Truly subcubic min-plus product for less structured matrices, with applications

Citation: Williams, VV and Xu, Y. 2020. "Truly subcubic min-plus product for less structured matrices, with applications." Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, 2020-January.

Persistent URL: https://hdl.handle.net/1721.1/137947

Version: Original manuscript: author's manuscript prior to formal peer review

Truly Subcubic Min-Plus Product for Less Structured Matrices, with Applications

Virginia Vassilevska Williams* and Yinzhan Xu†

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 paper is to get truly subcubic algorithms for Min-Plus product for less structured inputs than what was previously known, and to apply them to versions of APSP and other problems. The results are as follows:

(1) Our main result is the first truly subcubic algorithm for the Min-Plus product of two $n \times n$ matrices A and B with polylog n bit integer entries, where B has a partitioning into $n^{\varepsilon} \times n^{\varepsilon}$ blocks (for any $\varepsilon > 0$) where each block is at most n^{δ}-far (for $\delta < 3 - \omega$, where $2 \leq \omega < 2.373$) in ℓ_∞ norm from a constant rank integer matrix. This result presents the most general case to date of Min-Plus product that is still solvable in truly subcubic time.

(2) The first application of our main result is a truly subcubic algorithm for APSP in a new type of geometric graph. Chan'10 solved APSP in truly subcubic time in geometric graphs whose edges have weights that are a function of the identities of the edge's end-points. Our result extends Chan's result in the case of integer edge weights by allowing the weights to differ from a function of the end-point identities by at most n^{δ} for small δ.

(3) In a second application we consider a batch version of the range mode problem in which one is given a sequence of numbers $a_1, \ldots, a_n$ and n intervals defining contiguous subsequences, and one is asked to compute the range mode of each subsequence. Chan et al.'14 showed that any $O(n^{1.5-\varepsilon})$ time combinatorial algorithm for $\varepsilon > 0$ for this problem can be used to solve Boolean matrix multiplication combinatorially in truly subcubic time. We give the first $O(n^{1.5-\varepsilon})$ time for $\varepsilon > 0$ algorithm for this batch range mode problem, showing that the hardness is indeed constrained to combinatorial algorithms.

(4) Our final application is to the Maximum Subarray problem: given an $n \times n$ integer matrix, find the contiguous subarray of maximum entry sum. We show that Maximum Subarray can be solved in truly subcubic, $O(n^{3-\varepsilon})$ (for $\varepsilon > 0$) time, as long as every entry of the input matrix is no larger than $O(n^{0.62})$ in absolute value. This is the first truly subcubic algorithm for an interesting case of Maximum Subarray. The Maximum Subarray problem with arbitrary integer entries is known to be subcubically equivalent to APSP, in that a truly subcubic, $O(n^{3-\varepsilon})$ time algorithm for $\varepsilon > 0$ for one problem would imply a truly subcubic algorithm for the other. Because of this it is believed that Maximum Subarray does not admit truly subcubic algorithms, without a restriction on the inputs.

We also improve all the known conditional hardness results for the d-dimensional variant of Maximum Subarray, showing that many of the known algorithms are likely tight.

*MIT EECS and CSAIL, virgi@mit.edu

†MIT EECS, xyzhan@mit.edu

1 Introduction

The All-Pairs Shortest Paths (APSP) problem is one of the most basic and well-studied problems in graph algorithms. Algorithms for APSP have been studied since the 1950s when the Floyd-Warshall algorithm achieved a running time of $O(n^3)$ in n-vertex graphs. Over the next six decades, some improvements over the cubic running time were developed, culminating in the current fastest $n^3/2^{\Theta(\sqrt{\log n})}$ time algorithm by Williams [28]. Unfortunately, no *truly subcubic*, $O(n^{3-\varepsilon})$ time for $\varepsilon > 0$ algorithm is known, and a hypothesis that such an algorithm does not exist for APSP has become prominent in the field of fine-grained complexity (see e.g. [25]).

The so called *Min-Plus* product of matrices A and B, defined as the matrix C with $C_{i,j} = \min_k(A_{i,k} + B_{k,j})$, is known to be asymptotically *equivalent* to APSP (see e.g. [12]) in the sense that a $T(n)$ time algorithm for the Min-Plus product of two $n \times n$ matrices implies an $O(T(n))$ time algorithm for APSP in n-node graphs, and vice versa. Because of this equivalence, research on APSP algorithms typically involves studying the Min-Plus product directly.

