Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths

Yuzhou Gu
MIT, Cambridge, MA, USA

Adam Polak

École Polytechnique Fédérale de Lausanne, Switzerland

Virginia Vassilevska Williams
MIT, Cambridge, MA, USA

Yinzhan Xu
MIT, Cambridge, MA, USA

Abstract

One of the most basic graph problems, All-Pairs Shortest Paths (APSP) is known to be solvable in $n^{3-o(1)}$ time, and it is widely open whether it has an $O(n^{3-\epsilon})$ time algorithm for $\epsilon > 0$. To better understand APSP, one often strives to obtain subcubic time algorithms for structured instances of APSP and problems equivalent to it, such as the Min-Plus matrix product.

A natural structured version of Min-Plus product is Monotone Min-Plus product which has been studied in the context of the Batch Range Mode [SODA'20] and Dynamic Range Mode [ICALP'20] problems. This paper improves the known algorithms for Monotone Min-Plus Product and for Batch and Dynamic Range Mode, and establishes a connection between Monotone Min-Plus Product and the Single Source Replacement Paths (SSRP) problem on an n-vertex graph with potentially negative edge weights in $\{-M, \ldots, M\}$.

SSRP with positive integer edge weights bounded by M can be solved in $\tilde{O}(Mn^{\omega})$ time, whereas the prior fastest algorithm for graphs with possibly negative weights [FOCS'12] runs in $O(M^{0.7519}n^{2.5286})$ time, the current best running time for directed APSP with small integer weights. Using Monotone Min-Plus Product, we obtain an improved $O(M^{0.8043}n^{2.4957})$ time SSRP algorithm, showing that SSRP with constant negative integer weights is likely easier than directed unweighted APSP, a problem that is believed to require $n^{2.5-o(1)}$ time.

Complementing our algorithm for SSRP, we give a reduction from the Bounded-Difference Min-Plus Product problem studied by Bringmann et al. [FOCS'16] to negative weight SSRP. This reduction shows that it might be difficult to obtain an $\tilde{O}(Mn^{\omega})$ time algorithm for SSRP with negative weight edges, thus separating the problem from SSRP with only positive weight edges.

2012 ACM Subject Classification Theory of computation → Shortest paths

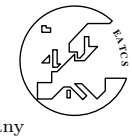
Keywords and phrases APSP, Min-Plus Product, Range Mode, Single-Source Replacement Paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.75

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.02806`

Funding *Adam Polak*: Supported by the Swiss National Science Foundation within the project *Lattice Algorithms and Integer Programming* (185030). Part of this work was done at Jagiellonian University, supported by Polish National Science Center grant 2017/27/N/ST6/01334.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.
Yinzhan Xu: Supported by NSF Grant CCF-1528078.



Acknowledgements This project was started as part of an open problem session held alongside MIT subject 6.890 taught by the third author. The authors would like to acknowledge the rest of the participants in the open problems session including but not limited to Angelos Pelecanos, Nicole Wein and Yuancheng Yu.

1 Introduction

The All-Pairs Shortest Paths problem (APSP) is one of the most well-known problems in graph algorithms. Following the classical Floyd–Warshall algorithm that solves APSP in $O(n^3)$ time in n-vertex edge-weighted graphs, a long list of papers have been dedicated to improving the APSP running time. The current best algorithm by Williams [26] runs in $n^3/2^{\Theta(\sqrt{\log n})}$ time. It is a big open problem whether a truly sub-cubic, $O(n^{3-\epsilon})$-time for $\epsilon > 0$, algorithm for APSP exists. In fact, the popular APSP hypothesis from Fine-Grained Complexity [23] asserts that this is not the case.

In the Min-Plus Product problem, one is given two $n \times n$ integer matrices A, B and is required to compute an $n \times n$ matrix $C = A \star B$ such that $C_{i,j} = \min_k\{A_{i,k} + B_{k,j}\}$. Fischer and Meyer [12] showed that Min-Plus Product is equivalent to APSP, in the sense that a $T(n)$ time algorithm for either of the problems immediately implies an $O(T(n))$ time algorithm for the other. The APSP hypothesis thus states that computing the min-plus product of $n \times n$ integer matrices requires $n^{3-o(1)}$ time[1].

Structured Min-Plus Products. To better understand the complexity of APSP, much research focuses on improving the running time for Min-Plus Product when one or both of the matrices have some structure, with the hope that eventually all instances can be handled. As a fundamental problem, Min-Plus Product can be used to solve many other problems. It turns out that in many cases a structured version of Min-Plus Product suffices [25, 4]. Thus, studying structured instances of Min-Plus Product has the potential to speed up the running times for many applications.

Alon, Galil and Margalit [3] first studied the Min-Plus Product of structured matrices. They showed, following ideas of Yuval [30], that if all entries of two $n \times n$ matrices A, B are integers in $\{-M, \ldots, M\} \cup \{\infty\}$, then one can compute the min-plus product of A and B in $\tilde{O}(Mn^\omega)$ time[2], where $\omega \in [2, 2.373)$ denotes the best possible exponent of square matrix multiplication [22, 17, 1].

Yuster [28] considered Min-Plus Product when one of the matrices has a small number of distinct entries in each row, generalizing [3]. Bringmann et al. [4] studied Min-Plus Product of bounded-difference matrices, generalizing [3, 28]. An integer matrix is called to have *bounded differences* if all pairs of adjacent entries (both horizontally and vertically) differ by at most $O(1)$. Bringmann et al. [4] gave an $O(n^{2.8244})$ time algorithm for computing the Min-Plus Product between two bounded-difference matrices. When $\omega = 2$, their algorithm runs in $O(n^{2.7554})$ time. They also studied variants of this problem including the case when only one matrix is guaranteed to have bounded differences, and the bounded-differences are only in the rows or only in the columns.

[1] In fine-grained complexity one needs to fix the model of computation for each hardness hypothesis, and the APSP hypothesis is typically stated for a word RAM with $O(\log n)$ bit words, which is the model the algorithms in our paper are in.

[2] Throughout the paper the $\tilde{O}$ notation hides subpolynomial factors.

Chan [5] gave a truly sub-cubic time algorithm for the min-plus product between *geometrically weighted* matrices using a geometric tool called the *partition theorem*. Recently, Vassilevska Williams and Xu [25] combined the approach of Bringmann et al. [4] and a geometric data structure to give a truly subcubic Min-Plus Product algorithm for integer matrices where one of the matrices has constant $O(1)$-approximate rank, further generalizing the results of [4] and partially [5].

Our contribution. In this work, we study the Min-Plus Product of a *monotone* integer matrix with an arbitrary[3] integer matrix. We defer the general definition of *Monotone Min-Plus Product* to Section 2. For now let us focus on an interesting special case: We are given an arbitrary $n \times n$ integer matrix A and an $n \times n$ matrix B whose entries are positive integers bounded by $O(n)$, and such that each row of B is non-decreasing.

The above special case already subsumes the Min-Plus Product of bounded-difference matrices studied by Bringmann et al.: Suppose we are asked to compute the min-plus product of matrices A and B where B has bounded differences. In other words, all pairs of adjacent entries (both horizontally and vertically) differ by at most some constant M. We can create a matrix B' so that $B'_{k,j} = B_{k,j} - B_{1,1} + jM$. It is easy to check that the matrix B' satisfies the above-mentioned special case definition of monotone matrix. Thus, we can use our algorithm to compute the min-plus product $C' = A \star B'$. Then it is easy to recover $C = A \star B$ by setting $C_{i,j} = C'_{i,j} + B_{1,1} - jM$. Therefore, Monotone Min-Plus Product is more general than the Min-Plus Product of an arbitrary matrix with a bounded-difference matrix.

Monotone Min-Plus Product was first studied by Vassilevska Williams and Xu [25] as a tool to give a fast algorithm for the *Batch Range Mode* problem. In their work, the authors devise a black-box reduction from Monotone Min-Plus Product to their Min-Plus Product algorithm for matrices with a small $O(1)$-approximate rank. Their algorithm runs in $\tilde{O}(n^{(15+\omega)/6}) = O(n^{2.8955})$ time for the above-mentioned special case. We improve and generalize their algorithm. Below is a special case of our main theorem, which will be introduced in Section 3.

▶ **Theorem 1.** *The min-plus product $A \star B$ of two $n \times n$ matrices where entries of B are non-negative integers bounded by $O(n)$ and each row of B is non-decreasing can be computed deterministically in $\tilde{O}(n^{\frac{12+\omega}{5}})$ time. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{2.8653})$.*

If $\omega = 2$, our improvement is from $\tilde{O}(n^{17/6}) \leq O(n^{2.8334})$ time to $\tilde{O}(n^{14/5}) = \tilde{O}(n^{2.8})$ time. We provide several interesting applications of our improved algorithm for Monotone Min-Plus Product.

