





# Core-Sparse Monge Matrix Multiplication



## Improved Algorithm and Applications

Paweł Gawrychowski  

University of Wrocław, Poland

Egor Gorbachev  

Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus,  
Saarbrücken, Germany

Tomasz Kociumaka  

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

---

### Abstract

*Min-plus matrix multiplication* is a fundamental tool for designing algorithms operating on distances in graphs and different problems solvable by dynamic programming. We know that, assuming the APSP hypothesis, no subcubic-time algorithm exists for the case of general matrices. However, in many applications the matrices admit certain structural properties that can be used to design faster algorithms. For example, when considering a planar graph, one often works with a *Monge matrix*  $A$ , meaning that the *density matrix*  $A^\square$  has non-negative entries, that is,  $A_{i,j}^\square := A_{i+1,j} + A_{i,j+1} - A_{i,j} - A_{i+1,j+1} \geq 0$ . The min-plus product of two  $n \times n$  Monge matrices can be computed in  $\mathcal{O}(n^2)$  time using the famous SMAWK algorithm.

In applications such as longest common subsequence, edit distance, and longest increasing subsequence, the matrices are even more structured, as observed by Tiskin [J. Discrete Algorithms, 2008]: they are (or can be converted to) *simple unit-Monge matrices*, meaning that the density matrix is a permutation matrix and, furthermore, the first column and the last row of the matrix consist of only zeroes. Such matrices admit an implicit representation of size  $\mathcal{O}(n)$  and, as shown by Tiskin [SODA 2010 & Algorithmica, 2015], their min-plus product can be computed in  $\mathcal{O}(n \log n)$  time. Russo [SPIRE 2010 & Theor. Comput. Sci., 2012] identified a general structural property of matrices that admit such efficient representation and min-plus multiplication algorithms: the *core size*  $\delta$ , defined as the number of non-zero entries in the density matrices of the input and output matrices. He provided an adaptive implementation of the SMAWK algorithm that runs in  $\mathcal{O}((n + \delta) \log^3 n)$  or  $\mathcal{O}((n + \delta) \log^2 n)$  time (depending on the representation of the input matrices).

In this work, we further investigate the core size as the parameter that enables efficient min-plus matrix multiplication. On the combinatorial side, we provide a (linear) bound on the core size of the product matrix in terms of the core sizes of the input matrices. On the algorithmic side, we generalize Tiskin's algorithm (but, arguably, with a more elementary analysis) to solve the core-sparse Monge matrix multiplication problem in  $\mathcal{O}(n + \delta \log \delta) \subseteq \mathcal{O}(n + \delta \log n)$  time, matching the complexity for simple unit-Monge matrices. As witnessed by the recent work of Gorbachev and Kociumaka [STOC'25] for edit distance with integer weights, our generalization opens up the possibility of speed-ups for weighted sequence alignment problems. Furthermore, our multiplication algorithm is also capable of producing an efficient data structure for recovering the witness for any given entry of the output matrix. This allows us, for example, to preprocess an integer array of size  $n$  in  $\tilde{\mathcal{O}}(n)$  time so that the longest increasing subsequence of any sub-array can be reconstructed in  $\tilde{\mathcal{O}}(\ell)$  time, where  $\ell$  is the length of the reported subsequence. In comparison, Karthik C. S. and Rahul [arXiv, 2024] recently achieved  $\mathcal{O}(\ell + n^{1/2} \text{polylog } n)$ -time reporting after  $\mathcal{O}(n^{3/2} \text{polylog } n)$ -time preprocessing.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Design and analysis of algorithms

**Keywords and phrases** Min-plus matrix multiplication, Monge matrix, longest increasing subsequence

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2025.74

**Related Version** *Full Version:* <https://arxiv.org/abs/2408.04613v2> [21]



© Paweł Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka;  
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 74; pp. 74:1–74:17

Leibniz International Proceedings in Informatics



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

**Funding** *Egor Gorbachev*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

## 1 Introduction

The *min-plus product* (also known as the distance product or the tropical product) of two matrices  $A$  and  $B$  is defined as a matrix  $C = A \otimes B$  such that  $C_{i,k} = \min_j (A_{i,j} + B_{j,k})$ . The task of computing the min-plus product of two  $n \times n$  matrices can be solved in  $n^3 / \exp(\Omega(\sqrt{\log n}))$  time [46], and it is fine-grained equivalent to the All-Pairs Shortest Path (APSP) problem [44], asking to compute the distances between every pair of vertices in a directed weighted graph on  $n$  vertices. While it is conjectured that APSP, and hence also the min-plus product, do not admit  $n^{3-\Omega(1)}$ -time solutions, faster algorithms exist for many special cases arising in numerous applications of the min-plus product; see, e.g., [2, 6, 10, 15–17, 23, 43, 47, 49]. Most of the underlying procedures rely on fast matrix multiplication for the standard  $(+, \cdot)$ -product.

*Monge matrices* constitute a notable exception: An  $n \times n$  matrix  $A$  is a Monge matrix if its *density matrix*  $A^\square$  is non-negative, that is,  $A_{i,j}^\square := A_{i+1,j} + A_{i,j+1} - A_{i,j} - A_{i+1,j+1} \geq 0$  holds for all  $i, j \in [0..n-1]$ . The min-plus product of two  $n \times n$  Monge matrices can be computed in  $\mathcal{O}(n^2)$  time using the SMAWK algorithm [1], and the resulting matrix still satisfies the Monge property. Monge matrices arise in many combinatorial optimization problems; see [7, 8] for surveys. One of the most successful applications is for planar graphs, where the distances between vertices on a single face satisfy the Monge property (see [18, Section 2.3]). This observation allowed for an  $\tilde{\mathcal{O}}(n)$ -time<sup>1</sup> single-source shortest path algorithm for planar graphs (with negative real weights) [18], and the resulting techniques now belong to the standard toolkit for designing planar graph algorithms; see, e.g., [5, 11, 25].

Another important application of Monge matrices is in sequence alignment problems, such as *edit distance* and *longest common subsequence* (LCS), as well as in the related *longest increasing subsequence* (LIS) problem. Already in the late 1980s, Apostolico, Atallah, Larmore, and McFaddin [3] noted that the so-called *DIST matrices*, which (among others) specify the weighted edit distances between prefixes of one string and suffixes of another string, satisfy the Monge property. A modern interpretation of this observation is that these matrices store boundary-to-boundary distances in planar *alignment graphs* [37]; see also [4, 26, 32] for further early applications of DIST matrices and their Monge property.

In the late 2000s, Tiskin [39, 42] observed that the DIST matrices originating from the unweighted variants of edit distance, LCS, and LIS problems are more structured. For this, he introduced the notions of *unit-Monge matrices*, whose density matrices are permutation matrices (that is, binary matrices whose rows and columns contain exactly a single 1 entry each) and *simple Monge matrices*, whose leftmost column and bottommost row consist of zeroes. He also proved that the product of two simple unit-Monge matrices still belongs to this class and can be computed in  $\mathcal{O}(n \log n)$  time provided that each matrix  $A$  is represented using the underlying permutation  $P_A$  [41]. By now, the resulting algorithm has found numerous applications, including for computing LCS and edit distance of compressed strings [19, 20, 24, 41], maintaining these similarity measures for dynamic strings [13, 22], approximate pattern matching [14, 40], parallel and distributed algorithms for similarity measures [31, 33], and oracles for substring similarity [12, 35, 36]. Furthermore, Tiskin's algorithm has been used to solve the LIS problem in various settings, such as dynamic [28],

<sup>1</sup> Throughout this paper, we use  $\tilde{\mathcal{O}}(\cdot)$  notation to suppress factors poly-logarithmic in the input size.

parallel [9], and distributed [30]. A disadvantage of Tiskin’s original description (and even the later informal descriptions [29]) is its dependence on the algebraic structure known as the monoid of seaweed braids, which natively supports unweighted LCS only (tasks involving edit distance need to be reduced to LCS counterparts). This makes the algorithm difficult to generalize to weighted problems and extend even to seemingly simple questions such as recovering (an implicit representation of) the witness indices  $j$  such that  $C_{i,k} = A_{i,j} + B_{j,k}$  [27].