A long line of research on APSP involves studying the Min-Plus product of structured matrices. The relationship between APSP and Min-Plus product extends to structured instances as well: Min-Plus product of structured matrices can be viewed as APSP in a structured layered graph with three layers. Conversely, in the case of graphs with integer weights but also in many other cases, APSP on structured instances is in truly subcubic time if Min-Plus product of an *arbitrary* matrix with a *structured* matrix (i.e. the graph's generalized adjacency matrix) is in truly subcubic time[1].

Studying structured instances of Min-Plus product/APSP is important for two main reasons:

- As an approach towards truly subcubic APSP: *What structure, if any, is needed to solve APSP in truly subcubic time?*

- As an approach to solve other problems faster: APSP is a very versatile problem, and many other important problems that sometimes, on the face of it, seem to have nothing to do with shortest paths, can be reduced to APSP. Often, the instances that are produced in these reductions are actually structured, and one could exploit this structure to get faster algorithms.

In the 1990s, Alon, Galil and Margalit [4] showed that the Min-Plus product of two $n \times n$ matrices of integers in $\{M, \dots, M\}$ can be computed in $O(Mn^\omega \log(Mn))$ time, where $\omega < 2.373$ [24, 16] is the matrix multiplication exponent; thus Min-Plus product is in truly subcubic time, as long as the matrix entries are small, $M < O(n^{3-\omega-\varepsilon})$ for $\varepsilon > 0$. This result is used over and over in shortest paths algorithms. For instance, it implies that APSP in undirected [20] and directed [31] graphs with small enough integer weights is in truly subcubic time.

Truly subcubic time algorithms for less and less structured versions of Min-Plus and APSP were developed over the years, e.g. [4, 30, 10, 23]. The most general structured Min-Plus algorithm to date is by Bringmann et al. [9]: Min-Plus product of two $n \times n$ integer matrices A and B is in truly subcubic time if A is arbitrary and for all rows (or similarly, columns) of B, any two consecutive entries are close: $|B[i,j] - B[i,j+1]| \leq n^\delta$ for small enough $\delta > 0$. (B is then called a bounded difference matrix.)

Bringmann et al. showed that their result on bounded difference matrices subsumes all previous results on truly subcubic Min-Plus product. They also gave several applications of their new algorithm, most no-

[1] For graphs with integer edge weights, this is true regardless of the structure, as one can always leverage two types of approaches to APSP: (1) compute the distances on paths that have at most n^δ vertices by iterating the Min-Plus product n^δ times, and (2) compute the distances on paths that have at least n^δ vertices by random sampling and a SSSP algorithm such as Dijkstra's, after the usual removal of negative edge weights using Johnson's trick and a truly subcubic time SSSP algorithm such as [17].

tably for language edit distance and RNA folding, that were not possible with the prior results on structured Min-Plus product.

Even though it is very powerful, the Bringmann et al. Min-Plus product result is still not general enough to solve some well-structured Min-Plus instances. We give one simple example. Consider a matrix X such that for every i, j, $|X[i,j] + X[i+1,j+1] - X[i+1,j] - X[i,j+1]| \leq 1$; let's call X a bounded discrete derivative (BDD) matrix. BDD matrices are extremely special, and we won't be too surprised if their Min-Plus product can be done in truly subcubic time. A truly subcubic algorithm for Min-Plus product for BDD matrices would be useful, for instance, for finding a Maximum Subarray of a matrix with small entries, a well-studied problem with many applications.

Unfortunately, however, BDD matrices are not bounded difference matrices, and the Bringmann et al. algorithm does not apply to them. Even the general framework devised by Bringmann et al. cannot be used as is. (We will go into more details in a bit.) The main goal of this paper is to modify Bringmann et al.'s framework to support less structured matrices, and to apply the new framework to obtain the first substantial improvements on the complexity of several studied problems.

1.1 Our results

1.1.1 New Subcubic Min-Plus Products.

Our main result is a new algorithm for Min-Plus product for less structured matrices. We begin with defining the structure needed.

Definition 1.1 (W-approximate rank)**.** *For an $n \times n$ integer matrix M, its W-approximate rank is defined as*

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

This W-approximate rank definition resembles the ε-approximate rank definition of Alon et al. [5]. The difference is that we require the matrix X to be have integer entries.

