Subcubic Min-Plus Product of Structured Matrices

by

Yinzhan Xu

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 18, 2021

Certified by . . . . . .
Virginia Vassilevska Williams
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Subcubic Min-Plus Product of Structured Matrices

by

Yinzhan Xu

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The All-Pairs Shortest Paths (APSP) problem is one of the most basic problems in computer science. The fastest known algorithms for APSP in n-node graphs run in $n^{3-o(1)}$ time, and it is a big open problem whether a truly subcubic, $O(n^{3-\varepsilon})$ for $\varepsilon > 0$ time algorithm exists for APSP. The Min-Plus product of two $n \times n$ matrices is known to be equivalent to APSP, where the optimal running times of the two problems differ by at most a constant factor. A natural way to approach understanding the complexity of APSP is thus understanding what structure (if any) is needed to solve Min-Plus Product in truly subcubic time. The goal of this thesis is to get truly subcubic algorithms for Min-Plus products for less structured inputs than what was previously known, and to apply them to versions of APSP and other problems.

This thesis gives sub-cubic algorithms for two interesting cases of structured Min-Plus Products: Min-Plus product between matrices with a constant additive approximate rank and Min-Plus product between monotone matrices, whose definitions are deferred to the main text. These faster algorithms have a wide range of applications, including Geometric APSP, Maximum Subarray, Range Mode and Single Source Replacement Paths.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis is based a joint work with Virginia Vassilevska Williams [50], a joint work with Bryce Sandlund [43], and another joint work with Yuzhou Gu, Adam Polak and Virginia Vassilevska Williams [57]. I would like to thank all my coauthors for discussing and solving problems with me. In particular, I would like to thank my advisor Virginia Vassilevska Williams for her valuable guidance and heartening encouragements. I also want to thank my parents for their unconditional caring and love throughout my life.

Contents

List of Figures

Chapter 1

Introduction

The All-Pairs Shortest Paths (APSP) problem is one of the most fundamental problems in computer science. In the APSP problem, given an n-node directed or undirected graph G, with edge weights in $\{-n^c, \ldots, n^c\}$ for some constant c, one is asked to compute the shortest path distances between all pairs of vertices in G. The Floyd-Warshall algorithm devised in the 1950s is the first algorithm that achieves $O(n^3)$ time for APSP in the word-RAM model with $O(\log n)$-bit words. Despite a lot of efforts made by the computer science community, APSP still doesn't have a truly subcubic, i.e. $O(n^{3-\epsilon})$ for $\epsilon > 0$, time algorithm. The current best APSP algorithm by Williams [52] runs in $n^3/2^{\Theta(\sqrt{\log n})}$ time. The popular APSP hypothesis of fine-grained complexity theory, states that APSP has no truly subcubic time algorithm in the word-RAM model with $O(\log n)$-bit words.

In the Min-Plus Product problem, given two $n \times n$ matrices A, B whose entries are all integers in $\{-n^c, \ldots, n^c\}$ for some constant c, one is asked to compute an $n \times n$ matrix $C = A \star B$ where $C_{i,j} = \min_k A_{i,k} + B_{k,j}$. It is known that APSP is equivalent to Min-Plus Product up to constant factors [23]. This means that if Min-Plus Product has a $T(n)$ time algorithm, then APSP also has an $O(T(n))$ time algorithm, and vice versa. Therefore, under the APSP hypothesis, Min-Plus Product requires $n^{3-o(1)}$ time.

There are two main reasons to study structured instances of Min-Plus Product. First, the techniques developed by studying structured Min-Plus Product could be useful for truly subcubic APSP. Second, since Min-Plus Product is a fundamental problem, a large number of other problems can be reduced to it. These problems are often reduced to structured

instances of Min-Plus Product, so by studying faster algorithms for structured Min-Plus Product, we could obtain faster algorithms for these problems as well.

Even though APSP and Min-Plus Product are equivalent, Min-Plus Product is structurally simpler than APSP. One evidence for this is that an $O(n^3)$ time algorithm for Min-Plus Product is straightforward while any $O(n^3)$ time algorithm for APSP requires some thinking. Also, structure on the input is better preserved in Min-Plus Product output. For instance, when the input matrices for Min-Plus Product have entries in $\{1, \ldots, M\}$, the output matrix has entries bounded by $2M$; however, when the input graph has weights in $\{1, \ldots, M\}$ in APSP, the distances can still be as large as $\Theta(nM)$. Therefore, it is more natural to focus on the structured version of Min-Plus Product.

1.1 Related Works

There has been a number of works on structured Min-Plus Product.

In the 1990s, Alon, Galil and Margalit [6] showed how to compute the Min-Plus product of two matrices whose entries are in $\{-M, \ldots, M\} \cup \{\infty\}$ in $\tilde{O}(Mn^\omega)$ time[1], where $\omega < 2.373$ denotes the matrix multiplication exponent [47, 28, 4]. When $M = O(n^{1-\epsilon})$ for some $\epsilon > 0$, their algorithm is truly subcubic. This algorithm has been used in a wide range of applications, including APSP in undirected graphs with small integer weights [44], APSP in directed unweighted graphs [58], and distance oracles on graphs with small integer weights [56]. The key idea in the algorithm of [6] is to reduce this structured Min-Plus Product problem to integer matrix multiplication where the entries can be as large as to $n^{O(M)}$. Since it takes $O(n^\omega)$ arithmetic operations to compute integer matrix multiplication, and each arithmetic operation takes $\tilde{O}(M)$ time for integers bounded by $n^{O(M)}$, the running time of their algorithm is $\tilde{O}(Mn^\omega)$.

Yuster [55] considered another type of structure: when each row of matrix A has L distinct values, Yuster's algorithm can compute $C = A \star B$ in time $\tilde{O}(Ln^{(3+\omega)/2})$. Note that as long as $L = O(n^{(3-\omega)/2-\epsilon})$ for $\epsilon > 0$, this algorithm is subcubic. This result can be used to compute, for instance, APSP where the weights of the incident edges of each vertex

[1]Throughout the thesis the $\tilde{O}$ notation hides subpolynomial factors.

only have L distinct values. To obtain this algorithm, we first consider a special case where all entries of A are either 0 or ∞. In this case, we can compute the Min-Plus product of A and B in $\tilde{O}(n^{(3+\omega)/2})$ time by grouping elements in each column of B with respect to their ranks, finding a group as candidate values for each $C_{i,j}$ via fast matrix multiplication, and finally enumerating all elements in each candidate group to determine the exact values for each $C_{i,j}$. To solve the original problem where each row of A has at most L distinct values, each time we pick one value from each row and use the previous special case. Overall, we call the special case L times, and thus the algorithm runs in $\tilde{O}(Ln^{(3+\omega)/2})$ time.

Chan [13] gave a truly subcubic time algorithm for the Min-Plus product of *geometrically weighted* matrices. Let $\mathcal{W}$ be a set of constant-degree, constant-piece piecewise-algebraic functions over $\mathbb{R}^d \times \mathbb{R}^d$ for some constant d. A matrix A is called a *geometrically weighted* matrix if there are points $p_1, \ldots, p_n$ such that $A_{i,j} \in \{w(p_i, p_j) : w \in \mathcal{W}\} \cup \{\infty\}$. He showed that we can compute the Min-Plus product of a geometrically weighted matrix A and an arbitrary matrix B in $\tilde{O}(|\mathcal{W}|n^{3-(3-\omega)/(\gamma+1)})$ time, where κ is a constant depending on the elements of $\mathcal{W}$. The main technical tool in this result is a well-known computational geometry result called the *partition theorem*. This work generalizes Alon et al.'s result (but with a worse running time), since when the input matrices have entries in $\{-M, \ldots, M\} \cup \{\infty\}$, we can associate one constant function $f(\cdot, \cdot) = i$ with $\mathcal{W}$ for each $i \in \{-M, \ldots, M\}$. Furthermore, Chan's work also generalizes Yuster's algorithm, since in the special case when all entries of A are either 0 or ∞, we can just take $\mathcal{W}$ to contain the zero function. Chan used this algorithm for several applications, including real-vertex-weighted APSP.

Bringmann, Grandoni, Saha and Vassilevska Williams [11] designed a Min-Plus Product algorithm for a quite general class of matrices called bounded differences matrices. A matrix X with integer entries is an M-bounded differences matrix if for every row i and column j, we have that $|X_{i,j} - X_{i+1,j}| \leq M$ and $|X_{i,j} - X_{i,j+1}| \leq M$. When $M = O(1)$, their algorithm runs in $O(n^{2.7554})$ randomized time and $O(n^{2.8603})$ deterministic time. Moreover, as long as $M = O(n^{3-\omega-\epsilon})$, their algorithm runs in truly subcubic time. Note that if all entries of a matrix are in the range $\{-M, \ldots, M\}$, the matrix is a $(2M)$-bounded differences matrix. Thus, Bringmann et al.'s algorithm is more general than Alon et al.'s algorithm,

in the sense that for any pair of input matrices whose Min-Plus product can be computed by Alon et al.'s algorithm in truly subcubic time, it can also be computed by Bringmann et al.'s algorithm in truly subcubic time (a small exception is when the input matrices contain ∞ entries since Bringmann et al. did not explicitly handle ∞ entries).

Their algorithm uses a three-phase approach. In the first phase, they compute an additive approximation $\tilde{C}$ of the target matrix C. In the second phase, they sample rows of A and columns of B. After that, they create matrices $\tilde{A}, \tilde{B}$ that are clever linear combinations of $A, B, \tilde{C}$ and the sampled rows and columns so C can be easily computed from $\tilde{A} \star \tilde{B}$ in $O(n^2)$ time. Then they set entries of $\tilde{A}, \tilde{B}$ that are too large to ∞ and use Alon et al.'s algorithm [6] to compute $\tilde{A} \star \tilde{B}$. Since they set some entries of $\tilde{A}$ and $\tilde{B}$ to ∞, they can no longer fully recover C. However, they show that in order to recover C, it remains to enumerate a small number of triples (i, k, j) and use $A_{i,k} + B_{k,j}$ to update $C_{i,j}$. In the third phase, they find such triples (i, k, j) efficiently.

Bringmann et al. also considered some more general versions of bounded differences matrices, including the case when only one of the two input matrices have bounded-differences, and the case when only the condition $|X_{i,j} - X_{i+1,j}| \leq M$ is needed. Their algorithm has multiple applications, including Language Edit Distance, RNA-folding and Optimum Stack Generation.

1.2 Our Results

Our main result is an algorithm generalizing Bringmann et al.'s algorithm [11]. We first define the notion of W-approximate rank before we introduce the structure of input matrices our algorithm needs.

Definition 1.2.1 (W-approximate rank)**.** *For an n by n integer matrix M, its W-approximate rank is defined as*

$$\min \left\{ \textit{rank}(X) : X \in \mathbb{Z}^{n \times n}, |X - M|_\infty \leq W \right\}.$$

Informally, X has small W-approximate rank if it can be approximated by a low-rank

matrix.

Let $\delta > 0$ be a constant and let $W \geq 0$ be an integer. Consider an $n \times n$ integer matrix B with the following structure. First partition B into blocks of dimension $n^\delta \times n^\delta$. We use $B^{a,b}$ to denote the block containing the entries $B_{i,j}$ where $i \in (an^\delta, (a+1)n^\delta], j \in (bn^\delta, (b+1)n^\delta]$. We require that every block $B^{a,b}$ has W-approximate rank at most $O(1)$. Roughly speaking, each block of B can be approximated by a constant-rank matrix.

The main result is described below.

Theorem 1.2.1. *Let $\delta \in (0, 1]$. Let A and B be two $n \times n$ matrices whose entries are* polylog n *bit integers, and B has all its $n^\delta \times n^\delta$ blocks of W-approximate rank at most d for $1 \leq d = O(1)$. If we are given the rank decompositions of the low rank matrices that approximate the blocks of B, then we can compute the Min-Plus product of A and B in time*

$$\tilde{O}\left(n^{3-\frac{\delta}{\lfloor (d+1)/2 \rfloor}} + W^{1/4} n^{(9+\omega)/4}\right).$$

This algorithm generalizes Bringmann et al.'s algorithm. In the most general case of Bringmann et al., the input matrix A can be arbitrary integer matrix, and the matrix B has to satisfy $|B_{i,j} - B_{i,j+1}| \leq M$ for some small M. Their algorithm is subcubic as long as $M = O(n^{3-\omega-\epsilon})$ for $\epsilon > 0$. We notice that for each $n^\delta \times n^\delta$ block $B^{a,b}$ of B, we can find a rank 1 integer matrix $X^{a,b}$ such that $|X^{a,b} - B^{a,b}| \leq n^\delta M$. Namely, $X^{a,b}$ is the first column of $B^{a,b}$ multiplied by the all ones row vector. Therefore, B has all its $n^\delta \times n^\delta$ blocks of Mn^δ-approximate rank at most 1. We can apply Theorem 1.2.1 to get an $\tilde{O}(n^{3-\delta} + (Mn^\delta)^{1/4} n^{(9+\omega)/4})$ time algorithm for computing the Min-Plus product between A and B. When $M = (n^{3-\omega-\epsilon})$, the running time of the algorithm is $\tilde{O}(n^{3-\delta} + n^{3-\epsilon/4+\delta/4})$, which is truly subcubic by setting $\delta = \epsilon/2$.

There is a variant of the main result in which the structure of the matrices is a type of monotonicity. For the purpose of applications of this structured Min-Plus product, we slightly generalize the matrix sizes and consider Min-Plus products $A \star B$ where A has dimension $n \times n^\beta$ and B has dimension $n^\beta \times n$ for $\beta \geq 0$.

Here, $\omega(1, \beta, 1)$ denotes the rectangular matrix multiplication exponent for multiplying an $n \times n^\beta$ matrix and an $n^\beta \times n$ matrix.An $n^\beta \times n$ matrix B is called monotone if for

every $k \in [n]$, $B_{k,j}$ is non-decreasing in $j \in [n]$. For a monotone matrix B, we define its total range as $\sum_{k\in[n^\beta]}(\max_{j\in[n],B_{k,j}\neq\infty} B_{k,j} - B_{k,1} + 1)$. Note that we allow the matrix B to contain entries of value ∞, but these entries can only appear at the end of each row by the definition of a monotone matrix.

Following a similar 3-phase algorithm as the main result, we obtain the following theorem for monotone matrix Min-Plus product.

Theorem 1.2.2. *Let $\beta \geq 0$ be a constant. Let A be an $n \times n^\beta$ matrices whose entries are* polylog n *bit integers and let B be an $n^\beta \times n$ monotone matrices whose entries are* polylog n *bit integers or ∞ with total range $O(n^{\beta+\eta})$. We can compute the Min-Plus product of A and B in time $\tilde{O}(n^2 + n^{\frac{1}{5}(7+4\beta+\eta+\omega(1,\beta,1))})$.*

Here, $\omega(1, \beta, 1)$ denotes the rectangular matrix multiplication exponent for multiplying an $n \times n^\beta$ matrix and an $n^\beta \times n$ matrix. We will use this theorem in the special case when $0 \leq \beta \leq 1$ in our applications. In this case, we use the simple bound $\omega(1, \beta, 1) \leq 2 - 2\beta + \beta\omega$ to get an $\tilde{O}(n^2 + n^{\frac{1}{5}(9+(2+\omega)\beta+\eta)})$ running time. For a more careful analysis of running times using fast rectangular matrix multiplication, please refer to [57].

1.3 Applications of The Main Results

We also find several applications of our sub-cubic time strutured Min-Plus Product algorithms.

1.3.1 Geometric APSP

Let W be the weight adjacency matrix for a graph G. Using a fairly standard technique, we could show that if we could compute the Min-Plus product between W and an arbitrary matrix in truly subcubic time, we could compute APSP of G in truly subcubic time as well. Therefore, as an almost immediate corollary of the main algorithm, we see that if W satisfies the conditions required for Theorem 1.2.1, we could compute APSP of G in truly subcubic time. More formally, define a (W, d, δ)-geometrically weighted clustered graph as follows. $G = (V, E)$ is a (W, d, δ)-geometrically weighted clustered graph if

- V is partitioned into $t = n^{1-\delta}$ subsets $V_1, V_2, \ldots, V_t$ of size $O(n^\delta)$,

- for every $i, j \in \{1, \ldots, t\}$, each $v \in V_i$ is assigned a d-dimensional integer vector $p^{i,j}(v)$, and each $u \in V_j$ is assigned a d-dimensional integer vector $q^{i,j}(u)$, and

- for $v \in V_i$, $u \in V_j$, $|w(v,u) - p^{i,j}(v)^T q^{i,j}(u)| \leq W$. In other words, the edge weights in $V_i \times V_j$ are determined by a matrix whose W-approximate rank is at most d.

Our results show that when $\delta > 0, W \leq O(n^{3-\omega-\epsilon})$ and $d = O(1)$, we can compute APSP in (W, d, δ)-geometrically weighted clustered graphs in truly subcubic time.