Russo [34] identified the number of non-zero elements in  $A^\square$ ,  $B^\square$ , and  $(A \otimes B)^\square$ , called the *core size*  $\delta$ , as the parameter that enables fast min-plus matrix multiplication. It is easy to see that  $A$ ,  $B$ , and  $A \otimes B$  can be stored in  $\mathcal{O}(n + \delta)$  space using a *condensed representation*, consisting of the *boundary entries* (the leftmost column and bottommost row, e.g., suffice) and *core elements*, i.e., the non-zero entries of the density matrix. Then, Russo’s algorithm for the *core-sparse Monge matrix multiplication* problem computes the condensed representation of  $A \otimes B$  in  $\mathcal{O}((n + \delta) \log^2 n)$  time when provided with constant-time random access to  $A$  and  $B$ , and in  $\mathcal{O}((n + \delta) \log^3 n)$  time given condensed representations of  $A$  and  $B$ .<sup>2</sup> Russo’s algorithm has a very different structure than Tiskin’s: it relies on a clever adaptation of the SMAWK algorithm [1] to provide efficient random access to  $C$  and then employs binary search to find individual non-zero entries of  $C^\square$ . This brings the question of unifying both approaches and understanding the complexity of core-sparse Monge matrix multiplication.

**Our Results.** We consider the core-sparse Monge matrix multiplication problem from the combinatorial and algorithmic points of view, and confirm that the core size is the right parameter that enables fast min-plus matrix multiplication. Let  $\delta(A)$ , or the core size of  $A$ , denote the number of non-zero elements in  $A^\square$ . We begin with analyzing, in Section 3, how the core size of  $A \otimes B$  depends on the core sizes of  $A$  and  $B$ .

► **Theorem 1.1.** *Let  $A$  be a  $p \times q$  Monge matrix, and let  $B$  be a  $q \times r$  Monge matrix. We have  $\delta(A \otimes B) \leq 2 \cdot (\delta(A) + \delta(B))$ .*

We stress that this is the first bound on  $\delta(A \otimes B)$  in terms of  $\delta(A)$  and  $\delta(B)$ : the complexity analysis of Russo’s algorithm [34] requires a bound on all  $\delta(A)$ ,  $\delta(B)$ , and  $\delta(A \otimes B)$ .

Next, in Section 4, we generalize Tiskin’s algorithm (but fully avoiding the formalism of the seaweed product) to solve the core-sparse Monge matrix multiplication problem. We believe that the more elementary interpretation makes our viewpoint not only more robust but also easier to understand. At the same time, the extension from simple unit-Monge matrices to general core-sparse Monge matrices introduces a few technical complications handled in the full version of the paper [21]. Notably, we need to keep track of the leftmost column and the bottommost row instead of assuming they are filled with zeroes. Further, the core does not form a permutation so splitting it into two halves of the same size requires some calculations.

► **Theorem 1.2.** *There is a (deterministic) algorithm that, given the condensed representations of a  $p \times q$  Monge matrix  $A$  and a  $q \times r$  Monge matrix  $B$ , in time  $\mathcal{O}(p + q + r + (\delta(A) + \delta(B)) \log(1 + \delta(A) + \delta(B)))$  computes the condensed representation of  $A \otimes B$ .<sup>3</sup>*

<sup>2</sup> More precisely, Russo’s algorithm builds a data structure that provides  $\mathcal{O}(\log n)$ -time random access to the entries of  $A \otimes B$ . Consequently, for repeated multiplication, we cannot assume constant-time access to the input matrices. Hence,  $\mathcal{O}((n + \delta) \log^3 n)$  is a more realistic bound.

<sup>3</sup> In the technical sections of the paper we use the  $\mathcal{O}(\cdot)$ -notation conservatively. Specifically, we interpret  $\mathcal{O}(f(x_1, \dots, x_k))$  as the set of functions  $g(x_1, \dots, x_k)$  for which there are constants  $c_g, N_g > 0$  such that  $g(x_1, \dots, x_k) \leq c_g \cdot f(x_1, \dots, x_k)$  holds for all valid tuples  $(x_1, \dots, x_k)$  satisfying  $\max_i x_i \geq N_g$ .

The above complexity improves upon Russo's [34] and matches Tiskin's [41] for the simple unit-Monge case. Thanks to the more direct description, we can easily extend our algorithm to build (in the same complexity) an  $\mathcal{O}(n + \delta)$ -size data structure that, given  $(i, k)$ , in  $\mathcal{O}(\log n)$  time computes the smallest *witness*  $j$  such that  $C_{i,k} = A_{i,j} + B_{j,k}$ .

**Applications.** As an application of our witness recovery functionality, we consider the problem of range LIS queries. This task asks to preprocess an integer array  $s[0..n)$  so that, later, given two indices  $0 \leq i < j \leq n$ , the longest increasing subsequence (LIS) of the sub-array  $s[i..j)$  can be reported efficiently. Tiskin [38, 41] showed that the LIS size  $\ell$  can be reported in  $\mathcal{O}(\log n)$  time after  $\mathcal{O}(n \log^2 n)$ -time preprocessing. It was unclear, though, how to efficiently report the LIS itself. The recent work of Karthik C. S. and Rahul [27] achieved  $\mathcal{O}(n^{1/2} \log^3 n + \ell)$ -time reporting (correct with high probability) after  $\mathcal{O}(n^{3/2} \log^3 n)$ -time preprocessing. It is fairly easy to use our witness recovery algorithm to deterministically support  $\mathcal{O}(\ell \log^2 n)$ -time reporting after  $\mathcal{O}(n \log^2 n)$ -time preprocessing; see Section 5 for an overview of this result. As further shown in the full version of the paper [21], the reporting time can be improved to  $\mathcal{O}(\ell \log n)$  and, with the preprocessing time increased to  $\mathcal{O}(n \log^3 n)$ , all the way to  $\mathcal{O}(\ell)$ .

► **Theorem 1.3.** *For every parameter  $\alpha \in [0, 1]$ , there exists an algorithm that, given an integer array  $s[0..n)$ , in time  $\mathcal{O}(n \log^{3-\alpha} n)$  builds a data structure that can answer range LIS reporting queries in time  $\mathcal{O}(\ell \log^\alpha n)$ , where  $\ell$  is the length of the reported sequence.*

*In particular, there is an algorithm with  $\mathcal{O}(n \log^3 n)$  preprocessing and  $\mathcal{O}(\ell)$  reporting time and an algorithm with  $\mathcal{O}(n \log^2 n)$  preprocessing and  $\mathcal{O}(\ell \log n)$  reporting time.*

In parallel to this work, Gorbachev and Kociumaka [22] used core-sparse Monge matrix multiplication for the *weighted edit distance* with integer weights. Theorem 1.2 allowed for saving two logarithmic factors in the final time complexities compared to the initial preprint using Russo's approach [34]. Weighted edit distance is known to be reducible to unweighted LCS only for a very limited class of so-called uniform weight functions [38], so this application requires the general core-sparse Monge matrix multiplication.

**Open Problem.** An interesting open problem is whether any non-trivial trade-off can be achieved for the *weighted* version of range LIS queries, where each element of  $s$  has a weight, and the task is to compute a maximum-weight increasing subsequence of  $s[i..j)$ : either the weight alone or the whole subsequence. Surprisingly, as we show in the full version of the paper [21], if our bound  $\delta(A \otimes B) \leq 2 \cdot (\delta(A) + \delta(B))$  of Theorem 1.1 can be improved to  $\delta(A \otimes B) \leq c \cdot (\delta(A) + \delta(B)) + \tilde{\mathcal{O}}(p + q + r)$  for some  $1 \leq c < 2$ , then our techniques automatically yield a solution with  $\tilde{\mathcal{O}}(n^{1+\log_2 c})$  preprocessing time,  $\tilde{\mathcal{O}}(1)$  query time, and  $\tilde{\mathcal{O}}(\ell)$  reporting time.

## 2 Preliminaries

► **Definition 2.1.** *A matrix  $A$  of size  $p \times q$  is a Monge matrix if it satisfies the following Monge property for every  $i \in [0..p-1)$  and  $j \in [0..q-1)$ :*

$$A_{i,j} + A_{i+1,j+1} \leq A_{i,j+1} + A_{i+1,j}.$$

*Furthermore,  $A$  is an anti-Monge matrix if the matrix  $-A$  (with negated entries) is a Monge matrix.*

---

Accordingly, whenever the expression inside  $\mathcal{O}(\cdot)$  depends on multiple parameters, we sometimes add 1 or 2 to the arguments of logarithms to ensure formal correctness in corner cases.

In some sources, the following equivalent (but seemingly stronger) condition is taken as the definition of Monge matrices.

