Tensor Ranks and the Fine-Grained Complexity of Dynamic Programming

Josh Alman
Columbia University, New York, NY, USA

Ethan Turok
Columbia University, New York, NY, USA

Hantao Yu
Columbia University, New York, NY, USA

Hengzhi Zhang
Columbia University, New York, NY, USA

Abstract

Generalizing work of Künnemann, Paturi, and Schneider [ICALP 2017], we study a wide class of high-dimensional dynamic programming (DP) problems in which one must find the shortest path between two points in a high-dimensional grid given a tensor of transition costs between nodes in the grid. This captures many classical problems which are solved using DP such as the knapsack problem, the airplane refueling problem, and the minimal-weight polygon triangulation problem. We observe that for many of these problems, the tensor naturally has low tensor rank or low slice rank.

We then give new algorithms and a web of fine-grained reductions to tightly determine the complexity of these problems. For instance, we show that a polynomial speedup over the DP algorithm is possible when the tensor rank is a constant or the slice rank is 1, but that such a speedup is impossible if the tensor rank is slightly super-constant (assuming SETH) or the slice rank is at least 3 (assuming the APSP conjecture).

We find that this characterizes the known complexities for many of these problems, and in some cases leads to new faster algorithms.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Problems, reductions and completeness; Theory of computation → Algebraic complexity theory

Keywords and phrases Fine-grained complexity, Dynamic programming, Least-weight subsequence

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.4

Related Version *Full Version*: `https://arxiv.org/abs/2309.04683` [4]

Funding Supported in part by NSF Grant CCF-2238221 and a grant from the Simons Foundation (Grant Number 825870 JA).

1 Introduction

Dynamic programming (DP) is one of the most common algorithmic paradigms, used throughout the theory and practice of diverse computational domains. See [16] chapter 14 for a detailed introduction.

When one solves a problem using DP, a natural question arises: is this the fastest algorithm to solve the problem? Recently, fine-grained complexity has been used to show that for many important problems, the answer is yes. For instance, researchers have established conditional lower bounds for the longest common subsequence [1, 13], edit distance [6], Fréchet distance [12], and regular expression matching [7], showing that there is no algorithm (whether or not it uses DP) that is faster than the standard DP algorithm by a polynomial factor.

On the other hand, there are some notable examples where a natural DP formulation is *not* the fastest known way to solve a problem. Consider, for instance, the polygon triangulation problem from computational geometry. In this problem, we are given as input a convex polygon with n nodes, where each node i has a weight w_i. For each triple i, j, k of nodes, a triangle with those nodes as vertices has weight $w_i \cdot w_j \cdot w_k$. The weight of a triangulation of the polygon is the sum of the weights of its constituent triangles. The goal in the problem is to find the triangulation of the polygon with minimum weight. This problem has applications in point visibility [23], mesh generation [8], computer graphics [38], and even in visual cryptography [41].

Polygon triangulation has a natural DP formulation as follows. Let $T[i, j]$ denote the minimum weight of a triangulation of the polygon consisting of just nodes $i, i+1, i+2, \ldots, j$ with an edge drawn between nodes i and j. Thus our goal is to compute $T[1, n]$, and these values satisfy the recurrence

$$T[i, j] = \min_{i<k<j} \Big\{ T[i, k] + T[k, j] + w_i \cdot w_j \cdot w_k \Big\}.$$

(Since there is an edge from i to j in the polygon, there must be a triangle involving those two nodes and a third node k; the recurrence corresponds to iterating over the choices of that third node.)

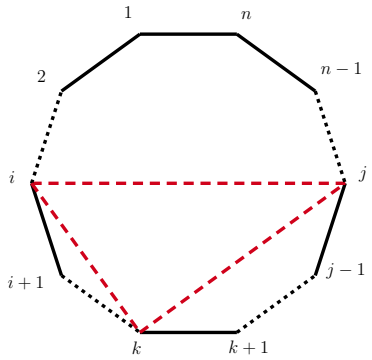


Figure 1 An example polygon triangulation problem. The polygon $P(i, j)$ is partitioned into 3 parts by choosing k and forming a triangle (i, j, k) whose weight is $w_i \cdot w_j \cdot w_k$.

This recurrence leads to a DP algorithm which solves the problem in time $O(n^3)$. However, Hu and Shing showed in [26, 27] that the problem can actually be solved much faster, in time $O(n \log n)$. This is a surprising example where geometric techniques lead to a faster algorithm than the natural DP formulation.

1.1 Least Weight Subsequence Problems

Künnemann, Paturi, and Schneider [33] initiated a general study of these phenomena. They studied a general class of problems intended to capture many one-dimensional DP problems called Least Weight Subsequence (LWS) problems: Given as input a positive integer n and an $n \times n$ matrix w, compute the value $T[n]$ defined by the recurrence:

$$T[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{0 \le i < j} \Big\{ T[i] + w[i, j] \Big\} & \text{otherwise.} \end{cases} \tag{1}$$

LWS was first introduced by Hirschbert and Lamore [24] to capture many known DP problems, including longest increasing subsequence [19], airplane refueling [24], coin change [33], nested boxes [33], pretty printing [31], and various knapsack problems.

For illustration purposes, consider the longest increasing subsequence (LIS) problem: given an array of n integers $X = [x_1, \ldots, x_n]$, return the length of the longest strictly increasing subsequence in X [19]. LIS can be formulated as an LWS problem by setting

$$w[i,j] = \begin{cases} -1 & \text{if } x_i < x_j \\ \infty & \text{otherwise.} \end{cases}$$

Notice that $w[i,j]$ equals negative one when x_i can be added to a subsequence which ends in x_j, thus increasing the length of a strictly increasing subsequence by 1. Since LIS is a maximization problem and LWS is a minimization problem, the weights are -1, not 1, and the solution is given by $-T[n]$. Many algorithmic problems can be formulated as an LWS instance by appropriately setting the weight matrix w.