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 $n^\delta \times n^\delta$ blocks $B^{a,b}$ (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 submatrix $B^{a,b}$ has W-approximate rank at most $O(1)$.

Our main result is:

Theorem 1.1. *Let $\delta \in (0,1]$. The Min-Plus product of two $n \times n$ matrices A and B 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 $1 \leq d = O(1)$ can be computed in time*

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

Notice that the matrix A is arbitrary, as long as its entries do not get too huge, larger than $2^{\omega(\text{polylog } n)}$. We would like arithmetic operations on the matrix entries to take $\tilde{O}(1)$ time, so that this entry size is not much of a restriction. The algorithm can handle larger entries as well. If the entries of A and B are β-bit integers, the algorithm gets a $\tilde{O}(\beta)$ overhead.

The running time of the algorithm is truly subcubic for any constant d and any constant $\delta > 0$, as long as $W = O(n^{3-\omega-\varepsilon})$ for some $\varepsilon > 0$.

Let us discuss first why Theorem 1.1 subsumes all previous results on truly subcubic structured Min-Plus product. We only need to show that a bounded differences matrix also has constant W-approximate rank blocks, as by the discussion in [9], all other known cases of truly subcubic Min-Plus can be reduced to multiplying a bounded differences matrix with an arbitrary integer matrix. Suppose that B is such that for every i and j, $|B_{i,j} - B_{i,j+1}| \leq Q$ for small Q. Now consider the $n^\delta \times n^\delta$ sub-blocks $B^{a,b}$ of B (for any choice of $\delta > 0$). All columns of $B^{a,b}$ differ entrywise from the first column $B^{a,b}(1)$ by at most Qn^δ. Thus, if we consider the rank one matrix that has n^δ columns identical to $B^{a,b}(1)$, we see that $B^{a,b}$ has Qn^δ-approximate rank one. Hence by Theorem 1.1, we get that for any $Q = O(n^{3-\omega-\varepsilon})$ for $\varepsilon > 0$, we can pick $\delta = \varepsilon/2$ and we'll get a truly subcubic time algorithm to Min-Plus multiply an arbitrary integer matrix A by B.

Theorem 1.1 is very general and can handle much more than just bounded difference matrices. For instance, it is not hard to see that the aforementioned BDD matrices have constant W-approximate rank blocks, but also many other structured instances can be solved using Theorem 1.1, as we will see in our applications.

To prove Theorem 1.1 we modify the Min-Plus framework of Bringmann et al. [9] and combine it with a result from computational geometry on halfspace intersection reporting.

We will give a brief overview on how we modify the Bringmann et al. framework. The framework from [9] for computing the Min-Plus product C of integer matrices A and B consists of Phase 1, Phase 2 and Phase 3.

Phase 1 computes a matrix C' which is close in ℓ_∞ norm to the desired output C. This phase is not hard to perform for the type of matrices we are considering; also, as shown by Bringmann et al., often this Phase can be avoided by scaling, and the real difficulty is in Phase 2, and especially Phase 3.

Phase 2 iteratively takes random samples of rows of A and columns of B, and repeatedly creates new matrices $\tilde{A}$ and $\tilde{B}$ whose entries are clever linear combinations of entries of A, B, the sampled row and column, and C', so that most entries of the Min-Plus product C of A and B can be easily computed from the Min-Plus products $\tilde{C}$ of $\tilde{A}$ and $\tilde{B}$ in $O(n^2)$ time. To perform Phase 2 efficiently, Bringmann et al. replace any entries of $\tilde{A}$ and $\tilde{B}$ that are larger than some M by ∞ and use the $\tilde{O}(Mn^\omega)$ time algorithm [4] to perform the Phase 2 Min-Plus products. By removing the large entries, some entries of C will not be recoverable from the computed Min-Plus products $\tilde{C}$ in the Phase 2 iterations. Bringmann et al. show that at most a truly subcubic number of sums $A_{i,k} + B_{k,j}$ that might be close to the Min-Plus product entries will be missed in the computation.