► **Observation 2.2.** *A matrix  $A$  of size  $p \times q$  is a Monge matrix if and only if it satisfies the following inequality for all integers  $0 \leq a \leq b < p$  and  $0 \leq c \leq d < q$ :*

$$A_{a,c} + A_{b,d} \leq A_{a,d} + A_{b,c}.$$

**Proof.** It suffices to sum the inequality in Definition 2.1 for all  $i \in [a..b)$  and  $j \in [c..d)$ . ◀

► **Definition 2.3.** *The min-plus product of a matrix  $A$  of size  $p \times q$  and a matrix  $B$  of size  $q \times r$  is a matrix  $A \otimes B = C$  of size  $p \times r$  satisfying  $C_{i,k} = \min_{j \in [0..q)} A_{i,j} + B_{j,k}$  for all  $i \in [0..p)$  and  $k \in [0..r)$ .*

*For  $i \in [0..p)$  and  $k \in [0..r)$ , we call  $j \in [0..q)$  a witness of  $C_{i,k}$  if and only if  $C_{i,k} = A_{i,j} + B_{j,k}$ . We define the  $p \times r$  witness matrix  $\mathcal{W}^{A,B}$  such that  $\mathcal{W}_{i,k}^{A,B}$  is the smallest witness of  $C_{i,k}$  for each  $i \in [0..p)$  and  $k \in [0..r)$ .*

The min-plus product of two Monge matrices is also Monge.

► **Fact 2.4** ([48, Corollary A]). *Let  $A, B$ , and  $C$  be matrices such that  $A \otimes B = C$ . If  $A$  and  $B$  are Monge, then  $C$  is also Monge.*

In the context of Monge matrices, it is useful to define the core and the density matrix.

► **Definition 2.5.** *The density matrix of a matrix  $A$  of size  $p \times q$  is a matrix  $A^\square$  of size  $(p-1) \times (q-1)$  satisfying  $A_{i,j}^\square = A_{i,j+1} + A_{i+1,j} - A_{i,j} - A_{i+1,j+1}$  for  $i \in [0..p-1)$ ,  $j \in [0..q-1)$ . We define the core of the matrix  $A$  as*

$$\text{core}(A) := \{(i, j, A_{i,j}^\square) \mid i \in [0..p-1), j \in [0..q-1), A_{i,j}^\square \neq 0\}$$

*and denote the core size of the matrix  $A$  by  $\delta(A) := |\text{core}(A)|$ . Furthermore, we define the core sum of the matrix  $A$  as the sum of the values of all core elements of  $A$ , that is,*

$$\delta^\Sigma(A) := \sum_{i \in [0..p-1)} \sum_{j \in [0..q-1)} A_{i,j}^\square.$$

Note that, for a Monge matrix  $A$ , all entries of its density matrix are non-negative, and thus  $\text{core}(A)$  consists of triples  $(i, j, v)$  with some positive values  $v$ .

For any matrix  $A$  of size  $p \times q$  and integers  $0 \leq a < b \leq p$  and  $0 \leq c < d \leq q$ , we write  $A[a..b][c..d)$  to denote the contiguous submatrix of  $A$  consisting of all entries on the intersection of rows  $[a..b)$  and columns  $[c..d)$  of  $A$ . Matrices  $A[a..b][c..d]$ ,  $A(a..b)[c..d)$ , etc., are defined analogously.

### 3 Properties of $\delta$ and $\delta^\Sigma$

In this section, we provide some useful properties of  $\delta$  and  $\delta^\Sigma$ . Most importantly, we show how to bound  $\delta(A \otimes B)$  in terms of  $\delta(A)$  and  $\delta(B)$ .

The following observation is a straightforward consequence of the definitions of  $\delta^\Sigma$  and  $A^\square$ .

► **Observation 3.1.** *For a matrix  $A$  of size  $p \times q$ , and integers  $0 \leq a \leq b < p$  and  $0 \leq c \leq d < q$ ,*

$$A_{a,c} + A_{b,d} + \delta^\Sigma(A[a..b][c..d]) = A_{a,d} + A_{b,c}.$$

**Proof.** By the definition of  $A^\square$ , we have  $A_{i,j} + A_{i+1,j+1} + A_{i,j}^\square = A_{i,j+1} + A_{i+1,j}$  for all  $i \in [a..b)$  and  $j \in [c..d)$ . The desired equality follows by summing up all these equalities. ◀

An application of Observation 3.1 for  $a = c = 0$  implies that every value of  $A$  can be uniquely reconstructed from  $A$ 's core and the values in the topmost row and the leftmost column of  $A$ .

► **Definition 3.2.** *The condensed representation of a matrix  $A$  consists of the core of  $A$  as well as the values in the topmost row and the leftmost column of  $A$ .*

► **Observation 3.3.** *Any submatrix  $A'$  (not necessarily contiguous) of a Monge matrix  $A$  is Monge.*

**Proof.** By Observation 2.2, if  $A$  is Monge, the Monge property holds for every (not necessarily contiguous)  $2 \times 2$  submatrix of  $A$ . In particular, it holds for contiguous  $2 \times 2$  submatrices of  $A'$ , and thus  $A'$  satisfies Definition 2.1. ◀

We next show a crucial property of the witness matrix.

► **Lemma 3.4** ([48, Theorem 1]). *For any two Monge matrices,  $A$  of size  $p \times q$  and  $B$  of size  $q \times r$ , the witness matrix  $\mathcal{W}^{A,B}$  is non-decreasing by rows and columns.*

**Proof.** We prove that  $\mathcal{W}^{A,B}$  is non-decreasing by columns. The claim for rows is analogous. Fix some  $i \in [0..p-1)$  and  $k \in [0..r)$ . Let  $j^* := \mathcal{W}_{i,k}^{A,B}$  and  $j \in [0..j^*)$ . As  $j^*$  is the smallest witness for  $(i, k)$ , we have  $A_{i,j} + B_{j,k} > A_{i,j^*} + B_{j^*,k}$ . Using this fact and the Monge property for  $A$ , we derive

$$\begin{aligned} A_{i+1,j} + B_{j,k} &\geq (A_{i,j} + A_{i+1,j^*} - A_{i,j^*}) + B_{j,k} \\ &= (A_{i,j} + B_{j,k}) + (A_{i+1,j^*} - A_{i,j^*}) \\ &> (A_{i,j^*} + B_{j^*,k}) + (A_{i+1,j^*} - A_{i,j^*}) \\ &= A_{i+1,j^*} + B_{j^*,k}. \end{aligned}$$

Since  $A_{i+1,j} + B_{j,k} > A_{i+1,j^*} + B_{j^*,k}$  for all  $j \in [0..j^*)$ , we conclude that  $\mathcal{W}_{i+1,k}^{A,B} \geq j^* = \mathcal{W}_{i,k}^{A,B}$  holds as required. ◀

We now show how to bound  $\delta^\Sigma(A \otimes B)$  in terms of  $\delta^\Sigma(A)$  and  $\delta^\Sigma(B)$ .

► **Lemma 3.5.** *Let  $A$  be a  $p \times q$  Monge matrix, and let  $B$  be a  $q \times r$  Monge matrix. Then,  $\delta^\Sigma(A \otimes B) \leq \min\{\delta^\Sigma(A), \delta^\Sigma(B)\}$ .*

**Proof.** Let  $C := A \otimes B$ . Denote  $j := \mathcal{W}_{0,0}^{A,B}$  and  $j' := \mathcal{W}_{p-1,r-1}^{A,B}$ , where  $j \leq j'$  due to Lemma 3.4. We have  $C_{0,0} = A_{0,j} + B_{j,0}$  and  $C_{p-1,r-1} = A_{p-1,j'} + B_{j',r-1}$ . Furthermore,  $C_{p-1,0} \leq A_{p-1,j} + B_{j,0}$  and  $C_{0,r-1} \leq A_{0,j'} + B_{j',r-1}$  due to the definition of  $C$ . Hence, due to Observation 3.1, we obtain

$$\begin{aligned} \delta^\Sigma(C) &= C_{p-1,0} + C_{0,r-1} - C_{0,0} - C_{p-1,r-1} \\ &\leq (A_{p-1,j} + B_{j,0}) + (A_{0,j'} + B_{j',r-1}) - (A_{0,j} + B_{j,0}) - (A_{p-1,j'} + B_{j',r-1}) \\ &= A_{p-1,j} + A_{0,j'} - A_{0,j} - A_{p-1,j'} \\ &= \delta^\Sigma(A[0..p][j..j']) \\ &\leq \delta^\Sigma(A). \end{aligned}$$

The inequality  $\delta^\Sigma(C) \leq \delta^\Sigma(B)$  can be obtained using a symmetric argument. ◀

The following corollary says that every core element of  $A \otimes B$  can be attributed to at least one core element of  $A$  and at least one core element of  $B$ .