1.1 Applications

Single Source Replacement Paths. The main contribution of this paper is establishing a relationship between Monotone Min-Plus Product and the Single-Source Replacement Paths (SSRP) problem. In the SSRP problem, one is given a directed edge-weighted graph G and a source vertex s, and is asked to compute for each edge e, $d_G(s, v, e)$'s, the shortest path distances from s to each vertex v in $G \setminus \{e\}$. Note that the interesting case is when e belongs to a shortest paths tree rooted at s, so that there are only $O(n^2)$ distances to report.

[3] Throughout the paper we assume the entries of the matrices are polylog(n)-bit integers or ∞ unless otherwise stated.

The trivial algorithm for SSRP runs in $\tilde{O}(n^3)$ time: For each edge e on the shortest path tree rooted at s, run Dijkstra's algorithm on the graph with e removed. Negative edge weights can be handled with Johnson's trick [16], without increasing the asymptotic complexity. Vassilevska Williams and Williams [24] showed that APSP and SSRP are sub-cubically equivalent. Hence, assuming the APSP hypothesis, there is no $O(n^{3-\epsilon})$ time algorithm for SSRP in graphs with arbitrary integer weights, for any $\epsilon > 0$. There seems to be little hope to improve upon the trivial algorithm for the general case.

Grandoni and Vassilevska Williams [14] studied SSRP in graphs with integer edge weights of small absolute value. They gave an algorithm that solves SSRP in directed n-vertex graphs with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time, which would become $\tilde{O}(M^{0.5} n^{2.5})$ if $\omega = 2$. For positive weights only, they reduce the runtime to $\tilde{O}(Mn^{\omega})$.

Let us consider the special case $M = 1$. Here, the algorithms of [14] solve SSRP with positive weights 1 in $\tilde{O}(n^{\omega})$ time, while the $\tilde{O}(n^{2+\frac{1}{4-\omega}})$ runtime for SSRP with weights in $\{-1, 0, 1\}$ is the same as the runtime for APSP with weights in $\{-1, 0, 1\}$.

As APSP in graphs with arbitrary integer weights is fine-grained equivalent to SSRP with arbitrary integer weights [24], it is possible that APSP with weights in $\{-1, 0, 1\}$ could be fine-grained equivalent to SSRP with weights in $\{-1, 0, 1\}$.

It is believed that APSP in directed graphs with weights in $\{-1, 0, 1\}$ (and even for unweighted graphs) requires $n^{2.5-o(1)}$ time [19, 27], as the best known algorithm by Zwick [31] would run in $\Omega(n^{2.5})$ time even if $\omega = 2$. As SSRP with small *positive* weights is in $O(n^{2.5-\varepsilon})$ time for $\varepsilon > 0$ [14], it is likely not fine-grained equivalent to directed unweighted APSP. Beating the APSP runtime for SSRP with *negative* weights is an open problem.

This leads to the following interesting questions.

(1) Is SSRP with negative weights inherently harder than SSRP with only positive weights?
(2) Or, is it possible to improve the running time of SSRP with negative weights, possibly below $n^{2.5}$, thus showing that it is likely not as hard as directed unweighted APSP?

Quite surprisingly, we give positive answers to both of these questions. First, we improve over the running time of [14] for negative weights.

▶ **Theorem 2.** *There is a randomized algorithm that solves SSRP in a directed n-vertex graph with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(M^{0.8043} n^{2.4957})$.*

Notably, when M is small enough, the running time $O(M^{0.8043} n^{2.4957})$ is polynomially faster than $n^{2.5}$, and hence faster than the best known running time of APSP in directed unweighted graphs which is $\Omega(n^{2.5})$ even if $\omega = 2$. This answers our question (2) above. If $\omega = 2$, our running time for SSRP with negative weights is $\tilde{O}(M^{5/9} n^{22/9}) \leq O(M^{0.556} n^{2.445})$.

APSP in directed graphs with edge weights in $\{-1, 0, 1\}$ is one of long list of so-called *intermediate* graph and matrix problems [19, 27], whose running time lies between $\tilde{O}(n^{\omega})$ and $\tilde{O}(n^3)$ and becomes $\tilde{O}(n^{2.5})$ when $\omega = 2$. Our result shows that SSRP with bounded negative integer weights is not an intermediate problem. We remark that recently Grandoni et al. [13] showed that another (ex-)candidate intermediate problem, All-Pairs LCA in DAGs, can actually be solved faster than $O(n^{2.5})$ time.

We prove Theorem 2 by improving the runtime of the so-called *subpath problem*, which is the bottleneck in the algorithm of [14]. Grandoni and Vassilevska Williams solve it by reducing to APSP in directed graphs with edge weights in $\{-M, \ldots, M\}$, and applying Zwick's APSP algorithm [31]. We show that the APSP computation can be rearranged so that certain min-plus products that appear throughout involve monotone matrices.

Next, we identify an obstacle to obtaining a $\tilde{O}(Mn^{\omega})$ time algorithm for SSRP with negative weights, addressing our question (1).

▶ **Theorem 3.** *If there exists a $T(n)$ time algorithm for SSRP in n-vertex graphs with edge weights in $\{-1, 0, 1\}$, then there exists an $O(T(n)\sqrt{n})$ time algorithm for the Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

Theorem 3 gives the following argument why SSRP with negative weights might be hard. The current best algorithm for Bounded-Difference Min-Plus Product runs in $O(n^{2.7554})$ time even if $\omega = 2$. If SSRP with weights $\{-1, 0, 1\}$ could be solved in $\tilde{O}(n^2)$ time (when $\omega = 2$), then Bounded-Difference Min-Plus Product could be solved in $\tilde{O}(n^{2.5})$ time, which would be a breakthrough in structured Min-Plus Product algorithms.

Recently replacement paths problems have received increased attention [8, 2, 9, 10]. None of these works is directly related to ours, because they focus either on the s-t Replacement Paths problem (with both source and target nodes fixed), or on combinatorial algorithms (i.e. without fast matrix multiplication) for sparse graphs.

Range Mode. Given an array a of elements, a range mode query asks for the most frequent element in a contiguous interval of a. In the Batch Range Mode problem the array a is fixed and all range mode queries are given in advance. In the Dynamic Range Mode problem one starts with an empty array and has to support insertions and deletions, and handle queries in an online fashion.

Vassilevska Williams and Xu [25] were the first to use structured Min-Plus Product in range mode algorithms. They reduced Batch Range Mode to a Min-Plus Product instance where both matrices have some monotone structures. Their techniques give an $O(n^{1.4854})$ time algorithm for Batch Range Mode on an array of size n and n queries. Since we improve over their Monotone Min-Plus Product algorithm, we naturally obtain a faster Batch Range Mode algorithm.

▶ **Theorem 4.** *The Batch Range Mode problem can be solved deterministically in time $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{1.4805})$.*

There are multiple algorithms that solve Dynamic Range Mode in $\tilde{O}(n^{2/3})$ time per update and query on an array of size bounded by n [6, 11]. Recently, Sandlund and Xu [21] improved both update and query time to $O(n^{0.6560})$ by using a so-called *Min-Plus-Query-Witness* problem. During the preprocessing phase of the Min-Plus-Query-Witness problem, one is given two matrices A, B. During the query phase, given two indices i, j and a set S, one is asked to compute $\arg\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$, where the set S can be viewed as the set of elements recently deleted in the array. In the Min-Plus-Query-Witness instances reduced from Dynamic Range Mode, the matrices A, B have the monotone property, so our techniques for Monotone Min-Plus Product can also apply to these Min-Plus-Query-Witness instances, leading to a faster Dynamic Range Mode algorithm.

▶ **Theorem 5.** *The Dynamic Range Mode problem can be solved deterministically in $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$ worst-case time per query with $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$ space. Using the current best bound on fast rectangular matrix multiplication improves the running time to $O(n^{0.6524})$ and the space complexity to $O(n^{1.3262})$.*

1.2 Overview of the Monotone Min-Plus Product Algorithm

Our improvement is achieved by extending Vassilevska Williams and Xu's [25] framework so that it can handle the more general monotone matrices. The algorithm has three phases. Say we would like to compute the min-plus product $C = A \star B$, where B is a monotone matrix.

In Phase 1, we compute a matrix $\tilde{C}$ which is close in ℓ_∞ norm to the desired output C. This can be done by, e.g., computing the min-plus product of $\lfloor \frac{A}{W} \rfloor$ and $\lfloor \frac{B}{W} \rfloor$, the downscaled versions of A and B, for some small parameter W.

In Phase 2, we repeatedly sample columns of B, and create new matrices A^r and B^r (for $r = 1, 2, \ldots$) whose entries are simple linear combinations of A, B and $\tilde{C}$. We replace large-magnitude entries of A with ∞, so that all finite entries of A^r are of small absolute values, and that the min-plus product $A^r \star B^r$ can be computed efficiently. The results are collected in a way such that by the end of Phase 2, we have found, for every pair (i, j),

$$\min_k \{A_{i,k} + B_{k,j} : A^r_{i,k} \neq \infty \text{ for some } r\}.$$

In Phase 3, we deal with (i, k) such that $A^r_{i,k} = \infty$ for all r. Such (i, k) are called *uncovered*. It can be shown that the number of *relevant* triples (i, k, j) with (i, k) uncovered is small, and we can afford enumerating all such triples. The enumeration is done by performing a witness-listing version of min-plus product of the downscaled versions of A and B.