Phase 3 strives to recover the parts of the output matrix C that are missed by Phase 2. We know that at most a truly subcubic number of relevant sums $A_{i,k} + B_{k,j}$ need to be considered. If we knew which triples i, k, j are involved in such sums, then we could finish the Min-Plus product in truly subcubic time by computing the sums explicitly. However, the main difficulty lies exactly in finding these triples. In particular, in the case of BDD matrices, there doesn't seem to be enough structure for one to be able to recover the remaining relevant triples in Step 3 efficiently.

One of the main insights in this work is that one can offload more work to Phase 2 so that in Phase 3 there is enough structure left to recover the remaining relevant triples efficiently. In particular, instead of removing the large entries from both $\tilde{A}$ and $\tilde{B}$ in Phase 2, we only remove them from $\tilde{A}$. Then intuitively, Phase 2 does more work, and it turns out that in Phase 3, in truly subcubic time, one can find the remaining triples that one needs to consider to compute the entire Min-Plus product of A and B, using a halfspace intersection reporting data structure from computational geometry.

However, now in Phase 2 we need to compute the Min-Plus product of an *arbitrary* integer matrix with a matrix with ∞ entries and finite entries bounded by M. This type of Min-Plus product is no longer known to be in $\tilde{O}(Mn^\omega)$ time. An $\tilde{O}(Mn^{(3+\omega)/2})$ time algorithm follows from prior work (e.g. [30], Lemma 3.3). We are able to improve the dependence on M, thus allowing for a faster truly subcubic final algorithm for Theorem 1.1.

Theorem 1.2. *The* $(\min, +)$*-product of two* $n \times n$ *integer matrices* A *and* B*, where* A *has entries in* $\{-M, \ldots, M\} \cup \{\infty\}$ *for some* $M \geq 1$ *and* B *is arbitrary can be computed in* $\tilde{O}(\sqrt{M} n^{(3+\omega)/2})$ *time.*

1.1.2 Applications.

To highlight the power of our new Min-Plus algorithm, we apply Theorem 1.1 to obtain the first improvements in the running times for several problems: a new geometric version of APSP, a batch range mode problem considered by Chan et al. [11] and the Maximum Subarray problem.

Geometric APSP. As we discussed earlier, 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 1.1 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 1.1 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 [10], 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 1.1 uses (Theorem 2.1) with the partition theorem in [2], we can achieve APSP for arbitrary algebraic functions as in [10], 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 [10] 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.

Batch Range Mode. Given a sequence a of length n, the range mode query on a range $[l, r]$ asks for the frequency of the most frequent element in the subsequence between the l-th and r-th element of a. Chan et al. [11] designed a linear space data structure that answers any range mode query in $\tilde{O}(\sqrt{n})$ time. Because the preprocessing step of the data structure is fast, this implies a $\tilde{O}(n^{1.5})$ time algorithm for the *batch* range mode problem in which one is given a sequence and n range mode queries to answer in batch.

Chan et al. [11] showed that any combinatorial algorithm for the batch range mode problem running in $O(n^{1.5-\varepsilon})$ time for $\varepsilon > 0$ would imply an $O(n^{3-\delta})$ time combinatorial algorithm for $\delta > 0$ that computes the product of two n by n Boolean matrices. This suggests that it might be hard to find such a combinatorial algorithm for batch range mode, as Boolean matrix multiplication is often conjectured to require $n^{3-o(1)}$ time using a combinatorial algorithm (see e.g. [25]). This leads to a natural question: if we do not limit to combinatorial algorithms, what should the complexity of batch range mode be? Prior to this work, no noncombinatorial $n^{1.5-o(1)}$ lower bounds (even conditional ones), and no $O(n^{1.5-\varepsilon})$ time (for $\varepsilon > 0$) algorithms were known to exist.

As another application of Theorem 1.1 we obtain a $\tilde{O}(n^{1.4854})$ time algorithm for batch range mode, giving the first ever $O(n^{1.5-\varepsilon})$ time (for $\varepsilon > 0$) algorithm for the problem. Note that in this application, we use $d = 1$ in Theorem 1.1, so each block of matrix B is a bounded difference matrix. Thus Bringmann et al.'s algorithm suffices to give an $O(n^{1.5-\varepsilon})$ (for $\varepsilon > 0$) time algorithm for batch range mode.

Maximum Subarray. 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 [8], the problem has many applications, for instance in graphics (see [22]) and in databases [3, 14, 15, 13, 29].