1.3.2 Maximum Subarray

In the Maximum Subarray problem, one is given a real valued square matrix and is asked to find the contiguous submatrix with maximum entry sum. Our algorithm focuses on the case when the array has a dimension of 2. Tamaki et al. [46] and Takaoka [45] showed how to use divide-and-conquer to efficiently reduce the 2D Maximum Subarray problem on an $n \times n$ grid to the Min-Plus product of two $n \times n$ matrices. The best algorithm for 2D Maximum Subarray thus runs in $n^3/2^{\Theta(\sqrt{\log n})}$ by using the fastest APSP algorithm by Williams [52]. Also, Vassilevska W. and Williams [49] showed that an $O(n^{3-\epsilon})$ time algorithm for 2D Maximum Subarray for $\epsilon > 0$ would imply an $O(n^{3-\epsilon'})$ time algorithm for Min-Plus product (and hence APSP), for $\epsilon' > 0$. Thus, 2D Maximum Subarray is subcubically equivalent to Min-Plus Product.

For many problems related to APSP, if the input weights are small integer values, the problems become easier. However, it was not known whether this holds true for 2D Maximum Subarray. Therefore, we study 2D Maximum Subarray for the case when all the entries in the array are integers in $\{-M, \ldots, M\}$. By using the divide-and-conquer approach of Tamaki et al. and Takaoka, this problem reduces to a Min-Plus product problem between A, B where $|X_{i,j} + X_{i+1,j+1} - X_{i,j+1} - X_{i+1,j}| \leq M$ for all indices i, j and $X \in \{A, B\}$. It turns out that we can use Theorem 1.2.1 to handle Min-Plus product between such matrices. The running time of our algorithm is $\tilde{O}(n^{\frac{15+\omega}{6}} M^{1/6})$, which is truly subcubic as long as $M = O(n^{3-\omega-\epsilon})$ for $\epsilon > 0$.

1.3.3 Range Mode

In the Batch Range Mode problem, given an array of length n, and n ranges of the array, one is asked to compute the most frequent element for each of the ranges. 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.

Chan et al. [14] showed an $\tilde{O}(n^{1.5})$ time algorithm for this problem. They also showed that any combinatorial algorithm (algorithms that do not use any "algebraic" techniques) for the Batch Range Mode problem requires $n^{1.5-o(1)}$ time, assuming the Boolean Matrix Multiplication Conjecture (see e.g. [48]). We improve the running time of the batch range mode problem to $\tilde{O}(n^{(21+2\omega)/(15+\omega)})$ using Theorem 1.2.2. It does not violate the Boolean Matrix Multiplication Conjecture since Theorem 1.2.2 relies heavily on fast matrix multiplication.

Chan et. al's reduction can be strengthened for Dynamic Range Mode by reducing from the Online Matrix-Vector Multiplication problem [35]. Using $O(n)$ Dynamic Range Mode operations on a sequence of length $O(n)$, we can multiply a $\sqrt{n} \times \sqrt{n}$ boolean matrix with $\sqrt{n}$ boolean vectors given one at a time. This indicates that a Dynamic Range Mode data structure taking $O(n^{1/2-\epsilon})$ time per operation for $\epsilon > 0$ is not possible with current knowledge. Previous attempts indicate the higher $O(n^{2/3})$ per operation cost as the bound to beat [14, 21]. Indeed, $\tilde{O}(n^{2/3})$ time per operation can be achieved with a variety of techniques, but crossing the $O(n^{2/3})$ barrier appears much harder. We show that 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.

1.3.4 Single Source Replacement Paths

In the Single Source Replacement Paths (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.

Grandoni and Vassilevska Williams [32] 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 improved the runtime to $\tilde{O}(Mn^{\omega})$.

We improve their algorithm when there are negative edges. Specifically, we show that 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.

1.4 Preliminary

The Min-Plus product of an $m \times n$ matrix A and an $n \times p$ matrix B is the $m \times p$ matrix $C = A \star B$ such that $C_{i,j} = \min_k \{A[i,k] + B[k,j]\}$.

We use ω to denote the square matrix multiplication exponent, i.e. the smallest real number such that two $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time. The current bound on ω is $2 \leq \omega < 2.373$ [28, 47, 4]. In this thesis, we will use fast rectangular matrix multiplication. Analogous to the square case, we use $\omega(1, \beta, 1)$ to denote the exponent of rectangular matrix multiplication, i.e., the smallest real number such that an $n \times n^{\beta}$ matrix and an $n^{\beta} \times n$ matrix can be multiplied in $n^{\omega(1,\beta,1)+o(1)}$ time. Le Gall and Urrutia [29] computed smallest upper bounds to date for various values of β. In this thesis, we will mostly use the following two simple bounds for better presentation: when $0 \leq \beta \leq 1$, we can split each matrix to $n^{1-\beta}$ matrices of sizes $n^{\beta} \times n^{\beta}$ and multiply each pair of pieces, so $\omega(1, \beta, 1) \leq \beta\omega + 2 - 2\beta$; when $\beta \geq 1$, we can split each matrix to $n^{\beta-1}$ matrices of sizes $n \times n$ and multiply the corresponding pairs, so $\omega(1, \beta, 1) \leq \omega + \beta - 1$.

Chapter 2

Main Algorithms

In this chapter, we describe our main algorithms. In Section 2.1, we introduce a lemma that is useful in both main algorithms. In Section 2.2 we prove Theorem 1.2.1 but with a randomized algorithm. In Section 2.3, we derandomize the algorithm. In Section 2.4, we prove Theorem 1.2.2.

2.1 Improvement over Min-Plus Product with One Bounded-Entry Matrix

We slightly improve on the dependence on the entry size for computing the Min-Plus product of an arbitrary matrix and one matrix with small entries (absolute value smaller than some $M \geq 1$). The first version of this theorem appeared in [50], and was later generalized and improved in [16].

Theorem 2.1.1. *For $0 \leq \beta$, the Min-Plus product of an $n \times n^\beta$ integer matrix A and an $n^\beta \times n$ integer matrix B, where A has entries in $\{-M, \ldots, M\} \cup \{\infty\}$ for some $M \geq 1$ and B has arbitrary* polylog n *bit integer entries can be computed in $\tilde{O}(\sqrt{M} n^{(\omega(1,\beta,1)+\beta+2)/2})$ time.*

Proof. Let $\hat{C}$ be an $n \times n$ matrix, the output of our algorithm. Initialize all entries in $\hat{C}$ to ∞. Let Δ to be a small polynomial in n that will be determined later. We sort each column j of B, and arrange the elements in each column into buckets of size Δ, based on the order

of the elements. Specifically, the smallest Δ elements in column j will be in the first bucket in column j, and the second smallest Δ elements will be in the second bucket, etc. We use $P_{j,\ell}$ to denote the set of row indices k such that $B_{k,j}$ is in the ℓ-th bucket of column j. Let the smallest entry in the ℓ-th bucket be $S_{j,\ell}$ and let the largest entry in the ℓ-th bucket be $L_{j,\ell}$.

Next, for every bucket index $\ell \in [n^\beta/\Delta]$, create a matrix B^ℓ. For the j-th column, if $L_{j,\ell} - S_{j,\ell} > 2M$ (large bucket), we set $B^\ell_{k,j}$ to ∞ for every k; otherwise $L_{j,\ell} - S_{j,\ell} \leq 2M$ (small bucket), and we set $B^\ell_{k,j} := B_{k,j} - S_{j,\ell} - M$ for every $k \in P_{j,\ell}$, and set $B^\ell_{k,j}$ to ∞ for every $k \notin P_{j,\ell}$. Notice that when $L_{j,\ell} - S_{j,\ell} \leq 2M$, $B^\ell_{k,j} = B_{k,j} - S_{j,\ell} - M \in [-M, M]$ for any $k \in P_{j,\ell}$. Thus, we can compute $C^\ell = A \star B^\ell$ in $\tilde{O}(Mn^{\omega(1,\beta,1)})$ time since entries of both A and B^ℓ are in $\{-M, \ldots, M\} \cup \{\infty\}$. We use $C^\ell_{i,j} + S_{j,\ell} + M$ to update $\hat{C}_{i,j}$. Since for every $k \in P_{j,\ell}$ when $P_{j,\ell}$ is a small bucket, $A_{i,k} + B^\ell_{k,j} + S_{j,\ell} + M = A_{i,k} + B_{k,j}$, we are essentially using $A_{i,k} + B_{k,j}$ to update $\hat{C}_{i,j}$ for every $k \in P_{j,\ell}$, if $P_{j,\ell}$ is a small bucket. Thus, after this part of the algorithm, $\hat{C}_{i,j} = \min_{k \in SB(j)}\{A_{i,k} + B_{k,j}\}$, where $SB(j)$ is the union of indices in small buckets in column j. This step takes $\tilde{O}(Mn^{\omega(1,\beta,1)+\beta}/\Delta)$ time since we compute $O(n^\beta/\Delta)$ instances of Min-Plus product of two matrices whose entries are in $\{-M, \ldots, M\} \cup \{\infty\}$.

Thus, for each pair (i, j), we only need to calculate $\min_{k \notin SB(j)}\{A_{i,k} + B_{k,j}\}$. In order to compute this, we first need to find the set of large buckets that contain an index k where $A_{i,k} < \infty$. Formally, for each i, j, we want to find

$$\{\ell : P_{j,\ell} \text{ is a "large" bucket, and there exists } k \in P_{j,\ell} \text{ such that } A_{i,k} < \infty\}.$$

We can do this in n/Δ iterations. In each iteration ℓ, we create a $\{0, \infty\}$-matrix $\bar{A}$ such that $\bar{A}_{i,k} = 0$ if and only if $A_{i,k} < \infty$. We also create a $\{0, \infty\}$-matrix $\bar{B}^\ell$ such that $\bar{B}^\ell_{k,j} = 0$ if and only if $B_{k,j}$ belongs to the ℓ-th bucket in column j. The result $\bar{C}^\ell = \bar{A} \star \bar{B}^\ell$ can be computed in $O(n^{\omega(1,\beta,1)})$ time. If $\bar{C}^\ell_{i,j} = 0$, then bucket $P_{j,\ell}$ contains an index k such that $A_{i,k} < \infty$. This step takes $\tilde{O}(n^{\omega(1,\beta,1)+\beta}/\Delta)$ time since we compute $O(n^\beta/\Delta)$ instances of Min-Plus product with entries in $\{0, \infty\}$.

Naively, for each pair (i, j), we want to enumerate indices in all large buckets $P_{j,\ell}$ that

contains an index k where $A_{i,k} < \infty$. However, it is not necessary. Consider three large buckets $\ell_1 < \ell_2 < \ell_3$ (the order here means the entries in bucket ℓ_1 are smallest, and the entries in bucket ℓ_3 are largest). Pick any $k_1 \in P_{j,\ell_1}, k_3 \in P_{j,\ell_3}$ such that $A_{i,k_1} < \infty$ and $A_{i,k_3} < \infty$. Note that $A_{i,k_1} + B_{k_1,j} \leq M + L_{j,\ell_1}$. Since buckets are ordered, the largest entry in bucket P_{j,ℓ_1} is at most the smallest entry in bucket P_{j,ℓ_2}. Thus, $A_{i,k_1} + B_{k_1 j} \leq M + S_{j,\ell_2}$. Similarly, $A_{i,k_3} + B_{k_3,j} \geq -M + S_{j,\ell_3} \geq -M + L_{j,\ell_2}$. Since P_{j,ℓ_2} is a large bucket, $L_{j,\ell_2} - S_{j,\ell_2} > 2M$, which leads to $A_{i,k_1} + B_{k_1,j} < A_{i,k_3} + B_{k_3,j}$. It means that if we have two buckets P_{j,ℓ_1} and P_{j,ℓ_2} that each contains an index k where $A_{i,k} < \infty$, all buckets that are larger than them won't give a better candidate k. Therefore, for each (i, j), we only need to enumerate the first two large buckets that contain indices k where $A_{i,k} < \infty$. Thus, it takes $\tilde{O}(n^2\Delta)$ time to cover large buckets.

In total, the running time of the algorithm is $\tilde{O}(Mn^{\omega(1,\beta,1)+\beta}/\Delta + n^2\Delta)$. Setting $\Delta = \sqrt{M}n^{(\omega(1,\beta,1)+\beta-2)/2}$ gives the claimed $\tilde{O}(\sqrt{M}n^{(\omega(1,\beta,1)+\beta+2)/2})$ time. □

2.2 Main Algorithm

Our algorithm will use the following efficient data structure for half-space query in $\mathbb{R}^d$ for constant d.

Theorem 2.2.1 ([40]). *For any constant $d \geq 2$, there exists a data structure that supports*

- *Given a set P of n points in $\mathbb{R}^d$, preprocess them in $\tilde{O}(n)$ time;*
- *Given a halfspace $\lambda = \{x \in \mathbb{R}^d | v^T x \leq b\}$, test whether $|P \cap \lambda| > 0$ in $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor})$ time.*
- *Given a halfspace $\lambda = \{x \in \mathbb{R}^d | v^T x \leq b\}$, report all points in $P \cap \lambda$ in $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor} + k)$ time, where $k = |P \cap \lambda|$.*

Let Δ be a positive integer that is a small polynomial in n. Assume for simplicity that n is a multiple of Δ. Then we can partition $[n]$ into n/Δ groups by setting $I(i') = \{i : i' - \Delta < i \leq i'\}$ for any i' divisible by Δ. For any i', j' that are multiples of Δ, we can group all entries $A_{i,j}$ where $i \in I(i'), j \in I(j')$ into a sub-matrix of size $\Delta \times \Delta$, thus partitioning

A into sub-matrices of size $\Delta \times \Delta$. We can similarly partition B into sub-matrices of size $\Delta \times \Delta$.

In Theorem 2.2.2 below we will show that if each of the $\Delta \times \Delta$ sub-matrices of B are close in ℓ_∞ norm to an $O(1)$-rank matrix, then we can compute $A \star B$ in truly sub-cubic time. In other words, we need the blocks of B to have constant n^ε-approximate rank for small $\varepsilon > 0$.

Theorem 2.2.2. *Let A, B be two given $n \times n$ matrices whose entries are* polylog n *bit integers. Let W be a nonnegative integer and let $d \geq 1$ be an integer with $d = O(1)$. Suppose that for all k', j' multiples of Δ, we can find two d by Δ integer matrices $X_{k',j'}$ and $Y_{k',j'}$, such that for any $(k, j) \in I(k') \times I(j')$, $\left|B_{k,j} - X_{k',j'}(k)^T Y_{k',j'}(j)\right| \leq W$. Then, for any integer $\rho \geq 1$, there exists a*

$$\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (d+1)/2 \rfloor} + \rho\sqrt{W} n^{(3+\omega)/2} + n^3/\rho)$$

time algorithm that computes $A \star B$.

To obtain Theorem 1.2.1 from Theorem 2.2.2, we set ρ to $\lceil n^{(3-\omega)/4} W^{-1/4} \rceil$ when $W \leq n^{3-\omega}$; otherwise we can run the trivial cubic time algorithm for Min-Plus product.

The algorithm starts with the framework behind the Bringmann et al. algorithm [11] that computes the Min-Plus product of two matrices with bounded differences. However, each of the three steps in the framework requires a completely different approach due to the less structured nature of matrix B. The resulting algorithm is a strong generalization of the algorithm of [11].

In the rest of this section, we will use $C = A \star B$ to denote the desired Min-Plus product, and use $\hat{C}$ as the output of our algorithm. The algorithm contains three phases. In the first phase, we will compute a matrix $\tilde{C}$, such that every entry of $\tilde{C}$ is an additive approximation of the corresponding entry in the desired output C. In the second phase, we will compute $\hat{C}$ by calculating the Min-Plus product of some small weight matrices generated by A, B and $\tilde{C}$ using fast matrix multiplication. In the third phase, we will correct all entries of $\hat{C}$ by efficiently enumerating all $A_{ik} + B_{kj}$ that can possibly improve $\hat{C}_{ij}$.

2.2.1 Phase 1: Approximated Min-Plus Product

For each triple (i', k', j') such that all i', k', j' are multiples of Δ, if we can compute an additive approximation $\tilde{C}^{i',k',j'}$ of the Min-Plus product $A_{I(i'),I(k')} \star B_{I(k'),I(j')}$, then we can, in $O(n^3/\Delta)$ time, compute $\tilde{C}_{i,j} = \min_{k':\Delta|k'} \tilde{C}^{i',k',j'}_{i,j}$ where $i \in I(i'), j \in I(j')$. We will use the geometric data structure from Theorem 2.2.1 to approximate $A_{I(i'),I(k')} \star B_{I(k'),I(j')}$.

Lemma 2.2.1. *There exists a $\tilde{O}(\Delta^{3-1/\lfloor (d+1)/2 \rfloor})$ time algorithm that computes a W-additive approximation $\tilde{C}^{i',k',j'}$ of $A_{I(i'),I(k')} \star B_{I(k'),I(j')}$, for any i', k', j' multiples of Δ.*

Proof. By the structure of B, for any $(k, j) \in I(k') \times I(j')$, we have

$$\left| B_{k,j} - X_{k',j'}(k)^T Y_{k',j'}(j) \right| \leq W.$$

Therefore, if we can accurately compute

$$\tilde{C}^{i',k',j'}_{i,j} = \min_{k \in I(k')} \left\{ A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) \right\},$$

we immediately get a W-additive approximation of $A_{I(i'),I(k')} \star B_{I(k'),I(j')}$.