We base on the fact that, when a monotone matrix B has very small entries, the number of *changes*, i.e. pairs (k, j) for which $B_{k,j} \neq B_{k,j+1}$, can be upper-bounded. Then for each fixed i we let j iterate through its range, and we maintain the set $\{A_{i,k} + B_{k,j}\}$ during the iteration. The total number of updates to the set is exactly the number of changes. This gives us an efficient way to compute min-plus product (and its witness-listing version) of the downscaled matrices, and leads to an efficient running time for Phases 1 and 3.

What makes our improvement possible is that we focus directly on the structure of monotone matrices, instead of going through a lossy black-box reduction to the Min-Plus Product of bounded-difference matrices, like the previous work [25] did.

2 Preliminaries

▶ **Definition 6** (Rectangular Matrix Multiplication Exponent). *Let α, β, γ be non-negative real numbers. Define $\omega(\alpha, \beta, \gamma)$ to be the smallest number such that the product of an $n^\alpha \times n^\beta$ matrix by an $n^\beta \times n^\gamma$ matrix can be computed in $\tilde{O}(n^{\omega(\alpha,\beta,\gamma)})$ time.*

▶ **Definition 7.** *Let α, β, γ, θ be non-negative real numbers. Define $g(\alpha, \beta, \gamma, \theta)$ to be the smallest number such that the min-plus product of an $n^\alpha \times n^\beta$ matrix whose entries are in $\{-n^\theta, \ldots, n^\theta\} \cup \{\infty\}$ by an arbitrary $n^\beta \times n^\gamma$ matrix can be computed in $\tilde{O}(n^{g(\alpha,\beta,\gamma,\theta)})$ time.*

▶ **Definition 8** (Bounded-Difference Matrix). *An $n \times m$ matrix A is called a bounded-difference matrix if $|A_{i,j} - A_{i,j+1}| \leq 1$ for every $1 \leq i \leq n, 1 \leq j < m$ and $|A_{i,j} - A_{i+1,j}| \leq 1$ for every $1 \leq i < n, 1 \leq j \leq m$.*

▶ **Problem 9** (Bounded-Difference Min-Plus Product). *Given two bounded-difference integer matrices A and B, compute $A \star B$.*

▶ **Definition 10** (Monotone Matrix). *An $n^\beta \times n^\gamma$ matrix B is called monotone if for every $k \in [n^\beta]$, $B_{k,j}$ is non-decreasing in $j \in [n^\gamma]$. For a monotone matrix B, we define its total range as $\sum_{k \in [n^\beta]} (\max_{j \in [n^\gamma], B_{k,j} \neq \infty} B_{k,j} - B_{k,1} + 1)$.*

▶ **Problem 11** (Monotone Min-Plus Product). *Given an $n^\alpha \times n^\beta$ matrix A and an $n^\beta \times n^\gamma$ matrix B where B is monotone and has total range $O(n^{\beta+\eta})$, where α, β, γ, η are non-negative real numbers, compute the min-plus product $A \star B$.*

▶ **Definition 12.** *Define $m(\alpha, \beta, \gamma, \eta)$ to be the smallest number such that Monotone Min-Plus Product with parameters α, β, γ, η, can be computed in $\tilde{O}(n^{m(\alpha,\beta,\gamma,\eta)})$ time.*

In our applications, we only need the case $\alpha = \gamma$. Because $m(c\alpha, c\beta, c\gamma, c\eta) = cm(\alpha, \beta, \gamma, \eta)$ for any $c \geq 0$, it suffices to consider only the case $\alpha = \gamma = 1$.

2.1 Upper bounds for g

We prove some useful upper bound for the function $g(\cdot, \cdot, \cdot, \cdot)$ introduced in Definition 7. We use the following bound from [7], which is a straightforward generalization of Theorem 1.2 from [25] to the case of rectangular matrices.

▶ **Lemma 13.** *For any non-negative real numbers $\alpha, \beta, \gamma, \theta$,*

$$g(\alpha, \beta, \gamma, \theta) \leq \min_{0 \leq \delta \leq \beta} \max\{\omega(\alpha, \beta, \beta + \gamma - \delta) + \theta, \delta + \alpha + \gamma\}. \tag{1}$$

We only need the following special cases of (1).

▶ **Corollary 14.** *For any non-negative real numbers β and θ,*

$$g(1, \beta, 1, \theta) \leq \frac{1}{2}(2 + \beta + \omega(1, \beta, 1) + \theta), \tag{2}$$

$$g(1, 1, 1, \theta) \leq \min_{0 \leq \delta \leq 1} \max\{\omega(1, 1, 2 - \delta) + \theta, 2 + \delta\}. \tag{3}$$

Proof. Consider the term $\omega(\alpha, \beta, \beta + \gamma - \theta)$ in (1). By splitting matrix B into $n^{\beta-\delta}$ matrices along its second dimension and computing $n^{\beta-\delta}$ independent instances of matrix multiplications, we get

$$\omega(\alpha, \beta, \beta + \gamma - \delta) \leq \omega(\alpha, \beta, \gamma) + \beta - \delta. \tag{4}$$

We plug (4) into (1), and take $\delta = \min\{\beta, \frac{1}{2}(\omega(\alpha, \beta, \gamma) + \beta + \theta - \alpha - \gamma)\}$, to get

$$g(\alpha, \beta, \gamma, \theta) \leq \max\{\omega(\alpha, \beta, \gamma) + \theta, \frac{1}{2}(\alpha + \beta + \gamma + \omega(\alpha, \beta, \gamma) + \theta)\}. \tag{5}$$

However, if $\omega(\alpha, \beta, \gamma) + \theta \geq \frac{1}{2}(\alpha + \beta + \gamma + \omega(\alpha, \beta, \gamma) + \theta)$, then both sides of (5) are at least $\alpha + \beta + \gamma$, and we can compute the min-plus product in $O(n^{\alpha+\beta+\gamma})$ time using a trivial algorithm. Therefore we get (2).

(3) is a simple substitution $\alpha = \beta = \gamma = 1$ to (1). ◀

▶ Remark 15. In the rectangular case $\alpha = \gamma = 1$, we use (4) so that in the final expression (2) we only need to deal with terms of the form $\omega(1, \beta, 1)$, whose value can be bounded by [18]. We know of no handy upper bounds for $\omega(\alpha, \beta, \gamma)$ when all three parameters are distinct.

In the square case $\alpha = \beta = \gamma = 1$, we do not need to use the simplification (4). This is because the upper bound of $\omega(1, \beta, 1)$ in [18] is by a bilinear algorithm. Thus by [15], the same upper bound works for $\omega(1, 1, \beta)$.

3 Monotone Min-Plus Product

▶ **Theorem 16.** *The min-plus product of an $n \times n^\beta$ matrix A and an $n^\beta \times n$ matrix B where B is monotone with total range $O(n^{\beta+\eta})$ can be deterministically computed, for any $\theta \in [0, \eta]$, in time $\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \frac{1}{2}(2+\beta+g(1,\beta,1,\theta))\}})$. In other words,*

$$m(1, \beta, 1, \eta) \leq \min_{0 \leq \theta \leq \eta} \max\{1 + \beta + \eta - \theta, \frac{1}{2}(2 + \beta + g(1, \beta, 1, \theta))\}.$$

Proof. Proof of Theorem 16 follows the three-phase framework of [4, 25], which we have briefly described in Section 1.2. Here we present the full algorithm.

Phase 1. Let $\theta \in [0, \eta]$ be a parameter, and let $W = \lfloor n^\theta \rfloor$. We define two matrices $\tilde{A}$ and $\tilde{B}$ as $\tilde{A}_{i,k} = \lfloor \frac{A_{i,k}}{W} \rfloor$ and $\tilde{B}_{k,j} = \lfloor \frac{B_{i,k}}{W} \rfloor$. We compute the min-plus product $\tilde{A} \star \tilde{B}$ and let $\tilde{C}_{i,j} = (\tilde{A} \star \tilde{B})_{i,j} W$. Then $||\tilde{C} - C||_\infty \leq 2W$.

We compute $\tilde{A} \star \tilde{B}$ using the following lemma, which is a simple algorithm that works fast when the total range is very small.

▶ **Lemma 17.** $m(\alpha, \beta, \gamma, \eta) \leq \max\{\alpha + \gamma, \beta + \gamma, \alpha + \beta + \eta\}$.

Proof. Say we would like to compute $C = A \star B$. For a fixed row $i \in [n^\alpha]$, we iterate through columns $j \in [n^\gamma]$, maintaining the multi-set $\{A_{i,k} + B_{k,j} : k \in [n^\beta]\}$.

Each time j increases, we need to update the multi-set for those k where $B_{k,j} \neq B_{k,j-1}$. The total number of (k, j) satisfying $B_{k,j} \neq B_{k,j-1}$ is $O(n^{\beta+\eta})$ by monotonicity and the bound on total range.

