]) \geq 3/4,$$

with overall runtime

$$O\left(\gamma^{\frac{\log 6}{\log 7}} + \gamma\left((\beta_1\sqrt{\log \beta_1})^{\frac{\log(6/7)}{\log(\alpha_1/7)}} + (\beta_2\sqrt{\log \beta_2})^{\frac{\log(6/7)}{\log(\alpha_2/7)}}\right)\right)$$

Proof. Our strategy will be to take multiple independent executions and average them so that the resulting sample means have relative variance $\epsilon^2/4$. By Chebyshev's inequality, the resulting sample means are then within $(1 \pm \epsilon)C_{ij}$ with probability at least $3/4$. To that end, let us set

$$s = \left\lceil \min\{\log_7 \gamma, \log_{\alpha_1/7}(\beta_1\sqrt{\log \beta_1}), \log_{\alpha_2/7}(\beta_2\sqrt{\log \beta_2})\} \right\rceil$$

It can be shown that $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ and $(\alpha_2/7)^s/\sqrt{s} \leq O(\psi_3/\psi_2)$. Now, by collecting terms in Proposition 12, and noting that $q_i = 1/d_i$ for $i = 1, 2, 3$, and using the bounds $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ and $(\alpha_2/7)^s/\sqrt{s}$ as well as the observation that $C_{ij}^{(2)} \leq C_{ij}$, we get the estimate:

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \leq O(\psi_3/7^s)$$

Thus, if we repeat the process for $\Omega(1 + \epsilon^{-2}/(7^s/\psi_3))$ iterations and take the mean of all observations, the overall relative variance of the resulting sample mean is reduced to $\epsilon^2/4$, and hence satisfies the required bounds. (We can scale by a known proportionality constant to get an unbiased estimator). Overall, the runtime is then

$$O(6^s(1 + \epsilon^{-2}/(7^s/\psi_3)))$$

which after some simplifications, gives the claimed runtime bounds. ◀

► **Corollary 15.** *Suppose matrices A, B are non-negative. With appropriate choice of parameters, the real-valued algorithm can compute C with relative error $1 \pm \epsilon$ with probability $3/4$ in runtime*

$$O\left((\psi_3/\epsilon^2)^{\frac{\log 6}{\log 7}} + \frac{\psi_1^{0.122}\psi_3^{0.878} + \psi_2^{0.259}\psi_3^{0.741}}{\epsilon^2}\right)$$

In particular, if the matrix is square of dimension n , then the runtime is $O(n^{2.77}/\epsilon^2)$.

57:12 Matrix Multiplication via Sampling

As usual, we can boost the success probability to any desired value $1 - \delta$ by using median amplification, with a further $O(\log(1/\delta))$ increase in runtime. Despite the (relatively) crisp formula, Corollary 15 is not directly relevant to practical parameter sizes. Many of the hidden constants are large, and the bounds are all somewhat crude. Since our main goal is to obtain a *practical* algorithm we need to consider more finely how the algorithm scales for *small* values of n and s ; we consider this in the next section.

The analysis for the Boolean case is similar to the real-valued case.

► **Theorem 16.** *Let $\delta \in (0, 1)$, and define parameters $\gamma = \psi_3 \log(1/\delta)$ and $\beta_1 = \psi_3/\psi_1$ and $\beta_2 = \psi_3/\psi_2$. With appropriate choice of parameters, we can obtain an estimated matrix \tilde{C} , such that, for any entry i, j , there holds*

$$\Pr(\tilde{C}_{ij} = C_{ij}) \geq 1 - \delta,$$

with overall runtime

$$O\left(\gamma^{\frac{\log 6}{\log 7}} + \gamma\left((\beta_1 \sqrt{\log \beta_1})^{\frac{\log(6/7)}{\log(\alpha_1/7)}} + (\beta_2 \sqrt{\log \beta_2})^{\frac{\log(6/7)}{\log(\alpha_2/7)}}\right)\right)$$

Proof. The proof is completely analogous to Theorem 14 and is omitted. ◀

► **Corollary 17.** *With appropriate choice of parameters, we can compute the overall matrix C correctly with probability at least $1 - 1/\text{poly}(\psi_3)$ using runtime*

$$O\left(\psi_3 \log \psi_3)^{\frac{\log 6}{\log 7}} + \psi_1^{0.122} \psi_3^{0.878} + \psi_2^{0.259} \psi_3^{0.741}\right)$$