Straightforward DP solves the LWS problem in $O(n^2)$ time. Since the input matrix w has $\Omega(n^2)$ entries, it requires quadratic time to read the input, so a faster algorithm isn't possible in general. However, if the w matrix is given in a smaller, compressed form, then one may hope for subquadratic-time algorithms.[1]

One example that [33] focuses on is the case when w is a low-rank matrix. If w has rank $r < n^{o(1)}$, then one can be given as input matrices $A, B \in \mathbb{R}^{n \times r}$ such that $w = A \times B^T$, so the input size to the problem is only $n^{1+o(1)}$.

Interestingly, Künnemann showed via fine-grained reductions that this problem is subquadratic-time *equivalent* to the well-studied Min-IP problem for vectors of dimension r: Given as input $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{R}^r$, find the x_i, y_j minimizing the inner product $\langle x_i, y_j \rangle$. This problem can be solved in time $O(n^{2-1/r})$ using geometric techniques [50, 36, 2], and thus has a truly-subquadratic time algorithm whenever r is a constant. On the other hand, it is known that assuming the Strong Exponential Time Hypothesis (SETH), the Min-IP problem requires time $n^{2-o(1)}$ even when r is slightly super-constant $r = 2^{\log^* n}$ [14], and thus the DP algorithm is essentially optimal. (Here $\log^*$ denotes the very slowly-growing iterated logarithm function.)

In this paper, we investigate the optimality of higher-dimensional DP formulations. We focus especially on generalizing LWS with low-rank matrices. As we will see, the known one-dimensional reductions do not generalize in a straightforward way, leading to a variety of generalizations and an intricate landscape of results. We will find that many classical higher-dimensional DP problems like the polygon triangulation problem are captured by our generalizations.

There are two choices to be made when generalizing LWS with low-rank matrices to higher dimensions: what is the higher-dimensional generalization of matrix rank (Section 1.2) and what is the higher-dimensional generalization of the LWS recurrence (Section 1.3).

[1] There has also been prior work on algorithms which assume w has some additional structure which does not mean w is compressible, but which lets one solve the problem without looking at most entries of w. For instance, [24] and [46] give $O(n \log n)$ and $O(n)$ time algorithms, respectively, for solving LWS with concave weights, i.e., when the entries of the matrix w are promised to satisfy a quadrangle inequality. See also [20, 29, 37].

1.2 Generalizations of matrix rank

The rank of a matrix has many equivalent definitions. However, when these definitions are generalized to higher-dimensional tensors, they lead to different notions. Prominent examples with applications in algebra, combinatorics, algorithm design, and complexity theory include the rank, subrank, border rank, slice rank, flattening rank, analytic rank, and geometric rank (see, e.g., [40, 32]). It is thus not clear, in general, which notion to use when generalizing results involving low-rank matrices.

We focus here on the two notions which arise naturally in known DP problems: tensor rank and slice rank.

1.2.1 Tensor Rank

A d-dimensional (order-d) tensor $w \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ has rank 1 if there are vectors $x_1 \in \mathbb{R}^{n_1}, \ldots, x_d \in \mathbb{R}^{n_d}$ such that, for all $i_1 \in [n_1], \ldots, i_d \in [n_d]$ we have $w[i_1, \ldots, i_d] = x_1[i_1] \cdot x_2[i_2] \cdots x_d[i_d]$. More generally, the rank of tensor w is the minimum non-negative integer k such that there are rank 1 tensors $w_1, \ldots, w_k$ for which $w = w_1 + \cdots + w_k$. This notion is sometimes also called canonical polyadic decomposition (CPD) rank.

For instance, in the polygon triangulation discussed earlier, the tensor w whose entry $w[i, j, k]$ gives the weight of triangle (i, j, k) has rank 1 because the weight of the triangle (i, j, k) is $w[i, j, k] = x_i \cdot x_j \cdot x_k$.

For another example, consider the airplane refueling problem: an airplane is traveling on a grid with dimension k such that each point in the grid is a refueling airport. The airplane starts at location $(1, \ldots, 1)$ and wants to arrive at location $(n, \ldots, n)$. The cost of flying from $(i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, \ldots, i_k)$ to $(i_1, \ldots, i_k)$ is $w[i_1, \ldots, i_k, j_\ell]$ (the airplane can only flies on the grid). The problem asks to minimize the cost of traveling.

One commonly studied cost of traveling from $(i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, \ldots, i_k)$ to $(i_1, \ldots, i_k)$ is $w[i_1, \ldots, i_k, j_\ell] = (k - (i_\ell - j_\ell))^2$ for a fixed constant k [24], which has rank 4 since

$$(k - (i_\ell - j_\ell))^2 = i_\ell^2 \cdot 1 + 1 \cdot j_\ell^2 + (i_\ell - k) \cdot (-2j_\ell) + (i_\ell - \frac{k}{2}) \cdot (-2k).$$

1.2.2 Slice Rank

A d-dimensional (order-d) tensor $w \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ has slice rank 1 if there is a $j \in [d]$, a vector $a \in \mathbb{R}^{n_j}$, and a $(d-1)$-dimensional tensor $b \in \mathbb{R}^{n_1 \times \cdots \times n_{j-1} \times n_{j+1} \times \cdots \times n_d}$ such that, for all $i_1 \in [n_1], \ldots, i_d \in [n_d]$ we have $w[i_1, \ldots, i_d] = a[i_j] \cdot b[i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_d]$. More generally, the slice rank of tensor w is the minimum non-negative integer k such that there are slice rank 1 tensors $w_1, \ldots, w_k$ for which $w = w_1 + \cdots + w_k$. Slice rank was recently introduced in the context of solving the cap set problem from extremal combinatorics [17, 44, 45, 11], but it has since found applications in algorithm design and complexity theory [11, 5, 3, 10, 9] and other areas of combinatorics [18, 39, 42, 35].