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 [8]) achieves a linear, $O(n)$ running time. Bentley [7] 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. [22] and Takaoka [21] 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 [28], 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 Maximum Subarray. Backurs et al. [6] and Vassilevska W. and Williams [26] 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 [22, 21], 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 1.1 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 before:

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

As BDD matrix Min-Plus product is a special case of Theorem 1.1 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. [6] 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. [25]):

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. [6] 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 1.3. *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. [18, 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 [18]) 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) [27, 18] 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 1.4. *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.

2 Preliminaries

We use $\tilde{O}(f(n))$ to denote $f(n)$polylog n. For a matrix X, we denote by $X(i)$ the ith column of X.

The Min-Plus or $(\min, +)$-product of two $n \times n$ matrices A and B is the $n \times n$ matrix $C = A \star B$ with $C[i,j] = \min_k\{A[i,k] + B[k,j]\}$. The All-Pairs Shortest Paths problem (APSP) is given a graph $G = (V, E)$ with integer edge weights $w(\cdot)$, determine for all $u, v \in V$, the shortest path distance $d(u, v)$ from u to v.

We let ω be the exponent of square matrix multiplication, i.e. the smallest real number such that $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time. It is known that $2 \leq \omega < 2.373$ [16, 24].

It is known [4] that for any $M \geq 1$, the Min-Plus product of two $n \times n$ matrices with entries in $\{-M, \ldots, M\} \cup \{\infty\}$ can be computed in time $\tilde{O}(Mn^\omega)$.

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

Theorem 2.1 ([19]). *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 $k = |P \cap \lambda|$.*

3 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 $M \geq 1$). Previously, the best algorithm for this runs in $\tilde{O}(Mn^{(3+\omega)/2})$ time.

Proof of Theorem 1.2. 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 $\ell \in [n/\Delta]$, 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 $L_{j,\ell} - S_{j,\ell} \le 2M$ (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 $L_{j,\ell} - S_{j,\ell} \le 2M$, $B^\ell_{k,j} = B_{k,j} - S_{j,\ell} - M \in [-M, M]$ for any $k \in P_{j,\ell}$. Thus, we can compute $C^\ell = A \star B^\ell$ in $\tilde{O}(Mn^\omega)$ 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 $\tilde{O}(Mn^{\omega+1}/\Delta)$ time since we compute $O(n/\Delta)$ 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 $O(n^\omega)$ 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 $\tilde{O}(n^{\omega+1}/\Delta)$ time since we compute $O(n/\Delta)$ 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 $A_{i,k_1} + B_{k_1,j} \le M + L_{j,\ell_1}$. 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, $A_{i,k_1} + B_{k_1 j} \le M + S_{j,\ell_2}$. Similarly, $A_{i,k_3} + B_{k_3,j} \ge -M + S_{j,\ell_3} \ge -M + L_{j,\ell_2}$. 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 $\tilde{O}(Mn^{1+\omega}/\Delta + n^2\Delta)$. Setting $\Delta = \sqrt{M}n^{(\omega-1)/2}$ gives the claimed $\tilde{O}(\sqrt{M}n^{(3+\omega)/2})$ time. □

4 Main Algorithm

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 $I(i') = \{i : i' - \Delta < i \le i'\}$ 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 4.1 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 4.1. *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 1.1 from Theorem 4.1, 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 [9] that computes the $(\min, +)$-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 [9].

In the rest of this section, we will use $C = A \star B$ to denote the desired $(\min, +)$-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 $(\min, +)$-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 $\hat{C}_{ij}$.

4.1 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 $(\min, +)$-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 2.1 to approximate $A_{I(i'),I(k')} \star B_{I(k'),I(j')}$.

Lemma 4.1. *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 2.1 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 $\tilde{C}^{i',k',j'}_{i,j}$.

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 4.2. *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 4.1. 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 $\tilde{C}_{i,j} = \min_{k' : \Delta | k'} \tilde{C}^{i',k',j'}_{i,j}$ in $O(n^3 \cdot \Delta^{-1})$ time. □

4.2 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 1.2.

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 $(\min, +)$-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 4.1. *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 4.3. *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. □

4.3 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 4.4. *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

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

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

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

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

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

By Lemma 4.4, 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 2.1. For each $(i,j) \in I(i') \times I(j')$, we create the following half-space:

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

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

5 Application I: Geometric APSP

In this section, we study an algorithm for APSP where the edge weights of the input graph can be approximated by a low dimensional geometric function.