We mention one important difference in our analysis of the real-valued and Boolean cases. For both algorithms, the analysis depends on the set $H \subseteq T_s$. For the Boolean algorithm, this is only used for purposes of analysis; the actual algorithm does not need to use H . For the real-valued algorithm, we need H to define the matrices G^A, G^B, G^C (which are part of the algorithm itself)

7 Concrete complexity estimates

Since our goal is to get a new practical algorithm, the asymptotic estimates are of limited value on their own. In this section, we consider costing the matrix multiplication algorithms for problem sizes which are at the upper limits of a practical problem size. In particular, we want to compare our algorithm to Strassen's algorithm and the KK algorithm. There are two main issues to consider. First, we need a realistic estimate of the runtime; second, we need a realistic estimate of the algorithm accuracy and/or success probability. The asymptotic estimates are much too crude for our purposes here.

Let us first consider the runtime. The work for the algorithm will involve 6^s many iterations of the recursive partitioning as given in Theorem 3. The cost of this step will have a quadratic component (adding and rearranging the relevant submatrices) as well as a nearly-cubic component (coming from the recursive subproblems). Assuming that the base case parameters are chosen to be sufficiently large, the first of these should be negligible; overall, we can estimate the work as $6^s W_b$ where W_b is the base-case cost.

Let us compare this runtime with other matrix-multiplication algorithms. To avoid issues with padding, let us assume for convenience that $d_1 = d_2 = d_3$ and d is a power of two. In this case, both Strassen's algorithm and the KK algorithm would run s many recursive iterations, and then again pass to a base case. Assuming again that the base case is sufficiently large to hide the quadratic parts of the algorithm, these would have costs of $7^s W_b$ and $6^s W_b$ respectively.

Estimating W_b itself is much trickier. As a starting point, we might estimate $W_b \approx b_1 b_2 b_3$ in terms of arithmetic operations. But there are many additional factors to consider: a CPU with SIMD registers of width w may be able to perform w operations in parallel, which is especially powerful for GF(2) arithmetic. Furthermore, for GF(2), there are other optimizations such as Four-Russians. Fortunately, in order to compare our algorithm with Strassen or KK, the precise value of W_b does not matter; it is common to all the algorithms.

To simplify further, let us assume that $b_1 = b_2 = b_3$, and let b denote this common value, which we regard as a fixed constant. In this case, for Strassen and KK, the parameter s must be set to a fixed value $s = s_0 = \log_2(n/b)$. Summarizing, the runtimes of our algorithm, Strassen's algorithms, and the KK algorithm, are proportional to values $6^s, 7^{s_0}, 6^{s_0}$, where $s_0 = \log_2(n/b)$, respectively.

At this point, let us bring up another important issue with costing the algorithms. As s increases, the memory used by the algorithm also increases. Storing the original matrices requires roughly n^2 memory, and both Strassen and KK algorithms use essentially this same amount of memory as well. Our algorithm, by contrast, may require 6^s memory to store the expanded \bar{A}, \bar{B} ; if s is large, this may significantly increase the memory cost. Note that, for very large scale problems, the computational costs of the algorithm (i.e. total number of arithmetic operations) are dominated by the communication costs (i.e. moving memory across multiple nodes of a computer cluster.) In order to keep this section relatively contained, we will not investigate further the memory costs.

Next, we need to analyze the algorithm accuracy. Let us first consider the Boolean algorithm, where the comparison between our algorithm and the other candidate algorithms is more straightforward. Assuming we are using the reduction from Boolean matrix multiplication to GF(2) matrix multiplication, all three algorithms are randomized. Let us denote by f the maximum failure probability for any entry C_{ij} , if we use the matrix multiplication algorithm directly. Assuming we want to boost the algorithm to a much lower failure probability δ , we need to repeat for $\lceil \log(1/\delta)/\log(1/f) \rceil$ iterations. Indeed, if we want to compute the entire matrix C correctly, the value δ may need to be quite small (on the order of $1/n^2$). In order to put the algorithms on an even footing, let us therefore define a cost parameter

$$M = \frac{\text{Runtime}}{\log(1/f)} \quad (1)$$

The value f may depend on the number of witness k with $A_{ik}B_{kj} = 1$. The extremal case for all the algorithms appears to be the case where there is only one witness. In this case, the KK algorithm has failure probability $f = 1 - (7/8)^s/2$ and Strassen's algorithm has failure probability $1/2$. We thus have

$$M_{\text{Strassen}} = 7^s / \log 2, \quad M_{\text{KK}} = \frac{6^s}{-\log(1 - (7/8)^s/2)} \sim 2 \cdot (48/7)^s \approx 2 \cdot 6.857^s$$

To compare, we will compute the value M for our algorithm in the next section.