It is immediate that if a tensor w has rank d, then it has slice rank at most d. However, there are many natural situations where the slice rank of a tensor may be much lower than its rank, and we could hope to take advantage of this to design faster algorithms.

For example, another reasonable cost function for the airplane refueling problem is the one that depends only on the destinations, e.g., each airport charges a fee for landing at that airport. In this scenario, the cost function would have slice rank 1 but very large rank. We discuss the details in Section 5.1.

1.3 Higher-dimensional LWS recurrences

Many problems solvable by LWS can be naturally generalized to higher dimensions, which motivates us to study high dimensional version of the LWS recurrence. We focus on two new recurrences which capture most examples. The first, which we call kD LWS, is perhaps the most general.

▶ **Definition 1** (kD LWS). *For a positive integer k, consider $(k+1)$-dimensional tensors $w_1, \ldots, w_k$ of size $n \times n \times \cdots \times n$, where $w_\ell[i_1, \ldots, i_k, j] \in \{-W, \ldots, W, \infty\}$ for all $1 \leq \ell \leq k$. The kD LWS problem asks, given as input $w_1, \ldots, w_k$, to determine $T[n, \ldots, n]$ given the dynamic programming recurrence relation:*

$$T\Big[j_1, j_2, \ldots, j_k\Big] = \begin{cases} 0 & \text{if } j_1 = j_2 = \ldots = j_k = 1 \\ \min\limits_{1 \leq \ell \leq k} \Big\{ \min\limits_{1 \leq i_\ell < j_\ell} \Big\{ T\Big[j_1, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1}, \ldots, j_k\Big] + w_\ell\Big[j_1, j_2, \ldots, j_k, i_\ell\Big] \Big\} \Big\} & \text{otherwise.} \end{cases}$$

Intuitively, to compute $T[j_1, j_2, \ldots, j_k]$, we look at all *previous* terms in the table T that differ from $(j_1, j_2, \ldots, j_k)$ by *one* coordinate. For example, when $k = 2$, 2D LWS can be expressed as

$$T[i,j] = \begin{cases} 0 & \text{if } i = j = 1 \\ \min \begin{cases} \min\limits_{1 \leq k < i} \{T[k,j] + w_1[i,j,k]\} \\ \min\limits_{1 \leq k < j} \{T[i,k] + w_2[i,j,k]\} \end{cases} & \text{otherwise.} \end{cases}$$

kD LWS captures high-dimensional analogs of many previous problems solved by LWS and also some new problems which we discuss below. This includes higher dimensional airplane refueling (see Section 5.1 below), kMin-IP (Section 3.1), all-pairs shortest paths (Section 3.2), multiple nested box chains (Section 5.2), etc. An illustration of 2D LWS is shown in Figure 2.

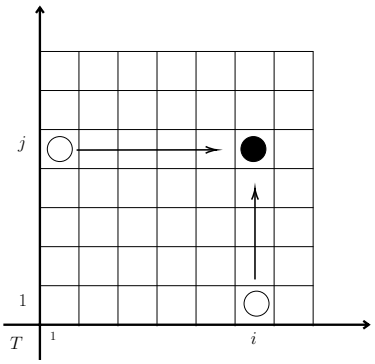


Figure 2 2D LWS. To compute $T[i,j]$, we take the minimum of all possible white circles (plus their respective tensor values w) such that their coordinates differ from the target by one coordinate.

Static kD LWS

Similar to [33], we also define a notion of "static" kD LWS in which we are given some entries in the DP table, and we would like to compute new entries which depend only on the given entries. The main idea of [Static]kD LWS is that we have the information $(T[i_1, \ldots, i_k])$ for

all $(i_1, \ldots, i_k)$ on a "band" $D_{a,a+N}$ and we want to compute $T[i_1, \ldots, i_k]$ for all $(i_1, \ldots, i_k)$ on the next "band" $D_{a+N,a+2N}$. A band $D_{\alpha,\beta}$ is defined to be all $(i_1, \ldots, i_k)$ such that their sum $i_1 + \cdots + i_k$ is in the interval $[\alpha, \beta)$.

▶ **Definition 2.** ([Static]kD LWS) *Given intervals $D_{a,a+N}, D_{a+N,a+2N}$ together with correctly computed values $T[i_1, \ldots, i_k]$ for all $1 \le \ell \le k$ and $(i_1, \ldots, i_k) \in D_{a,a+N}$,* [Static]kD LWS *asks to determine*

$$T'\Big[j_1, \ldots, j_k\Big] = \min_{1 \le \ell \le k} \left\{ \min_{a - I_\ell \le i_\ell < a+N-I_\ell} \left\{ T\Big[j_1, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1}, \ldots, j_k\Big] + w_\ell\Big[j_1, j_2, \ldots, j_k, i_\ell\Big] \right\} \right\}$$

for all $(j_1, j_2, \ldots, j_k) \in D_{a+N,a+2N}$.

For illustration purposes, consider the [Static]2D LWS problem: given correctly computed values $T[i, j]$ for all $(i, j) \in D_{a,a+N}$, determine

$$T'[i,j] = \min \begin{cases} \min_{a-i \le k < a+N-i} \{T[k, j] + w_1[i, j, k]\} \\ \min_{a-j \le k < a+N-j} \{T[i, k] + w_2[i, j, k]\} \end{cases}$$

for all $(i, j) \in D_{a+N,a+2N}$. Figure 3 depicts the idea.