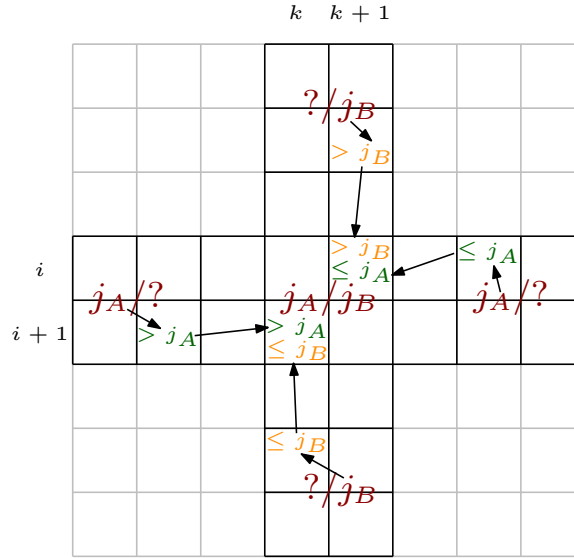
► **Corollary 3.6.** *Let  $A$  be a  $p \times q$  Monge matrix, let  $B$  be a  $q \times r$  Monge matrix, and let  $C := A \otimes B$ . Consider integers  $i \in [0..p-1)$  and  $k \in [0..r-1)$  such that  $C_{i,k}^\square \neq 0$ . There exist integers  $j_A, j_B \in [\mathcal{W}_{i,k}^{A,B}.. \mathcal{W}_{i+1,k+1}^{A,B})$  such that  $A_{i,j_A}^\square \neq 0$  and  $B_{j_B,k}^\square \neq 0$ .*

**Proof.** Let  $j = \mathcal{W}_{i,k}^{A,B}$  and  $j' = \mathcal{W}_{i+1,k+1}^{A,B}$ . By monotonicity of  $\mathcal{W}^{A,B}$  (Lemma 3.4), we have  $j \leq \mathcal{W}_{i,k+1}^{A,B}, \mathcal{W}_{i+1,k}^{A,B} \leq j'$ . Thus,  $C[i..i+1][k..k+1] = A[i..i+1][j..j'] \otimes B[j..j'][k..k+1]$ . Due to Lemma 3.5, we have  $0 < C_{i,k}^\square = \delta^\Sigma(C[i..i+1][k..k+1]) \leq \delta^\Sigma(A[i..i+1][j..j'])$ . Thus, there exists a core element in  $A^\square[i..i+1][j..j']$ .

Symmetrically,  $0 < C_{i,k}^\square = \delta^\Sigma(C[i..i+1][k..k+1]) \leq \delta^\Sigma(B[j..j'][k..k+1])$ , and there exists a core element in  $B^\square[j..j'][k..k+1]$ . ◀

We now use Corollary 3.6 to derive a bound on  $\delta(A \otimes B)$  in terms of  $\delta(A)$  and  $\delta(B)$ . The underlying idea is to show that every core element of  $C$  is either the first or the last one (in the lexicographic ordering) that the mapping of Corollary 3.6 attributes to the corresponding core element of  $A$  or  $B$  (see Figure 1 for why the opposite would lead to a contradiction).

► **Theorem 1.1.** *Let  $A$  be a  $p \times q$  Monge matrix, and let  $B$  be a  $q \times r$  Monge matrix. We have  $\delta(A \otimes B) \leq 2 \cdot (\delta(A) + \delta(B))$ .*



■ **Figure 1** We consider the matrix of witnesses  $\mathcal{W}^{A,B}$ , drawn as an array with rows in  $[0..p)$  indexed from top to bottom and columns in  $[0..r)$  indexed from left to right. In the cells, we write bounds on the corresponding witnesses. In the grid nodes, we write the two values in  $[0..q-1)$  derived from Corollary 3.6 for each core element of  $C$ . The center of the picture corresponds to a core value  $C_{i,k}^\square$ , attributed to  $A_{i,j_A}^\square$  and  $B_{j_B,k}^\square$ , respectively. For a proof by contradiction, suppose that there are some core elements to the left and right of  $(i, k)$  attributed to  $A_{i,j_A}^\square$  and some core elements above and below  $(i, k)$  attributed to  $B_{j_B,k}^\square$ . The arrows represent implications (based on Corollary 3.6 and Lemma 3.4), from which a contradiction follows:  $j_A < \mathcal{W}_{i+1,k}^{A,B} \leq j_B < \mathcal{W}_{i,k+1}^{A,B} \leq j_A$ .



**Proof.** Let  $C := A \otimes B$ . Define a function  $f_A : \text{core}(C) \rightarrow \text{core}(A)$  that maps every core element  $(i, k, C_{i,k}^\square) \in \text{core}(C)$  to  $(i, j, A_{i,j}^\square)$  for the smallest  $j \in [\mathcal{W}_{i,k}^{A,B} \dots \mathcal{W}_{i+1,k+1}^{A,B}]$  with  $A_{i,j}^\square \neq 0$ ; such a  $j$  exists due to Corollary 3.6. Analogously, we define a function  $f_B : \text{core}(C) \rightarrow \text{core}(B)$  that maps every core element  $(i, k, C_{i,k}^\square) \in \text{core}(C)$  to  $(j, k, B_{j,k}^\square)$  for the smallest  $j \in [\mathcal{W}_{i,k}^{A,B} \dots \mathcal{W}_{i+1,k+1}^{A,B}]$  with  $B_{j,k}^\square \neq 0$ ; again, such a  $j$  exists due to Corollary 3.6.

▷ **Claim 3.7.** Every core element  $c \in \text{core}(C)$  is the lexicographically minimal or maximal one in its pre-image  $f_A^{-1}(f_A(c))$  under  $f_A$  or its pre-image  $f_B^{-1}(f_B(c))$  under  $f_B$ .

**Proof.** For a proof by contradiction, pick an element  $c := (i, k, C_{i,k}^\square) \in \text{core}(C)$  violating the claim. Let  $(i, j_A, A_{i,j_A}^\square) := a := f_A(c)$  and  $(j_B, k, B_{j_B,k}^\square) := b := f_B(c)$ . We make four symmetric arguments (all illustrated in Figure 1), with the first one presented in more detail.

1. Since  $c$  is not the minimal core element in  $f_A^{-1}(a)$ , we have  $f_A(c') = a$  for some core element  $(i', k', C_{i',k'}^\square) := c' \in \text{core}(C)$  that precedes  $c$  in the lexicographical order, denoted  $c' \prec c$ . Due to  $f_A(c') = a = (i, j_A, A_{i,j_A}^\square)$ , we must have  $i' = i$  and  $j_A < \mathcal{W}_{i+1,k'+1}^{A,B}$ . Since  $i' = i$ , then  $c' \prec c$  implies  $k' + 1 \leq k$ . The monotonicity of the witness matrix (Lemma 3.4) thus yields  $\mathcal{W}_{i+1,k'+1}^{A,B} \leq \mathcal{W}_{i+1,k}^{A,B}$ . Overall, we conclude that  $j_A < \mathcal{W}_{i+1,k}^{A,B}$ .
2. Since  $c$  is not the maximal core element in  $f_A^{-1}(a)$ , we have  $f_A(c') = a$  for some core element  $c' \succ c$ . In particular,  $c' = (i, k', C_{i,k'}^\square) \in \text{core}(C)$  for some  $k' > k$ . Then,  $\mathcal{W}_{i,k+1}^{A,B} \leq \mathcal{W}_{i,k'}^{A,B} \leq j_A$  follows from Lemma 3.4 and  $f_A(c') = a$ , respectively.
3. Since  $c$  is not the minimal core element in  $f_B^{-1}(b)$ , we have  $f_B(c') = b$  for some core element  $c' \prec c$ . In particular,  $c' = (i', k, C_{i',k}^\square) \in \text{core}(C)$  for some  $i' < i$ . Then,  $\mathcal{W}_{i,k+1}^{A,B} \geq \mathcal{W}_{i'+1,k+1}^{A,B} > j_B$  follows from Lemma 3.4 and  $f_B(c') = b$ , respectively.
4. Since  $c$  is not the maximal core element in  $f_B^{-1}(b)$ , we have  $f_B(c') = b$  for some core element  $c' \succ c$ . In particular,  $c' = (i', k, C_{i',k}^\square) \in \text{core}(C)$  for some  $i' > i$ . Then,  $\mathcal{W}_{i+1,k}^{A,B} \leq \mathcal{W}_{i',k}^{A,B} \leq j_B$  follows from Lemma 3.4 and  $f_B(c') = b$ , respectively.