Let W be an integer, $d \ge 1$ 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 $v \in V_i$, $u \in V_j$, $|w(v,u) - p^{i,j}(v)^T q^{i,j}(u)| \le W$. 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 [17].

The following is a direct corollary of Theorem 1.1:

Corollary 5.1. *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 5.1, 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 5.1. *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 $W \le n^{3-\omega-4\delta/\lfloor(d+1)/2\rfloor}$.*

Proof. Let B be the generalized adjacency matrix, and let ℓ be a parameter to be fixed later. For each $i \le \ell$, we can compute $B^{(i)}$ by iterating the product $B^{(i)} \leftarrow A \star B$ for $A = B^{(i-1)}$. By Corollary 5.1, 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 $W \le n^{3-\omega-4\delta/\lfloor(d+1)/2\rfloor}$, then $n^{(9+\omega)/4} \cdot W^{1/4} \le n^{3-\delta/\lfloor(d+1)/2\rfloor}$, 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 $\tilde{O}(n^{3-\delta/(2\lfloor(d+1)/2\rfloor)})$. □

6 Application II: Batch Range Mode

In this section, as an application of our Main Algorithm, we give an $O(n^{1.5-\varepsilon})$ time algorithm for the Batch Range Mode query problem for some $\varepsilon > 0$. In a high level, there are two steps in the algorithm. First we use the Main Algorithm to obtain a truly subcubic time $(\min, +)$-product for particularly structured matrices; then we show how to reduce range mode to this kind of structured $(\min, +)$-product.

Lemma 6.1. *Let A, B be two $n \times n$ integer matrices, where matrix B meets*

1) Each row of B is non-increasing;

2) *The difference between the sum of elements in the j-th column, and the sum of elements in the $(j+1)$-th column is at most m, for any j.*

When $m = \Omega(n^{(\omega-1)/2})$, there exists a $\tilde{O}(n^{(14+\omega)/6} \cdot m^{1/6})$ time algorithm that computes $A \star B$, which is truly sub-cubic as long as $m = O(n^{4-\omega-\varepsilon})$ for some $\varepsilon > 0$. When $m = O(n^{(\omega-1)/2})$, there exists a $\tilde{O}(n^{(9+\omega)/4})$ time algorithm.

Proof. Let $\Delta, \gamma \geq 1$ be small polynomials in n to be fixed later. Fix j' a multiple of Δ. Since $\sum_{k=1}^{n} B_{k,j} - \sum_{k=1}^{n} B_{k,j+1} \leq m$ for any $j \in I(j')$, we have

$$\sum_{k=1}^{n} B_{k,j'-\Delta+1} - \sum_{k=1}^{n} B_{k,j'} \leq \Delta m.$$

By averaging, there are at most $\Delta m/\gamma$ indices $k \in [n]$ such that $B_{k,j'-\Delta+1} - B_{k,j'} \geq \gamma$. For each k such that $B_{k,j'-\Delta+1} - B_{k,j'} \geq \gamma$, and for each $j \in I(j')$, we set $B_{k,j}$ as M, for some large enough integer M (larger than all entries in B). We call the matrix $\hat{B}$ after we do this transformation for every j'. Note that $\hat{B}$ differs with B in at most $nm\Delta/\gamma$ entries.

Notice that $\hat{B}$ has the following nice property: for each j', k' multiples of Δ, $\left|\hat{B}_{j,k} - \hat{B}_{j',k}\right| \leq \gamma$ for any $j \in I(j'), k \in I(k')$. Consider a set of 1-dimensional vectors $X_{k',j'}(k) = [\hat{B}_{j',k}]$, and $Y_{k',j'}(j) = [1]$, then $\left|\hat{B}_{j,k} - X_{k',j'}(k)^T Y_{k',j'}(k)\right| \leq \gamma$. Therefore, we can apply Theorem 1.1 using $d = 1$. This gives a

$$\tilde{O}(n^3/\Delta + n^{(9+\omega)/4} \cdot \gamma^{1/4})$$

time algorithm to compute $\hat{C} = A \star \hat{B}$.

