Removing the log Factor from $(\min, +)$-Products on Bounded Range Integer Matrices

Dvir Fried
Bar-Ilan University, Ramat-Gan, Israel

Tsvi Kopelowitz
Bar-Ilan University, Ramat-Gan, Israel

Ely Porat
Bar-Ilan University, Ramat-Gan, Israel

Abstract

We revisit the problem of multiplying two square matrices over the $(\min, +)$ semi-ring, where all entries are integers from a bounded range $[-\mathrm{M} : \mathrm{M}] \cup \{\infty\}$. The current state of the art for this problem is a simple $O(\mathrm{M}n^{\omega} \log \mathrm{M})$ time algorithm by Alon, Galil and Margalit [JCSS'97], where ω is the exponent in the runtime of the fastest matrix multiplication (FMM) algorithm. We design a new simple algorithm whose runtime is $O(\mathrm{M}n^{\omega} + \mathrm{M}n^2 \log \mathrm{M})$, thereby removing the $\log \mathrm{M}$ factor in the runtime if $\omega > 2$ or if $n^{\omega} = \Omega(n^2 \log n)$.

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

Keywords and phrases FMM, $(\min, +)$-product, FFT

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.57

Funding Supported by ISF grant no. 1926/19, by a BSF grant 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

1 Introduction

One of the most fundamental algorithmic tasks is to compute the product of two matrices. In particular, the classic problem of matrix multiplication on two n by n matrices over a ring has been researched for decades [1, 8, 10, 15, 16, 21, 27]. Let ω be the exponent of n in the fastest matrix multiplication (FMM) algorithm (that is, the runtime is $O(n^{\omega})$). The current best upper bound on ω was recently given by [27] who showed that $\omega < 2.37156$. On the lower bound front, it is straightforward to see that $2 \leq \omega$. Raz [18] showed that any matrix multiplication algorithm in bounded coefficient arithmetic circuits requires at least $\Omega(n^2 \log n)$ time, implying that $\omega > 2 + o(1)$.

In addition to the classic matrix multiplication problem, the algorithmic community has also focused on several other important definitions of matrix products, including $(\min, +)$-product, $(\max, \min)$-Product, Dominance-Product, Witness-Product and more (see [3, 9, 17, 19, 23]). The focus of this paper is on the $(\min, +)$-product.

$(\min, +)$-product

For a matrix A, we denote by $A_{i,j}$ (or sometimes by $(A)_{i,j}$) the entry of A at the ith row and jth column. In the $(\min, +)$-product problem, the input is two $n \times n$ matrices S and T, and the goal is to compute an $n \times n$ matrix P where $P_{t,r} = \min_{1 \leq j \leq n}\{S_{t,j} + T_{j,r}\}$. The $(\min, +)$-product has strong connections with the all pairs shortest path (APSP) problem, and various other algorithmic problems that have efficient dynamic programming solutions ([4, 12, 22]).

32nd Annual European Symposium on Algorithms (ESA 2024).
Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 57; pp. 57:1–57:6
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The trivial algorithm for solving the $(\min,+)$-product problem runs in $O(n^3)$ time. The current fastest algorithm by Williams [24] runs in $\frac{n^3}{2^{\Omega(\log n)^{1/2}}}$ time. We remark that reducing the 3 term in the exponent of n in the runtime would refute the APSP conjecture [25]. Thus, several papers have focused on computing the $(\min,+)$-product for special families of input matrices [2, 4, 5, 6, 13, 14, 26].

In particular, Galil and Margalit [13, 14] considered the $(\min,+)$-product problem over a bounded range where each entry in the input matrices is either ∞ or an integer between $-\mathrm{M}$ and M, for some integer parameter M. The current fastest algorithm for the bounded range $(\min,+)$-product is given by Alon, Galil and Margalit [2] with a runtime of $O(\mathrm{M}n^\omega \log \mathrm{M})$. Throughout this paper, we refer to the algorithm of [2] as the AGM algorithm.

In this paper we essentially remove the $\log \mathrm{M}$ term from the runtime of the AGM algorithm and prove the following theorem.

▶ **Theorem 1.** *There exists an algorithm for the bounded range* $(\min,+)$*-product problem whose runtime is* $O(\mathrm{M}n^\omega + \mathrm{M}n^2 \log \mathrm{M})$.

Since we may assume that $\mathrm{M} \leq n^{3-\omega} < n$ (as otherwise the naïve algorithm is faster), if $\omega > 2$ or $n^\omega = \Omega(n^2 \log n)$ (which is true for bounded coefficient arithmetic circuits [18]), we conclude that the runtime is $O(\mathrm{M}n^\omega)$.