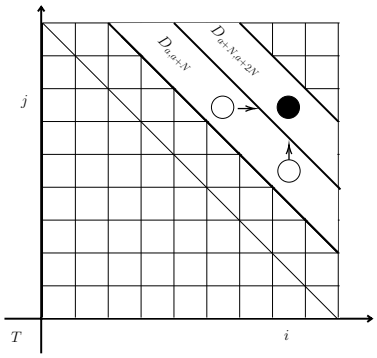


Figure 3 [Static]2D LWS. To calculate $T'[i, j]$ (black circle), we take the minimum over all possible white circles (plus their respective weight values w) such that they share all but one coordinate with $T[i, j]$.

[33] showed that in the $k = 1$ case, [Static]LWS is subquadratic-*equivalent* to the original LWS problem. We will find that the relationships among the higher-dimensional versions are more complicated.

1.4 Polygon Triangulation

kD LWS as we defined above captures many different high-dimensional DP problems, but it is not the only conceivable way to generalize LWS. We consider here another example we call 2D LWS$^{\mathsf{PT}}$, which captures optimization over sets that are counted by the Catalan numbers.

Recall that the Catalan numbers $C_0, C_1, C_2, \ldots$ can be defined recursively by $C_0 = 1$ and

$$C_n = \sum_{k=1}^{n} C_{k-1} \cdot C_{n-k}. \tag{2}$$

C_n counts many different combinatorial objects, such as the number of triangulations of a convex polygon with $n + 2$ vertices, and the number of binary trees with $n + 1$ leaves. (See,

e.g., [43].) The variable k being summed over in Equation (2) typically corresponds to ways to partition the combinatorial object into two smaller parts. This leads to our new definition, in which we want to instead *minimize* over all ways to partition the object:

▶ **Definition 3** (2D LWS$^{\text{PT}}$). *Given as input an $n \times n \times n$ tensor w, the* 2D LWS$^{\text{PT}}$ *problem asks to compute the value of $T[n,n]$ given the dynamic programming recurrence relation:*

$$T[i,j] = \begin{cases} 0 & \text{if } j - i \leq 1 \\ \min\limits_{i<k<j} \left\{ T[i,k] + T[k,j] + w[i,j,k] \right\} & \text{otherwise.} \end{cases}$$

For instance, this captures the polygon triangulation problem defined above when $w[i,j,k] = w_i \cdot w_j \cdot w_k$, which is unsurprising as polygon triangulations are counted by the Catalan numbers; this inspires the name 2D LWS$^{\text{PT}}$. We show in Section 6 below that this recurrence also captures other natural problems such as optimal binary search tree construction (optimizing over binary trees, which are counted by Catalan numbers) and matrix chain multiplication (optimizing over sequences of properly matched parentheses, again counted by Catalan numbers). Furthermore, in each of these examples, the rank (for polygon triangulation) or slice rank (for the other two examples) of w is 1.

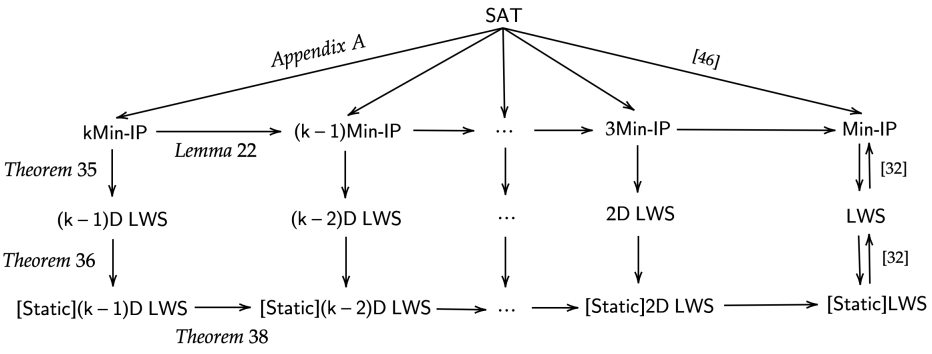
1.5 Main Results and Proof Structure Overview

Reduction notation. Before stating our results, we introduce one piece of notation for denoting the results of fine-grained reductions between problems with different running times. For computational problems P, Q, we say that P "reduces" to Q, denoted by $P \to Q$ if a polynomial speedup for Q yields a polynomial speedup for P. More precisely, suppose P, Q are solved in time T_p, T_q, respectively, via the straightforward algorithms. We say that P "reduces" to Q, denoted by $P \to Q$ if for every $\varepsilon > 0$ there exists a $\delta > 0$ such that, given a $O(T_q^{1-\varepsilon})$ time algorithm for Q, one gets a $O(T_p^{1-\delta})$ time algorithm for P.

For example, SAT $\to$ Min-IP$_{n,c\log n}$ means that if there is an algorithm for Min-IP$_{n,c\log n}$ with running time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$, then there is an algorithm for SAT with running time $O(2^{(1-\delta)n})$ for some $\delta > 0$. 3Min-IP $\to$ Min-IP means that if there is an algorithm for Min-IP with running time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$, then there is an algorithm for 3Min-IP with running time $O(n^{3-\delta})$ for some $\delta > 0$. When it may be unclear, the formal statements of our results are stated in the theorems below.

1.5.1 kD LWS Hierarchy and Hardness

We establish a hierarchy of kD LWS problems and describe their connections to kMin-IP, summarized by the following diagram.



In particular, assuming SETH, it follows that the straightforward algorithms for all the problems in the diagram cannot be substantially improved. The results depicted in this diagram, more precisely stated, are described by the following four theorems.