Overall, we derive a contradiction:  $j_A < \mathcal{W}_{i+1,k}^{A,B} \leq j_B < \mathcal{W}_{i,k+1}^{A,B} \leq j_A$ . ◁

As every core element of  $C$  is either the first or the last one in some pre-image under  $f_A$  or  $f_B$ , we get that  $\delta(C) \leq 2 \cdot (\delta(A) + \delta(B))$ . ◀

► **Example 3.8.** Note that the inequality for the core size from Theorem 1.1 is weaker than the inequality for the core sum from Lemma 3.5. We claim that this weakening is not an artifact of our proof. Consider the following Monge matrices

$$A = \begin{pmatrix} 0 & 4 & 5 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 0 \\ 0 & 1 & 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We have  $C = A \otimes B$ ,  $\delta(A) = 2$ ,  $\delta(B) = 3$ , and  $6 = \delta(C) > \delta(A) + \delta(B) > \min\{\delta(A), \delta(B)\}$ .

► **Remark 3.9.** To the best of our knowledge, Theorem 1.1 shows the first bound on the core size of  $A \otimes B$  in terms of the core sizes of  $A$  and  $B$ . Previously, such a bound was only known for unit-Monge matrices [41]. In particular, our bound allows simplifying some time complexities of already existing algorithms. For example, [34, Lemma 18] shows how to compute the product of two explicitly given Monge matrices  $A$  and  $B$  in time  $\mathcal{O}(p \log q + q \log r + \delta(A \otimes B) \log q \log r + (\delta(A) + \delta(B)) \log^2 q)$ . Assuming that  $r \in \text{poly}(q)$ , Theorem 1.1 tells us that we can drop the third summand in the time complexity.



## 4 Core-Sparse Monge Matrix Min-Plus Multiplication Algorithm

In this section, we present an algorithm that, given the condensed representations of two Monge matrices, computes the condensed representation of their  $(\min, +)$ -product. Even though the condensed representation itself does not provide efficient random access to the values of the matrix, the following fact justifies our choice of the representation of Monge matrices.

► **Fact 4.1** ([22, Lemma 4.4]). *There exists an algorithm that, given the condensed representation of a matrix  $A$  of size  $p \times q$ , in time  $\mathcal{O}(p + q + \delta(A) \log(1 + \delta(A)))$  builds a core-based matrix oracle data structure  $\text{CMO}(A)$  that provides  $\mathcal{O}(\log(2 + \delta(A)))$ -time random access to the entries of  $A$ .*

**Proof Sketch.** By Observation 3.1 applied for  $a = c = 0$ , it suffices to answer orthogonal range sum queries on top of  $\text{core}(A)$ . According to [45, Theorem 3], it can be done in  $\mathcal{O}(\delta(A) \log(1 + \delta(A)))$  preprocessing time and  $\mathcal{O}(\log(2 + \delta(A)))$  query time. ◀

► **Definition 4.2.** *Let  $A$  be a  $p \times q$  matrix. For every  $i \in [0..p-1]$ , denote  $\text{core}_{i,\cdot}(A) = \{(i, j, A_{i,j}^\square) \mid j \in [0..q-1], A_{i,j}^\square \neq 0\}$  and  $\delta_{i,\cdot}(A) = |\text{core}_{i,\cdot}(A)|$ . Analogously, for every  $j \in [0..q-1]$ , denote  $\text{core}_{\cdot,j}(A) = \{(i, j, A_{i,j}^\square) \mid i \in [0..p-1], A_{i,j}^\square \neq 0\}$  and  $\delta_{\cdot,j}(A) = |\text{core}_{\cdot,j}(A)|$ .*

► **Observation 4.3.**  $\delta(A) = \sum_{i \in [0..p-1]} \delta_{i,\cdot}(A) = \sum_{j \in [0..q-1]} \delta_{\cdot,j}(A)$ .

We now design another matrix oracle as an alternative to  $\text{CMO}(A)$  for the situation in which one wants to query an entry of  $A$  that is adjacent to some other already known entry.

► **Lemma 4.4.** *There is an algorithm that, given the condensed representation of a  $p \times q$  matrix  $A$ , in time  $\mathcal{O}(p + q + \delta(A))$  builds a local core oracle data structure  $\text{lco}(A)$  with the following interface.*

**Boundary access:** *given indices  $i \in [0..p)$  and  $j \in [0..q)$  such that  $i = 0$  or  $j = 0$ , in time  $\mathcal{O}(1)$  returns  $A_{i,j}$ .*

**Vertically adjacent recomputation:** *given indices  $i \in [0..p-1)$  and  $j \in [0..q)$ , in time  $\mathcal{O}(\delta_{i,\cdot}(A) + 1)$  returns  $A_{i+1,j} - A_{i,j}$ .*

**Horizontally adjacent recomputation:** *given indices  $i \in [0..p)$  and  $j \in [0..q-1)$ , in time  $\mathcal{O}(\delta_{\cdot,j}(A) + 1)$  returns  $A_{i,j+1} - A_{i,j}$ .*

**Proof.** The local core oracle data structure of  $A$  stores all the values of the topmost row and the leftmost column of  $A$ , as well as two collections of lists:  $\text{core}_{i,\cdot}(A)$  for all  $i \in [0..p-1)$  and  $\text{core}_{\cdot,j}(A)$  for all  $j \in [0..q-1)$ . The values of the topmost row and the leftmost column of  $A$  are already given. The lists  $\text{core}_{i,\cdot}(A)$  and  $\text{core}_{\cdot,j}(A)$  can be computed in  $\mathcal{O}(p + q + \delta(A))$  time from  $\text{core}(A)$ . Hence, we can build  $\text{lco}(A)$  in time  $\mathcal{O}(p + q + \delta(A))$ .

Boundary access can be implemented trivially. We now show how to implement the vertically adjacent recomputation. Suppose that we are given  $i \in [0..p-1)$  and  $j \in [0..q)$ . Due to Observation 3.1, we have  $A_{i,j} + A_{i+1,0} = A_{i+1,j} + A_{i,0} + \delta^\Sigma(A[i..i+1][0..j])$ . Note that the values  $A_{i,0}$  and  $A_{i+1,0}$  can be computed in constant time using boundary access queries, and  $\delta^\Sigma(A[i..i+1][0..j])$  can be computed from  $\text{core}_{i,\cdot}(A)$  in time  $\mathcal{O}(\delta_{i,\cdot}(A) + 1)$ . Hence,  $A_{i+1,j} - A_{i,j}$  can be computed in time  $\mathcal{O}(\delta_{i,\cdot}(A) + 1)$ .

The horizontally adjacent recomputation is implemented analogously so that  $A_{i,j+1} - A_{i,j}$  can be computed in time  $\mathcal{O}(\delta_{\cdot,j}(A) + 1)$ . ◀

Note that vertically and horizontally adjacent recomputations allow computing the neighbors of any given entry  $A_{i,j}$ .

► **Lemma 4.5.** *Given the condensed representation of a  $p \times q$  matrix  $A$ , the condensed representation of any contiguous submatrix  $A'$  of  $A$  can be computed in time  $\mathcal{O}(p + q + \delta(A))$ .*

**Proof.** Say,  $A' = A[i..i'][j..j']$ . Note that  $\text{core}(A')$  can be obtained in time  $\mathcal{O}(\delta(A))$  by filtering out all elements of  $\text{core}(A)$  that lie outside of  $[i..i'] \times [j..j']$ . It remains to obtain the top-most row and the leftmost column of  $A'$ . In time  $\mathcal{O}(p + q + \delta(A))$  we build  $\text{lco}(A)$  of Lemma 4.4. By starting from  $A_{i,0}$  and repeatedly applying the horizontally adjacent recomputation, we can compute  $A[i..i][0..q]$  (and thus  $A[i..i][j..j']$ ) in time  $\mathcal{O}(\sum_{j \in [0..q]} (\delta_{\cdot,j}(A) + 1)) = \mathcal{O}(q + \delta(A))$ . The leftmost column of  $A'$  can be computed in time  $\mathcal{O}(p + \delta(A))$  analogously. ◀

We now show a helper lemma that compresses “ultra-sparse” Monge matrices to limit their dimensions by the size of their core.