Create a set of $(d+1)$-dimensional points

$$P_i = \left\{ \begin{pmatrix} A_{i,k} \\ X_{k',j'}(k) \end{pmatrix} : k \in I(k') \right\},$$

and use the data structure in Theorem 2.2.1 to pre-process this set. Each set has size $O(\Delta)$, and there are $|I(i')| = \Delta$ such sets, so the total pre-processing time is $\tilde{O}(\Delta^2)$. For any $j \in I(j')$, we create a $(d+1)$-dimensional vector $v_j = \begin{pmatrix} 1 \\ Y_{k',j'}(j) \end{pmatrix}$. We observe that

$$A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) = v_j^T \begin{pmatrix} A_{i,k} \\ X_{k',j'}(k) \end{pmatrix},$$

so $\tilde{C}^{i',k',j'}_{i,j} = \min_{x \in P_i} v_j^T x$. In order to compute $\min_{x \in P_i} v_j^T x$ for every pair (i, j), we use the emptiness query of the geometric data structure. We want to find the minimum value of

b, so that there exists a point $x \in P_i$ where $v_j^T x \leq b$. This is equivalent to testing whether the half-space $\lambda = \{x \in \mathbb{R}^{d+1} | v_j^T x \leq b\}$ intersects P_i. Therefore, we can use binary search on the minimum value of b, which will be equal to $\tilde{C}_{i,j}^{i',k',j'}$.

Each emptiness query takes $\tilde{O}(\Delta^{1-1/\lfloor (d+1)/2 \rfloor})$ time, and we need to query $O(\log(|A|_\infty + |B|_\infty))$ time for each pair $(i,j) \in I(i') \times I(j')$, so in total it takes $\tilde{O}(\Delta^{3-1/\lfloor (d+1)/2 \rfloor})$ time to compute $\tilde{C}^{i',k',j'}$. □

Lemma 2.2.2. *There exists a $\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (d+1)/2 \rfloor})$ time algorithm that computes a W-additive approximation $\tilde{C}$ of $A \star B$.*

Proof. For every triple (i', k', j') where i', k', j' are multiples of Δ, we compute $\tilde{C}^{i',k',j'}$ using the algorithm in Lemma 2.2.1. Since there are $O((n/\Delta)^3)$ such triples, it takes $\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (d+1)/2 \rfloor})$ time in total. Then we compute $\tilde{C}$ using $\tilde{C}_{i,j} = \min_{k' : \Delta | k'} \tilde{C}_{i,j}^{i',k',j'}$ in $O(n^3 \cdot \Delta^{-1})$ time. □

2.2.2 Phase 2: Create Estimate Matrix $\hat{C}$ by Random Sampling

This phase of the algorithm consists of $10\rho \ln n$ rounds. For each round r, we sample $j^r \in [n]$ uniformly at random. Define A^r to be an $n \times n$ matrix where $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$, and define B^r such that $B^r_{k,j} := B_{k,j} - B_{k,j^r}$. If we compute $C^r = A^r \star B^r$, we can infer $C = A \star B$ via the relation $C_{i,j} = C^r_{i,j} + \tilde{C}_{i,j^r}$. However, it is not always possible to compute C^r efficiently, since the weights of A^r and B^r can be arbitrarily large. Therefore, we need to set the large entries in A^r to be ∞ in order to compute $A^r \star B^r$ efficiently. Specifically, we will set an entry of A^r to ∞ if its absolute value is more than $3W$. Then we can compute $C^r = A^r \star B^r$ in $\tilde{O}(\sqrt{W} n^{(3+\omega)/2})$ time by Theorem 2.1.1.

This phase deviates from the approach of Bringmann et al. Bringmann et al. set the large entries of both A^r and B^r to ∞. If we were to do that, we wouldn't be able to complete Phase 3 – there doesn't seem to be enough to finish the Min-Plus product computation in truly subcubic time. By only setting the large entries of A^r to ∞ and letting B^r keep all its entries, we offload enough work onto Phase 2, so that now Phase 3 can also be done in truly subcubic time.

Since there are ρ rounds, the total time complexity of this phase is $\tilde{O}(\rho\sqrt{W} n^{(3+\omega)/2})$.

Intuitively, fix any $i, j \in [n]$, if $A^r_{i,k}$ is not set to ∞, then $C^r_{i,j} \leq \left(A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}\right) + (B_{k,j} - B_{k,j^r}) = A_{i,k} + B_{k,j} - \tilde{C}_{i,j^r}$. Thus, if we take $\hat{C}_{i,j}$ to be $\min_r \left\{C^r_{i,j} + \tilde{C}_{i,j^r}\right\}$, then $\hat{C}_{i,j} \leq A_{i,k} + B_{k,j}$ as long as $A^r_{i,k} < \infty$ for at least one r. We will formalize this intuition and show that we only need to enumerate a sub-cubic number of (i, k, j) triples in order to correct all entries in $\hat{C}$ after $10\rho \ln n$ rounds.

Definition 2.2.1. *We call a triple* (i, k, j)

- *strongly relevant if* $A_{i,k} + B_{k,j} = C_{i,j}$*;*
- *weakly relevant if* $|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}| \leq 3W$*;*
- *uncovered if for all* $1 \leq r \leq 10\rho \ln n$, $|A^r_{i,k}| > 3W$.

Since whether a triple (i, k, j) is uncovered only depends on (i, k), we will also call a pair (i, k) uncovered if for all $1 \leq r \leq 10\rho \ln n$, $|A^r_{i,k}| > 3W$. A triple (pair) that is not uncovered will be called *covered*.

If a triple (i, k, j) is not strongly relevant, then even if $A^r_{i,k} = \infty$ for every round r, it doesn't affect whether $\hat{C}_{i,j} = C_{i,j}$. If a triple (i, k, j) is covered, then there exists a round r such that $A^r_{i,k}$ is not set to ∞. In this case, $\hat{C}_{i,j} \leq C^r_{i,j} + \tilde{C}_{i,j^r} \leq A_{i,k} + B_{k,j}$. Since only strongly relevant triples matter, and our algorithm already updates the answer for every covered triples, so we need to update $\hat{C}$ using triples that are both strongly relevant and uncovered. Specifically, if we can enumerate all strongly relevant and uncovered triples (i, k, j), and update $\hat{C}_{i,j}$ using $A_{i,k} + B_{k,j}$, we can correct all entries in $\hat{C}$.

However, it is hard to only enumerate strongly relevant and uncovered triples without enumerating some additional triples. Thus we allow the algorithm to enumerate some of the *weakly* relevant and uncovered triples, in addition to strongly relevant and uncovered triples. In this way, we can cover all strongly relevant and uncovered triples, while keeping the total number of triples small. Note that since $\tilde{C}$ is a W-additive approximation of C, a strongly relevant triple is always weakly relevant, so we care about the total number of weakly relevant and uncovered triples. The next lemma shows that the number of such triples is truly sub-cubic.

Lemma 2.2.3. *With high probability, the number of weakly relevant and uncovered triples is at most n^3/ρ.*

Proof. We say a pair (i, k) is bad if the number of weakly relevant triples (i, k, j) is greater than n/ρ.

Fix any bad (i, k). For a random $j \in [n]$, the probability that (i, k, j) is weakly relevant is at least $1/\rho$. Since we have $10\rho \ln n$ randomly sampled j^r, the probability that at least one j^r forms a weakly relevant triple (i, k, j^r) is at least $1 - (1 - 1/\rho)^{10\rho \ln n} \geq 1 - 1/n^{10}$. Suppose (i, k, j^r) is weakly relevant, then $|A^r_{i,k}| = |A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}| \leq 3W$. Thus, $A^r_{i,k}$ will not be set to ∞ in round r, so (i, k) is covered. By taking a union bound over all bad (i, k), we conclude that with probability at least $1 - 1/n^8$, all triples (i, k, j) will be covered when (i, k) is bad. It means that with high probability, these bad (i, k) pairs don't contribute any weakly relevant and uncovered triples.

For a pair (i, k) that is not bad, the number of j such that (i, k, j) is weakly relevant is at most n/ρ, by definition of a bad pair. Thus, these (i, k) pairs contribute at most n^3/ρ weakly relevant and uncovered pairs. □

2.2.3 Phase 3: Enumerate Strongly Relevant and Uncovered Triples

It remains to show how to quickly iterate through strongly relevant, uncovered triples. Fix i', k', j' multiples of Δ, we will show how to efficiently enumerate strongly relevant, uncovered triples in $I(i') \times I(k') \times I(j')$. We consider the set $S_{i',k',j'} \subseteq I(i') \times I(j') \times I(k')$, consisting of triples (i, j, k) such that $A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) \leq 2W + \tilde{C}_{i,j}$. The following lemma shows that it is sufficient to enumerate triples in this set.

Lemma 2.2.4. *The set $S_{i',k',j'}$ contains all strongly relevant triples in $I(i') \times I(j') \times I(k')$, and it contains only weakly relevant triples.*

Proof. Let (i, k, j) be any strongly relevant triple. Then

$$\begin{aligned}
&A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j} \\
=&A_{i,k} + B_{k,j} - C_{i,j} + \left(X_{k',j'}(k)^T Y_{k',j'}(j) - B_{k,j}\right) + \left(C_{i,j} - \tilde{C}_{i,j}\right) \\
\leq&2W,
\end{aligned}$$

so $(i,k,j) \in S_{i',k',j'}$.

In order to prove the second claim, we need to show $\left|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}\right| \leq 3W$ for every triple $(i,j,k) \in S_{i',k',j'}$. Since $\tilde{C}$ is a W-additive approximation of C, $A_{i,k} + B_{k,j} - \tilde{C}_{i,j} \geq -W$ holds for every triple (i,k,j). Since $(i,k,j) \in S_{i',k',j'}$, we have $A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) \leq 2W + \tilde{C}_{i,j}$, or equivalently:

$$A_{i,k} + B_{k,j} - \tilde{C}_{i,j} \leq 2W + (B_{k,j} - X_{k',j'}(k)^T Y_{k',j'}(j)).$$

Since $B_{k,j}$ differs at most W from $X_{k',j'}(k)^T Y_{k',j'}(j)$, we have $A_{i,k} + B_{k,j} - \tilde{C}_{i,j} \leq 3W$. $\square$

By Lemma 2.2.4, it suffices to enumerate uncovered triples in $S_{i',k',j'}$. For each $i \in I(i')$, create a set of $(d+1)$-dimensional points

$$Q_i = \left\{ \begin{pmatrix} A_{i,k} \\ X_{k',j'}(k) \end{pmatrix} : k \in I(k') \wedge (i,k) \text{ is uncovered} \right\},$$

and pre-process these points using the data structure in Theorem 2.2.1. For each $(i,j) \in I(i') \times I(j')$, we create the following half-space:

$$\lambda_{i,j} = \left\{ x \in \mathbb{R}^{d+1} | \begin{pmatrix} 1 \\ Y_{k',j'}(j) \end{pmatrix}^T x \leq 2W + \tilde{C}_{i,j} \right\}.$$

Then $Q_i \cap \lambda_{i,j}$ contains the set of $k \in I(k')$ such that $(i,k,j) \in S_{i',j',k'}$ and (i,k) is uncovered. Therefore, we can use the data structure in Theorem 2.2.1 to list the set of k in $\tilde{O}(\Delta^{1-1/\lfloor (d+1)/2 \rfloor} + |Q_i \cap \lambda|)$ time. Note that the total number of listed points is bounded by the number of weakly-relevant, uncovered triples, so the summation of the second term over all i', k', j', i, j is $\tilde{O}(n^3/\rho)$. The summation of the first term over all i', k', j', i, j is $\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (d+1)/2 \rfloor})$.

2.3 Derandomization of the Main Algorithm

The only randomness used by the algorithm is to sample random $j^r \in [n]$ in Phase 2. In order to remove this randomness, we need to first define the following notion of *approximately relevant triples*.

Definition 2.3.1. *A triple (i,k,j), where $k \in I(k'), j \in I(j')$ for some k', j' divisible by Δ, is called approximately relevant if $\left|A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j}\right| \leq 4W$.*

Approximately relevant triples are strongly related to weakly relevant triples by the following lemma.

Lemma 2.3.1. *Any weakly relevant triple (i,k,j) is also approximately relevant.*

Proof. Consider

$$\begin{aligned}&\left|\left(A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j}\right) - \left(A_{i,k} + B_{k,j} - \tilde{C}_{i,j}\right)\right| \\ =&\left|X_{k',j'}(k)^T Y_{k',j'}(j) - B_{k,j}\right| \leq W\end{aligned}$$

Therefore, by the simple inequality $||a| - |b|| \leq |a-b|$, we know that

$$\left|\left|A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j}\right| - \left|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}\right|\right| \leq W.$$

For any weakly relevant triple (i,k,j), $\left|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}\right| \leq 3W$ by definition. Since the difference between it and $\left|A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j}\right|$ is bounded by W, the latter cannot exceed $4W$, which means (i,k,j) is approximately relevant. □

Therefore, in order to cover approximately relevant triples, when computing $A^r \star B^r$, we need to keep all entries of A^r that have absolute value at most $5W$, but it won't change time complexity.

After we sample some j^r, if the number of uncovered, approximately relevant triples is $O(n^3/\rho)$, then by Lemma 2.3.1, the number of uncovered, weakly relevant triples is $O(n^3/\rho)$ as well. In the rest of this section, we show how to *deterministically* choose the

set of j^r, so that the number of uncovered, approximately relevant triples is $O(n^3/\rho)$ after computing $A^r \star B^r$ for all j^r.

We first notice that a triple (i,k,j) is approximately relevant if and only if $A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j} \leq 4W$, since this quantity can never be smaller than $-4W$. Fix $i', k', j', i \in I(i')$. For every $j \in I(j')$, we add the point $\begin{bmatrix} -\tilde{C}_{i,j} \\ Y_{k',j'}(j) \end{bmatrix}$ to the geometric data structure. This takes $\tilde{O}(n^3/\Delta)$ time. Then for each $k \in I(k')$, we use the geometric data structure to list points in the half-space $\begin{bmatrix} -A_{i,k} \\ X_{k',j'}(k) \end{bmatrix}^T x \leq 4W$. It will take $\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (d+1)/2 \rfloor}) + O(\text{total number of points listed})$. For each (i,k) pair, if we stop listing j as soon as we get n/ρ values of j, the total number of points listed would be $O(n^3/\rho)$.

Finally, for the (i,k) pairs that have less than n/ρ values of j listed, we ignore these pairs . For every other pair (i,k), we have a set $S(i,k)$ that contains n/ρ values of j such that (i,k,j) is approximately relevant. We need to find a set of j^r that intersects with each of these $S(i,k)$ sets. By the standard greedy algorithm for hitting set/set cover, we can choose $\tilde{O}(\rho)$ different j^r in $\tilde{O}(n^3/\rho)$ time, so that each $S(i,k)$ contains at least one j^r we choose.

The other parts of the algorithm proceed similarly, and it will have the same running time as the *randomized* version.

2.4 Algorithm for Monotone Min-Plus Product

Our Monotone Min-Plus Product algorithm depends on the following lemma, which is a simple algorithm that works fast when the total range is very small.

Lemma 2.4.1. *Let $\beta \geq 0$ be a constant. Let A be an $n \times n^\beta$ matrices whose entries are* polylog n *bit integers and let B be an $n^\beta \times n$ monotone matrices whose entries are* polylog n *bit integers or ∞ with total range $O(n^{\beta+\eta})$. We can compute the Min-Plus product of A and B in time $\tilde{O}(n^2 + n^{1+\beta+\eta})$.*

Proof. Say we would like to compute $C = A \star B$. For a fixed row $i \in [n]$, we iterate

through columns $j \in [n]$, 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 $i \in [n]$, we need to make $O(n^{\beta+\eta})$ updates and $O(n)$ 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 thus $\tilde{O}(n^2 + n^{1+\beta+\eta})$. □

We recall Theorem 1.2.2:

Theorem 1.2.2. *Let $\beta \geq 0$ be a constant. Let A be an $n \times n^\beta$ matrices whose entries are* polylog n *bit integers and let B be an $n^\beta \times n$ monotone matrices whose entries are* polylog n *bit integers or ∞ with total range $O(n^{\beta+\eta})$. We can compute the Min-Plus product of A and B in time $\tilde{O}(n^2 + n^{\frac{1}{5}(7+4\beta+\eta+\omega(1,\beta,1))})$.*

Proof of the Theorem also follows the previous three-phase framework.

Proof of Theorem 1.2.2. 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 $\tilde{A} \star \tilde{B}$ using Lemma 2.4.1 and let $\tilde{C}_{i,j} = (\tilde{A} \star \tilde{B})_{i,j} W$. Then $||\tilde{C} - C||_\infty \leq 2W$.

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[1]. Define matrix A^r as $A^r_{i,k} = A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r}$ if $A_{i,k} + B_{k,j^r} - \tilde{C}_{i,j^r} \leq 3W$ and $A^{r'}_{i,k} = \infty$ for all $r' < r$; $A^r_{i,k} = \infty$ otherwise. Define matrix B^r as $B^r_{k,j} = B_{k,j} - B_{k,j^r}$ if $B_{k,j^r} \neq \infty$; $B^r_{k,j} = 0$ otherwise. We compute $C^r = A^r \star B^r$ using Theorem 2.1.1 because A^r has bounded entries. Finally, for all $i, j \in [n]$, we make the update $\hat{C}_{i,j} \leftarrow \min\{\hat{C}_{i,j}, C^r_{i,j} + \tilde{C}_{i,j^r}\}$.