Building on Chen's reduction [14] from SAT to Min-IP$_{n,2^{O(\log^* n)}}$, we show that more generally, SAT also reduces to kMin-IP. (Note that kMin-IP reduces to (k − 1)Min-IP in a straightforward way, but a reduction in the other direction is not known.)

▶ **Theorem 4.** *Assuming* SETH*, there is no algorithm for* kMin-IP$_{n,2^{O(\log^* n)}}$ *running in time* $O(n^{k-\varepsilon})$ *for any* $\varepsilon > 0$*.*

Just as Min-IP reduces to LWS, we show that kMin-IP reduces to (k − 1)D LWS.

▶ **Theorem 5** (Theorem 35)**.** *Suppose there exists an algorithm for* (k − 1)D LWS *with rank* d *with running time* $O(n^{k-\varepsilon})$ *for some* $\varepsilon > 0$*, then there exists an algorithm for* kMin-IP *with rank* d *with running time* $O(n^{k-\delta})$ *for some* $\delta > 0$*.*

By a divide and conquer method similar to [33, Lemma 3.5], we show that kD LWS can be reduced to [Static]kD LWS.

▶ **Theorem 6** (Theorem 36)**.** *Suppose there exists an algorithm for* [Static]kD LWS$_{n,N,d}$ *with running time* $O(N^{2-\varepsilon} \cdot n^{k-1})$ *for some* $\varepsilon > 0$*, then there exists an algorithm for* kD LWS$_{n,d}$ *with running time* $O(n^{k+1-\delta})$ *for some* $\delta > 0$*.*

In addition, we show that [Static]kD LWS also exhibits a hierarchy similar to kMin-IP.

▶ **Theorem 7** (Theorem 38)**.** *Suppose there exists an algorithm for* [Static](k − 1)D LWS$_{n,N,d}$ *with running time* $O(N^{2-\varepsilon} \cdot n^{k-2})$ *for some* $\varepsilon > 0$*, then there exists an algorithm for* [Static]kD LWS$_{n,N,d}$ *with running time* $O(N^{2-\delta} \cdot n^{k-1})$ *for some* $\delta > 0$*.*

As one consequence of these reductions, [Static]kD LWS, kD LWS all reduce to Min-IP, which has a truly sub-quadratic algorithm when the vector length is constant, leading to a polynomial speedup:

▶ **Corollary 8.** *For every constant* $c > 0$ *there is an* $\varepsilon > 0$ *such that* kD LWS, [Static]kD LWS *can be solved in time* $O(n^{k+1-\varepsilon})$*,* $O(N^{2-\varepsilon} \cdot n^{k-1})$ *respectively if the rank of the tensor* w *is at most* c*.*

For one application of this corollary, we show in Section 5.1 that the generalized airplane refueling problem [24] in higher dimensions can be solved polynomially faster than the straightforward DP formulation.

Our reductions from SAT also give hardness of kD LWS assuming SETH, showing that the fast algorithm cannot extend much beyond constant rank:

▶ **Corollary 9.** *Under* SETH*, for any* $k > 1$ *and* $\varepsilon > 0$*, there is no algorithm running in time* $O(n^{k+1-\varepsilon})$ *for* kD LWS *when the weight tensor has rank* $2^{O(\log^* n)}$*.*

Since for rank r, the input size is nr, one could have imagined a better running time than $O(n^{k+1})$ for any $r \ll n^k$; our lower bound shows that it is (conditionally) impossible even for the slightly super-constant $r = 2^{\Omega(\log^* n)}$.

Note that the prior work [33] showed a subquadratic *equivalence* between LWS and its static version. We give a reduction from kD LWS to its static version, but not a reduction in the other direction; the techniques from prior work do not seem to generalize to the higher-dimensional setting.

1.5.2 Slice Rank in kD LWS

Slice rank is a another way to define tensor rank. If a tensor with rank d, then it trivially has slice rank at most d, which makes 2D LWS with slice rank more powerful. Indeed, we show that 2D LWS with slice rank becomes hard very quickly. Interestingly, this new hardness result builds on the APSP conjecture, rather than SETH; the APSP conjecture has not previously been connected to LWS problems to our knowledge.

▶ **Theorem 10** (Theorem 39). *Assuming the* APSP *conjecture, there is no truly sub-cubic algorithm for* 2D LWS *or* [Static]2D LWS *with slice rank* 3.

However, we can design a truly sub-cubic algorithm for 2D LWS with slice rank 1.

▶ **Theorem 11** (Theorem 41,Theorem 40). *There are truly sub-cubic time algorithms for* 2D LWS *and* [Static]2D LWS *with slice rank 1.*

We then use this to design faster algorithms for natural dynamic programming problems. For instance, we generalize the nested boxes problem defined in [33] to a multiple nested boxes problem and show that it can be formulated as kD LWS with slice rank 1, and thus it can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$.

▶ **Theorem 12** (Theorem 51). *Multiple nested boxes with dimension k can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$.*

We also show how to give a polynomial speedup for the airplane refueling problem in dimension k if the cost only depends on where the airplane lands, since that would mean the tensor has slice rank 1. We discuss the details in Section 5.1.

1.5.3 Hardness of Polygon Triangulation

We show similar algorithms and hardness for 2D LWS$^{\text{PT}}$.

▶ **Corollary 13** (Corollary 43). *Under* SETH*, there is no truly sub-cubic algorithm for* 2D LWS$^{\text{PT}}$ *with weight function whose rank is $2^{O(\log^* n)}$ or above.*

▶ **Corollary 14** (Corollary 45). *Under* APSP *conjecture, there is no truly sub-cubic algorithm for* 2D LWS$^{\text{PT}}$ *with weight function whose slice rank is* 3 *or above.*

These results are proved by making use of a reduction to 2D LWS$^{\text{PT}}$ from a *special case* of 2D LWS where the two tensors w_1, w_2 must be equal. We then show that our previous hardness results hold even in this special case, yielding the above corollaries.