► **Lemma 4.6 (Matrix compression).** *There are two algorithms **Compress** and **Decompress** with the following properties: The **Compress** algorithm, given the condensed representations of a  $p^* \times q^*$  Monge matrix  $A^*$  and a  $q^* \times r^*$  Monge matrix  $B^*$ , in time  $\mathcal{O}(p^* + q^* + r^* + \delta(A^*) + \delta(B^*))$ , builds a  $p \times q$  Monge matrix  $A$  and a  $q \times r$  Monge matrix  $B$  such that  $p \leq \delta(A^*) + 1$ ,  $q \leq \delta(A^*) + \delta(B^*) + 1$ ,  $r \leq \delta(B^*) + 1$ ,  $\delta(A) = \delta(A^*)$ , and  $\delta(B) = \delta(B^*)$ . The **Decompress** algorithm, given the condensed representations of  $A^*$ ,  $B^*$ , and  $A \otimes B$ , where  $(A, B) = \text{Compress}(A^*, B^*)$ , computes the condensed representation of  $A^* \otimes B^*$  in time  $\mathcal{O}(p^* + r^* + \delta(A^*) + \delta(B^*))$ .*

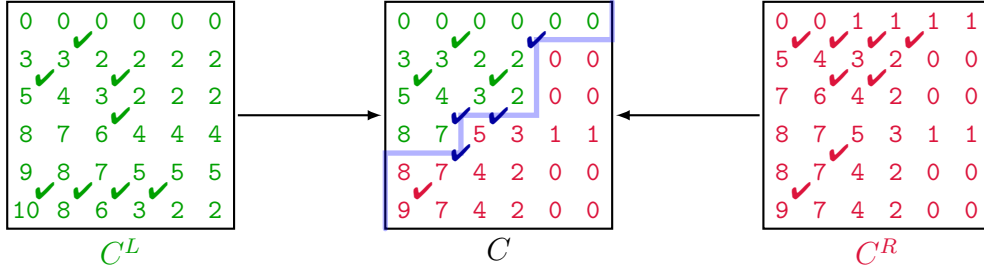
**Proof Sketch.** The **Compress** algorithm deletes all rows of  $A^*$  that do not contain any core elements and all columns of  $B^*$  that do not contain any core elements. This way,  $p \leq \delta(A^*) + 1$  and  $r \leq \delta(B^*) + 1$  are guaranteed. Furthermore, all core elements of these two matrices are preserved. If there are no core elements between rows  $i$  and  $i + 1$  of  $A^*$ , the corresponding entries of these two rows differ by some constant  $c$ , and thus all the entries of the row  $i + 1$  of  $A^* \otimes B^*$  can be obtained from the corresponding entries of row  $i$  of  $A^* \otimes B^*$  by adding  $c$ . Since we can recover  $c$  from  $A^*$ , no information about the answer is “lost” when deleting such rows. An analogous property holds for the removed columns of  $B^*$ . The **Decompress** algorithm reverses the row and column removals performed by the **Compress** algorithm.

The **Compress** algorithm also reduces the number  $q^*$  of columns of  $A^*$  and rows of  $B^*$  to  $q \leq \delta(A^*) + \delta(B^*) + 1$ . Observe that if there are no core elements between columns  $j$  and  $j + 1$  of  $A^*$  and between rows  $j$  and  $j + 1$  of  $B^*$ , then the values  $A_{i,j}^* + B_{j,k}^*$  and  $A_{i,j+1}^* + B_{j+1,k}^*$  differ by a constant independent of  $i$  and  $k$ . Depending on the sign of this constant difference, we can delete either the  $j$ -th column of  $A^*$  and the  $j$ -th row of  $B^*$  or the  $(j + 1)$ -th column of  $A^*$  and the  $(j + 1)$ -th row of  $B^*$  so that we do not change the min-plus product of these matrices. By repeating this process for all indices  $j$  that do not “contribute” to the cores of  $A^*$  and  $B^*$ , we get  $q \leq \delta(A^*) + \delta(B^*) + 1$ . The formal proof can be found in the full version of the paper [21]. ◀

Finally, we show our main Monge matrix multiplication algorithm.

► **Theorem 1.2.** *There is a (deterministic) algorithm that, given the condensed representations of an  $n_1 \times n_2$  Monge matrix  $M_1$  and an  $n_2 \times n_3$  Monge matrix  $M_2$ , in time  $\mathcal{O}(n_1 + n_2 + n_3 + (\delta(M_1) + \delta(M_2)) \log(1 + \delta(M_1) + \delta(M_2)))$  computes the condensed representation of  $M_1 \otimes M_2$ .*

**Proof Sketch.** We design a recursive divide-and-conquer procedure  $\text{Multiply}(A^*, B^*)$  (see Algorithm 1 for the details) and solve the problem by initially running  $\text{Multiply}(M_1, M_2)$ . Given the matrices  $A^*$  and  $B^*$ , we first apply the **Compress** algorithm of Lemma 4.6 to



■ **Figure 2** An example of how  $C$  is obtained from  $C^L$  and  $C^R$ . The blue ladder represents the border between the values that are inherited from  $C^L$  and the values that are inherited from  $C^R$ . Ticks correspond to the core elements: the green ticks are inherited from  $C^L$ , the red ticks are inherited from  $C^R$ , and the blue ticks represent the core element computed from scratch.

compress  $A^*$  and  $B^*$  into a  $p \times q$  matrix  $A$  and a  $q \times r$  matrix  $B$  respectively. We then compute the condensed representation of  $C := A \otimes B$  and finally use the **Decompress** algorithm of Lemma 4.6 to decompress  $C$  into  $A^* \otimes B^*$ .

If  $p = r = 1$ , we compute the matrix  $C$  trivially. Otherwise, we pick a splitting point  $m \in (0..q)$ , split the matrix  $A$  vertically into the matrix  $A^L$  of size  $p \times m$  and the matrix  $A^R$  of size  $p \times (q - m)$ , and split the matrix  $B$  horizontally into the matrix  $B^L$  of size  $m \times r$  and the matrix  $B^R$  of size  $(q - m) \times r$ . We pick  $m$  in such a way that it splits the cores of  $A$  and  $B$  almost equally across  $A^L$  and  $B^L$  and  $A^R$  and  $B^R$ , that is,  $\delta(A^L) + \delta(B^L)$  and  $\delta(A^R) + \delta(B^R)$  are at most  $(\delta(A) + \delta(B))/2$ . We recursively compute the condensed representations of the matrices  $C^L := A^L \otimes B^L$  and  $C^R := A^R \otimes B^R$ . The resulting matrix  $C$  can be obtained as the element-wise minimum of  $C^L$  and  $C^R$ . Furthermore, one can see that, due to Lemma 3.4, in some top-left region of  $C$ , the values are equal to the corresponding values of  $C^L$ , and in the remaining bottom-right region of  $C$ , the values are equal to the corresponding values of  $C^R$ ; see Figure 2. We find the boundary between these two regions by starting in the bottom-left corner of  $C$  and traversing it towards the top-right corner along this boundary. We use  $\text{lco}(C^L)$  and  $\text{lco}(C^R)$  to sequentially compute the subsequent entries along the boundary. Having determined the boundary, we construct the core of  $C$  by picking the core elements of  $C^L$  located to the top-left of this boundary, picking the core elements of  $C^R$  located to the bottom-right of this boundary, and computing the  $C^\square$  values on the boundary from scratch. The values in the topmost row and the leftmost column of  $C$  can be trivially obtained as element-wise minima of the corresponding values in  $C^L$  and  $C^R$ . This concludes the recursive procedure; see Algorithm 1 for the pseudocode. This algorithm follows the classical divide-and-conquer framework, and thus its time complexity can be easily derived from Theorem 1.1. The full description of the algorithm, the proof of its correctness, and the formal analysis of its time complexity can be found in the full version of the paper [21]. ◀

If the  $(\min, +)$ -product of matrices represents the lengths of the shortest paths in a graph, the corresponding witness matrix allows computing the underlying shortest paths themselves. For that, we extend the presented algorithm to allow witness reconstruction.