We can recover $C = A \star B$ from $\hat{C}$. Since B and $\hat{B}$ differ in at most $nm\Delta/\gamma$ entries, and $\hat{B}$ is larger on these entries, we can enumerate $A_{i,k} + B_{k,j}$ to update $C_{i,j}$, where $B_{k,j}$ differs from $\hat{B}_{k,j}$. This will take $O(n^2 m\Delta/\gamma)$ time.

The total complexity is $\tilde{O}(n^3/\Delta + n^{(9+\omega)/4} \cdot \gamma^{1/4} + n^2 m\Delta/\gamma)$.

When $m = \Omega(n^{(\omega-1)/2})$, we can balance by setting $\Delta = n^{(4-\omega)/6} m^{-1/6}$, and $\gamma = n^{(1-\omega)/3} m^{2/3}$. This gives a $\tilde{O}(n^{(14+\omega)/6} \cdot m^{1/6})$ time algorithm.

When $m = O(n^{(\omega-1)/2})$, we can balance by setting $\Delta = n^{(3-\omega)/4}$, and $\gamma = 1$ to get a $\tilde{O}(n^{(9+\omega)/4})$ time algorithm. □

Theorem 6.1. *Given a sequence $a_1, a_2, \ldots, a_n$, and n ranges $[l_1, r_1], [l_2, r_2], \ldots, [l_n, r_n]$, there exists a $\tilde{O}(n^{(27+2\omega)/(19+\omega)})$ time algorithm that computes the frequency of the most frequent element for each range $[l_i, r_i]$. Using $\omega \leq 2.373$, this algorithm runs in $\tilde{O}(n^{1.4854})$ time.*

Proof. 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 T times (infrequent elements), and elements that appear more than 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 T times, the number of of such intervals is at most $O(Tn)$. 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 $\tilde{O}(Tn)$ preprocess time and $\tilde{O}(1)$ query time.

Now consider the "frequent" elements in the array that occur more than T times. There are at most n/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 $\{i : a_i = x\}$, 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 $O(T)$, 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 $S_s, S_{s+1}, \ldots, S_t$ 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 $S_s, S_{s+1}, \ldots, S_t$, or some elements in $[l_i, r_i] \setminus \{S_s, S_{s+1}, \ldots, S_t\}$.

Suppose that the range mode of $[l_i, r_i]$ is not the range mode of $S_s, S_{s+1}, \ldots, S_t$. Then, we have a candidate list of $O(T)$ numbers (those to the left and right of $S_s, S_{s+1}, \ldots, S_t$ 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 $\tilde{O}(T)$ overhead to compute the range mode of $[l_i, r_i]$ once we know the range mode of $S_s, S_{s+1}, \ldots, S_t$. Thus, we can focus on the sub-problem of computing the range mode of the subinterval $S_s, S_{s+1}, \ldots, S_t$, where S_s is to the left of $n/2$, and S_t 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 heavy elements in $a_1, \ldots, a_n$. 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 $O(n/T)$ by $O(n/T)$ matrices.

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

Note that A, B are $O(n/T)$ by $O(n/T)$ matrices, each row of B is monotonically non-increasing, and the difference between the i-th column and $(i + 1)$-th column is at most T. Therefore, we can apply Lemma 6.1 to multiply $A \star B$ in $\tilde{O}((n/T)^{(14+\omega)/6} T^{1/6})$ time when $T = \Omega((n/T)^{(\omega-1)/2})$.

Therefore, the overall running time of the algorithm is $\tilde{O}((n/T)^{(14+\omega)/6} T^{1/6} + nT)$. By setting $T = n^{(8+\omega)/(19+\omega)}$, we get a $\tilde{O}(n^{(27+2\omega)/(19+\omega)})$ time algorithm. □

7 Application III: Maximum Subarray with Bounded Entries

In [22], Tamaki and Tokuyama reduced 2D maximum subarray problem to $(\min, +)$-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

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

The same property holds for B as well. If we can compute $(\min, +)$-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 7.1. *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 $\left|(\mathcal{D}^t B)_{i,j}\right| \leq M$ 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 7.1. *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 7.1. *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 7.1, 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 1.1 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 7.1. *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 [22], and apply Theorem 7.1 using $t = 1$ to immediately get this result. □

7.1 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. [6] 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 7.2 (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 7.3 (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 7.4 (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 7.5 (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. [6] 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 7.2. *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 $S := \{k \in [d] : k \not\equiv i \pmod{d}$ 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 7.3. *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 7.2, Lemma 7.3, together with the reduction from Central Maximum Subarray Sum to Maximum Subarray [6] imply Theorem 1.3.