In fact, previous work shows that in some special cases when w has rank 1, 2D LWS$^{\text{PT}}$ can be solved in truly sub-cubic time.

▶ **Theorem 15** ([26, 27, 25, 34]). *Suppose there exists $x_i \in \mathbb{N}, 1 \leq i \leq n$ such that $w[i,j,k] = x_i \cdot x_j \cdot x_k$ for all $1 \leq i,j,k \leq n$, then* 2D LWS$^{\text{PT}}$ *with tensor w can be solved in $O(n \log n)$ time.*

Our results help explain why the examples of 2D LWS$^{\text{PT}}$ problems where faster algorithms are known correspond to tensors w of rank or slice rank 1; see Section 6 for more details.

1.6 Organization

Section 2 contains the preliminaries: our notation, background on fine-grained complexity, and definitions of relevant problems. Section 3 discusses the kD LWS hierarchy and hardness, proving that a polynomial speedup over the standard DP algorithm is possible when the tensor rank is $O(1)$ but impossible when the tensor rank is $2^{O(\log^* n)}$ (assuming SETH). Section 4 discusses the polygon triangulation problem 2D LWS$^{\text{PT}}$ and its connections with 2D LWS. Sections 5 and 6 respectively discuss applications of kD LWS and 2D LWS$^{\text{PT}}$ to various real-world problems.

2 Preliminaries

In this section, we state our core problems and relevant problems in fine-grained complexity. We also state our notations for convenience.

For a problem P, we say that it is truly sub-quadratic (or sub-cubic) if there exists an algorithm for it with running time $O(n^{2-\varepsilon})$ (or $O(n^{3-\varepsilon})$) for any $\varepsilon > 0$. We say that P and Q are sub-quadratic (sub-cubic) equivalent if P is truly sub-quadratic (sub-cubic) if and only if Q is truly sub-quadratic (sub-cubic).

For a positive integer n, we let $[n] = \{1, \ldots, n\}$. We assume the word-RAM model with word size $\Theta(\log n)$ throughout this paper, and assume that all integer inputs are chosen from $\{-W, \ldots, W, \infty\}$ where W fits in a constant number of words. We usually use d to denote the length of vectors (rank of a tensor) in our problems, k to denote the dimension of our problems, and n to denote the input size. We put these parameters as subscripts of problems. Since we will discuss both rank and slice rank, we make it clear which we are talking about in the discussions. In this paper, "kD LWS has (slice) rank d" means the array w has (slice) rank d.

For a k-dimensional dynamic programming array T with entries $T[i_1, i_2, \ldots, i_k]$, we let I_ℓ denote the sum of all i_k's except i_ℓ. Let $D_{a,b}$ denote the set of all $(i_1, \ldots, i_k)$ such that $a \leq i_1 + \ldots + i_k < b$.

2.1 Strong Exponential Time Hypothesis and Min-IP

We state some important problems and definitions in fine-grained complexity that we will refer to later.

▶ **Conjecture 16** (Strong Exponential Time Hypothesis (SETH))**.** *For every $\varepsilon > 0$, there exists a positive integer k such that* kSAT *requires time $\Omega(2^{(1-\varepsilon)n})$.*

SETH is well-known for being a stronger version of P $\neq$ NP, i.e. it implies P $\neq$ NP.

▶ **Definition 17** ($\mathsf{OV}_{n,d}$)**.** *Given two sets of vectors $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}$ such that $a_i, b_j \in \{0,1\}^d$ for all i, j, the* Orthogonal Vectors *problem* ($\mathsf{OV}_{n,d}$) *asks to determine whether there exists $1 \leq i, j \leq n$ such that $\langle a_i, b_j \rangle = 0$.*

In [47], it is shown that assuming SETH, for any positive c there exists $\varepsilon > 0$ such that $\mathsf{OV}_{n,c\log n}$ cannot be solved in time $O(n^{2-\varepsilon})$.

▶ **Definition 18** (Min-IP$_{n,d}$)**.** *Given two sets of vectors $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}$ such that $a_i, b_j \in \{-W, \ldots, W, \infty\}^d$ for all i, j and a natural number $r \in \mathbb{Z}$, the* Minimal Inner Product (Min-IP$_{n,d}$) *asks to determine whether there exists $1 \leq i, j \leq n$ such that $\langle a_i, b_j \rangle \leq r$.*

$\mathsf{OV}_{n,d}$ trivially reduces to $\mathsf{Min\text{-}IP}_{n,d}$, and in fact these two problems are sub-quadratic equivalent [15]. It is known that this decision version of $\mathsf{Min\text{-}IP}_{n,d}$ is sub-quadratic equivalent to its counting version where it asks to output $\min_{1\le i,j\le n}\langle a_i, b_j\rangle$. We will use the counting version throughout the rest of the paper.

2.2 Higher Dimension OV, Min-IP

OV, Min-IP can be naturally generalize to higher dimensions as follows.

▶ **Definition 19** ($\mathsf{kOV}_{n,d}$). *Given k sets of vectors*

$$X_1 = \{x_{11}, \ldots, x_{1n}\}, \ldots, X_k = \{x_{k1}, \ldots, x_{kn}\}$$

such that $x_{ij} \in \{0,1\}^d$ for all i, j, $\mathsf{kOV}_{n,d}$ asks to determine whether there exists $1 \le i_1, \ldots, i_k \le n$ such that $\langle x_{1,i_1}, x_{2,i_2}, \ldots, x_{k,i_k}\rangle = 0$.