► **Theorem 4.7.** *The algorithm of Theorem 1.2 can be extended so that, within the same time complexity, it also builds a data structure that takes  $\mathcal{O}(n_1 + n_2 + n_3 + \delta(M_1) + \delta(M_2))$  space and provides  $\mathcal{O}(\log(2 + \delta(M_1) + \delta(M_2)))$ -time oracle access to  $\mathcal{W}^{M_1, M_2}$ .*

■ **Algorithm 1** The algorithm from Theorem 1.2. Given the condensed representations of the Monge matrices  $A^*$  and  $B^*$ , the algorithm returns the condensed representation of  $A^* \otimes B^*$ .

---

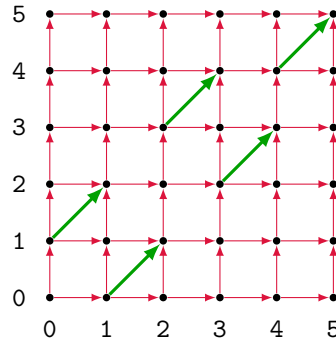
```

1 Multiply( $A^*, B^*$ ) begin
2    $A, B \leftarrow \text{Compress}(A^*, B^*)$ ;
3    $p \leftarrow \text{height of } A$ ;  $q \leftarrow \text{width of } A$ ;  $r \leftarrow \text{width of } B$ ;
4   if  $p = 1$  and  $r = 1$  then
5      $C \leftarrow 1 \times 1$  matrix with the entry equal to  $\min\{A_{0,j} + B_{j,0} \mid j \in [0..q]\}$ ;
6     return  $\text{Decompress}(A^*, B^*, C)$ ;
7   Sort  $\text{core}(A)$  by columns and  $\text{core}(B)$  by rows; // needed to find  $m$ 
8   Pick the largest  $m \in (0..q)$  with
9      $\delta(A[0..p][0..m]) + \delta(B[0..m][0..r]) \leq (\delta(A) + \delta(B))/2$ ;
10  Build the condensed representations of  $A[0..p][0..m]$  and  $B[0..m][0..r]$ ;
11   $C^L \leftarrow \text{Multiply}(A[0..p][0..m], B[0..m][0..r])$ ; Build  $\text{lco}(C^L)$  using Lemma 4.4;
12  Build the condensed representations of  $A[0..p][m..q]$  and  $B[m..q][0..r]$ ;
13   $C^R \leftarrow \text{Multiply}(A[0..p][m..q], B[m..q][0..r])$ ; Build  $\text{lco}(C^R)$  using Lemma 4.4;
14   $\text{lst} \leftarrow \text{list of length } p$ ; // vertical edges of the ladder from Figure 2
15   $i \leftarrow p - 1, j \leftarrow -1$ ; // current matrix coordinates while traversing the ladder
16  Maintain the well-defined values in  $C^L[i..i+1][j..j+1]$  and  $C^R[i..i+1][j..j+1]$ 
17  using  $\text{lco}(C^L)$  and  $\text{lco}(C^R)$ , and the values in  $C[i..i+1][j..j+1]$  as element-wise
18  minima;
19   $\text{Core} \leftarrow \{\}$ ;
20  while  $i \geq 0$  do
21    if  $i < p - 1$  and  $0 \leq j < r - 1$  and  $C_{i,j+1} + C_{i+1,j} \neq C_{i,j} + C_{i+1,j+1}$  then
22       $\text{Core} \leftarrow \text{Core} \cup \{(i, j, C_{i,j+1} + C_{i+1,j} - C_{i,j} - C_{i+1,j+1})\}$ ;
23    if  $j = r - 1$  or  $C_{i,j+1}^L > C_{i,j+1}^R$  then
24       $\text{lst}_i \leftarrow j$ ;
25       $i \leftarrow i - 1$ ;
26    else
27       $j \leftarrow j + 1$ ;
28   $\text{Core} \leftarrow \text{Core} \cup \{(i, j, C_{i,j}^{L\Box}) \in \text{core}(C^L) \mid j < \text{lst}_{i+1}\} \cup \{(i, j, C_{i,j}^{R\Box}) \in \text{core}(C^R) \mid$ 
29     $j > \text{lst}_i\}$ ;
30  Compute the topmost row and the leftmost column of  $C$  as the element-wise
31  minima of the topmost rows and the leftmost columns of  $C^L$  and  $C^R$ ;
32  Condensed representation of  $C \leftarrow (C[0..0][0..r], C[0..p][0..0], \text{Core})$ ;
33  return  $\text{Decompress}(A^*, B^*, C)$ ;

```

---

**Proof Sketch.** We slightly modify the algorithm of Theorem 1.2. For the leaf recursive calls with  $p = r = 1$ , we explicitly store the minimal witness of the only entry of  $C$ . In the non-leaf recursive calls, we store the correspondence between the indices in the compressed matrices  $A, B, C$  and the decompressed matrices  $A^*, B^*, C^*$ , as well as the border separating the values of  $C^L$  and  $C^R$  in  $C$  (the blue curve of Figure 2). Naively, this data takes space proportional to the time complexity of Theorem 1.2. Nevertheless, using bit-masks equipped with rank/select functionality, the total space complexity can be brought down to be linear in terms of the size of the input. To answer a query, we descend the recursion tree. In constant time we can find an entry of  $C$  corresponding to the queried entry of  $C^*$  and decide on which side of the border separating the values of  $C^L$  and  $C^R$  the queried entry of  $C$  is.



■ **Figure 3** The alignment graph for  $s = [1, 0, 3, 2, 4]$ , with red edges of weight 0 and green edges of weight 1. The length of the longest path from the  $i$ -th vertex in the bottom row to the  $j$ -th vertex in the top row of this graph for  $i < j$  is equal to  $\text{LIS}(s[i..j])$ .

After that, we recurse into either  $A^L \otimes B^L$  or  $B^L \otimes B^R$ . In the terminal recursion call with  $p = r = 1$  we return the minimal witness that is stored explicitly. The time complexity of the algorithm is proportional to the depth of recursion. See the full version of the paper [21] for a formal description of the algorithm. ◀

## 5 Range LIS Queries: Sketch

One of the original applications of Tiskin’s procedure for simple unit-Monge matrix multiplication [38, 41] is an algorithm that preprocesses a given sequence  $(s[i])_{i \in [0..n]}$  of integers in  $\mathcal{O}(n \log^2 n)$  time so that *Range Longest Increasing Subsequence* (Range LIS) queries can be answered in  $\mathcal{O}(\log n)$  time. We show an alternative way of obtaining the same result using Theorem 1.2 and avoiding the seaweed braid formalism. The extension of Theorem 4.7 allows recovering the underlying longest increasing subsequence of length  $\ell$  in  $\mathcal{O}(\ell \log^2 n)$  time. Further  $\mathcal{O}(n \log^3 n)$ -time preprocessing allows for  $\mathcal{O}(\ell)$ -time recovery, which improves upon the result of [27].

Without loss of generality we assume that  $(s[i])_{i \in [0..n]}$  is a permutation of  $[0..n]$ .<sup>4</sup> We use a popular tool for string similarity problems and interpret range LIS queries as computing the longest path between some pair of vertices of a corresponding *alignment graph*  $G^s$ ; see Figure 3. The crucial property of  $G^s$  is that it is planar, and thus due to [18, Section 2.3], the matrix  $M^s$  of longest distances from the vertices in the bottom row to the vertices of the top row of this graph is anti-Monge.<sup>5</sup> Thus, the problem of constructing a data structure for range LIS queries is reduced to the problem of computing the condensed representation of  $M^s$ .

As  $\text{LIS}(s[i..j]) \leq n$  for all  $i, j \in [0..n]$  with  $i < j$ , all entries of  $M^s$  are bounded by  $\mathcal{O}(n)$ , and thus  $\delta(M^s) = \mathcal{O}(n)$  holds due to Observation 3.1 and the fact that  $\delta(M^S) \leq \delta^\Sigma(M^S)$  holds for integer-valued matrices. We compute  $M^s$  in a divide-and-conquer fashion. We split the sequence  $s$  into subsequences  $s^{\text{lo}}$  and  $s^{\text{hi}}$  containing all values in  $[0.. \lfloor \frac{n}{2} \rfloor]$  and  $[\lfloor \frac{n}{2} \rfloor.. n]$  respectively, and recursively compute  $M^{s^{\text{lo}}}$  and  $M^{s^{\text{hi}}}$ . After that, we note that  $G^{s^{\text{lo}}}$  and  $G^{s^{\text{hi}}}$

<sup>4</sup> In what follows, we reserve the word “permutation” for permutations of  $[0..m]$  for some  $m \in \mathbb{Z}_+$ .