7.2 Lower Bound for Maximum Subarray with Bounded Weight

In Section 7.1, we showed a tight conditional lower bound for d-Dimensional Maximum Subarray with real valued weights. In Section 4, 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 1.4. We notice that the reduction from Two-sided d-Uniform Max-Weight Hyperclique problem to Central Maximum Subarray Sum (Lemma 7.3), and the reduction from Central Maximum Subarray Sum to Maximum Subarray (presented in [6]) 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 1.4 follows from the following lemma.

Lemma 7.4. *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 7.2. 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.

□

References

[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] 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.

[5] Noga Alon, Troy Lee, Adi Shraibman, and Santosh Vempala. The approximate rank of a matrix and its algorithmic applications: Approximate rank. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC 2013, pages 675–684, 2013.

[6] 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.

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

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

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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

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

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

[23] Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\Delta}$ time, with applications. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 225–231. ACM, 2006.

[24] 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.

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

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

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

[28] 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.

[29] 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.

[30] 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.

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

A 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 A.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 $\left|A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j}\right| \le 4W$.*

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

Lemma A.1. *Any weakly relevant triple (i, k, j) is also approximately relevant.*

Proof. Consider

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

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

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

For any weakly relevant triple (i, k, j), $\left|A_{i,k} + B_{k,j} - \tilde{C}_{i,j}\right| \le 3W$ 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 A.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 $A_{i,k} + X_{k',j'}(k)^T Y_{k',j'}(j) - \tilde{C}_{i,j} \le 4W$, 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 $\begin{bmatrix} -A_{i,k} \\ X_{k',j'}(k) \end{bmatrix}^T x \le 4W$. 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.

B Limitation of the Reduction Path for Constant Weight Maximum Subarray

Our conditional lower bound in Section 7.2 first reduces a hardness problem to the Central Maximum Subarray Sum problem, and then to the Maximum Subarray problem. Backurs et al. [6] use a similar strategy in their reduction. Vassilevska W. and Williams [26] also have an intermediate problem in their reduction from Negative Triangle to 2D Maximum Subarray. This intermediate problem, similar to the Central Maximum Subarray Sum problem, also weights a subarray based on the values on the corner of the subarray.

The Central Maximum Subarray Sum problem, though nicely fits in all these previous reductions to Maximum Subarray, has a major limitation as the intermidiate problem: if the weights of the array are bounded integers, then there exists an $\tilde{O}(n^{2d-4+\omega})$ time algorithm that solves the Central Maximum Subarray Sum problem. It means that, in order to prove an lower bound larger than $n^{2d-4+\omega}$ for Maximum Subarray, we need to find some other alternative intermediate problem.

Claim 1. *Given a d-Dimensional Central Array A, such that all entries of A are integers bounded by some constant, there exists a $\tilde{O}(n^{2d-4+\omega})$ time algorithm that computes Central Maximum Subarray Sum of A.*

Proof Sketch. When $d > 2$, we can exhaustively enumerate all possible values of the first $d-2$ dimensions, and weights of the remaining 2D problem can be at most 2^{d-2} times larger than the original weights. When d is a constant, the resulting 2D problem also has entries bounded by a constant. Therefore, it is sufficient to show an $\tilde{O}(n^{\omega})$ algorithm for the 2D case.

The 2D case is similar to Tamaki et al.'s algorithm for 2D Maximum Subarray [22]. Using Min-Plus product for matrices with bounded integer weights, we can compute in $\tilde{O}(n^{\omega})$ time, for every $x_1 < 0 < x_2$,

$$1)\ d_{x_1,x_2} = \max_{y_1<0} \left\{A_{x_1,y_1} + A_{x_2,y_1}\right\}, \qquad 2)\ u_{x_1,x_2} = \max_{y_2>0} \left\{A_{x_1,y_2} + A_{x_2,y_2}\right\}.$$

Then the central maximum subarray sum of A is $\max_{x_1<0<x_2}\left(d_{x_1,x_2} + u_{x_1,x_2}\right)$. □