▶ **Definition 20** ($\mathsf{kMin\text{-}IP}_{n,d}$). *Given k sets of vectors*

$$X_1 = \{x_{11}, \ldots, x_{1n}\}, \ldots, X_k = \{x_{k1}, \ldots, x_{kn}\}$$

such that $x_{ij} \in \{-W, \ldots, W, \infty\}^d$ for all i, j and a natural number $r \in \mathbb{Z}$, $\mathsf{kMin\text{-}IP}_{n,d}$ asks to determine whether there exists $1 \le i_1, \ldots, i_k \le n$ such that $\langle x_{1,i_1}, x_{2,i_2}, \ldots, x_{k,i_k}\rangle \le r$.

Just from the definitions, kOV trivially reduces to kMin-IP. In addition, it is not hard to show $\mathsf{kOV}_{n,d} \to (\mathsf{k}-1)\mathsf{OV}_{n,d}$ and $\mathsf{kMin\text{-}IP}_{n,d} \to (\mathsf{k}-1)\mathsf{Min\text{-}IP}_{n,d}$ for all $k \ge 3$.

▶ **Lemma 21.** *Suppose there exists an algorithm for $(\mathsf{k}-1)\mathsf{OV}_{n,d}$ that runs in time $O(n^{k-1-\varepsilon})$ for some $\varepsilon > 0$, then there exists an algorithm for $\mathsf{kOV}_{n,d}$ that runs in time $O(n^{k-\delta})$ for some $\delta > 0$.*

Proof. Given an $\mathsf{kOV}_{n,d}$ instance with sets $X_1, \ldots, X_k$, we trivially compute $\langle x_{1,i_1}, x_{2,i_2}\rangle$ for all $1 \le i_1, i_2 \le n$ using time $O(n^2 d)$. Now for each $1 \le i_1 \le n$, run the algorithm for $(\mathsf{k}-1)\mathsf{OV}_{n,d}$ with $x_{1,i_1} \cdot X_2, X_3, \ldots, X_k$. If there are no zeros then output no; otherwise output yes.

This algorithm is correct because we have covered all possible $\langle x_{1,i_1}, x_{2,i_2}, \ldots, x_{k,i_k}\rangle$. The running time is $O(n^2 d) + O(n^{k-1-\varepsilon}) \cdot n = O(n^{k-1-\varepsilon})$. ◀

▶ **Lemma 22.** *Suppose there exists an algorithm for $(\mathsf{k}-1)\mathsf{Min\text{-}IP}_{n,d}$ that runs in time $O(n^{k-1-\varepsilon})$ for some $\varepsilon > 0$, then there exists an algorithm for $\mathsf{kMin\text{-}IP}_{n,d}$ that runs in time $O(n^{k-\delta})$ for some $\delta > 0$.*

Proof. The proof is exactly the same as the proof of Lemma 21. ◀

2.3 All Pair Shortest Path

All Pair Shortest Path (APSP) is a well-known problem in fine grained complexity and graph theory.

▶ **Definition 23** (APSP). *Given a directed graph G with nodes $V = \{v_1, \ldots, v_n\}$, the* All Pair Shortest Path (APSP) *asks to determine the distance between v_i and v_j for all $1 \le i < j \le n$.*

Currently there is no algorithm for APSP that runs in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$, and it is conjectured that no such algorithm exists.

▶ **Conjecture 24** (APSP Conjecture). *There is no truly sub-cubic algorithm for* APSP.

It is known that APSP is sub-cubic equivalent to many problems such that min-plus matrix multiplication, negative triangle etc [49]. Therefore, assuming APSP conjecture, none of these problems are truly sub-cubic. We will use the hardness of these problems to obtain our hardness results.

▶ **Definition 25** ((min, +)MM). *Given two $n \times n$ matrices A, B, compute its min-plus product C where*

$$C[i,j] = \min_{1 \le k \le n} \{A[i,k] + B[k,j]\}$$

for all $1 \le i, j \le n$.

▶ **Definition 26** (NegativeTriangle). *Given an undirected, weighted graph, determines whether there exists a triangle such that its weight (the sum of weights of three sides) is negative.*

2.4 Least Weight Subsequence

We formally define LWS using the definition provided in [33].

▶ **Definition 27** (LWS). *Consider a sequence of n data items $x_1, \ldots, x_n$ and a weight matrix w of size $n \times n$ where $w[i,j] \in \{-W, \ldots, W, \infty\}$ for all $1 \le i < j \le n$. The* LWS *problem asks to determine $T[n]$, which is defined by the following DP formulation:*

$$T[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{0 \le i < j} \Big\{ T[i] + w[i,j] \Big\} & \text{otherwise.} \end{cases}$$

Given a sequence of n items, LWS computes a subsequence of those items which minimizes the total weight from the items chosen. We assume all the entries of w can be accessed in $O(1)$ time. Figure 4 captures the idea of LWS.

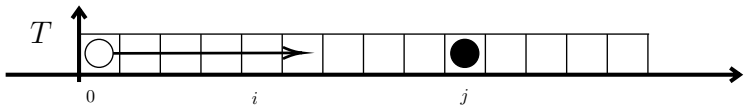


Figure 4 LWS. To compute the value of $T'[j]$ (black circle), we start from $T[0]$ and go through all possible $T[i]$ such that $1 \le i < j \le n$ and takes the minimum of all possible values (plus their respective weight values w).

[33] also defines a "static" version of LWS which is central to their reductions.