[1]For simplicity of presentation, we use randomness here. Similar to Section 2.3, this part can easily be derandomized.

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.

Following the algorithm for Theorem 1.2.1, 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.

The following lemma is a slight generalization of Lemma 2.2.3 which also works for rectangular matrices.

Lemma 2.4.2. *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, $A^r_{i,k} \neq \infty$ 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 $\tilde{O}(n^{(\omega(1,\beta,1)+\beta+2+\theta)/2})$ time by Theorem 2.1.1, so in total Phase 2 costs $\tilde{O}(n^{(\omega(1,\beta,1)+\beta+2+\theta+2\rho)/2})$ time.

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 2.4.3. *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 2.4.4. *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 2.4.3, it suffices to enumerate over moderately relevant and uncovered triples to recover all of C. By Lemmas 2.4.2 and 2.4.4, the number of moderately relevant and uncovered triples is at most $O(n^{2+\beta-\rho})$, with high probability.

Lemma 2.4.5. *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 2.4.1 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 $B_{k,j} \neq B_{k,j-1}$ 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 2.4.1, plus the number of triples emitted, which, with high probability, is at most $O(n^{2+\beta-\rho})$, by Lemma 2.4.2. □

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

Summary. The overall running time of our algorithm is

$$\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta,(\omega(1,\beta,1)+\beta+2+\theta+2\rho)/2,2+\beta-\rho\}}).$$

By setting $\theta = \frac{1}{5}(\beta + 4\eta - \omega(1,\beta,1) - 2)$ and $\rho = \frac{1}{5}(\beta - \eta - \omega(1,\beta,1) + 3)$, we get $\tilde{O}(n^2 + n^{\frac{1}{5}(7+4\beta+\eta+\omega(1,\beta,1))})$ as claimed. Note that by setting θ and ρ this way, they could potentially be negative. However, if it happens, then the naive $\tilde{O}(n^{2+\beta})$ time algorithm or the $\tilde{O}(n^2 + n^{1+\beta+\eta})$ time algorithm from Lemma 2.4.1 already beat the claimed running time. $\square$

Chapter 3

Geometric APSP

3.1 Introduction

Typically, an algorithm for a structured version of Min-Plus Product implies an algorithm for a structured version of APSP. An almost immediate consequence of Theorem 1.2.1 is that APSP for graphs whose generalized adjacency matrix has $n^\delta \times n^\delta$ blocks of constant W-approximate rank and whose entries are polylog n bit integers can be solved in truly subcubic time when $\delta > 0$ and $W \leq O(n^{3-\omega-\varepsilon})$ for some $\varepsilon > 0$.

The proof is fairly standard: iterate the Min-Plus product of Theorem 1.2.1 L times, where in the ith iteration B is the generalized adjacency matrix of the graph and A is the matrix computed in the $(i-1)$-th iteration (in the first iteration $A = B$). Then in the Lth iteration one has computed the shortest paths in the graph using at most L edges. To handle the paths longer than L one computes SSSP from a random sample of $\tilde{O}(n/L)$ vertices, and L is chosen to balance the running times.

Let us discuss what the graphs that we can handle look like: Define a (W, d, δ)-geometrically weighted clustered graph, (W, d, δ)-GWC for short as follows. $G = (V, E)$ is (W, d, δ)-GWC if

- V is partitioned into $t = n^{1-\delta}$ subsets $V_1, V_2, \ldots, V_t$ of size $O(n^\delta)$,
- for every $i, j \in \{1, \ldots, t\}$, each $v \in V_i$ is assigned a d-dimensional integer vector $p^{i,j}(v)$, and each $u \in V_j$ is assigned a d-dimensional integer vector $q^{i,j}(u)$, and

- for $v \in V_i$, $u \in V_j$, $|w(v,u) - p^{i,j}(v)^T q^{i,j}(u)| \leq W$. In other words, the edge weights in $V_i \times V_j$ are determined by a matrix whose W-approximate rank is at most d.

Notice that (W, d, δ)-GWC graphs can simulate a lot of structure. For instance, imagine that each vertex j is represented by an integer x_j, and the weights are determined by some degree d (for $d = O(1)$) polynomial function p of x_i and x_j, up to an error at most W. Then, the weights can be represented (up to noise at most W in each entry) with inner products of vectors v_i and v'_j of length d^2, where $v_i[a,b]$ is the monomial of $p(x'_i, x'_j)$ corresponding to $(x'_i)^a \cdot (x'_j)^b$ with the corresponding coefficient coming from p, evaluated at $x'_i = x_i$ and $x'_j = 1$, and $v'_j[a,b]$ is x_j^b; then we get that $v_i^T v'_j = p(x_i, x_j)$. A similar argument can be carried over if the x_i are $O(1)$ dimensional vectors and p is a polynomial in the entries of x_i and x_j.

In [13], Chan studied a related version of geometrically weighted APSP where the weights between two vertices can be arbitrary algebraic functions, instead of just dot products between two vectors or polynomials. We remark that if we replace the geometric data structure that our Theorem 1.2.1 uses (Theorem 2.2.1) with the partition theorem in [2], we can achieve APSP for arbitrary algebraic functions as in [13], as long as the produced edge weights are integers. Moreover, our algorithm allows the edge weights to disagree with the function of their endpoints by an additive error, while the algorithm in [13] requires the edge weights to exactly agree with the function. In other words, in the case of integer edge weights, we obtain a more powerful geometric APSP algorithm.

3.2 Algorithm

Let W be an integer, $d \geq 1$ be a constant integer and let $\delta \in (0, 1]$ be a constant. Let us define (as in the introduction) a (W, d, δ)-geometrically weighted clustered graph, (W, d, δ)-GWC for short as follows. $G = (V, E)$ is (W, d, δ)-GWC if

- V is partitioned into $t = n^{1-\delta}$ subsets $V_1, V_2, \ldots, V_t$ of size $O(n^\delta)$,
- for every $i, j \in \{1, \ldots, t\}$, each $v \in V_i$ is assigned a d-dimensional integer vector $p^{i,j}(v)$, and each $u \in V_j$ is assigned a d-dimensional integer vector $q^{i,j}(u)$, and

- for $v \in V_i$, $u \in V_j$, $|w(v,u) - p^{i,j}(v)^T q^{i,j}(u)| \leq W$. In other words, the edge weights in $V_i \times V_j$ are determined by a matrix whose W-approximate rank is at most d,
- the absolute value of any edge weight is at most $O(n^c)$ for some constant c.

The last bullet is only needed so that SSSP in such graphs can be performed in truly subcubic time even if there are negative edge weights, e.g. as in Goldberg [30].

The following is a direct corollary of Theorem 1.2.1:

Corollary 3.2.1. *For any integer matrix A and B the generalized adjacency matrix of a (W, d, δ)-GWC graph, we can compute $C = A \star B$ in $\tilde{O}(n^{3-\delta/\lfloor (d+1)/2 \rfloor} + n^{(9+\omega)/4} \cdot W^{1/4})$ time.*

Using Corollary 3.2.1, we can compute the shortest distance between two vertices among all paths with a small length. Using a standard technique in APSP algorithms, we can compute shortest paths among the long paths by randomly sampling vertices.

Theorem 3.2.1. *We can compute APSP for a (W, d, δ)-GWC graph in*

- *$\tilde{O}(W^{1/8} n^{(21+\omega)/8})$ time whenever $W > n^{3-\omega-4\delta/\lfloor (d+1)/2 \rfloor}$, and*
- *$\tilde{O}(n^{3-\delta/(2\lfloor (d+1)/2 \rfloor)})$ time if $W \leq n^{3-\omega-4\delta/\lfloor (d+1)/2 \rfloor}$.*

Proof. Let B be the generalized adjacency matrix, and let ℓ be a parameter to be fixed later. For each $i \leq \ell$, we can compute $B^{(i)}$ by iterating the product $B^{(i)} \leftarrow A \star B$ for $A = B^{(i-1)}$. By Corollary 3.2.1, this step will take $\tilde{O}(\ell \cdot n^{3-\delta/\lfloor (d+1)/2 \rfloor} + \ell \cdot n^{(9+\omega)/4} \cdot W^{1/4})$ time.

We can randomly sample $\tilde{\Theta}(n/\ell)$ vertices S, and perform Dijkstra's algorithm to and from these vertices in S (after the usual Johnson's preprocessing to get rid of any negative weights, and using say Goldberg's SSSP algorithm which works in truly subcubic time since the edge weights are assumed to be polynomial in n). With high probability, S hits a shortest path between every two vertices that have a shortest path containing at least ℓ vertices. We can perform this step in $\tilde{\Theta}(n^3/\ell)$ time.

The first step gives the shortest path between two vertices that uses at most ℓ vertices, and the second step gives the shortest path that uses more than ℓ vertices. Thus, by taking the smaller one over these two, we can correctly compute the APSP.

The total time complexity is $\tilde{O}(\ell \cdot n^{3-\delta/\lfloor (d+1)/2 \rfloor} + \ell \cdot n^{(9+\omega)/4} \cdot W^{1/4} + n^3/\ell)$.

If $W > n^{3-\omega-4\delta/\lfloor (d+1)/2 \rfloor}$, then $n^{(9+\omega)/4} \cdot W^{1/4} > n^{3-\delta/\lfloor (d+1)/2 \rfloor}$, so the running time is

$$\tilde{O}(\ell \cdot n^{(9+\omega)/4} \cdot W^{1/4} + n^3/\ell).$$

We can set ℓ to be $n^{(3-\omega)/8}/W^{1/8}$, balancing the two terms of the runtime and thus minimizing it at $\tilde{O}(W^{1/8} n^{(21+\omega)/8})$.

Otherwise, if $W \leq n^{3-\omega-4\delta/\lfloor (d+1)/2 \rfloor}$, then $n^{(9+\omega)/4} \cdot W^{1/4} \leq n^{3-\delta/\lfloor (d+1)/2 \rfloor}$, so the running time is

$$\tilde{O}(\ell \cdot n^{3-\delta/\lfloor (d+1)/2 \rfloor} + n^3/\ell).$$

Then it makes sense to set $\ell = n^{\delta/(2\lfloor (d+1)/2 \rfloor)}$, minimizing the runtime to $\tilde{O}(n^{3-\delta/(2\lfloor (d+1)/2 \rfloor)})$.

□

Chapter 4

Maximum Subarray

4.1 Introduction

In the Maximum Subarray problem, one is given a real valued square matrix and is asked to find the contiguous submatrix of maximum entry sum. First studied by Bentley [9], the problem has many applications, for instance in graphics (see [46]) and in databases [3, 25, 26, 24, 54].

The Maximum Subarray problem can be generalized to arbitrary dimension d: here one is given a d-dimensional grid (or tensor) with n coordinates in each dimension (i.e. $[n]^d$), each point in the grid has a real value, and one is asked to return the contiguous subgrid of maximum entry sum. In 1D, Kadane's algorithm (presented in [9]) achieves a linear, $O(n)$ running time. Bentley [8] showed how to use Kadane's algorithm to solve the 2D variant of the Maximum Subarray problem in $O(n^3)$ time; the same approach gives an $O(n^{2d-1})$ time algorithm, "Kadane's algorithm", for the d dimensional version for all d. Tamaki et al. [46] and Takaoka [45] showed how to use divide-and-conquer to efficiently reduce the 2D Maximum Subarray problem on an $n \times n$ grid to the Min-Plus product of two $n \times n$ matrices. Using the fastest APSP algorithm to date by Williams [52], one obtains the fastest 2D Maximum Subarray algorithm to date, running in $n^3/2^{\Theta(\sqrt{\log n})}$ time. This algorithm can be used to give the fastest known running time $n^{2d-1}/2^{\Theta(\sqrt{\log n})}$ for the d-dimensional version of the problem as well.

In recent years, fine-grained complexity has yielded conditional lower bounds for Max-

imum Subarray. Backurs et al. [7] and Vassilevska W. and Williams [49] showed that an $O(n^{3-\varepsilon})$ time algorithm for 2D Maximum Subarray for $\varepsilon > 0$ would imply an $O(n^{3-\varepsilon'})$ time algorithm for Min-Plus product (and hence APSP), for $\varepsilon' > 0$. Together with the reductions of [46, 45], this implies that the 2D Maximum Subarray problem is subcubically equivalent to APSP. One of the main hardness hypotheses of fine-grained complexity is that APSP requires $n^{3-o(1)}$ time in graphs with integer weights (in the word RAM model with $O(\log n)$ bit words). Under this hypothesis, the best known algorithms for 2D Maximum Subarray are essentially optimal, up to $n^{o(1)}$ factors, for arbitrary integer matrices.

An intriguing question is whether the 2D Maximum Subarray problem can be solved in truly subcubic, $O(n^{3-\varepsilon})$ time for $\varepsilon > 0$ when the entries of the input matrix are small integers in absolute value. Such an algorithm would be very interesting in practice, as the matrix values often represent such small discrete values.

Due to the equivalence between Min-Plus Product and Maximum Subarray and since Min-Plus Product can be solved in truly subcubic time when the matrix entries are small integers, it stands to reason that a truly subcubic algorithm might exist for the small entry Maximum Subarray problem as well. Unfortunately, the known reductions from Maximum Subarray to Min-Plus Product blow up the matrix entries, so that even if the maximum subarray entries are in $\{-1, 0, 1\}$, the resulting matrices whose Min-Plus product we want to compute might have entries that are quadratic in n. Thus, one cannot simply use the known faster algorithms for small entry Min-Plus product to speed-up the Maximum Subarray problem with small entries. On the lower bound end, there doesn't seem to be a way to take an instance of Min-Plus product with arbitrarily large entries and to create a Maximum Subarray instance with small entries. Thus, there is no obvious way to show that the small entry case is hard.

We show that Theorem 1.2.1 can be used to obtain a truly subcubic algorithm for 2D Maximum Subarray with bounded entries.

Examining Tamaki et al. and Takaoka's reduction of Maximum Subarray to Min-Plus Product, it can be seen that starting with a Maximum Subarray instance with entries in

$\{-M, \ldots, M\}$, one obtains $n \times n$ matrices A and B that are BDD as described below:

$$\forall X \in \{A, B\}, \forall i, j \in [n-1],\ |X[i,j] + X[i+1, j+1] - X[i, j+1] - X[i+1, j]| \leq M.$$

We show that BDD Matrix Min-Plus Product is a special case of Theorem 1.2.1, and thus we immediately obtain a truly subcubic time algorithm for Maximum Subarray for matrices with entries bounded in absolute value by $O(n^{0.62})$.

Conditional lower bounds for d-Dimensional Maximum Subarray. Backurs et al. [7] showed that the d-Dimensional Maximum Subarray problem requires $n^{3d/2-o(1)}$ time (in the word-RAM model of computation) under the following popular hardness assumption (see e.g. [48]):

Hypothesis 1 (Max-Weight k-Clique Hypothesis)**.** *In the word-RAM model with $O(\log n)$ bit words, there is no $O(n^{k-\varepsilon})$ time algorithm for $\varepsilon > 0$ that can find a k-Clique of maximum weight in a given n-node graph with edge weights in $\{-n^{ck}, \ldots, n^{ck}\}$ for large enough constant c.*

The fastest known algorithm for d-Dimensional Maximum Subarray runs in $n^{2d-1-o(1)}$ time which is much higher than the Backurs et al. [7] conditional lower bound. A natural question is thus, is there a faster algorithm for $d > 2$, or can the conditional lower bounds be improved?

Our first hardness result is an improvement of the lower bound of Backurs et al., showing that Kadane's algorithm for d-Dimensional Maximum Subarray is conditionally tight:

Theorem 4.1.1. *Under the Max-Weight k-Clique Hypothesis, in the word-RAM model with $O(\log n)$ bit words, the d-Dimensional Maximum Subarray problem requires $n^{2d-1-o(1)}$ time.*

We were able to show that the 2D Maximum Subarray problem can be solved faster when the matrix entries are bounded. One might wonder whether such an improvement is possible for $d > 2$ as well? The simple reduction from d-Dimensional Maximum Subarray to 2-Dimensional Maximum Subarray, unfortunately blows up the entries, and one cannot

use the subcubic algorithm that we developed in a straightforward way. While an improvement is still possible for larger d, we show under a popular hardness assumption that at best one would be able to save a factor of $n^{1+o(1)}$ over the runtime of Kadane's algorithm.

The hardness assumption we use is the ℓ-Uniform Hyperclique assumption used in prior works (see e.g. [39, 1]):

Hypothesis 2 (ℓ-Uniform k-Hyperclique Hypothesis)**.** *Let $k > \ell \geq 3$ be integers. In the word-RAM model with $O(\log n)$ bit words, there is no $O(n^{k-\varepsilon})$ time algorithm for $\varepsilon > 0$ that can find a hyperclique on k nodes in a given n-node ℓ-uniform hypergraph.*

The hypothesis is very believable for a variety of reasons. It is known (see [39]) that the natural extension of the techniques used to solve k-clique (in graphs) will not solve k-hyperclique in ℓ-uniform hypergraphs faster than n^k. Moreover, there are known reductions from notoriously difficult problems such as Exact Weight k-Clique (a problem harder than Max Weight k-Clique) [1], Max ℓ-SAT and even harder Constrained Satisfaction Problems (CSPs) [51, 39] to k-hyperclique in ℓ-uniform hypergraphs so that if the hypothesis is false, then all of these problems have surprisingly improved algorithms.