Next, let us consider a scenario for real-valued matrix multiplication. To make things concrete, let us suppose that A, B are non-negative matrices and we want to estimate the value C_{ij} up to $(1 \pm \epsilon)$ for some constant value $\epsilon = 1/2$ and with success probability $3/4$. In order to do so, we need to run the matrix multiplication algorithm for

$$L = \frac{4\mathbb{V}[\tilde{C}_{ij}]}{\epsilon^2\mathbb{E}[\tilde{C}_{ij}]^2} = \frac{16\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2}$$

trials. The runtime for our algorithm will thus be

$$6^s \cdot 16\mathbb{V}[\tilde{C}_{ij}]/\mathbb{E}[\tilde{C}_{ij}]^2$$

This comparison is among the favorable possible for our algorithm. Strassen’s algorithm has error probability zero and has perfect accuracy.

Although [11] did not discuss it, it is also possible to use KK for real-valued matrix multiplication (where we assume a random permutation of the rows and columns of A, B). We can estimate its runtime as $6^{s_0} \cdot 16(8/7)^{s_0}$, while Strassen’s algorithm will take time 7^{s_0} .

7.1 Case study: non-negative matrix multiplication

Let us consider a sample problem instance with $d_1 = d_2 = d_3 = 2^{25}$; as a reasonable choice for base case, let us take $b_1 = b_2 = b_3 = 2^7$. Note that storing the matrices already requires 2^{51} words of memory, which is at the upper limit of practical. Beyond this size, the problem would likely require a highly distributed memory system, and the runtime would be mostly determined by scheduling communications (as opposed to counting arithmetic operations).

With these parameters, Strassen and KK will set $s_0 = 18$, and they have runtimes of respectively

$$2^{50.5}, 2^{54.0}$$

In order to compare our algorithm to these two algorithms, to need to choose parameter s and we also need to determine the matrices G^A, G^B, G^C . We do not know how to select the matrices G in the optimal way, and the choice in Proposition 12 would be suboptimal by large constant factors. For our purposes, we used a local search method to select G , in order to minimize relative variance. (We note that, if instead we simply fixed all entries of G^A, G^B, G^C to be equal to one, the variance would only increase by roughly 10% – 20%)

Figure 1, following, shows the relative variance and runtime for various choices of parameter s , and for our chosen accuracy parameter $\varepsilon = 1/2$ and success probability $3/4$.

s	$\mathbb{V}[\tilde{C}_{ij}]/\mathbb{E}[\tilde{C}_{ij}]^2$	Runtime
10	25.9	55.8
11	23.1	55.6
12	20.3	55.4
13	17.6	55.2
14	14.8	55.0
15	12.1	54.8
16	9.4	54.8
17	6.9	54.8
18	4.5	55.0
19	2.4	55.6
20	0.7	56.4
21	-0.7	57.6

■ **Figure 1** Runtime of our algorithm for the sample problem. To handle the wide dynamic range, all figures are given in log base two; e.g., for $s = 10$, relative variance is $2^{25.9}$ and runtime is $2^{55.8}$.

The optimal value appears to be roughly $s \approx 15$, with runtime $2^{54.8}$. This is slightly worse than KK algorithm, and much worse than Strassen’s algorithm. We emphasize that this comparison is still unfair to Strassen’s algorithm, which provides precise and deterministic calculations of C_{ij} . Also, the base case choice here is still probably smaller than optimal, again making the comparisons unfairly favorable to the randomized algorithms.

Thus, despite the asymptotic advantages, it will probably never be profitable to our algorithm, or the KK algorithm, for real-valued matrix multiplication.

7.2 Estimating performance for Boolean matrix multiplication

We now turn to estimating the failure probability of our algorithm for the Boolean setting. Unlike the KK algorithm, the random process involved is much more complex and we cannot obtain a simple closed-form expression. Our starting point is to consider the random process of Theorem 9, and to try to estimate the probability p of the event $\bigwedge_{u \in T_s} \bar{E}_u$; note that $f \leq p$.

Let us consider a sample problem with $d_1 = d_2 = d_3 = 2^{25}$; following heuristics of [1], we might take the base case size as $b_1 = b_2 = b_3 = 2^{10}$. With these parameters, we can calculate

$$M_{\text{Strassen}} = 2^{42.64}, M_{\text{KK}} = 2^{42.61}$$

with both algorithms using $s_0 = 15$. (Recall the definition of M from Eq. (1).) To compare these with our algorithm, we use both empirical simulations (with 10^6 trials) on the hand, and Theorem 8 combined with Theorem 9 on the other hand, to obtain estimates of failure probability. Specifically, we write $\tilde{\kappa}$ for the estimate coming from Theorem 8 and we write $\hat{\kappa} = \log(1/\hat{p})$ for the Monte Carlo estimate.