For some direct applications, our new algorithm improves the runtimes of the algorithms of Shoshan and Zwick [20] (after being adjusted by [11]), Chan, Vassilevska-Williams and Xu [8] for undirected graphs, and Zwick [28] for the APSP problem on undirected graphs, where the edge weights are integers in the range $\{1, ..., \mathrm{M}\}$.

2 Polynomials and FFT

The description of our algorithm relies on understanding the basic flow of the FFT based algorithm [7] for multiplying two polynomials. Thus, we begin with an overview of the FFT algorithm, with some notation that will assist in proving Theorem 1.

The FFT algorithm

For a polynomial $A(x) = \sum_{j=0}^{k} a_j x^j$, the degree of $A(x)$ is the largest power of x whose coefficient in $A(x)$ is non-zero. For a natural ℓ, the ℓ *roots of unity* are the (complex) numbers ${\omega_\ell}^t$ for integers $0 \leq t < \ell$, where[1] $\omega_\ell = \exp\left(\frac{-2\pi i}{\ell}\right)$.

The FFT algorithm receives as input a polynomial A of degree k and an integer $\ell > k$ that is a power of 2, and performs a discrete Fourier transform (DFT) on the coefficients of A, which produces the evaluation of A at the ℓ roots of unity.

Formally, let $\gamma_A = (a_0, \ldots, a_k)$ be the *coefficient* representation of A, and for $\ell > k$ that is a power of 2, let $\phi_{A,\ell} = (A(\omega_\ell^0), \ldots, A(\omega_\ell^{\ell-1}))$ be the ℓ*-sample* representation of A, which is the evaluation of A at the ℓ roots of unity. The DFT on γ_A is $\phi_{A,\ell}$.

The FFT algorithm is also used to invert the DFT[2], so that given $\phi_{A,\ell}$ where ℓ is a power of 2 and the degree of A is $k < \ell$, the inversion returns γ_A.

The cost of the FFT algorithm for both computing and inverting the DFT is $O(\ell \log \ell)$ time ([7]).

[1] Recall that $i^2 = -1$.

[2] Since the DFT on γ_A can be expressed as multiplying a Vandermonde matrix with γ_A, and since the Vandermonde matrix is invertible, the DFT is also invertible.

Operations on ℓ-sample representations

Let $\ell > 0$ be a power of 2. Let A and B be two polynomials of degrees k_A and k_B, respectively, such that $\max(k_A, k_B) < \ell$. Let $\phi_{A,\ell}$ and $\phi_{B,\ell}$ be the ℓ-*sample* representations of A and B, respectively.

The *pointwise addition* of $\phi_{A,\ell}$ and $\phi_{B,\ell}$ is $(A(\omega_\ell^0)+B(\omega_\ell^0), \ldots, A(\omega_\ell^{\ell-1})+B(\omega_\ell^{\ell-1}))$. Since the degree of $A(x) + B(x)$ is $\max(k_A, k_B) < \ell$, then the pointwise addition of $\phi_{A,\ell}$ and $\phi_{B,\ell}$ is $\phi_{A+B,\ell}$. The *pointwise multiplication* of $\phi_{A,\ell}$ and $\phi_{B,\ell}$ is $(A(\omega_\ell^0) \cdot B(\omega_\ell^0), \ldots, A(\omega_\ell^{\ell-1}) \cdot B(\omega_\ell^{\ell-1}))$. Since the degree of $A(x) \cdot B(x)$ is $k_A + k_B$, if $k_A + k_B < \ell$ then the pointwise multiplication of $\phi_{A,\ell}$ and $\phi_{B,\ell}$ is $\phi_{A \cdot B,\ell}$

Given $\phi_{A,\ell}$ and $\phi_{B,\ell}$, both pointwise addition and pointwise multiplication of $\phi_{A,\ell}$ and $\phi_{B,\ell}$ can be trivially computed in $O(\ell)$ time.

Multiplying polynomials using the FFT algorithm

Let $A(x)$ and $B(x)$ be two polynomials, each with degree at most k, represented by γ_A and γ_B, respectively. Our goal is to return γ_C where $C(x) = A(x) \cdot B(x)$. Notice that the degree of C is at most $2k$.

For $\ell > 2k$ that is a power of 2, the algorithm applies the FFT algorithm to compute the DFTs on γ_A and γ_B, which produces $\phi_{A,\ell}$ and $\phi_{B,\ell}$. Next, since $\phi_{C,\ell}$ is the pointwise multiplication of $\phi_{A,\ell}$ and $\phi_{B,\ell}$, the algorithm computes $\phi_{C,\ell}$ in $O(\ell)$ time. Finally, the algorithm uses the FFT algorithm to compute the inverse DFT on $\phi_{C,\ell}$, and returns γ_C. The total runtime is $O(\ell \log \ell)$.