We prove:

Theorem 4.1.2. *Fix any $d \geq 3$. Under the 3-Uniform $(2d-2)$-Hyperclique Hypothesis, in the word-RAM model with $O(\log n)$ bit words, the d-Dimensional Maximum Subarray problem on matrices with entries in $\{-2^{O(d)}, \ldots, 2^{O(d)}\}$ requires $n^{2d-2-o(1)}$ time.*

That is, for any constant d, solving the problem in matrices with entries bounded by a constant is $n^{2d-2-o(1)}$-hard.

4.2 Algorithm

In [46], Tamaki and Tokuyama reduced 2D Maximum Subarray problem to Min-Plus product of two matrices A, B, using a divide-and-conquer approach. In this reduction, if the absolute values of the entries of the input array are bounded by M, then the matrix A has the property that

$$\forall i, j, |A_{i+1,j+1} - A_{i,j+1} - A_{i+1,j} + A_{i,j}| \leq M.$$

The same property holds for B as well. If we can compute Min-Plus product of matrices with this property in sub-cubic time, then we can solve the maximum subarray problem with bounded entry in sub-cubic time as well.

Motivated by this application, we define the following notion of finite difference operator.

Definition 4.2.1. *The finite difference operator $\mathcal{D}$ acts on a matrix such that*

$$(\mathcal{D}A)_{i,j} = A_{i+1,j+1} - A_{i,j+1} - A_{i+1,j} + A_{i,j}.$$

Using this definition, we can rephrase the property of matrices related with the maximum subarray problem as $|(\mathcal{D}A)_{i,j}| \leq M$.

In the rest of this section, we will show how to compute $A \star B$ in sub-cubic time when $|(\mathcal{D}^t B)_{i,j}| \leq M$ for some constant t. The following lemma shows that matrices with bounded entries after the operator $\mathcal{D}^t$ can be approximated with a low rank matrix.

Lemma 4.2.1. *For an arbitrary matrix B where $|(\mathcal{D}^t B)_{i,j}| \leq M$, there exist $2n$ integer vectors of $(2t)$-dimension $X(1), X(2), \ldots, X(n)$ and $Y(1), Y(2), \ldots, Y(n)$, such that $\left|B_{i,j} - X(i)^T Y(j)\right| = O(Mn^{2t})$.*

Proof. We prove this by induction on t. When $t = 0$, the claim is trivially true.

When $t > 0$, assume the claim is true for $t-1$. Let $A = \mathcal{D}B$. Since $\mathcal{D}^{t-1}A = \mathcal{D}^t B$, by induction, there exists $(2t-2)$-dimension vectors $P(i), Q(j)$ such that $\left|A_{i,j} - P(i)^T Q(j)\right| = O(Mn^{2t-2})$. Define $E_{i,j} = A_{i,j} - P(i)^T Q(j)$ to be the error term, whose absolute value is

bounded by $O(Mn^{2t-2})$. Since $A = \mathcal{D}B$,

$$\begin{aligned}
B_{i,j} &= \left(\sum_{a=1}^{i-1}\sum_{b=1}^{j-1} A_{a,b}\right) - B_{1,1} + B_{i,1} + B_{1,j} \\
&= \left(\sum_{a=1}^{i-1}\sum_{b=1}^{j-1} \left(P(a)^T Q(b) + E_{a,b}\right)\right) - B_{1,1} + B_{i,1} + B_{1,j} \\
&= \left(\sum_{a=1}^{i-1} P(a)\right)^T \left(\sum_{b=1}^{j-1} Q(b)\right) - B_{1,1} + B_{i,1} + B_{1,j} + \left(\sum_{a=1}^{i-1}\sum_{b=1}^{j-1} E_{a,b}\right) \\
&= \begin{bmatrix} 1 \\ -B_{1,1} + B_{i,1} \\ \sum_{a=1}^{i-1} P(a) \end{bmatrix}^T \begin{bmatrix} B_{1,j} \\ 1 \\ \sum_{b=1}^{j-1} Q(b) \end{bmatrix} + \left(\sum_{a=1}^{i-1}\sum_{b=1}^{j-1} E_{a,b}\right)
\end{aligned}$$

Therefore, if we set

$$X(i) := \begin{bmatrix} 1 \\ -B_{1,1} + B_{i,1} \\ \sum_{a=1}^{i-1} P(a) \end{bmatrix}, \text{ and } Y(j) := \begin{bmatrix} B_{1,j} \\ 1 \\ \sum_{a=1}^{i-1} Q(a) \end{bmatrix},$$

we will have

$$\left|B_{i,j} - X(i)^T Y(j)\right| = \left|\sum_{a=1}^{i-1}\sum_{b=1}^{j-1} E_{a,b}\right| = O(Mn^{2t}),$$

which completes the induction. □

Theorem 4.2.1. *For two integer matrices A and B, if $|(\mathcal{D}^t B)_{i,j}| \leq M$ for some constant $t \geq 1$, then there exists an algorithm that computes $A \star B$ in $\tilde{O}(n^{3-\frac{3-\omega}{2t^2+4}} M^{1/(2t^2+4)})$ time.*

Proof. Let Δ be a small polynomial in n. For any $\Delta \times \Delta$ sub-matrix of B, the t-th discrete difference is also bounded by M. Therefore, by Lemma 4.2.1, for each i', j' multiples of Δ, there exist $2t$-dimensional vectors $X_{i',j'}(i), Y_{i',j'}(j)$ such that $X_{i',j'}(i)^T Y_{i',j'}(j)$ is an $O(M\Delta^{2t})$-additive approximation of $B_{i,j}$. In other word, every $\Delta \times \Delta$ sub-matrices of B has an $O(M\Delta^{2t})$-approximate rank at most $2t$. Therefore, we can apply Theorem 1.2.1 to

get an algorithm that computes $A \star B$ in time

$$\tilde{O}(n^3 \cdot \Delta^{-1/\lfloor (2t+1)/2 \rfloor} + n^{(9+\omega)/4} \cdot (M\Delta^{2t})^{1/4}).$$

By setting $\Delta = \left(n^{(3-\omega)/2} \cdot M^{-1/2}\right)^{\frac{t}{t^2+2}}$, we get a $\tilde{O}(n^{3-\frac{3-\omega}{2t^2+4}} M^{1/(2t^2+4)})$ time algorithm. □

Corollary 4.2.1. *Given an $n \times n$ array A, where the absolute value of each entry is bounded by M. There exists an algorithm that finds the maximum subarray of A in $\tilde{O}(n^{\frac{15+\omega}{6}} M^{1/6})$ time. Use $\omega < 2.373$, this gives an $\tilde{O}(n^{2.8955} M^{1/6})$ time algorithm, which is truly subcubic when $M = o(n^{0.627})$.*

Proof. We can use Tamaki and Tokuyama's reduction in [46], and apply Theorem 4.2.1 using $t = 1$ to immediately get this result. □

4.3 Lower Bounds

4.3.1 Tight Lower Bound for d-Dimensional Maximum Subarray

In this section, we show the conditional lower bound for the d-Dimensional Maximum Subarray problem, where the entries of the input array can have arbitrary real values. Backurs et al. [7] showed an $n^{d+\lfloor d/2 \rfloor - o(1)}$ conditional lower bound for d-Dimensional Maximum Subarray, based on the hardness of the Max-Weight $(d + \lfloor d/2 \rfloor)$-Clique problem. Their lower bound is only tight for $d = 2$, since Kadane's algorithm for d-Dimensional Maximum Subarray runs in $O(n^{2d-1})$ time.

We show an $n^{2d-1-o(1)}$ conditional lower bound for the d-Dimensional Maximum Subarray problem, based on the hardness of the Max-Weight $(2d-1)$-Clique problem. In our reduction, we will introduce two intermediate problems defined as following.

Definition 4.3.1 (Two-sided d-Uniform Hypergraph)**.** *A complete hyperedge-weighted d-uniform hypergraph whose vertex set is partitioned into $2d$ sets $U_1, U_2, \ldots, U_d, V_1, V_2, \ldots, V_d$, each with n vertices is* two-sided *if any d-hyperedge $(w_1, \ldots, w_d)$ not in the form of $w_1 \in U_1 \cup V_1, w_2 \in U_2 \cup V_2, \ldots, w_d \in U_d \cup V_d$, has zero weight.*

Definition 4.3.2 (Two-sided d-Uniform Max-Weight Hyperclique)**.** *Given a two-sided d-uniform hypergraph, find one vertex from each vertex set, so that the sum of hyperedge weights between these vertices is maximized.*

Definition 4.3.3 (Central d-Dimensional Array)**.** *A d-dimensional array A with side length $2n+1$ is called a central array if the index set of it is $\{-n, -n+1, \ldots, n-1, n\}^d$.*

Definition 4.3.4 (Central Maximum Subarray Sum)**.** *Given a central d-dimensional array A, find*

$$\max_{\substack{i\in[n]^d \\ \delta\in[2n]^d \\ -n\le i-\delta<0}} \sum_{j\in\{0,1\}^d} A[i-\delta\odot j],$$

where $\odot$ denotes the componentwise product of two vectors.

Central Maximum Subarray Sum asks to find a subarray whose 2^d corners are in each of the 2^d quadrants, such that the sum of values on its corners is maximized. Backurs et al. [7] showed an $O(n^d)$ time reduction from the Central Maximum Subarray Sum problem to the Maximum Subarray problem in d-dimension. Thus, any (higher than n^d) lower bound for the Central Maximum Subarray Sum problem would imply the same lower bound for the Maximum Subarray problem. In the rest of this section, we will first show a reduction from Max-Weight $(2d-1)$-Clique problem to Two-sided d-Uniform Max-Weight Hyperclique problem, and then show a reduction from the Two-sided d-Uniform Max-Weight Hyperclique problem to the Central Maximum Subarray Sum problem. If the well-known Max-Weight $(2d-1)$-Clique Hypothesis is true, the Central Maximum Subarray Sum problem would have an $n^{2d-1-o(1)}$ lower bound, and thus the Maximum Subarray problem would share the $n^{2d-1-o(1)}$ lower bound due to Backurs et al.'s reduction.

Lemma 4.3.1. *If there exists an $O(n^{2d-1-\epsilon})$ time algorithm (for $\epsilon > 0$) for the Two-sided d-Uniform Max-Weight Hyperclique problem, then there exists an $O(n^{2d-1-\epsilon})$ time algorithm for Max-Weight $(2d-1)$-Clique problem.*

Proof. Let $G = (V_1 \cup V_2 \cup \cdots \cup V_{2d-1}, E)$ be a $(2d-1)$-partite graph. We will construct a Two-sided d-Uniform Hypergraph $G' = (U_1 \cup U_2 \cup \cdots \cup U_{2d}, E')$ such that the maximum $(2d-1)$-clique weight of G is equal to the maximum $(2d)$-hyperclique weight of G'. For

simplicity, assume n is a power of 2, and we will index the vertices in each vertex set from 0.

The first $2d-1$ vertex sets of G' are copies of the vertex sets of G. Specifically, U_i is a copy of V_i for any $i \leq 2d-1$. U_{2d}, however, encodes something different. Assume we pick $v_i \in V_i$ to be the s_i-th vertex in V_i, then intuitively, U_{2d} encodes $s_{d+1} \oplus s_{d+2} \oplus \cdots \oplus s_{2d-1}$, where $\oplus$ is the bitwise exclusive-or operation.

We initialize all hyperedge weights of G' to 0, and increase these weights incrementally by considering edges of G one by one.

For any $1 \leq i < j \leq 2d-1$, pick an edge $(v_i, v_j) \in V_i \times V_j$, with weight $w(v_i, v_j)$. Let u_i, u_j be the copies of v_i, v_j in the hypergraph G'. First consider the case when $j \neq i+d$. This is the case when there exist arbitrarily weighted hyperedges that contain both u_i and u_j. Let $S := \{k \in [d] : k \not\equiv i \pmod{d} \text{ and } k \not\equiv j \pmod{d}\}$. We enumerate every n^{d-2} combinations of vertices $u'_k \in U_k$ for $k \in S$, and add $w(v_i, v_j)$ to the hyperedge between the d vertices u_i, u_j and u'_k where $k \in S$.

The case when $j = i+d$ is more interesting, since all hyperedges in G' that contain both u_i and u_j must have zero weight, because of the definition of Two-sided d-Uniform Hypergraph. However, we can encode $w(v_i, v_j)$ via the extra vertex set U_{2d}. Let u_j be the s_j-th vertex in U_j. We enumerate all n^{d-2} combinations of indices $s'_{d+1}, s'_{d+2}, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_{2d}$, such that $s'_{d+1} \oplus s'_{d+2} \oplus \cdots \oplus s'_{j-1} \oplus s_j \oplus s'_{j+1} \oplus \cdots \oplus s'_{2d-1} = s'_{2d}$. Let the s'_k-th vertex in U_k be u'_k for any $k \in \{d+1, d+2, \ldots, j-1, j+1, \ldots, 2d\}$. We add $w(v_i, v_j)$ to the hyperedge that consists of u_i and u'_k for every $k \in \{d+1, d+2, \ldots, j-1, j+1, \ldots, 2d\}$.

Finally, enumerate all combinations of $s_{d+1}, s_{d+2}, \ldots, s_{2d}$ such that $s_{d+1} \oplus s_{d+2} \oplus \cdots s_{2d-1} \neq s_{2d}$. Let u_k be the s_k-th vertex in U_k, for every $d+1 \leq k \leq 2d$. We set the weight of the hyperedge that consists of $u_{d+1}, u_{d+2}, \ldots, u_{2d}$ to $-M'$ for some large enough M'. If all edge weights in G are numbers in $[-M, M]$, we can set M' to be $100d^{10}M$.

The construction of G' takes $O(n^d)$ time, since for each edge (v_i, v_j) in G, we enumerate $O(n^{d-2})$ hyperedges. It remains to show that the maximum weight of $(2d-1)$-cliques in G is equal to the maximum $(2d)$-hyperclique weight of G'.

Pick any $2d$ indices $s_1, s_2, \ldots, s_{2d}$. Let u_i be the s_i-th vertex in U_i. If $s_{d+1} \oplus s_{d+2} \oplus \cdots \oplus s_{2d-1} \neq s_{2d}$, then there will be a $-M'$ weight on the hyperedge $(u_{d+1}, u_{d+2}, \ldots, u_{2d})$,

so the weight of the hyperclique $u_1, u_2, \ldots, u_{2d}$ can never be maximum. Therefore, we are forced to pick $s_{2d} = s_{d+1} \oplus s_{d+2} \oplus \cdots \oplus s_{2d-1}$. In this case, the weight of the hyperclique $(u_1, u_2, \ldots, u_{2d})$ is equal to the weight of the clique $(v_1, v_2, \ldots, v_{2d-1})$, where v_i is a copy of u_i for each $i < 2d$. Thus, if we invoke the $O(n^{2d-1-\epsilon})$ algorithm for the Two-sided d-Uniform Max-Weight Hyperclique problem on G', we get the Max-Weight $(2d-1)$-Clique on G. □

Lemma 4.3.2. *If there exists an $O(n^{2d-1-\epsilon})$ time algorithm for the d-Dimensional Central Maximum Subarray Sum problem, then there exists an $O(n^{2d-1-\epsilon})$ time algorithm for the Two-sided d-Uniform Max-Weight Hyperclique problem.*

Proof. Take any Two-sided d-Uniform Hypergraph $G = (V_1 \cup V_2 \cdots V_d \cup U_1 \cup U_2 \cdots \cup U_d, E)$, we index the vertices in V_i and U_i from 1. We will construct a d-Dimensional Central Array A based on G such that the central maximum subarray sum of A is equal to the maximum $2d$-hyperclique of G. If any entry of vector $i \in \{-n, \ldots, n\}^d$ is 0, we set $A[i]$ to be 0, since they are not relevant to the central maximum subarray sum of A. For any other index i, we choose d vertices $w_1, w_2, \ldots, w_d$ from the graph G based on i. If $i_r > 0$ for some r, we choose w_r to be the i_r-th vertex in V_r; otherwise, we choose w_r to be the $(-i_r)$-th vertex in U_r. We set $A[r]$ to be the weight of the hyperedge connecting $w_1, w_2, \ldots, w_d$.

Pick any $2d$ vertices $v_1 \in V_1, v_2 \in V_2, \ldots, v_d \in V_d, u_1 \in U_1, u_2 \in U_2 \ldots, u_d \in U_d$. Define two d-dimensional vectors $\vec{v}$ and $\vec{u}$, such that $\vec{v}_i$ is the index of v_i in V_i, and $\vec{u}_i$ is the index of u_i in V_i. For any $j \in \{0, 1\}^d$, let i be a d-dimensional vector such that $i_r = \vec{v}_r$ if $j_r = 0$, and $i_r = -\vec{u}_r$ if $j_r = 1$. Also, let W be a set of d vertices $\bigcup_{1 \le r \le d}\{\text{if } j_r = 0 \text{ then } v_r \text{ else } u_r\}$. The entry $A[i]$ is exactly the weight of the hyperedge between vertices in W. Thus, the sum of all corners of the subarray whose two opposite corners are $\vec{v}$ and $-\vec{u}$ is equal to the weight of the hyperclique $(v_1, v_2, \ldots, v_d, u_1, u_2, \ldots u_d)$.