By way of comparison, we can also calculate $\mu = 7^s q'_1 q'_2 q'_3$ to be the expected number of events E_u which hold. In an ideal setting, if the events E_u were completely independent, the failure probability would be $e^{-\mu}$, and we would have $\kappa = \mu$. See Figure 2.

s	μ	$\tilde{\kappa}$	$\hat{\kappa}$	M (upper bound)	M (empirical)
7	-26.3	-26.4		44.5	
8	-23.5	-23.6		44.2	
9	-20.7	-20.8	-19.9	44.0	43.2
10	-17.9	-18.0	-18.9	43.8	44.8
11	-15.1	-15.2	-15.2	43.6	43.5
12	-12.3	-12.5	-12.5	43.5	43.5
13	-9.5	-9.8	-9.7	43.4	43.3
14	-6.7	-7.3	-7.1	43.5	43.2
15	-3.9	-5.0	-4.5	43.7	43.3
16	-1.1	-2.9	-2.1	44.3	43.5
17	1.7	-1.2	-0.1	45.2	44.0
18	4.5	0.2	1.6	46.4	44.9
19	7.3	14	3.0	47.7	46.1
20	10.1	2.5		49.2	
21	13.0	3.5		50.8	

■ **Figure 2** Possible failure parameters for the algorithm. Figures are given in log base two. Some empirical estimates are left blank because statistically significant estimates could not be obtained.

The optimal value s appears to be roughly $s = 14$, and for this value we indeed have $\tilde{\kappa} \approx \hat{\kappa}$. Thus, a good heuristic to select s would be to use the calculated value $\tilde{\kappa}$ and choose s to minimize the resulting value M . This is much faster than empirical estimation, and it is also safer in that the resulting failure probability is a provable upper bound on the true failure probability.

8 Potential improvements

In the algorithms as we describe, the hash functions f_ℓ are chosen uniformly at random. The number of preimages of any $i \in [d_\ell]$ is thus a Binomial random variable with mean

$$\mu = \frac{2^s b_\ell}{d_\ell} = 2^s q_\ell$$

There are likely better ways to choose the random hash functions. For example, one could use a dependent rounding scheme to ensure that each $i \in [d_\ell]$ has precisely $\lfloor \mu \rfloor$ or $\lceil \mu \rceil$ preimages. One could even go further, aiming to ensure that the set of preimages $x \in f_\ell^{-1}(i)$ is “balanced” in terms of the Hamming weight of its elements.

These types of modifications would likely reduce the variance of the statistical estimates we develop. For practical parameter sizes, they could make a critical difference in the performance of the algorithm. However, these modifications are also much harder to analyze precisely. We leave as an open question the benefit of such improvements and whether they can make this algorithm fully practical.

References

- 1 Martin Albrecht, Gregory Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over $GF(2)$. *ACM Transactions on Mathematical Software (TOMS)*, 37(1):1–14, 2010.
- 2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
- 3 Vladimir L’vovich Arlazarov, Yefim A Dinitz, MA Kronrod, and Igor Aleksandrovich Faradzhev. On economical construction of the transitive closure of an oriented graph. In *Doklady Akademii Nauk*, volume 194, pages 487–488. Russian Academy of Sciences, 1970.
- 4 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen’s matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 193–204, 2012.
- 5 Austin R Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. *ACM SIGPLAN Notices*, 50(8):42–53, 2015.
- 6 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.
- 7 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.
- 8 Jianyu Huang, Tyler M Smith, Greg M Henry, and Robert A Van De Geijn. Strassen’s algorithm reloaded. In *SC’16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 690–701. IEEE, 2016.
- 9 Svante Janson, Tomasz Luczak, and Andrzej Rucinski. An exponential bound for the probability of nonexistence of a specified subgraph in a random graph. In *Random graphs*, volume 87, pages 73–87, 1990.
- 10 Matti Karppa. Techniques for similarity search and Boolean matrix multiplication. PhD thesis, 2020.
- 11 Matti Karppa and Petteri Kaski. Probabilistic tensors and opportunistic Boolean matrix multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–515. SIAM, 2019.

- 12 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012.
- 13 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Information and Computation*, 261:240–247, 2018.