3 $(\min,+)$-product With Bounded Range

The AGM algorithm

We describe the AGM algorithm, following along the lines of the description given in [5]. The AGM algorithm constructs two monomial matrices $S(x), T(x)$ from S, T as follows:

For each $t, j \in [n] \times [n]$ and for each $j, r \in [n] \times [n]$,

$$S(x)_{t,j} = \begin{cases} 0 & \text{if } S_{t,j} = \infty, \\ x^{\mathrm{M}-S_{t,j}} & \text{otherwise} \end{cases} \qquad T(x)_{j,r} = \begin{cases} 0 & \text{if } T_{j,r} = \infty, \\ x^{\mathrm{M}-T_{j,r}} & \text{otherwise} \end{cases}$$

Notice that the entries in both $S(x)$ and $T(x)$ are all monomials of degree at most $2\mathrm{M}$. Let S_γ and T_γ be two $n \times n$ matrices where for each $t, j \in [n] \times [n]$ we have $(S_\gamma)_{t,j} = \gamma_{S(x)_{t,j}}$, and for each $j, r \in [n] \times [n]$ we have $(T_\gamma)_{j,r} = \gamma_{T(x)_{j,r}}$. Notice that computing both S_γ and T_γ from S and T costs $O(n^2\mathrm{M})$ time in a trivial manner.

The AGM algorithm computes $P_\gamma = S_\gamma \cdot T_\gamma$ using a single application of an FMM algorithm over the matrices S_γ and T_γ, where each multiplication in the FMM algorithm is executed by applying the FFT-based polynomial multiplication algorithm between the coefficient representation of two polynomials. Notice that the output of each multiplication or addition during the execution of the FMM algorithm is a coefficient representation of a polynomial of degree at most $4\mathrm{M}$ (see [2]). Moreover, if $P(x) = S(x) \cdot T(x)$ then for each $t, r \in [n] \times [n]$ we have that $(P_\gamma)_{t,r} = \gamma_{P(x)_{t,r}}$.

Let $k_{t,r}$ be the degree of $P(x)_{t,r}$. The following lemma enables a method for extracting $P(x)$ from P_γ.

▶ **Lemma 2** ([2]). *If $P(x)_{t,r} = 0$ then $P_{t,r} = \infty$. Otherwise, $P_{t,r} = 2\mathrm{M} - k_{t,r}$.*

Proof. If $P(x)_{t,r} = 0$, then for each $j \in [n]$ it holds that $S(x)_{t,j} = 0 \vee T(x)_{j,r} = 0$, which implies that $S_{t,j} = \infty \vee T_{j,r} = \infty$, and so $P_{t,r} = \min_{1 \le j \le n}\{S_{t,j} + T_{j,r}\} = \infty$.

Otherwise, notice that for each $t, r \in [n] \times [n]$ we have $P(x)_{t,r} = \sum_{j \in [n]} x^{2\mathrm{M}-(S_{t,j}+T_{j,r})}$. Let $j^* = \arg\min\{S_{t,j}+T_{j,r}\} = \arg\max\{-(S_{t,j}+T_{j,r})\} = \arg\max\{2\mathrm{M}-(S_{t,j}+T_{j,r})\}$. Thus, $k_{t,r} = \max\{2\mathrm{M}-(S_{t,j}+T_{j,r})\} = 2\mathrm{M}-(S_{t,j^*}+T_{j^*,r}) = 2\mathrm{M}-P_{t,r}$, and so $P_{t,r} = 2\mathrm{M}-k_{t,r}$. ◀

Following Lemma 2, for each $t, r \in [n] \times [n]$, the algorithm scans $(P_\gamma)_{t,r}$ to deduce whether $P(x)_{t,r} = 0$, and, if so, set $P_{t,r} = \infty$. Otherwise, the algorithm computes $k_{t,r}$ and sets $P_{t,r} = 2\mathrm{M} - k_{t,r}$.

The construction of S_γ and T_γ from S and T costs $O(n^2\mathrm{M})$ time, and the construction of P from P_γ costs $O(n^2\mathrm{M})$ time. The execution of the FMM algorithm on S_γ and T_γ costs $O(n^\omega)$ multiplication or addition operations on 4M-sample representations of polynomials, and each such operation costs at most $O(\mathrm{M}\log\mathrm{M})$ time, so the total cost is $O(n^\omega\mathrm{M}\log\mathrm{M})$ time.