Conversely, for any subarray of A whose 2^d corners are in different quadrants, there exists a hyperclique in G whose $2d$ vertices are from different vertex sets, by a similar argument.

Thus, the central maximum subarray sum of A is equal to the max-weight $2d$-hyperclique

of G, so we can invoke the $O(n^{2d-1-\epsilon})$ algorithm of d-Dimensional Central Maximum Subarray Sum problem to output the max-weight $2d$-hyperclique of G. □

Lemma 4.3.1, Lemma 4.3.2, together with the reduction from Central Maximum Subarray Sum to Maximum Subarray [7] imply Theorem 4.1.1.

4.3.2 Lower Bound for Maximum Subarray with Bounded Weight

In Section 4.3.1, we showed a tight conditional lower bound for d-Dimensional Maximum Subarray with real valued weights. In Section 4.2, we also showed an algorithm that is better than this conditional lower bound, for 2D Maximum Subarray with bounded integer weights. A natural question arises: Can we prove some conditional lower bound for d-Dimensional Maximum Subarray when the numbers in the array are bounded integers?

In this section, we answer this question positively by proving Theorem 4.1.2. We notice that the reduction from Two-sided d-Uniform Max-Weight Hyperclique problem to Central Maximum Subarray Sum (Lemma 4.3.2), and the reduction from Central Maximum Subarray Sum to Maximum Subarray (presented in [7]) only increase the largest absolute value of weights by a constant factor. Therefore, we only need to show a conditional lower bound for Two-sided d-Uniform Max-Weight Hyperclique when the weights of the hyperedges are bounded integers. Therefore, Theorem 4.1.2 follows from the following lemma.

Lemma 4.3.3. *If there exists an $O(n^{2d-2-\epsilon})$ algorithm (for $\epsilon > 0$) for the Two-sided d-Uniform Max-Weight Hyperclique problem where the hyperedges have bounded integer weights, then there exists an $O(n^{2d-2-\epsilon})$ algorithm for the 3-Uniform $(2d-2)$-Hyperclique problem.*

Proof. The proof has a similar spirit as the proof to Lemma 4.3.1. For simplicity, we denote a 3-Uniform Hypergraph G as $(V_1 \cup V_2 \cup \ldots \cup V_{d-1} \cup U_1 \cup U_2 \cup \cdots \cup U_{d-1}, E)$. Note that even though the vertex sets of G are not partitioned to two parts naturally, we used V_i for one half and U_i for the other half. Also assume n is a power of 2 for simplicity.

Create a two-sided d-uniform hypergraph $G' = (V_1' \cup V_2' \cup \cdots \cup V_d' \cup U_1' \cup U_2' \cup \cdots \cup U_d', E')$, where V_i' is a copy of V_i for any $i \leq d-1$, and U_i' is a copy of U_i for any $i \leq d-1$. If we pick the s_i-th vertex v_i' from V_i', then V_d' encodes the information $s_1 \oplus s_2 \oplus \cdots \oplus s_{d-1}$. Similarly,

if we pick the t_i-th vertex u'_i from U'_i, then U'_d encodes the information $t_1 \oplus t_2 \oplus \cdots \oplus t_{d-1}$. We call V_i and U_i corresponding vertex sets; we also call V'_i and U'_i corresponding vertex sets. For any vertex set S, we use $S[k]$ to denote the k-th vertex in S, indexed from 0. We initialize all edge weights of G' to 0.

Every 3-hyperedge $(a, b, c) \in E$ will increase the weight of some hyperedges in G' by 1. First assume no two vertices in $\{a, b, c\}$ are from a pair of corresponding vertex sets. Let a', b', c' be a, b, c's copies in G', respectively. Take any hyperedege e' in G' that contains $\{a', b', c'\}$ and $d - 3$ other vertices from the first half of the vertex sets, so that every pair of corresponding vertex sets contains exactly one vertex. We increment the weight of any such hyperedge e' by 1.

If two vertices in $\{a, b, c\}$ are from a pair of corresponding vertex sets, then without loss of generality, we can assume $a \in V_i, b \in U_i$. When $c \in V_j$ for some $j \neq i$, we increment all hyperedges consisting of vertices $V'_1[s_1], \ldots, V'_{i-1}[s_{i-1}], U'_i[t_i], V'_{i+1}[s_{i+1}], \ldots, V'_d[s_d]$, where $U_i[t_i] = b$, $V_j[s_j] = c$ and $V_i\left[\bigoplus_{1\leq k\leq d, k\neq i} s_k\right] = a$. It is symmetric when $c \in U_j$ for some $j \neq i$: we increment all hyperedges consisting of vertices

$$U'_1[t_1], \ldots, U'_{i-1}[t_{i-1}], V'_i[s_i], U'_{i+1}[t_{i+1}], \ldots, U'_d[t_d],$$

where $V_i[s_i] = a$, $U_j[t_j] = c$ and $U_i\left[\bigoplus_{1\leq k\leq d, k\neq i} t_k\right] = b$.

Finally, for any $s_1, s_2, \ldots, s_d$ such that $s_1 \oplus s_2 \oplus \cdots \oplus s_{d-1} \neq s_d$, we set the weight of the edge consisting of $V'_1[s_1], V'_2[s_2], \ldots, V'_d[s_d]$ to be $-M$ for $M = 100d^{10}$. Symmetrically, for any $t_1, t_2, \ldots, t_d$ such that $t_1 \oplus t_2 \oplus \cdots \oplus t_{d-1} \neq t_d$, we set the weight of the edge consisting of $U'_1[t_1], U'_2[t_2], \ldots, U'_d[t_d]$ to be $-M$.

The maximum absolute value of hyperedge weight of G' is M, which is a constant when d is a constant. By construction, G has a $(2d-2)$-hyperclique if and only if the max-weight hyperclique of G' has weight $\binom{2d-2}{3}$. Thus, we can solve 3-Uniform $(2d-2)$-Hyperclique by invoking the assumed algorithm for the Two-sided d-Uniform Max-Weight Hyperclique problem.

□

Chapter 5

Range Mode

5.1 Introduction

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 range mode queries in an online fashion. The mode of a sequence of elements is one of the most basic data statistics, along with the median and the mean. It is frequently computed in data mining, information retrieval, and data analytics.

The range mode problem seeks to answer multiple queries on distinct intervals of the data sequence without having to recompute each answer from scratch. Its study in the data structure community has shown that the mode is a much more challenging data statistic to maintain than other natural range queries: while range sum, min or max, median, and majority all support linear space dynamic data structures with poly-logarithmic or better time per operation [42, 15, 34, 27, 22], the current fastest dynamic range mode data structure previously requires a stubborn $O(n^{2/3})$ time per operation [21]. Indeed, range mode is one of few remaining classical range queries to which our currently known algorithms may be far from optimal. As originally stated by Brodal et al. [12] and mentioned by Chan et al. [14] in 2011 and 2014, respectively, "The problem of finding the most frequent element within a given array range is still rather open."

The current best conditional lower bound, by Chan et al. [14], reduces multiplication of two $\sqrt{n} \times \sqrt{n}$ Boolean matrices to n range mode queries on a fixed array of size $O(n)$. This indicates that if the current algorithm for Boolean matrix multiplication is optimal, then answering n range mode queries on an array of size $O(n)$ cannot be performed in time better than $O(n^{3/2-\epsilon})$ time for $\epsilon > 0$ with combinatorial techniques, or $O(n^{\omega/2-\epsilon})$ time for $\epsilon > 0$ in general. This reduction can be strengthened for dynamic range mode by reducing from the Online Matrix-Vector Multiplication problem [35]. Using $O(n)$ dynamic range mode operations on a sequence of length $O(n)$, we can multiply a $\sqrt{n} \times \sqrt{n}$ Boolean matrix with $\sqrt{n}$ Boolean vectors given one at a time. This indicates that a dynamic range mode data structure taking $O(n^{1/2-\epsilon})$ time per operation for $\epsilon > 0$ is not possible with current knowledge.

Previous attempts indicate the higher $O(n^{2/3})$ per operation cost as the bound to beat [14, 21] for Dynamic Range Mode. Indeed, $\tilde{O}(n^{2/3})$ time per operation can be achieved with a variety of techniques, but crossing the $O(n^{2/3})$ barrier appears much harder.

As an application of Theorem 1.2.2 we obtain an $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}}) = O(n^{1.481})$ time algorithm for Batch Range Mode, breaking the $n^{1.5-o(1)}$ combinatorial lower bound.

Theorem 5.1.1. *The Batch Range Mode problem can be solved deterministically in time* $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$.

We also break the $O(n^{2/3})$ time per operation barrier for Dynamic Range Mode. We achieve this by defining the following new type of data structure problem on the Min-Plus Product that can be applied to Dynamic Range Mode, which may be of independent interest. Then we combine this data structure problem with our Monotone Min-Plus Product algorithm to achieve faster Dynamic Range Mode.

Problem 5.1.2 (*Min-Plus-Query* problem)**.** *During initialization, we are given two matrices* A, B*. For each query, we are given three parameters* i, j, S*, where* i, j *are two integers, and* S *is a set of integers. The query asks* $\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.

Our performance theorem is the following.

Theorem 5.1.3. *The Dynamic Range Mode problem can be solved deterministically in* $\tilde{O}(n^{\frac{\omega+9}{\omega+15}})$ *worst-case time per operation/query with* $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$ *space.*

We can show that finding the range mode is equivalent to finding the frequency of the range mode, up to logarithmic factors. Given the mode itself, we can query its frequency via binary search trees in $O(\log n)$ time. Suppose we have an algorithm for finding the frequency of the mode, then for query $[l, r]$, we ask the frequency of the mode in range $[l, (l+r)/2]$. If it is the same, we repeat the search with right endpoint in range $[(l+r)/2, r]$; if it is not, we repeat the search with right endpoint in range $[l, (l + r)/2]$. Using this method, we can binary search until we determine when the frequency of the mode changes, thus finding the element that is mode in an additional $O(\log n)$ queries. Therefore, in our algorithms, we will focus on finding the frequency of the range modes.

5.1.1 Related Works

The range mode problem was first studied formally by Krizanc et al. [37]. They study space-efficient data structures for static range mode, achieving a time-space tradeoff of $O(n^{2-2\epsilon})$ space and $O(n^{\epsilon} \log n)$ query time for any $0 < \epsilon \leq 1/2$. They also give a solution occupying $O(n^2 \log \log n / \log n)$ space with $O(1)$ time per query.

Chan et al. [14] also study static range mode, focusing on linear space solutions. They achieve a linear space data structure supporting queries in $O(\sqrt{n})$ time via clever use of arrays, which can be improved to $O(\sqrt{n/\log n})$ time via bit-packing tricks. Their paper also introduces the conditional lower bound which reduces multiplication of two $\sqrt{n} \times \sqrt{n}$ boolean matrices to n range mode queries on an array of size $O(n)$. As mentioned, combined with the presumed hardness of the Online Matrix-Vector problem [35], this result indicates that a Dynamic Range Mode data structure must take greater than $O(n^{1/2-\epsilon})$ for $\epsilon > 0$ time per operation. Finally, Chan et al. [14] also give the first data structure for Dynamic Range Mode. At linear space, their solution achieves $O(n^{3/4} \log n / \log \log n)$ worst-case time per query and $O(n^{3/4} \log \log n)$ amortized expected time update, and at $O(n^{4/3})$ space, their solution achieves $O(n^{2/3} \log n / \log \log n)$ worst-case time query and amortized expected update time.

Recently, Hicham et al. [21] improved the runtime of Dynamic Range Mode to worst-case $O(n^{2/3})$ time per operation while simultaneously improving the space usage to linear.

A cell-probe lower bound for static range mode has been devised by Greve et al. [33].

Their result states that any range mode data structure that uses S memory cells of w-bit words needs $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a query.

Both static and dynamic range mode have been studied in approximate settings [10, 33, 20].

5.2 Algorithm for Batch Range Mode

In this section, as an application of Algorithm 1.2.2, we give an $O(n^{1.5-\varepsilon})$ time algorithm for the Batch Range Mode problem for some $\varepsilon > 0$. Recall the following theorem:

Theorem 5.1.1. *The Batch Range Mode problem can be solved deterministically in time* $\tilde{O}(n^{\frac{21+2\omega}{15+\omega}})$.

Proof. Let the array be a and let the n queries be $[l_i, r_i]$ for $1 \leq i \leq n$. Without loss of generality, we assume $l_i \leq n/2 < r_i$. Otherwise, we can use a divide-and-conquer approach to first compute the queries that satisfy $l_i \leq n/2 < r_i$, then recurse on the two halves $[1, n/2]$ and $(n/2, n]$ to compute answers. Since the proposed time complexity is $\Omega(n^{1+\varepsilon})$ for some $\varepsilon > 0$, the total time complexity does not change by the Master Theorem.

Let t be a parameter of the algorithm that controls the block size as well as a threshold frequency for frequent elements and infrequent elements. We handle elements that appear at most n^t times (infrequent elements), and elements that appear more than n^t times (frequent elements) differently.

Fix some infrequent elements x. For any $a_j = a_k = x$ where $j \leq k$, we create an interval $[j, k]$, whose weight is the number of occurrence of x in the range $[j, k]$. Since x occurs at most n^t times, the number of of such intervals is at most $O(n^{1+t})$. To query the largest frequency in a range $[l_i, r_i]$, it is equivalent to ask the largest weight of intervals that are contained in the interval $[l_i, r_i]$. This problem can be solved by, for instance, using a persistent balanced search tree, in $\tilde{O}(n^{1+t})$ preprocessing time and $\tilde{O}(1)$ query time.

Now consider the "frequent" elements in the array that occur more than n^t times. There are at most n^{1-t} distinct frequent elements in the array. For each of these elements x, we create a balanced binary search tree B_x, whose elements are the set of occurrences $\{i \in [n] : a_i = x\}$, augmented with the size of the subtree rooted at each node. We split the

whole sequence $a_1, \ldots, a_n$ into consecutive blocks of size $O(n^t)$, so that $n/2$ is the right boundary of one block and the left boundary of the next block.

For a range $[l_i, r_i]$, let $S_u, S_{u+1}, \ldots, S_v$ be the maximum set of blocks in this range, then the range mode of $[l_i, r_i]$ is either the range mode of the subinterval $S_u, S_{u+1}, \ldots, S_v$, or some elements in $[l_i, r_i] \setminus \{S_u, S_{u+1}, \ldots, S_v\}$.

Suppose that the range mode of $[l_i, r_i]$ is not the range mode of $S_u, S_{u+1}, \ldots, S_v$. Then, we have a candidate list of $O(n^t)$ numbers (those to the left and right of $S_u, S_{u+1}, \ldots, S_v$ in $[l_i, r_i]$) that can possibly be the range mode of the interval $[l_i, r_i]$. For each of these numbers x, we can query its occurrence in the range $[l_i, r_i]$ by querying the number of elements between $[l_i, r_i]$ in B_x which takes $O(\log n)$ time due to the augmentation.

Therefore, it takes $\tilde{O}(n^t)$ overhead to compute the range mode of $[l_i, r_i]$ once we know the range mode of $S_u, S_{u+1}, \ldots, S_v$. Thus, we can focus on the sub-problem of computing the range mode of the subinterval $S_u, S_{u+1}, \ldots, S_v$, where S_u is to the left of $n/2$, and S_v is to the right of $n/2$ and some pair of blocks S_{i*}, S_{i*+1} end and start (respectively) at $n/2$. Call these last two the middle blocks.

We create two matrices A and B. The columns of A and rows of B are indexed by the frequent elements. The rows of A and columns of B are indexed by j such that S_j is one of the blocks of size T that we partitioned $a_1, \ldots, a_n$ into. Hence both A and B are $O(n^{1-t})$ by $O(n^{1-t})$ matrices.

More concretely, for each S_u to the left of $n/2$, we create a row u in matrix A, where $A_{u,k}$ is the negated number of occurrences of element k in the subinterval $S_u, \ldots, S_{i*}$ (recall that S_{i*} ends at $n/2$); for each S_v to the right of $n/2$, we create column t in matrix B where $B_{k,t}$ is the negated number of occurrences of element k in the subinterval $S_{i*+1}, \ldots, S_v$ (recall that S_{i*+1} starts at $n/2 + 1$). Therefore, the negated Min-Plus product entry $-(A \star B)_{u,v}$ will be the frequency of the range mode in the full subinterval $S_u, S_{u+1}, \ldots, S_v$.

Note that A, B are $O(n^{1-t})$ by $O(n^{1-t})$ matrices, each row of B is monotonically non-increasing, and the total range is at most n. Therefore, we can apply Theorem 1.2.2 by setting $\beta = 1$ and $\eta = \frac{t}{1-t}$ to multiple $A \star B$ in $\tilde{O}(n^{\frac{1-t}{5}(9+(2+\omega)\cdot 1+\frac{t}{1-t})}) = \tilde{O}(n^{\frac{1}{5}(11+\omega-10t-\omega t)})$ time.