For each $i \in [n^\alpha]$, we need to make $O(n^{\beta+\eta})$ updates and $O(n^\gamma)$ queries for the minimum number in the multi-set. We can use a balanced BST to maintain the multi-set so that each update and query costs $\tilde{O}(1)$ time. The total running time is $\tilde{O}(n^{\max\{\alpha+\gamma, \beta+\gamma, \alpha+\beta+\eta\}})$. ◀

Total range of $\tilde{B}$ is $O(n^{\beta+\eta-\theta})$, so running time of Phase 1 is $\tilde{O}(n^{\max\{2, 1+\beta+\eta-\theta\}})$.

Phase 2. In Phase 2 we compute a matrix $\hat{C}$ which upper bounds $C = A \star B$ and agrees with it on most entries. Initially, let $\hat{C}_{i,j} \leftarrow \infty$ for all $i, j \in [n]$.

Phase 2 consists of $(10 + \beta)n^\rho \log n$ *rounds*, for a parameter $\rho \geq 0$ to be chosen later. In the r-th round, we choose $j^r \in [n]$ uniformly at random[4]. Define matrix A^r and B^r as

$$A^r_{i,k} = \begin{cases} A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r} & \text{if } A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r} \leq 3W \text{ and } A^{r'}_{i,k} = \infty \forall r' < r, \\ \infty & \text{otherwise,} \end{cases}$$

$$B^r_{k,j} = \begin{cases} B_{k,j} - B_{k,j^r} & \text{if } B_{k,j^r} \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

We compute $C^r = A^r \star B^r$ using Corollary 14 because A^r has bounded entries. Finally, for all $i, j \in [n]$, we make the update $\hat{C}_{i,j} \leftarrow \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r}\}$.

In other words, in the end we have $\hat{C}_{i,j} = \min_r\{C^r_{i,j} + \tilde{C}_{i,j^r}\}$. If $A^r_{i,k} \neq \infty$, then for all j, we have $\hat{C}_{i,j} \leq C^r_{i,j} + \tilde{C}_{i,j^r} \leq A^r_{i,k} + B^r_{i,k} + \tilde{C}_{i,j^r} = A_{i,k} + B_{k,j}$. Thus in this case we have effectively updated $\hat{C}_{i,j}$'s using $A_{i,k} + B_{k,j}$ for all j.

[4] For simplicity of presentation, we use randomness here. The derandomization is deferred to the full version of the paper.

Following [25], we make the following definitions: a triple $(i,k,j) \in [n] \times [n^\beta] \times [n]$ is *strongly relevant*, if $A_{i,k} + B_{k,j} = C_{i,j}$; *weakly relevant*, if $A_{i,k} + B_{k,j} - \tilde{C}_{i,j} \leq 3W$; *covered*, if $A^r_{i,k} \neq \infty$ for some r; *uncovered*, if it is not covered. We use the following lemma from [25].

▶ **Lemma 18** ([25, Lemma 4.3]). *With probability $1 - n^{-9}$, the number of triples that are weakly relevant and uncovered is at most $n^{2+\beta-\rho}$.*

Proof. Fix some pair of (i,k). If the number of j such that (i,k,j) is weakly relevant is at least $n^{1-\rho}$, then with probability at least $1 - (1-n^{-\rho})^{(10+\beta)n^\rho \log n} \geq 1 - n^{-10-\beta}$, we will sample a j^r such that (i,k,j^r) is weakly relevant. If so, $A^r_{i,k} \neq \infty$ for some r and thus (i,k,j) will be covered for all j. Therefore, with probability at least $1 - n^{-9}$, all (i,k) that are in at least $n^{1-\rho}$ weakly relevant triples will be covered. The number of remaining weakly relevant triples is at most $n^{2+\beta-\rho}$. ◀

Each round costs $\tilde{O}(n^{g(1,\beta,1,\theta)})$ time. Phase 2 costs $\tilde{O}(n^{\rho+g(1,\beta,1,\theta)})$ time in total.

Phase 3. We define a triple (i,k,j) to be *moderately relevant* if $\tilde{A}_{i,k} + \tilde{B}_{k,j} \leq (\tilde{A} \star \tilde{B})_{i,j} + 1$. In Phase 3, we enumerate over moderately relevant and uncovered triples to complete matrix C.

▶ **Lemma 19.** *Every strongly relevant triple is also moderately relevant.*

Proof. Suppose (i,k,j) is strongly relevant. If $\tilde{A}_{i,k'} + \tilde{B}_{k',j} = (\tilde{A} \star \tilde{B})_{i,j}$ for some k', then because $A_{i,k} + B_{k,j} \leq A_{i,k'} + B_{k',j}$, we have

$$\tilde{A}_{i,k} + \tilde{B}_{k,j} \leq \tilde{A}_{i,k'} + \tilde{B}_{k',j} + 1 = (\tilde{A} \star \tilde{B})_{i,j} + 1.$$

Hence (i,k,j) is moderately relevant. ◀

▶ **Lemma 20.** *Every moderately relevant triple is also weakly relevant.*

Proof. Suppose (i,k,j) is moderately relevant. Then

$$\begin{aligned} A_{i,k} + B_{k,j} - \tilde{C}_{i,j} &\leq (\tilde{A}_{i,k} + 1)W + (\tilde{B}_{k,j} + 1)W - (\tilde{A} \star \tilde{B})_{i,j}W \\ &\leq (\tilde{A}_{i,k} + \tilde{B}_{k,j} - (\tilde{A} \star \tilde{B})_{i,j})W + 2W \leq 3W. \end{aligned}$$

Hence (i,k,j) is weakly relevant. ◀

By Lemma 19, it suffices to enumerate over moderately relevant and uncovered triples to recover all of C. By Lemmas 18 and 20, the number of moderately relevant and uncovered triples is at most $O(n^{2+\beta-\rho})$, with high probability.

▶ **Lemma 21.** *With high probability, it takes time $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$ to enumerate all moderately relevant and uncovered triples.*

Proof. Define matrix $\check{A}$ as $\check{A}_{i,k} = \tilde{A}_{i,k}$ if (i,k) is uncovered; and $\check{A}_{i,k} = \infty$ otherwise.

We proceed on computing $\check{A} \star \tilde{B}$ in a way similar to Lemma 17 from Phase 1. For each row i, we maintain the set $\{(\check{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [n^\beta]\}$ as j iterates over $[n]$. Each time j increases, we need to update the multi-set for those k where $\tilde{B}_{k,j} \neq \tilde{B}_{k,j-1}$. The total number of (k,j) satisfying $\tilde{B}_{k,j} \neq \tilde{B}_{k,j-1}$ is $O(n^{\beta+\eta-\theta})$.

For each (i,j), we enumerate the elements in the multi-set in the increasing order, and stop as soon as we observe a k where $\check{A}_{i,k} + \tilde{B}_{k,j} > (\tilde{A} \star \tilde{B})_{i,j} + 1$. Therefore we enumerate exactly the moderately relevant uncovered triples. The running time is the running time from Lemma 17, plus the number of triples emitted, which, with high probability, is at most $O(n^{2+\beta-\rho})$, by Lemma 18. ◀

Phase 3 runs in time $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$.

Summary. Overall running time of our algorithm is $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,\rho+g(1,\beta,1,\theta),2+\beta-\rho\}})$. Note that $g(1,\beta,1,\theta) \leq 2+\beta$. So we can take $\rho = \frac{1}{2}(2+\beta-g(1,\beta,1,\theta))$. Also note that $\rho + g(1,\beta,1,\theta) \geq 2$, so the 2 in the max expression can be ignored. In the end we get $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\frac{1}{2}(2+\beta+g(1,\beta,1,\theta))\}})$ as claimed. ◀

As a benchmark, let us consider the case $\alpha = \beta = \gamma = \eta = 1$.

▶ **Theorem 1.** *The min-plus product $A \star B$ of two $n \times n$ matrices where entries of B are non-negative integers bounded by $O(n)$ and each row of B is non-decreasing can be computed deterministically in $\tilde{O}(n^{\frac{12+\omega}{5}})$ time. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(n^{2.8653})$.*

Proof. In this case, Theorem 16 simplifies (via (3)) to

$$\begin{aligned} m(1,1,1,1) &\leq \min_{0\leq\theta\leq 1} \max\{3-\theta, \frac{1}{2}(3+g(1,1,1,\theta))\} \\ &\leq \min_{0\leq\theta\leq 1} \max\{3-\theta, \frac{1}{2}(3+\min_{0\leq\delta\leq 1} \max\{\omega(1,1,2-\delta)+\theta, 2+\delta\})\}. \end{aligned} \tag{6}$$

Without using rectangular matrix multiplication, we can use $\omega(1,1,2-\delta) \leq 1-\delta+\omega$ and take $\theta = \frac{3-\omega}{5}$ and $\delta = \frac{2\omega-1}{5}$, so (6) takes value $\frac{12+\omega}{5}$.