3.1 Proof of Theorem 1

Proof. The algorithm is obtained by replacing the computation of P_γ from S_γ and T_γ in the AGM algorithm with a more efficient procedure. Thus, we describe how to compute P_γ more efficiently, and the correctness follows from the correctness of the AGM algorithm.

Let $\hat{\mathrm{M}} = 2^{\lceil \log \mathrm{M} \rceil}$. Let S_ϕ and T_ϕ be two $n \times n$ matrices of $8\hat{\mathrm{M}}$-sample representations of the polynomials in $S(x)$ and $T(x)$, respectively. That is, for each $t, j \in [n] \times [n]$, $(S_\phi)_{t,j} = \phi_{S(x)_{t,j},8\hat{\mathrm{M}}}$, and $(T_\phi)_{t,j} = \phi_{T(x)_{t,j},8\hat{\mathrm{M}}}$. To compute S_ϕ and T_ϕ, for every entry in S_γ and T_γ the algorithm computes the DFT of the entry in total $O(\mathrm{M}n^2\log\mathrm{M})$ time.

Next, the algorithm computes $P_\phi = S_\phi \cdot T_\phi$, by executing an FMM algorithm on S_ϕ and T_ϕ, where each multiplication and addition operation during the execution of the FMM algorithm is performed on the $8\hat{\mathrm{M}}$-sample representations of two polynomials, each with degree at most 4M. Thus, at the end of the execution of the FMM algorithm, we have that for each $t, r \in [n] \times [n]$, $(P_\phi)_{t,r} = \phi_{P(x)_{t,r},8\hat{\mathrm{M}}}$. Finally, for each $t, r \in [n] \times [n]$, the algorithm computes the inverse DFT of $(P_\phi)_{t,r}$ to obtain $(P_\gamma)_{t,r}$.

Computing S_ϕ and T_ϕ costs $O(\mathrm{M}n^2\log\mathrm{M})$ time. During the execution of the FMM algorithm, each multiplication and addition is between two $8\hat{\mathrm{M}}$-sample representations, which costs $O(\mathrm{M})$ time, for a total of $O(\mathrm{M}n^\omega)$ time for the FMM execution. Finally, computing P_γ from P_ϕ costs $O(\mathrm{M}n^2\log\mathrm{M})$ time. Thus, the total time cost for computing P_γ from S_γ and T_γ is $O(\mathrm{M}n^\omega + \mathrm{M}n^2\log\mathrm{M})$ time. In addition, the rest of the operations of the AGM algorithm cost $O(\mathrm{M}n^2)$ time, for a total of $O(\mathrm{M}n^\omega + \mathrm{M}n^2\log\mathrm{M})$ time. ◀

References

1. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *SODA 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
2. Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.
3. Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 417–426. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267748.
4. Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.*, 48(2):481–512, 2019. doi:10.1137/17M112720X.

5 Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1435–1447. SIAM, 2022. doi:10.1137/1.9781611977073.60.

6 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, 2022*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

7 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

8 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

9 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 384–391. SIAM, 2009. doi:10.1137/1.9781611973068.43.

10 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2129–2138. IEEE, 2023. doi:10.1109/FOCS57990.2023.00130.

11 Pavlos Eirinakis, Matthew D. Williamson, and K. Subramani. On the shoshan-zwick algorithm for the all-pairs shortest path problem. *J. Graph Algorithms Appl.*, 21(2):177–181, 2017. doi:10.7155/JGAA.00410.

12 Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-dyck edit distance problem. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3650–3669. SIAM, 2022. doi:10.1137/1.9781611977073.144.

13 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.

14 Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *J. Comput. Syst. Sci.*, 54(2):243–254, 1997. doi:10.1006/jcss.1997.1385.

15 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

16 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *SODA 2018*, pages 1029–1046, 2018. doi:10.1137/1.9781611975031.67.

17 Jirí Matousek. Computing dominances in eˆn. *Inf. Process. Lett.*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-O.

18 Ran Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369, 2003. doi:10.1137/S0097539702402147.

19 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011. doi:10.1007/s00453-009-9328-x.

20 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 605–615. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814635.

21 Volker Strassen. Gaussian elimination is not optimal. *Matematika*, 13(5):354–356, 1969.

22 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

23 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory Comput.*, 5(1):173–189, 2009. doi:10.4086/toc.2009.v005a009.

24 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

25 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

26 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *SODA 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.

27 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega, 2023. arXiv:2307.07970.

28 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *CoRR*, cs.DS/0008011, 2000. URL: https://arxiv.org/abs/cs/0008011.