Therefore, the overall running time of the algorithm is $\tilde{O}(n^{\frac{1}{5}(11+\omega-10t-\omega t)} + n^{1+t})$. By

setting $t = \frac{6+\omega}{15+\omega}$, we get an $\tilde{O}(n^{(21+2\omega)/(15+\omega)})$ time algorithm as claimed. □

5.3 Algorithm for Dynamic Range Mode

Despite the $\tilde{O}(n^{(21+2\omega)/(15+\omega)})$ time algorithm for Batch Range Mode, we cannot directly apply the result to Dynamic Range Mode. The main issue is the element deletion operation. In many Batch Range Mode algorithms, critical points are chosen evenly distributed in the array, and the algorithm precomputes the range modes of intervals between every pair of critical points. Our Batch Range Mode improvement is achieved via a faster precomputation algorithm, which uses a Min-Plus Product algorithm for structured matrices. However, if element deletion is allowed, the results stored in the precomputation will not be applicable. For example, an interval between two critical points could contain x copies of element a, $x-1$ copies of element b, and many other elements with frequencies less than $x-1$. During precomputation, the range mode of this interval would be a. However, if we delete two copies of a, there is no easy way to determine that the mode of this interval has now changed to b.

We overcome this difficulty by introducing the Min-Plus-Query problem. Intuitively, in the Min-Plus-Query problem, a large portion of the work of the Min-Plus product is put off until the query. It also supports more flexible queries. Using the Min-Plus-Query problem as a subroutine, we will be able to query the most frequent element excluding a set S of forbidden elements. For instance, in the preceding example, we would be able to query the most frequent element that is not a.

Problem 5.3.1 (*Min-Plus-Query* problem)**.** *During initialization, we are given two matrices A, B. For each query, we are given three parameters i, j, S, where i, j are two integers, and S is a set of integers. The query asks* $\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.

Sometimes our algorithm is also required to output a witness for each Min-Plus-Query, which is defined as the following problem.

Problem 5.3.2 (*Min-Plus-Query-Witness* problem)**.** *During initialization, we are given two matrices A, B. For each query, we are given three parameters i, j, S, where i, j are two*

integers, and S is a set of integers. We must output an index $k^* \notin S$ such that $A_{i,k^*}+B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.

If A is an $n \times n^\beta$ matrix and B is an $n^\beta \times n$ matrix, then the naive algorithm for Min-Plus-Query just enumerates all possible indices k for each query, which takes $O(n^\beta)$ time per query. In order to get a faster algorithm for dynamic range mode, we need to achieve $\tilde{O}(n^{2+\beta-\epsilon})$ preprocessing time and $\tilde{O}(n^{\beta-\epsilon} + |S|)$ query time, for some $\epsilon > 0$, where A, B are some special matrices generated by the range mode instance. Specifically, matrix B is a monotone matrix with a small total range.

We will show that nontrivial data structures exist for the Min-Plus-Query problem for arbitrary integer matrix A and such input matrix B. Such a data structure will lead to a faster algorithm for Dynamic Range Mode.

In the following lemma, we show a data structure for the Min-Plus-Query problem when both input matrices have integer weights small in absolute value.

Lemma 5.3.1. *Let $\beta \geq 0$ be a constant. Let A and B be two integer matrices of dimension $n \times n^\beta$ and $n^\beta \times n$, respectively, with entries in $\{-M, \ldots, M\} \cup \{\infty\}$ for some $M \geq 0$. Then we can solve the Min-Plus-Query problem of A and B in $\tilde{O}(Mn^{\omega(1,\beta,1)})$ preprocessing time and $\tilde{O}(|S|)$ query time. The space complexity is $\tilde{O}(Mn^2 + n^{1+\beta})$.*

Proof. The algorithm uses the idea by Alon, Galil and Margalit in [6], which computes the Min-Plus product of A, B in $\tilde{O}(Mn^{\omega(1,\beta,1)})$ time.

In their algorithm, they first construct matrix A' defined by

$$A'_{i,k} = \begin{cases} (n^\beta + 1)^{A_{i,k}+M} & \text{if } A_{i,k} \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

We can define B' similarly. Then the product $A'B'$ captures some useful information about the Min-Plus product of A and B. Namely, for each entry $(A'B')_{i,j}$, we can uniquely write it as $\sum_{t \geq 0} r_t^{i,j}(n^\beta+1)^t$ for integers $0 \leq r_t^{i,j} \leq n^\beta$. Note that $r_t^{i,j}$ exactly equals the number of k such that $A_{i,k} + B_{k,j} = t - 2W$. Thus, we can use $A'B'$ to compute the Min-Plus Product of A and B.

In our algorithm, we use a range tree to maintain the sequence $r_t^{i,j}$ for each pair of i, j. The preprocessing takes $\tilde{O}(Mn^{\omega(1,\beta,1)})$ time, which is the time to compute $A'B'$ and the sequences $r_t^{i,j}$.

During each query, we are given i, j, S. We enumerate each $k \in S$, and decrement $r^{i,j}_{A_{i,k}+B_{k,j}+2M}$ in the range tree if $A_{i,j} + B_{k,j} < \infty$. After we do this for every $k \in S$, we query the range tree for the smallest t such that $r_t^{i,j} \neq 0$, so $t - 2M$ is the answer to the Min-Plus-Query query. After each query, we need to restore the values of $r^{i,j}$, which can also be done efficiently. The query time is $\tilde{O}(|S|)$, since each update and each query of range tree takes $\tilde{O}(1)$ time. The space complexity should be clear from the algorithm. □

In the previous lemma, the data structure only answers the Min-Plus-Query problem. In all subsequent lemmas, the data structure will be able to handle the Min-Plus-Query-Witness problem.

In the next lemma, we use Lemma 5.3.1 as a subroutine to show a data structure for the Min-Plus-Query-Witness problem when only matrix A has small integer weights in absolute value.

Lemma 5.3.2. *Let $\beta \geq 0$ be a constant. Let A and B be two integer matrices of dimensions $n \times n^\beta$ and $n^\beta \times n$, respectively, where A has entries in $\{-M, \ldots, M\} \cup \{\infty\}$ for some $M \geq 1$, and B has arbitrary integer entries represented by* polylog n *bit numbers. Then for every integer $1 \leq P \leq n^\beta$, we can solve the Min-Plus-Query-Witness problem of A and B in $O(\frac{n^\beta}{P} Mn^{\omega(1,\beta,1)})$ preprocessing time and $O(|S| + P)$ query time. The space complexity is $\tilde{O}(\frac{Mn^{2+\beta}}{P} + \frac{n^{1+2\beta}}{P})$.*

Proof. For simplicity, assume P is a factor of n^β. We sort each column of matrix B and put entries whose rank is between $(\ell - 1)P + 1$ and ℓP into the ℓ-th bucket. We use $K_{j,\ell}$ to denote the set of row indices of entries in the ℓ-th bucket of the column j. We use $L_{j,\ell}$ to denote the smallest entry value of the bucket $K_{j,\ell}$, and use $H_{j,\ell}$ to denote the largest entry value. Formally,

$$L_{j,\ell} = \min_{k \in K_{j,\ell}} B_{k,j} \quad \text{and} \quad H_{j,\ell} = \max_{k \in K_{j,\ell}} B_{k,j}.$$

For each $\ell \in [n^\beta/P]$, we do the following[1]. We create an $n^\beta \times n$ matrix B^ℓ and

[1]We use $[n]$, with n integer, to denote the set $\{1, 2, \ldots, n\}$.

initialize all its entries to ∞. Then for each column j, if $H_{j,\ell} - L_{j,\ell} \leq 2W$ (we will call it a small bucket), we set $B^{\ell}_{k,j} := B_{k,j} - L_{j,\ell} - W$ for all $k \in K_{j,\ell}$. We will handle the case $H_{j,\ell} - L_{j,\ell} > 2W$ (large bucket) later. Clearly, all entries in B^{ℓ} have values in $\{-W, \ldots, W\} \cup \{\infty\}$, so we can use the algorithm in Lemma 5.3.1 to preprocess A and B^{ℓ} and store the data structure in D^{ℓ}. Also, for each pair (i, j), we create a range tree $\mathcal{T}^{i,j}_{\text{small}}$ on the sequence $(A \star B^1)_{i,j}, (A \star B^2)_{i,j}, (A \star B^3)_{i,j}, \ldots, (A \star B^{n^{\beta}/P})_{i,j}$, which stores the optimal Min-Plus values when k is from a specific small bucket. This part takes $\tilde{O}(\frac{n^{\beta}}{P} M n^{\omega(1,\beta,1)})$ time. The space complexity is $\frac{n^{\beta}}{P}$ times more than the space complexity of Lemma 5.3.1, so space complexity of this part is $\tilde{O}(\frac{Mn^{2+\beta}}{P} + \frac{n^{1+2\beta}}{P})$.

We also do the following preprocessing for buckets where $H_{j,\ell} - L_{j,\ell} > 2M$. We first create a $0/1$ matrix $\bar{A}$ where $\bar{A}_{i,k} = 1$ if and only if $A_{i,k} \neq \infty$. Then for each $\ell \in [n^{\beta}/P]$, we create a $0/1$ matrix $\bar{B}^{\ell}$ such that $\bar{B}^{\ell}_{k,j} = 1$ if and only if $k \in K_{j,\ell}$ and $H_{j,\ell} - L_{j,\ell} > 2M$. Then we use fast matrix multiplication to compute the product $\bar{A}\bar{B}^{\ell}$. If $K_{j,\ell}$ is a large bucket, the (i, j)-th entry of $\bar{A}\bar{B}^{\ell}$ is the number of $k \in K_{j,\ell}$ such that $A_{i,k} < \infty$; if $K_{j,\ell}$ is a small bucket, the (i, j)-th entry is 0. For each pair (i, j), we create a range tree $\mathcal{T}^{i,j}_{\text{large}}$ on the sequence $(\bar{A}\bar{B}^1)_{i,j}, (\bar{A}\bar{B}^2)_{i,j}, (\bar{A}\bar{B}^3)_{i,j}, \ldots, (\bar{A}\bar{B}^{n^{\beta}/P})_{i,j}$. This part takes $\tilde{O}(\frac{n^{\beta}}{P} n^{\omega(1,\beta,1)})$ time, which is dominated by the time for small buckets. The space complexity is also dominated by the data structures for small buckets.

Now we describe how to handle a query (i, j, S). First consider small buckets. In $O(|S|)$ time, we can compute the set of small buckets $K_{j,\ell}$ that intersect with S. For each such $K_{j,\ell}$, we can query the data structure D^{ℓ} with input $(i, j, S \cap K_{j,\ell})$ to get the optimum value when $k \in K_{j,\ell}$. For each small bucket that intersects with S, we can set its corresponding value in the range tree $\mathcal{T}^{i,j}_{\text{small}}$ to ∞, then we can compute the optimum value of all small buckets that do not intersect with S by querying the minimum value of the range tree $\mathcal{T}^{i,j}_{\text{small}}$. After this query, we need to restore all values in the range tree. It takes $\tilde{O}(|S|)$ time to handle small buckets on query.

Now consider large buckets. Intuitively, we want to enumerate indices in all large buckets $K_{j,\ell}$ such that there exists an index $k \in K_{j,\ell} \cap ([n^{\beta}] \setminus S)$ where $A_{i,k} < \infty$. However, doing so would be prohibitively expensive. We will show that we only need two such buckets. Consider three large buckets $l_1 < l_2 < l_3$. Pick any $k_1 \in T_{j,l_1}, k_3 \in T_{j,l_3}$ such that

$A_{i,k_1} < \infty$. Since

$$A_{i,k_1} + B_{k_1,j} \leq M + L_{j,l_2} < M + H_{j,l_2} - 2M < A_{i,k_3} + B_{k_3,j},$$

k_3 can never be the optimum. Thus, it suffices to find the smallest two buckets such that there exists an index $k \in K_{j,\ell} \cap ([n^\beta] \setminus S)$ where $A_{i,k} < \infty$, and then enumerate all indices in these two buckets. To find such two buckets, we can enumerate over all indices $k \in S$, and if $A_{i,k} < \infty$ we can decrement the corresponding value in the range tree $\mathcal{T}^{i,j}_{\text{large}}$. Thus, we can compute the two smallest buckets by querying the two earliest nonzero values in the range tree. We also need to restore the range tree after the query. The range tree part takes $\tilde{O}(|S|)$ time and scanning the two large buckets requires $O(P)$ time. Thus, this step takes $\tilde{O}(|S| + P)$ time.

At this point, we will know the bucket that contains the optimum index k^*. Thus, we can iterate all indices in this bucket to actually get the witness for the Min-Plus-Query-Witness query. It takes $O(P)$ time to do so.

In summary, the preprocessing time, query time, and space complexity meet the promise in the lemma statement.

□

In the following lemma, we show a data structure for the Min-Plus-Query-Witness problem when the matrix B is a monotone matrix with a small total range.

Lemma 5.3.3. *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 $0 \leq \theta \leq \eta$, $\rho \geq 0$ *and* $0 \leq \sigma \leq \beta$.

Proof. Preprocessing is in three steps, corresponding to the three phases of the algorithm for Theorem 1.2.2.

Preprocessing Step 1. This step is slightly more complicated than Phase 1 of Theorem 1.2.2. 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.

Running time of this step is $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta\}})$. Space complexity is also $\tilde{O}(n^{\max\{2,1+\beta+\eta-\theta\}})$ because we use persistent data structures.

Preprocessing Step 2. As in Phase 2 of Theorem 1.2.2, 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

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

$$B^r_{k,j} = B_{k,j} - B_{k,j^r}.$$

Then for each r, we apply the data structure in Lemma 5.3.2 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 similarly to Theorem 1.2.2.

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

$$0 \leq \tilde{A}_{i,k} + \tilde{B}_{k,j} - \frac{1}{W}\tilde{C}'_{i,j} \leq 1.$$

Note that for such triples, we must have

$$|A_{i,k} + B_{k,j} - \tilde{C}'_{i,j}| \leq 3W.$$

So the number of uncovered and almost relevant triples is $\tilde{O}(n^{2+\beta-\rho})$.

We run an algorithm similar to Lemma 2.4.5 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 $T_{i,j}$ defined as

$$\{(A_{i,k} + B_{k,j}, k) : (i,k,j) \text{ is uncovered and almost relevant}\}.$$

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 5.3.2) 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 $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \frac{1}{W}\tilde{C}'_{i,j} \leq -1$. 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 5.1.3. For clarity, we will use *element* to refer to a specific item a_i of the sequence and use *value* to refer to all elements of a given type. Given a pointer to an element of the sequence a_i, we assume the ability to look up its index i in the sequence in $\tilde{O}(1)$ time by storing all elements of the sequence in a balanced binary search tree with worst-case time guarantees (e.g. a red-black tree). Thus we can go from index i to element a_i and back via appropriate rank and select queries on the balanced binary search tree. We may also add or remove an element a_i from the sequence, and thus the binary search tree, in $\tilde{O}(1)$ time.

Let t_1, t_2, t_3 be three parameters of the algorithm. Parameter t_1 is a threshold that controls the number of "frequent" colors, t_2 controls how frequently the data structure is rebuilt, and t_3 represents the size of blocks in the algorithm.

We call values that appear more than n^{1-t_1} times *frequent* and all other values *infrequent*. Thus, there are at most n^{t_1} frequent values at any point in time. Note that a fixed value can change from frequent to infrequent, or from infrequent to frequent, via a deletion or insertion.

Infrequent Values

First, we discuss how to handle infrequent values. We maintain n^{1-t_1} balanced search

trees $\mathcal{BST}_1, \ldots, \mathcal{BST}_{n^{1-t_1}}$. For balanced search tree $\mathcal{BST}_k$, we prepare the key/value pairs in the following way. Fix a given value of the sequence. Say all its occurrences are at indices $i_1, i_2, \ldots, i_t$. Then we insert the key/value pairs (i_x, i_{x+k-1}) to $\mathcal{BST}_k$ for every $1 \leq x \leq t - k + 1$. However, the indices themselves would need updating when sequence a is updated. Instead of inserting the indices themselves, we insert corresponding pointers to the nodes of the binary search tree that holds sequence a. That way we can perform all comparisons using binary search tree operations in $\tilde{O}(1)$ time, without needing to update indices when sequence a changes. We also augment each balanced search tree $\mathcal{BST}_i$ so that every subtree stores the smallest value y of any pair (x, y) in the subtree. After an insertion or deletion, we need to update a total of $O((n^{1-t_1})^2)$ pairs. Thus, we can maintain these balanced search trees in $\tilde{O}((n^{1-t_1})^2)$ time per operation.

During a query $[l, r]$, we iterate through all the balanced search trees $\mathcal{BST}_1, \ldots, \mathcal{BST}_{n^{1-t_1}}$. If there exists a pair $(i_1, i_2) \in \mathcal{BST}_k$ such that $l \leq i_1 \leq i_2 \leq r$, then the range mode is at least k. Thus, if the range mode is an infrequent value, we can find its frequency and corresponding value by querying the balanced search trees. The query time is $\tilde{O}(n^{1-t_1})$, which is not the dominating term.