Using the rectangular matrix multiplication upper bounds in [18] (see also Remark 15), we find that when $\theta = 0.1348$, $\delta = 0.7305$, expression (6) takes value ≤ 2.8653. ◀

4 Single Source Replacement Paths

We show our algorithm and lower bound for SSRP in this section. We use $d_G(u,v)$ to denote the length of a shortest path from u to v in a graph G, and we use $d_G(u,v,e)$ as a shorthand for $d_{G\setminus\{e\}}(u,v)$. When it is clear from the context, we sometimes omit G.

4.1 Algorithm

In this section we present our improved algorithm for SSRP, proving Theorem 2.

▶ **Theorem 2.** *There is a randomized algorithm that solves SSRP in a directed n-vertex graph with edge weights in $\{-M,\ldots,M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using the current best bound on fast rectangular matrix multiplication the running time improves to $O(M^{0.8043}n^{2.4957})$.*

To this end we improve the bottleneck in Grandoni-Vassilevska Williams algorithm [14], hence let us begin with a high level overview of that algorithm. This is however just to give a context and intuition, and our formal proof of Theorem 2 follows from a black-box[5] application of Lemmas 23 and 27.

[5] See end of the proof of Lemma 27 for a discussion why the $\tilde{O}(Mn^{\omega})$ component from Lemma 23 can be omitted.

Algorithm overview and the subpath problem. The algorithm first computes a shortest paths tree (from the source vertex s), and splits it into a subpolynomial number of subtrees. By using balanced separators, the subtrees can be roughly of the same size. Then, for each such subtree T, values $d_G(s, v, e)$ such that both vertex v and edge e belong to T are deferred to a recursive call on a graph obtained from G by carefully compressing its parts outside T. The only remaining interesting values $d_G(s, v, e)$ (i.e. such that they might be different from $d_G(s, v)$) are such that vertex v belongs to subtree T and edge e lies on the path from s to the root of T in the shortest paths tree. The problem of computing those remaining values is called *subpath problem*, which we now define formally.

▶ **Definition 22** (Subpath problem). *Given an n-vertex directed graph G with edge weights in $\{-M, \ldots, M\}$, a source vertex s, and a tree T which is a subtree of a shortest paths tree from s, compute $d_G(s, v, e)$ for every $v \in T$ and every e on the path from s to the root t of T in the shortest paths tree.*

Using the ideas outlined above, Grandoni and Vassilevska Williams formally reduce SSRP to the subpath problem.

▶ **Lemma 23** (Lemma 5.1 in [14]). *Given an algorithm that solves the subpath problem in time $\tilde{O}(M^\alpha n^\beta)$, with high probability, for constants $\beta \geq \alpha + 1 \geq 1$, there is an algorithm that solves SSRP in time $\tilde{O}(Mn^\omega + M^\alpha n^\beta)$, with high probability.*

Jumping paths and departing paths. We proceed to show how to solve the subpath problem. Let $P = (s = s_1 \to s_2 \to \cdots \to s_{|P|} = t)$ be the s-t path in the shortest paths tree. A replacement path witnessing $d_G(s, v, e)$ has to depart from P somewhere before e and then can either (1) join P back somewhere after e, and thus reach v through t, or (2) never use any other edge of P after departing. Paths of the first type are called *jumping paths*, and of the second type – *departing paths*. Grandoni and Vassilevska Williams [14] use the fact that jumping paths can be found by solving the *s-t replacement paths* problem, i.e. computing all $d_G(s, t, e)$'s for fixed t, which can be computed in $\tilde{O}(Mn^\omega)$ time (see Lemma 24). We just follow their approach in that regard.

▶ **Lemma 24** (Theorem 1.1 in [14]). *There is a randomized algorithm that solves s-t replacement paths problem in $\tilde{O}(Mn^\omega)$ time, with high probability.*

Improved algorithm for departing paths. Let $\widetilde{G}$ denote the graph obtained from G by removing all edges on path P. Note that the length of a shortest departing replacement path to v avoiding $e = (s_i, s_{i+1})$ equals to $\min_{j \leq i} d_G(s, s_j) + d_{\widetilde{G}}(s_j, v)$. Grandoni and Vassilevska Williams [14] simply feed $\widetilde{G}$ to Zwick's APSP algorithm [31], running in time $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$, to compute all $d_{\widetilde{G}}(s_j, v)$'s. We take a different approach and employ our truly subcubic algorithm for Monotone Min-Plus Product. We remark that any truly subcubic algorithm would yield an improvement.

Let $\zeta \in [0, 1]$ be a parameter to be determined later. We say that a departing replacement path is *hop-long* if it visits at least n^ζ nodes after departing P, otherwise it is *hop-short*. We handle the two types of paths separately.

Hop-short paths. To find hop-short paths we use a modification of Zwick's APSP algorithm [31], already described in [14]. Zwick's algorithm consists of $O(\log n)$ iterations, and in the i-th iteration it computes the shortest paths which use at most $(3/2)^i$ nodes. By running only first few iterations we can compute all hop-short shortest paths in time $\tilde{O}(Mn^{\zeta+\omega(1,1-\zeta,1)})$, which is faster than it would take to compute all shortest paths (given that ζ is small enough). For a formal proof of this statement we refer to [14].

▶ **Lemma 25** (Corollary 3.1 in [14]). *The distances between all pairs of nodes that have shortest paths on at most n^ζ nodes can be computed in time $\tilde{O}(Mn^{\zeta+\omega(1,1-\zeta,1)})$, with high probability.*

Hop-long paths. To find hop-long paths, first we sample (with replacement) $c \cdot n^{1-\zeta} \ln n$ nodes, for a large enough constant c. Let $B \subseteq V$ denote the set of sampled nodes. For the sake of analysis let us fix a set $\mathcal{S}$ of shortest hop-long departing replacement for all nodes $v \in T$ and all edges $e \in P$. When there is more than one such path of the smallest length for a given pair (v, e), we choose an arbitrary one. Note that for paths in $\mathcal{S}$, we only include the portions after they depart P so that they only contain edges in $\widetilde{G}$. Since the definition of hop-long paths only concerns the length of the part of a path after it departs P, all paths in $\mathcal{S}$ have length at least n^ζ. By a standard proof, with high probability, every path in $\mathcal{S}$ contains a node from B which lies in that path's middle third part.

Then we construct Yuster-Zwick distance oracle for graph $\widetilde{G}$ (see Lemma 26 below) and use it compute all $\tilde{O}(n \cdot n^{1-\zeta})$ shortest paths to and from B using at least $(1/3) \cdot n^\zeta$ nodes. In total it takes time $\tilde{O}(Mn^\omega + n^{3-2\zeta})$.

▶ **Lemma 26** (implicit in [29], Lemma 2.3 in [14]). *Given an n-vertex directed graph G, with edge weights in $\{-M, \ldots, M\}$, one can compute in $\tilde{O}(Mn^\omega)$ time an $n \times n$ matrix D, so that the (i, j)-th entry of the min-plus product $D \star D$ is the distance from node i to j in G. Furthermore, by the properties of D, the length of a shortest $i \rightsquigarrow j$ path containing at least n^ζ nodes can be computed in $\tilde{O}(n^{1-\zeta})$ extra time, with high probability.*

Let $d^{YZ}(\cdot, \cdot)$ denote such computed distances. The length of a shortest hop-long departing replacement path to v avoiding $e = (s_i, s_{i+1})$ equals $\min_{j \le i} \min_{b \in B} d_G(s, s_j) + d^{YZ}(s_j, b) + d^{YZ}(b, v)$.

We create two matrices A and B, of dimensions at most $n \times |B|$ and $|B| \times n$, respectively, such that

$$A_{i,b} = \min_{j \le i} d_G(s, s_j) + d^{YZ}(s_j, b), \quad \text{and} \quad B_{b,v} = d^{YZ}(b, v).$$

We need to compute $A \star B$. Note that $A_{i+1,b} \le A_{i,b}$, i.e. columns of A are monotone. Moreover, finite entries of A are of absolute value at most $2 \cdot nM$, so we can compute $A \star B = (B^T \star A^T)^T$ in time $\tilde{O}(n^{m(1,1-\zeta,1,1+\log_n M)})$.

Wrap-up and runtime analysis. Now we can sum up the running time and then balance the terms. The proof of the following lemma is deferred to the full version of the paper.

▶ **Lemma 27.** *There is a randomized algorithm that solves the subpath problem in a directed n-vertex graph with edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}})$ time, with high probability. Using fast rectangular matrix multiplication improves the running time to $O(M^{0.8043} n^{2.4957})$.*

4.2 Lower Bound

In this section, we prove our conditional lower bound for SSRP.