<sup>5</sup> Note that, in reality, some entries of this matrix are infinite. In the formal description of the algorithm, we augment the alignment graph so that the finite entries of this matrix are preserved and the infinite entries become finite.

are essentially compressed versions of the lower half and the upper half of  $G^s$ , respectively. Based on this, we transform  $M^{s^{lo}}$  in  $\mathcal{O}(n)$  time into the matrix of the longest distances from the vertices in the bottom row of  $G^s$  to the vertices in the middle row of  $G^s$ . Analogously, we transform  $M^{s^{hi}}$  into the matrix of the longest distances from the middle row of  $G^s$  to the top row of  $G^s$ . To obtain  $M^s$ , it remains to  $(\max, +)$ -multiply these two matrices using Theorem 1.2. Every single recursive call takes time  $\mathcal{O}(n \log n)$  dominated by the algorithm of Theorem 1.2. The whole divide-and-conquer procedure takes  $\mathcal{O}(n \log^2 n)$  time. Given the condensed representation of  $M^s$ , we use Fact 4.1 to create an oracle for  $\mathcal{O}(\log n)$ -time range LIS queries, thus replicating the result of [41].

Compared to the results of [41], this algorithm operates directly on the local LIS values and thus can be easily converted into an algorithm for the reporting version of range LIS queries, where we want to not only find the length of the longest path in the alignment graph but to also compute the structure of the underlying path itself. For that, we simply use the witness reconstruction oracle of Theorem 4.7 to find the midpoint of the path and descend the recursion tree. This costs  $\mathcal{O}(\log n)$  time per recursive call and allows reconstructing the entire length- $\ell$  LIS in time  $\mathcal{O}(\ell \log^2 n)$ . A similar recursive LIS reporting scheme is used in [9].

By treating increasing subsequences of length at most  $\log^2 n$  separately, we further obtain an algorithm with  $\mathcal{O}(n \log^3 n)$  preprocessing time and  $\mathcal{O}(\ell)$  reporting time. It improves the result of [27] with  $\mathcal{O}(n^{3/2} \text{polylog } n)$ -time preprocessing and  $\mathcal{O}(\ell + n^{1/2} \text{polylog } n)$ -time reporting.

The formal proofs of the results in this section are given in the full version of the paper [21].

---

## References

- 1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.
- 2 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997. doi:10.1006/JCSS.1997.1388.
- 3 Alberto Apostolico, Mikhail J. Atallah, Lawrence L. Larmore, and Scott McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, 19(5):968–988, 1990. doi:10.1137/0219066.
- 4 Gary Benson. A space efficient algorithm for finding the best nonoverlapping alignment score. *Theoretical Computer Science*, 145(1&2):357–369, 1995. doi:10.1016/0304-3975(95)92848-R.
- 5 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.
- 6 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 Journal on Computing*, 48(2):481–512, 2019. doi:10.1137/17M112720X.
- 7 Rainer E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, 2007. doi:10.1016/J.EJOR.2005.04.050.
- 8 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.
- 9 Nairen Cao, Shang-En Huang, and Hsin-Hao Su. Nearly optimal parallel algorithms for longest increasing subsequence. In Kunal Agrawal and Julian Shun, editors, *35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023*, pages 249–259. ACM, 2023. doi:10.1145/3558481.3591078.



- 10 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.
- 11 Panagiotis Charalampopoulos, Paweł Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *Journal of the ACM*, 70(2):12:1–12:50, 2023. doi:10.1145/3580474.
- 12 Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. An almost optimal edit distance oracle. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 48:1–48:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.48.
- 13 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Shay Mozes. Dynamic string alignment. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 9:1–9:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.9.
- 14 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance: A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 698–707. IEEE, 2022. doi:10.1109/FOCS54457.2022.00072.
- 15 Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 1435–1447. SIAM, 2022. doi:10.1137/1.9781611977073.60.
- 16 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.
- 17 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Information Processing Letters*, 181:106358, 2023. doi:10.1016/j.ipl.2023.106358.
- 18 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.
- 19 Arun Ganesh, Tomasz Kociumaka, Andrea Lincoln, and Barna Saha. How compression and approximation affect efficiency in string distance measures. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2867–2919. SIAM, 2022. doi:10.1137/1.9781611977073.112.
- 20 Paweł Gawrychowski. Faster algorithm for computing the edit distance between SLP-compressed strings. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *19th International Symposium on String Processing and Information Retrieval, SPIRE 2012*, volume 7608 of *LNCS*, pages 229–236. Springer, 2012. doi:10.1007/978-3-642-34109-0\_24.
- 21 Paweł Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka. Core-sparse Monge matrix multiplication: Improved algorithm and applications, 2024. arXiv:2408.04613v2.
- 22 Egor Gorbachev and Tomasz Kociumaka. Bounded edit distance: Optimal static and dynamic algorithms for small integer weights. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2157–2166, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3717823.3718168.
- 23 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.75.

- 24 Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013. doi:10.1007/S00453-011-9590-6.
- 25 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.
- 26 Sampath Kannan and Eugene W. Myers. An algorithm for locating nonoverlapping regions of maximum alignment score. *SIAM Journal on Computing*, 25(3):648–662, 1996. doi:10.1137/S0097539794262677.
- 27 Karthik C. S. and Saladi Rahul. Range longest increasing subsequence and its relatives: Beating quadratic barrier and approaching optimality, 2024. doi:10.48550/arXiv.2404.04795.
- 28 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 640–653. ACM, 2021. doi:10.1145/3406325.3451026.
- 29 Jaehyun Koo. On range LIS queries, 2023. Accessed: 2025-04-22. URL: <https://codeforces.com/blog/entry/111625>.
- 30 Jaehyun Koo. An optimal MPC algorithm for subunit-Monge matrix multiplication, with applications to LIS. In Kunal Agrawal and Erez Petrank, editors, *36th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2024*, pages 145–154. ACM, 2024. doi:10.1145/3626183.3659974.
- 31 Peter Krusche and Alexander Tiskin. New algorithms for efficient parallel string comparison. In Friedhelm Meyer auf der Heide and Cynthia A. Phillips, editors, *22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2010*, pages 209–216. ACM, 2010. doi:10.1145/1810479.1810521.
- 32 Gad M. Landau and Michal Ziv-Ukelson. On the common substring alignment problem. *Journal of Algorithms*, 41(2):338–359, 2001. doi:10.1006/JAGM.2001.1191.
- 33 Nikita Mishin, Daniil Berezun, and Alexander Tiskin. Efficient parallel algorithms for string comparison. In Xian-He Sun, Sameer Shende, Laxmikant V. Kalé, and Yong Chen, editors, *50th International Conference on Parallel Processing, ICPP 2021*, pages 50:1–50:10. ACM, 2021. doi:10.1145/3472456.3472489.
- 34 Luís M. S. Russo. Monge properties of sequence alignment. *Theoretical Computer Science*, 423:30–49, 2012. doi:10.1016/J.TCS.2011.12.068.
- 35 Yoshifumi Sakai. A substring-substring LCS data structure. *Theoretical Computer Science*, 753:16–34, 2019. doi:10.1016/J.TCS.2018.06.034.
- 36 Yoshifumi Sakai. A data structure for substring-substring LCS length queries. *Theoretical Computer Science*, 911:41–54, 2022. doi:10.1016/J.TCS.2022.02.004.
- 37 Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998. doi:10.1137/S0097539795288489.
- 38 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2007. arXiv:0707.3619.
- 39 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008. doi:10.1016/J.JDA.2008.07.001.
- 40 Alexander Tiskin. Threshold approximate matching in grammar-compressed strings. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference 2014*, pages 124–138. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2014. URL: <http://www.stringology.org/event/2014/p12.html>.
- 41 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. *Algorithmica*, 71(4):859–888, 2015. doi:10.1007/S00453-013-9830-Z.

- 42 Alexandre Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008. doi:10.1007/S11786-007-0033-3.
- 43 Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in  $n^{3-\Delta}$  time, with applications. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing, STOC 2006*, pages 225–231. ACM, 2006. doi:10.1145/1132516.1132550.
- 44 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians, ICM 2018*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880\_0188.
- 45 Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985. doi:10.1137/0214019.
- 46 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.
- 47 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In Shuchi Chawla, editor, *31st ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.
- 48 F. Frances Yao. Speed-up in dynamic programming. *SIAM Journal on Algebraic Discrete Methods*, 3(4):532–540, 1982. doi:10.1137/0603055.
- 49 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 950–957. SIAM, 2009. doi:10.1137/1.9781611973068.103.