Newly Modified Values

We now consider how to handle frequent values. We handle newly modified values and unmodified values differently. We will rebuild our data structure after every n^{t_2} operations, and call values that are inserted or deleted after the last rebuild *newly modified values*.

For every value, we maintain a balanced search tree of occurrences of this value in the sequence. It takes $\tilde{O}(1)$ time per operation to maintain such balanced search trees. Thus, given an interval $[l, r]$, it takes $\tilde{O}(1)$ time to query the number of occurrences of a particular value in the interval. We use this method to query the number of occurrences of each newly modified value. Since there can be at most n^{t_2} such values, this part takes $\tilde{O}(n^{t_2})$ time per operation.

Data Structure Rebuild

It remains to handle the frequent, not newly modified values during each rebuild. In this case, we will assume we can split the whole array roughly equally into a left half and right

half. We can recursively build the data structure on these two halves so that we may assume a range mode query interval has left endpoint in the left half and right endpoint in the right half. The recursive construction adds only a poly-logarithmic factor to the complexity.

We split the left half and the right half into consecutive segments of length at most n^{t_3}, so that there are $O(n^{1-t_3})$ segments. We call the segments $P_1, P_2, \ldots, P_m$ in the left half and $Q_1, Q_2, \ldots, Q_m$ in the right half, where segments with a smaller index are closer to the middle of the sequence.

Let $v_1, v_2, \ldots, v_l$ be the frequent values during the rebuild. We create a matrix A such that $A_{i,k}$ equals the negation of the number of occurrences of v_k in segments $P_1, \ldots, P_i$; similarly, we create a matrix B such that $B_{k,j}$ equals the negation of the number of occurrences of v_k in segments $Q_1, \ldots, Q_j$. Note that the negation of the value $A_{i,k} + B_{k,j}$ is the frequency of value v_k in the interval from P_i to Q_j. It is not hard to verify that matrix B satisfies the requirement of Lemma 5.3.3. We take the negation here since Lemma 5.3.3 handles Min-Plus Product instead of Max-Plus Product. Then we use the preprocessing part of Lemma 5.3.3 with matrices A, B, and $\lambda = t_2$. In the notation of Lemma 5.3.3, $n = O(n^{1-t_3})$ and $n^\beta = O(n^{t_1})$, so $\beta = \frac{t_1}{1-t_3}$ and $\lambda = t_2$. Thus, by Lemma 5.3.3 the rebuild takes

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

time, 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. Since we perform the rebuild every n^{t_2} operations, the amortized cost of rebuild is

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

per operation.

Now we discuss how to handle queries for frequent, unmodified elements. For a query interval $[l, r]$, we find all the segments inside the interval $[l, r]$. The set of such segments must have the form $P_1, \cup \cdots \cup P_i \cup Q_1 \cup \cdots \cup Q_j$ for some i, j. We scan through all elements in $[l, r] \setminus (P_1 \cup \cdots \cup P_i \cup Q_1 \cup \cdots \cup Q_j)$, and use their frequency to update the answer. Since the size of segments is $O(n^{t_3})$, the time complexity to do so is $\tilde{O}(n^{t_3})$.

For the segments $P_1, \ldots, P_i, Q_1, \ldots, Q_j$, we query the data structure in Lemma 5.3.3

with S being the set of newly modified elements. The answer will be the most frequent element in the interval from P_i to Q_j that is not newly modified. By Lemma 5.3.3 this takes $O(|S|) = O(n^{t2})$ time per operation.

Time and Space Complexity

In summary, the amortized cost per operation is 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 $t_1, t_2, t_3 \in [0,1]$, $\beta = \frac{t_1}{1-t_3}$, $\eta = \frac{1-t_1}{1-t_3}$, $0 \le \theta \le \eta$, $\rho \ge 0$, $0 \le \sigma \le \beta$.

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 $\tilde{O}(n^{2-t_1}) = \tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$.

The space usage has two potential bottlenecks. The first is the space to store $\mathcal{BST}_1, \ldots, \mathcal{BST}_{n^{1-t_1}}$ for handling infrequent elements, which is $\tilde{O}(n^{2-t_1})$. The second is the space used by Lemma 5.3.3, which is

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

By plugging in the values, the space complexity becomes $\tilde{O}(n^{\frac{3\omega+39}{2\omega+30}})$, with the $\tilde{O}(n^{2-t_1})$ term being the dominating term.

Worst-Case Time Complexity

By applying the *global rebuilding* technique of Overmars [41], we can achieve a worst-case time bound. The basic idea is that after n^{t_2} operations, we don't immediately rebuild the Min-Plus-Query-Witness data structure. Instead, we rebuild the data structure during the next n^{t_2} operations, spreading the work evenly over each operation. To answer queries during these n^{t_2} operations, we use the previous build of the Min-Plus-Query-Witness data structure. By this technique, the per-operation runtime can be made worst-case.

□

Chapter 6

Single Source Replacement Paths

6.1 Introduction

The main contribution of this chapter 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.

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 [36], without increasing the asymptotic complexity. Vassilevska Williams and Williams [49] 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 [32] 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 [32] 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 [49], 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 [38, 53], as the best known algorithm by Zwick [58] 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 $\varepsilon > 0$ [32], 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 [32] for negative weights.

Theorem 6.1.1. *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.*

Remark 6.1.2. *Using the current best bound on fast rectangular matrix multiplication the running time can be improved to $O(M^{0.8043} n^{2.4957})$. Please refer to [57] for the analysis using fast rectangular matrix multiplication.*

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 $\tilde{O}(M^{5/9} n^{22/9})$.

APSP in directed graphs with edge weights in $\{-1, 0, 1\}$ is one of long list of so-called *intermediate* graph and matrix problems [38, 53], 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. [31] 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 6.1.1 by improving the runtime of the so-called *subpath problem*, which is the bottleneck in the algorithm of [32]. 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 [58]. 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 6.1.3. *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 6.1.3 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 [17, 5, 18, 19]. 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.

6.2 Algorithm

In this section we present our improved algorithm for SSRP, proving Theorem 6.1.1.

Theorem 6.1.1. *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.*

To this end we improve the bottleneck in Grandoni-Vassilevska Williams algorithm [32], 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 6.1.1 follows from a black-box[1] application of Lemmas 6.2.1 and 6.2.5.

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 6.2.1 (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 6.2.1 (Lemma 5.1 in [32])**.** *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*

[1] See end of the proof of Lemma 6.2.5 for a discussion why the $\tilde{O}(Mn^\omega)$ component from Lemma 6.2.1 can be omitted.

paths, and of the second type – *departing paths*. Grandoni and Vassilevska Williams [32] 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 6.2.2). We just follow their approach in that regard.

Lemma 6.2.2 (Theorem 1.1 in [32])**.** *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 $\min_{j \le i} d_G(s, s_j) + d_{\widetilde{G}}(s_j, v)$. Grandoni and Vassilevska Williams [32] simply feed $\widetilde{G}$ to Zwick's APSP algorithm [58], 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 [58], already described in [32]. 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 [32].

Lemma 6.2.3 (Corollary 3.1 in [32])**.** *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 6.2.4 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 6.2.4 (implicit in [56], Lemma 2.3 in [32])**.** *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 $A \star B = (B^T \star A^T)^T$ using Theorem 1.2.2.

Wrap-up and runtime analysis. Now we can sum up the running time and then balance the terms.

Lemma 6.2.5. *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.*

Proof. Let $M = n^{\mu}$. By summing bounds from Lemma 6.2.2, Lemma 6.2.3, and the hop-long path part (Yuster-Zwick distance oracle, and a Monotone Min-Plus Product), we get that the running time is

$$\tilde{O}\left(n^{\mu+\omega} + n^{\mu+\zeta+\omega(1,1-\zeta,1)} + n^{3-2\zeta} + n^{\frac{1}{5}(9+(2+\omega)(1-\zeta)+1+\mu)}\right).$$

To express the running time as a function of square matrix multiplication exponent ω, we need to implement rectangular matrix multiplication by cutting rectangular matrices into square matrices, which lets us bound $\omega(1, 1-\zeta, 1) \leq \omega \cdot (1-\zeta) + 2\zeta$. By setting $\zeta = 4 \cdot \frac{3-\omega-\mu}{17-4\omega}$, we get running time

$$\tilde{O}(M^{\frac{5}{17-4\omega}} n^{\frac{36-7\omega}{17-4\omega}}). \tag{6.1}$$

We remark that if the value ζ we pick is negative, then our claimed running time is worse than the naive $\tilde{O}(n^3)$ running time, so the claimed bound will trivially hold.

□

6.3 Lower Bound

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

Theorem 6.1.3. *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.*

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

Problem 6.3.1 (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. [16] for a reduction from Min-Plus Product with small integer weights to unweighted directed APSP.

Lemma 6.3.1. *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 [16], 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 $1 \leq i < n$. 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 $i' \geq i$, then choose some path p_k in the middle, then go to $b_{j'}$ for $j' \leq j$, 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 $2(i' - i) + X'_{i',k} \geq X'_{i,k}$ and $Y'_{k,j'} + 2(j - j') \geq Y'_{k,j}$. 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 6-1. 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. □

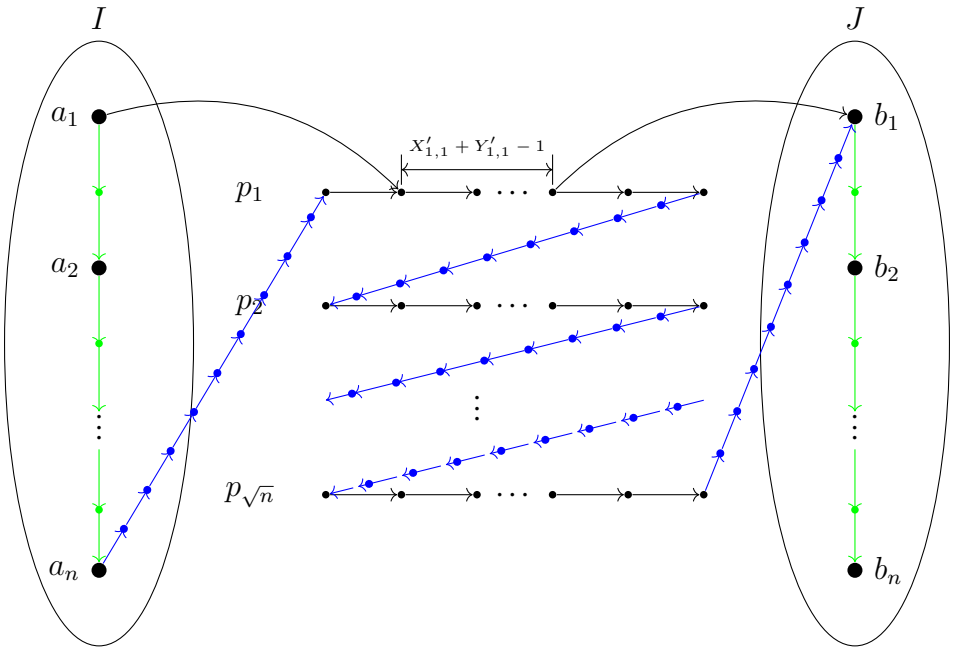

Figure 6-1: Reduction from Bounded-Difference Min-Plus Product to Ham-APSP (Lemma 6.3.1). Most edges between the parts I, J and the middle paths $p_1, \ldots, p_{\sqrt{n}}$ are omitted for clarity. The green portions are the first type paths, and the blue portions are the second type paths.

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

Lemma 6.3.2. *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 6-2:

1. We add an edge from v'_i to v'_{i-1} of weight -1 for every $1 \leq i \leq n+1$;

2. We add an edge from v'_i to v_i of weight 0 for every $1 \leq i \leq n$;

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'.

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 $1 \leq j \leq n$ must move from v'_{n+1} to v'_k for some $i \leq k \leq n$, then take the weight 0 edge to v_k, and finally move to v_j in the copy of graph G. Therefore, $d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) = \min_{i \leq k \leq n}(k - n - 1) + d_G(v_k, v_j)$. We show that $\min_{i \leq k \leq n}(k - n - 1) + d_G(v_k, v_j) = (i - n - 1) + d_G(v_i, v_j)$. Clearly, $\min_{i \leq k \leq n}(k - n - 1) + d_G(v_k, v_j) \leq (i - n - 1) + d_G(v_i, v_j)$ 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, $d_G(v_i, v_k) + d_G(v_k, v_j) \geq d_G(v_i, v_j)$. Since G has a Hamiltonian path $v_1 \to \cdots \to v_n$, it holds that $d_G(v_i, v_k) \leq k - i$. Hence,

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

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 $d_G(v_i, v_j) = d_{G'}(v'_{n+1}, v_i, (v'_i, v'_{i-1})) - (i - n - 1)$. □

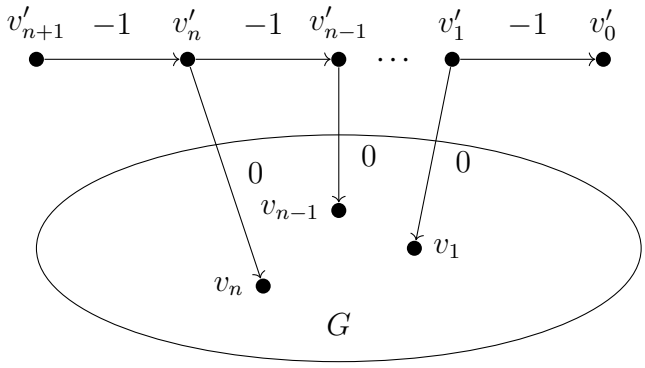

Figure 6-2: Reduction from Ham-APSP to SSRP with weights in $\{-1, 0, 1\}$ (Lemma 6.3.2).

Bibliography

[1] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 253–266, 2018.

[2] Pankaj K. Agarwal and Jirí Matousek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11(4):393–418, 1994.

[3] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993.*, pages 207–216, 1993.

[4] 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.

[5] 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.

[6] 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.

[7] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 81:1–81:13, 2016.

[8] Jon Bentley. Programming pearls: Perspective on performance. *Commun. ACM*, 27(11):1087–1092, November 1984.

[9] Jon Louis Bentley. Algorithm design techniques. *Commun. ACM*, 27(9):865–871, 1984.

[10] Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate range mode and range median queries. In *Annual Symposium on Theoretical Aspects of Computer Science*, 2005.

[11] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly sub-cubic algorithms for language edit distance and rna-folding via fast bounded-difference min-plus product. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 375–384, 2016.

[12] Gerth Stølting Brodal, Beat Gfeller, Allan Jørgensen, and Peter Sanders. Towards optimal range medians. In *Theoretical Computer Science*, 2011.

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

[14] 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(4):719–741, 2014.

[15] Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 77. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Hicham El-Zein, Meng He, J. Ian Munro, Yakov Nekrich, and Bryce Sandlund. On approximate range mode and range selection. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*, 2019.

[21] 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.

[22] Amr Elmasry, Meng He, J Ian Munro, and Patrick K Nicholson. Dynamic range majority data structures. In *International Symposium on Algorithms and Computation*, pages 150–159. Springer, 2011.

[23] 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.

[24] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Constructing efficient decision trees by using optimized numeric association rules. In *Proceedings of 22th International Conference on Very Large Data Bases, VLDB 1996, September 3-6, 1996, Mumbai (Bombay), India*, pages 146–155, 1996.

[25] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized accociation rules: Scheme, algorithms, and visualization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*, pages 13–23, 1996.

[26] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.*, 26(2):179–213, 2001.

[27] Travis Gagie, Meng He, and Gonzalo Navarro. Compressed dynamic range majority data structures. In *Data Compression Conference (DCC), 2017*, pages 260–269. IEEE, 2017.

[28] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

[29] 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.

[30] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995.

[31] 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.

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

[33] Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell probe lower bounds and approximations for range mode. In *International Colloquium on Automata, Languages, and Programming*, 2010.

[34] Meng He, J. Ian Munro, and Patrick K. Nicholson. Dynamic range selection in linear space. In *International Symposium on Algorithms and Computation*, 2011.

[35] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, page 21–30, 2015.

[36] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[37] Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 2005.

[38] 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.

[39] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.

[40] Jiri Matousek. Reporting points in halfspaces. *Computational Geometry*, 2(3):169–186, 1992.

[41] Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983.

[42] Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 117–122. Society for Industrial and Applied Mathematics, 2010.

[43] Bryce Sandlund and Yinzhan Xu. Faster dynamic range mode. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020.

[44] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999, 17-18 October, 1999, New York, NY, USA*, pages 605–615, 1999.

[45] Tadao Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electr. Notes Theor. Comput. Sci.*, 61:191–200, 2002.

[46] Hisao Tamaki and Takeshi Tokuyama. Algorithms for the maxium subarray problem based on matrix multiplication. In *SODA*, volume 1998, pages 446–452, 1998.

[47] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

[48] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians*, page to appear, 2018.

[49] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

[50] 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.

[51] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

[52] 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.

[53] 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.

[54] Kunikazu Yoda, Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Computing optimized rectilinear regions for association rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining KDD 1997, Newport Beach, California, USA, August 14-17, 1997*, pages 96–103, 1997.

[55] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957, 2009.

[56] 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.

[57] Virginia Vassilevska Williams Yuzhou Gu, Adam Polak and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, page to appear. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[58] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.