▶ **Theorem 3.** *If there exists a $T(n)$ time algorithm for SSRP in n-vertex graphs with edge weights in $\{-1, 0, 1\}$, then there exists an $O(T(n)\sqrt{n})$ time algorithm for the Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

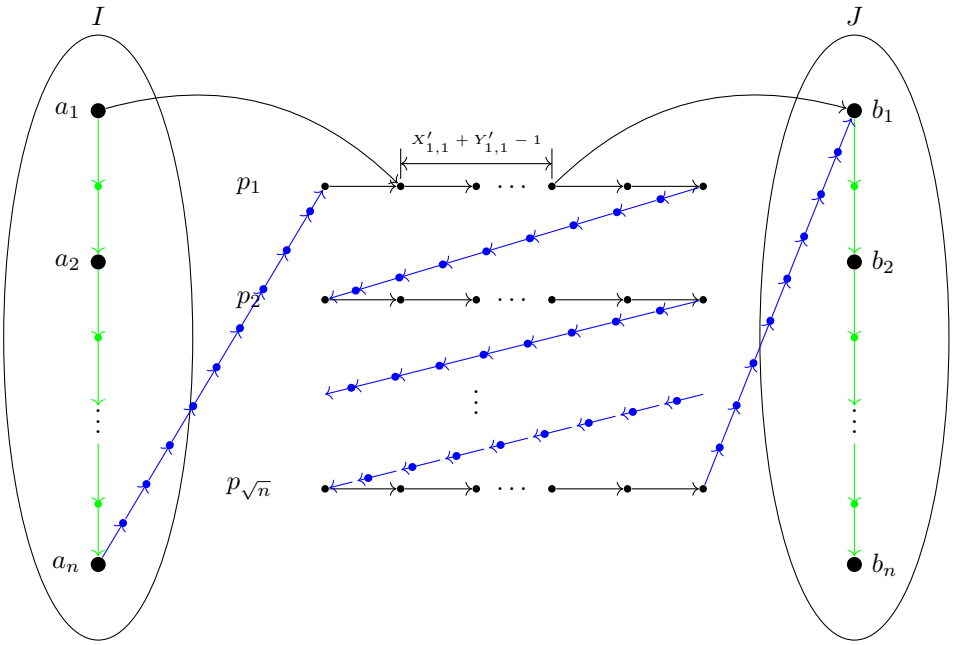


Figure 1 Reduction from Bounded-Difference Min-Plus Product to Ham-APSP (Lemma 29). Most edges between the parts I, J and the middle paths $p_1, \ldots, p_{\sqrt{n}}$ are omitted for clarity. The green portions are the first type paths, and the blue portions are the second type paths.

We first reduce Bounded-Difference Min-Plus Product to a problem called Ham-APSP, then we further reduce Ham-APSP to SSRP.

▶ **Problem 28** (Ham-APSP). *Given a directed unweighted graph G with vertex set $\{v_1, \ldots, v_n\}$ and a Hamiltonian path $v_1 \to \cdots \to v_n$ of G, compute all pairs shortest path distances in G.*

The key idea in the following reduction was used by Chan et al. [7] for a reduction from Min-Plus Product with small integer weights to unweighted directed APSP.

▶ **Lemma 29.** *If there exists a $T(n)$ time algorithm for Ham-APSP in a graph with n vertices, then there exists an $O(T(n)\sqrt{n})$ time algorithm for Bounded-Difference Min-Plus Product of $n \times n$ matrices.*

Proof. Given two $n \times n$ bounded-difference matrices A and B, we first split the columns of A and rows of B to $O(\sqrt{n})$ pieces. For each pair of pieces, we need to compute the min-plus product of an $n \times \sqrt{n}$ bounded-difference matrix X and a $\sqrt{n} \times n$ bounded-difference matrix Y. We will use a single call of the assumed $T(n)$ time algorithm for Ham-APSP to compute the min-plus product between each pair of pieces, yielding an $O(T(n)\sqrt{n})$ overall running time.

We create a new matrix X' such that $X'_{i,k} = X_{i,k} - X_{i,1}$. Since $|X_{i,k} - X_{i,k+1}| \leq 1$ for any i, k, all entries of X' are bounded by $\sqrt{n}$. We can create Y' similarly by setting $Y'_{k,j} = Y_{k,j} - Y_{1,j}$ so that all entries of Y' are bounded by $\sqrt{n}$ as well. We will later use the Ham-APSP algorithm to compute $X' \star Y'$, which immediately gives $X \star Y$ via the relation $(X \star Y)_{i,j} = (X' \star Y')_{i,j} + X_{i,1} + Y_{1,j}$.

In [7], Min-Plus Product of an $n \times \sqrt{n}$ and a $\sqrt{n} \times n$ matrices with weights up to $\sqrt{n}$ is reduced to unweighted directed APSP on n-node graphs. Here is a description of that reduction. We create a vertex set I of n vertices $\{a_1, \ldots, a_n\}$ and a vertex set J of n vertices $\{b_1, \ldots, b_n\}$. We also create $\sqrt{n}$ paths $p_1, \ldots, p_{\sqrt{n}}$ each of length $2\sqrt{n}$. From each a_i to p_k,

we add a directed edge from a_i to the $(\sqrt{n} - X'_{i,k})$-th node on p_k; similarly, from each p_k to b_j, we add a directed edge from the $(\sqrt{n} + Y'_{k,j})$-th node on p_k to b_j. Then we can see that the distance from a_i to b_j equals $(X' \star Y')_{i,j} + 2$.

In order to have a Hamiltonian path in the graph, we need to add two types of additional paths.

For the first type, we add paths of length 2 from a_i to a_{i+1} and from b_i to b_{i+1} for every $1 \le i < n$. Clearly, we only add $O(n)$ vertices and $O(n)$ edges. Now consider the shortest path from a_i to b_j for some i, j. The shortest path has the option to go to some $a_{i'}$ for $i' \ge i$, then choose some path p_k in the middle, then go to $b_{j'}$ for $j' \le j$, and finally reach b_j. The cost of this path would be $2(i' - i) + 1 + X'_{i',k} + Y'_{k,j'} + 1 + 2(j - j')$. Because X and Y have bound differences, we have that $2(i' - i) + X'_{i',k} \ge X'_{i,k}$ and $Y'_{k,j'} + 2(j - j') \ge Y'_{k,j}$. Therefore, in one of the shortest paths from a_i to b_j, we have $i' = i$ and $j' = j$. Thus, the distance from a_i to b_j is exactly $2 + (X' \star Y')_{i,j}$, so we can recover $X' \star Y'$ by computing all the pairwise distances.

For the second type of paths, we add $O(\sqrt{n})$ paths of lengths $3\sqrt{n}$ to connect I, J and each p_k, as shown in Figure 1. The total number of vertices and number of edges added are both $O(n)$. Since all distances we care about are at most $2\sqrt{n} + O(1)$, adding those paths won't affect these distances.

This graph now has a Hamiltonian path: we can travel from a_1 to a_n via the first type of paths. Then we use the second type of paths to travel from a_n to the beginning of p_1 and then we can easily travel to the end of p_1 by using edges of p_1. Similarly, we can go through all vertices in $p_2, \ldots, p_{\sqrt{n}}$. Finally, we travel from the end of $p_{\sqrt{n}}$ to b_1 via the second type of paths, and then use the first type of paths to travel to b_n. ◀

In the following lemma, we further reduce Ham-APSP to SSRP.

▶ **Lemma 30.** *If there exists a $T(n)$ time algorithm for SSRP in a graph with n vertices whose edge weights are in $\{-1, 0, 1\}$, then there exists an $O(T(n))$ time algorithm for Ham-APSP in a graph with n vertices.*

Proof. Let G be an instance of the Ham-APSP problem. We first create a graph G' whose vertex set is $\{v'_0, v'_1, \ldots, v'_n, v'_{n+1}\} \cup \{v_1, \ldots, v_n\}$. Then we add the following three types of edges to G, as depicted in Figure 2:

1. We add an edge from v'_i to v'_{i-1} of weight -1 for every $1 \le i \le n + 1$;
2. We add an edge from v'_i to v_i of weight 0 for every $1 \le i \le n$;
3. We add an edge from v_i to v_j of weight 1 for every $(v_i, v_j) \in E(G)$. This part essentially pastes a copy of G to G'.

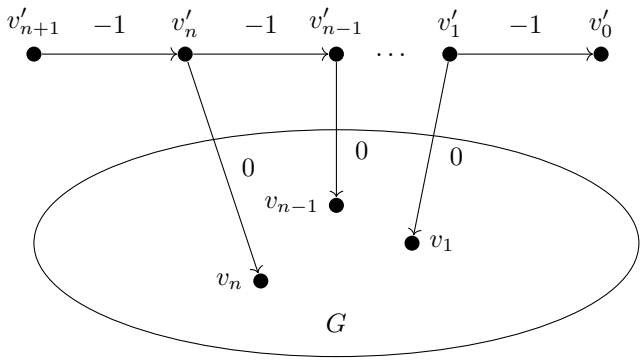


Figure 2 Reduction from Ham-APSP to SSRP with weights in $\{-1, 0, 1\}$ (Lemma 30).

If we cut the edge (v'_i, v'_{i-1}) in the graph G', then the shortest path from v'_{n+1} to v_j for some $1 \le j \le n$ must move from v'_{n+1} to v'_k for some $i \le k \le n$, then take the weight 0 edge to v_k, and finally move to v_j in the copy of graph G. Therefore, $d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) = \min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j)$. We show that $\min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j) = (i - n - 1) + d_G(v_i, v_j)$. Clearly, $\min_{i \le k \le n}(k - n - 1) + d_G(v_k, v_j) \le (i - n - 1) + d_G(v_i, v_j)$ since the right hand side is one of the terms we are minimizing over.

To show the other direction, we fix an arbitrary $k \in [i, n]$. By triangle inequality, $d_G(v_i, v_k) + d_G(v_k, v_j) \ge d_G(v_i, v_j)$. Since G has a Hamiltonian path $v_1 \to \cdots \to v_n$, it holds that $d_G(v_i, v_k) \le k - i$. Hence,

$$(k - n - 1) + d_G(v_k, v_j) \ge (k - n - 1) + d_G(v_i, v_j) - d_G(v_i, v_k) \ge (i - n - 1) + d_G(v_i, v_j).$$

We have shown that $d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) = (i - n - 1) + d_G(v_i, v_j)$. Thus, we can infer the pairwise distances in G by querying the assumed $T(n)$ time SSRP algorithm on graph G' since $d_G(v_i, v_j) = d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) - (i - n - 1)$. ◀

5 Range Mode

In this section, we show our improved algorithms for Batch Range Mode and Dynamic Range Mode.

5.1 Batch Range Mode

▶ **Theorem 4.** *The Batch Range Mode problem can be solved deterministically in time* $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$. *Using the current best bound on fast rectangular matrix multiplication the running time improves to* $O(n^{1.4805})$.

Proof. Via a binary search, Sandlund and Xu [21] showed that the Batch Range Mode problem can be reduced to finding the frequencies of the most frequent elements for all queries (with an $\tilde{O}(1)$ factor overhead), which is in turn reduced to Monotone Min-Plus Product in [21, Theorem 6.1], leading to a deterministic

$$\tilde{O}(n^{\min_{0 \le \tau \le 1} \max\{m(1-\tau, 1-\tau, 1-\tau, \tau), 1+\tau\}})$$

time algorithm for Batch Range Mode.

Expanding the expression using Theorem 16 and (3), we have

$$\begin{aligned}
&\min_{0 \le \tau \le 1} \max\{m(1-\tau, 1-\tau, 1-\tau, \tau), 1+\tau\} \\
&= \min_{0 \le \tau \le 1} \max\{(1-\tau)m(1, 1, 1, \frac{\tau}{1-\tau}), 1+\tau\} \\
&\le \min_{0 \le \tau \le 1} \max\{(1-\tau) \min_{0 \le \theta \le \frac{\tau}{1-\tau}} \max\{2 + \frac{\tau}{1-\tau} - \theta, \frac{1}{2}(3 + g(1, 1, 1, \theta))\}, 1+\tau\} \\
&= \min_{\substack{0 \le \tau \le 1 \\ 0 \le \theta \le \frac{\tau}{1-\tau}}} \max\{\tau + (1-\tau)(2-\theta), \frac{1-\tau}{2}(3 + g(1, 1, 1, \theta)), 1+\tau\} \\
&\le \min_{\substack{0 \le \tau \le 1 \\ 0 \le \theta \le \frac{\tau}{1-\tau} \\ 0 \le \delta \le 1}} \max\{\tau + (1-\tau)(2-\theta), \frac{1-\tau}{2}(3 + \max\{\omega(1, 1, 2-\delta) + \theta, 2 + \delta\}), 1+\tau\}.
\end{aligned} \tag{7}$$

By using $\omega(1, 1, 2-\delta) \le 1 - \delta + \omega$ and taking $\delta = \frac{4\omega-3}{9}, \theta = \frac{3-\omega}{9}, \tau = \frac{6+\omega}{15+\omega}$, we can upper bound (7) by $\frac{21+2\omega}{15+\omega}$. Using the upper bounds in [18], we find that when $\tau = 0.4804$, $\theta = 0.0754$, $\delta = 0.6984$, expression (7) takes value ≤ 1.4805. ◀

5.2 Dynamic Range Mode

▶ **Theorem 5.** *The Dynamic Range Mode problem can be solved deterministically in $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$ worst-case time per query with $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$ space. Using the current best bound on fast rectangular matrix multiplication improves the running time to $O(n^{0.6524})$ and the space complexity to $O(n^{1.3262})$.*

Our strategy is to improve [21, Lemma 11]. The following Min-Plus-Query-Witness problem defined in [21, Problem 7] plays a key role in the algorithm for Dynamic Range Mode.

▶ **Problem 31** (Min-Plus-Query-Witness problem)**.** *We are given two matrices A and B and are able to perform preprocessing before the first query. For each query, we are given two indices i, j and a set S of indices, and we must output an index $k^* \notin S$ such that*

$$A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}.$$

We recall the following lemma from [21].

▶ **Lemma 32** ([21, Lemma 9])**.** *Let β and θ be non-negative real numbers. The Min-Plus-Query-Witness problem where A is an $n \times n^\beta$ matrix whose entries are in $\{-n^\theta, \ldots, n^\theta\} \cup \{\infty\}$ and B is an $n^\beta \times n$ matrix can be solved with*

- *$\tilde{O}(n^{\theta+\omega(1,\beta,1)+\beta-\sigma})$ preprocessing time,*
- *$\tilde{O}(|S| + n^\sigma)$ worst-case time per query,*
- *and $\tilde{O}(n^{\max\{2+\beta+\theta, 1+2\beta\}-\sigma})$ space,*

for every $0 \le \sigma \le \beta$.

The following is our key lemma for the dynamic range mode algorithm.

▶ **Lemma 33.** *The Min-Plus-Query-Witness problem where A is an $n \times n^\beta$ matrix, B is an $n^\beta \times n$ monotone matrix with total range $O(n^{\beta+\eta})$, and the size of S for each query is $O(n^\lambda)$, can be solved with*

- *$\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \rho+\theta+\omega(1,\beta,1)+\beta-\sigma, 2+\beta-\rho\}})$ preprocessing time,*
- *$\tilde{O}(n^\lambda + n^{\rho+\sigma})$ worst-case time per query,*
- *and $\tilde{O}(n^{\max\{1+\beta+\eta-\theta, \rho+\max\{2+\beta+\theta, 1+2\beta\}-\sigma, 2+\beta-\rho\}})$ space,*

for any constants $0 \le \theta \le \eta$, $\rho \ge 0$ and $0 \le \sigma \le \beta$.

Proof. Preprocessing is in three steps, corresponding to the three phases of the algorithm for Theorem 16.

Preprocessing Step 1. This step is slightly more complicated than Phase 1 of Theorem 16. Let $0 \le \theta \le \eta$ be a parameter to be chosen. Let $W = \lfloor n^\theta \rfloor$. Define two matrices $\tilde{A}$ and $\tilde{B}$ as $\tilde{A}_{i,k} = \lfloor \frac{A_{i,k}}{W} \rfloor$ and $\tilde{B}_{k,j} = \lfloor \frac{B_{k,j}}{W} \rfloor$.

We iterate through $j \in [n]$, and in each iteration j, maintain for each $i \in [n]$ the set $L_{i,j} := \{(\tilde{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [n^\beta]\}$ using a *persistent* balanced BST.

Using the maintained information, we can compute a matrix $\tilde{C}'$, where each entry $\tilde{C}'_{i,j}$ is defined as W times the $(n^\lambda + 1)$-th smallest element in $\{\tilde{A}_{i,k} + \tilde{B}_{k,j} : k \in [n^\beta]\}$. Furthermore, for each (i, j) and for any t, we can enumerate the t smallest elements in $L_{i,j}$ in $\tilde{O}(t)$ time.

Time complexity and space complexity of this step are both $\tilde{O}(n^{\max\{2, 1+\beta+\eta-\theta\}})$.

Preprocessing Step 2. As in Phase 2 of Theorem 16, we sample $j^r \in [n]$ for $r \in [\tilde{O}(n^\rho)]$ for some parameter ρ to be chosen, and compute matrices A^r and B^r as

$$A^r_{i,k} = \begin{cases} A_{i,k} + B_{k,j^r} - \tilde{C}'_{i,j^r} & \text{if } |A_{i,k} + B_{k,j^r} - \tilde{C}'_{i,j^r}| \le 3W \text{ and } A^{r'}_{i,k} = \infty \forall r' < r, \\ \infty & \text{otherwise,} \end{cases}$$

$$B^r_{k,j} = \begin{cases} B_{k,j} - B_{k,j^r} & \text{if } B_{k,j^r} \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

Then for each r, we apply the data structure in Lemma 32 to A^r and B^r. Running time for this step is $\tilde{O}(n^{\rho+\theta+\omega(1,\beta,1)+\beta-\sigma})$. Space complexity for this step is $\tilde{O}(n^{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma})$.

We note that this part can be derandomized as well.

Preprocessing Step 3. For a pair (i,k) if $A^r_{i,k} \neq \infty$ for some r, we call (i,k) *covered*; otherwise we call it *uncovered*. We call a triple (i,k,j) *almost relevant* if $0 \le \tilde{A}_{i,k} + \tilde{B}_{k,j} - \frac{1}{W}\tilde{C}'_{i,j} \le 1$. Note that for such triples, we must have $|A_{i,k} + B_{k,j} - \tilde{C}'_{i,j}| \le 3W$. So the number of uncovered and almost relevant triples is $\tilde{O}(n^{2+\beta-\rho})$.

We run an algorithm similar to Lemma 21 to enumerate all uncovered and almost relevant triples (i,k,j). Then for each $i,j \in [n]$, we use a balanced BST to store the set $T_{i,j} := \{(A_{i,k} + B_{k,j}, k) : (i,k,j) \text{ is uncovered and almost relevant}\}$.

Running time for this step is $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,2+\beta-\rho\}})$. Space complexity for this step is $\tilde{O}(n^{2+\beta-\rho})$.

Query. Now let us describe how to handle a query. Let (S,i,j) be a query, and k^* be an optimal index, i.e., $A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}$. There are three cases.

Case 1: (i,k^*) is covered. For each r, we query the data structure (Lemma 32) for A^r and B^r with (S_r,i,j) where $S_r = S \cap \{k : A^r_{i,k} \neq \infty\}$. Because finite entries of A^r are disjoint for different r, we have $\sum_r |S_r| = |S|$. So the total query time for this case is $\tilde{O}(|S| + n^{\rho+\sigma})$.

Case 2: (i,k^*) is uncovered and almost relevant. In this case k^* must be among the $(|S|+1)$ smallest elements in $T_{i,j}$. We deal with this case by enumerating the $(|S|+1)$ smallest elements in $T_{i,j}$. The query time for this case is $\tilde{O}(|S|)$.

Case 3: (i,k^*) is uncovered and not almost relevant. Note that by definition of $\tilde{C}'_{i,j}$, in this case we must have $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \frac{1}{W}\tilde{C}'_{i,j} \le -1$. Therefore k^* must be among the $(n^\lambda + 1)$ smallest elements in $L_{i,j}$. We deal with this case by enumerating the $(n^\lambda+1)$ smallest elements in $L_{i,j}$. The query time for this case is $\tilde{O}(n^\lambda)$.

Summary. Total preprocessing time is $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}})$. Space complexity is $\tilde{O}(n^{\max\{1+\beta+\eta-\theta,\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$. Each query costs $\tilde{O}(n^\lambda + n^{\rho+\sigma})$ time. ◀

Proof of Theorem 5. The algorithm is exactly the same as in [21], except for replacing [21, Lemma 11] with Lemma 33. We skip most of the algorithm description and focus on analyzing the time and space complexity.

For the algorithm we need to choose three constants $t_1, t_2, t_3 \in [0,1]$. The algorithm has three parts.

Part 1: Infrequent values. In the first part, we handle values that appear at most n^{1-t_1} times. By maintaining n^{1-t_1} balanced BSTs, this part can be done in $\tilde{O}(n^{2-2t_1})$ time per operation. Space complexity is $\tilde{O}(n^{2-t_1})$.

Part 2: Newly modified values. We maintain a data structure holding frequent values that rebuilds every n^{t_2} operations. The data structure is discussed in Part 3. In Part 2, we deal with values that have been modified in the last n^{t_2} operations. This part can be done in $\tilde{O}(n^{t_2})$ time and $\tilde{O}(n^{t_2})$ space.

Part 3: Data structure. In Part 3, we build the data structure. For this part, we call Lemma 33 where A is an $n^{1-t_3} \times n^{t_1}$ matrix, B is an $n^{t_1} \times n^{1-t_3}$ monotone matrix with total range $O(n)$, and in the queries we have $|S| = O(n^{t_2})$. The dimensions $\alpha = \gamma = 1 - t_3$ come from choosing n^{1-t_3} evenly spaced points in $[n]$, and $\beta = t_1$ comes from the $O(n^{t_1})$ values which appear more than n^{1-t_1} times. In a query we pick the maximum interval whose endpoints are chosen points inside the query interval, and S is the set of values modified in the last n^{t_2} operations.

Rebuilding costs $\tilde{O}(n^{(1-t_3)\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}-t_2})$ time per operation[6], where $\beta = \frac{t_1}{1-t_3}$, $\eta = \frac{1-t_1}{1-t_3}$, and $0 \le \theta \le \eta$, $\rho \ge 0$, $0 \le \sigma \le \beta$ are constants to be chosen. One query in Lemma 33 costs $\tilde{O}(n^{t_2} + n^{(1-t_3)(\rho+\sigma)})$ time. Besides the above, we also need $\tilde{O}(n^{t_3})$ time per operation to deal with elements not covered by the maximum interval of chosen points inside the query interval.

Space cost of the data structure is $\tilde{O}(n^{(1-t_3)\max\{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$.

Summary. Running time per operation is

$$\tilde{O}(n^{2-2t_1} + n^{t_2} + n^{(1-t_3)\max\{1+\beta+\eta-\theta,\rho+\theta+\omega(1,\beta,1)+\beta-\sigma,2+\beta-\rho\}-t_2} + n^{(1-t_3)(\rho+\sigma)} + n^{t_3}),$$

subject to $t_1, t_2, t_3 \in [0,1]$, $\beta = \frac{t_1}{1-t_3}$, $\eta = \frac{1-t_1}{1-t_3}$, $0 \le \theta \le \eta$, $\rho \ge 0$, $0 \le \sigma \le \beta$.

Space cost is $\tilde{O}(n^{2-t_1} + n^{t_2} + n^{(1-t_3)\max\{\rho+\max\{2+\beta+\theta,1+2\beta\}-\sigma,2+\beta-\rho\}})$.

As an observation, in the optimum case we have $\beta \ge 1$, so we may use $\omega(1,\beta,1) \le \omega+\beta-1$. As a result, we can set $t_1 = \frac{\omega+21}{2\omega+30}, t_2 = t_3 = \frac{\omega+9}{\omega+15}, \theta = \frac{3-\omega}{6}, \rho = \frac{3-\omega}{4}$ and $\sigma = \frac{5\omega+9}{12}$ to upper bound the running time by $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$. The space complexity is dominated by Part 1, which is $\tilde{O}(n^{2-t_1}) = \tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$.

Taking $t_1 = 0.67385$, $t_2 = t_3 = 0.6523$, $\theta = 0.1239$, $\rho = 0.1859$, $\sigma = 1.6902$ and using fast rectangular matrix multiplication we get $O(n^{0.6524})$ running time per operation. The space complexity is again dominated by Part 1, which is $\tilde{O}(n^{2-t_1}) = O(n^{1.3262})$. ◀

References

1 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, 2021. doi:10.1137/1.9781611976465.32.

2 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

[6] The rebuilding is done every n^{t_2} operations, so a priori we get an amortized time complexity. However, as pointed out in [21], we can use the global rebuilding technique of [20] to achieve a worst-case time bound.

3 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.

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 Journal on Computing*, 48(2):481–512, 2019.

5 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

6 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.

7 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Algorithms, reductions and equivalences for small weight variants of all-pairs shortest paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, page to appear. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

8 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2090–2109. SIAM, 2019. doi:10.1137/1.9781611975482.126.

9 Shiri Chechik and Ofer Magen. Near optimal algorithm for the directed single source replacement paths problem. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 81:1–81:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.

10 Shiri Chechik and Moran Nechushtan. Simplifying and unifying replacement paths algorithms in weighted directed graphs. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.29.

11 Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund. Improved time and space bounds for dynamic range mode. In *Proceedings of the 26th Annual European Symposium on Algorithms*, pages 25:1–25:13, 2018.

12 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory, SWAT 1971*, pages 129–131. IEEE, 1971.

13 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Łukasiewicz, Nikos Parotsidis, and Przemysław Uznański. All-Pairs LCA in DAGs: Breaking through the $O(n^{2.5})$ barrier. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289, 2021. doi:10.1137/1.9781611976465.18.

14 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms*, 16(1):1–25, 2019.

15 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of complexity*, 14(2):257–299, 1998.

16 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM. doi:10.1145/2608628.2608664.

18 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.

19 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.

20 Mark H. Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1987.

21 Bryce Sandlund and Yinzhan Xu. Faster Dynamic Range Mode. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 94:1–94:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.94.

22 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, 2012. doi:10.1145/2213977.2214056.

23 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

24 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):1–38, 2018.

25 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 12–29. SIAM, 2020.

26 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 664–673. ACM, 2014.

27 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and apsp. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797, 2020. doi:10.1109/FOCS46700.2020.00078.

28 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. 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 950–957. SIAM, 2009.

29 Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 389–396, USA, 2005. IEEE Computer Society. doi:10.1109/SFCS.2005.20.

30 Gideon Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4(6):155–156, 1976. doi:10.1016/0020-0190(76)90085-5.

31 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.