▶ **Definition 28.** ([Static]LWS) *Fix an instance of* LWS *with matrix w. Given intervals $I = \{a, a+1, \ldots, a+N-1\}, J = \{a+N, a+N+1, \ldots, a+2N-1\}$, together with correctly computed values $T[i]$ for all $i \in I$, the* [Static]LWS *problem asks to determine*

$$T'[j] = \min_{i \in I} \Big\{ T[i] + w[i,j] \Big\} \qquad \text{for all } j \in J.$$

[Static]LWS is a parallel, batch version of LWS that applies the LWS recurrence relation to many values of j at once, rather than to just a single j value at a time. Indeed, we can compute all $T'[j]$ values where $j \in J$ in parallel because each $T'[j]$ only depends on the $T[i]$ values where $i \in I$, not on any other $T'[j]$ values.

We use the notation T' to highlight that $T'[j]$ may not equal $T[j]$ since $T'[j]$ is computed with partial information (in I). Figure 5 captures the idea of [Static]LWS.

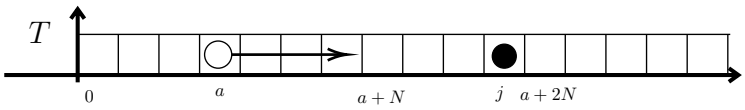


Figure 5 [Static]LWS. To compute $T'[j]$ (black circle), we take the minimum of all possible white circles from $T[a]$ to $T[a+N-1]$ (plus their respective weight values w).

2.5 Higher Dimension LWS

We now define the core problems that this paper discusses again, which is a generalization of LWS to higher dimensions.

▶ **Definition 29** (kD LWS). *Fix a positive integer k. Consider $(k+1)$-dimensional tensors $w_1, \ldots, w_\ell$ such that $w_\ell[i_1, \ldots, i_k, j] \in \{-W, \ldots, W, \infty\}$ for all $1 \leq i_1, \ldots, i_k, j \leq n, 1 \leq \ell \leq k$. The* kD LWS *problem asks to determine $T[n, \ldots, n]$ given the dynamic programming recurrence relation:*

$$T\Big[j_1, j_2, \ldots, j_k\Big] = \begin{cases} 0 & \text{if } j_1 = j_2 = \ldots = j_k = 1 \\ \min\limits_{1 \leq \ell \leq k} \Big\{ \min\limits_{1 \leq i_\ell < j_\ell} \Big\{ T\Big[j_1, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1}, \ldots, j_k\Big] + w_\ell\Big[j_1, j_2, \ldots, j_k, i_\ell\Big] \Big\} \Big\} & \text{otherwise.} \end{cases}$$

An illustration of 2D LWS can be found in Figure 2.

Like [33], we also define [Static]kD LWS, a generalization of [Static]LWS to higher dimensions which is central to our reductions.

▶ **Definition 30.** ([Static]kD LWS) *Given intervals $D_{a,a+N}, D_{a+N,a+2N}$ together with correctly computed values $T_\ell[i_1, \ldots, i_k]$ for all $1 \leq \ell \leq k$ and $(i_1, \ldots, i_k) \in D_{a,a+N}$,* [Static]kD LWS *asks to determine*

$$T'\Big[j_1, \ldots, j_k\Big] = \min_{1 \leq \ell \leq k} \left\{ \min_{a - I_\ell \leq i_\ell < a+N-I_\ell} \Big\{ T_\ell\Big[j_1, \ldots, j_{\ell-1}, i_\ell, j_{\ell+1}, \ldots, j_k\Big] + w_\ell\Big[j_1, j_2, \ldots, j_k, i_\ell\Big] \Big\} \right\}$$

for all $(j_1, j_2, \ldots, j_k) \in D_{a+N,a+2N}$.

An illustration of [Static]2D LWS can be found in Figure 3.

2.6 Tensor Ranks

We give definitions of rank and slice rank for tensors. For notational convenience, we say that a problem has rank d if its associated array/tensor has rank d.

▶ **Definition 31** (Rank). *We say that a k-dimension array w has* rank d *if there exists k sets*

$$X_1 = \{x_{11}, \ldots, x_{1n}\}, \ldots, X_k = \{x_{k1}, \ldots, x_{kn}\}$$

of vectors with length d such that $w[i_1, \ldots, i_k] = \langle x_{1,i_1}, x_{2,i_2}, \ldots, x_{k,i_k} \rangle$ for all $1 \leq i_1, \ldots, i_k \leq n$.

▶ **Definition 32** (Slice Rank). *A k-dimensional (order-k) tensor $w \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ has slice rank 1 if there is a $j \in [d]$, a vector $a \in \mathbb{R}^{n_j}$, and a $(d-1)$-dimensional tensor $b \in \mathbb{R}^{n_1 \times \cdots \times n_{j-1} \times n_{j+1} \times \cdots \times n_d}$ such that, for all $i_1 \in [n_1], \ldots, i_d \in [n_d]$ we have*

$$w[i_1, \ldots, i_d] = a[i_j] \cdot b[i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_d].$$

More generally, the slice rank of tensor w is the minimum non-negative integer k such that there are slice rank 1 tensors $w_1, \ldots, w_k$ for which $w = w_1 + \cdots + w_k$.

2.7 Polygon Triangulation

In computational geometry, a polygon triangulation is a partition of a polygon P into triangles. It is known that the number of partitions of a convex polygon is counted by the Catalan numbers [43]. We discuss the problem of finding the triangulation that minimizes the sum of weights of all triangles.

▶ **Definition 33** (Polygon Triangulation). *Let $P(n)$ denote an n-sided convex polygon and fix an ordering of the polygon vertices. A triangulation of $P(n)$ is a partition of $P(n)$ into disjoint triangles using straight, internal edges between pair of nodes of $P(n)$. For each triangle (v_i, v_j, v_k), let $w(i,j,k)$ be its weight, and the weight of a partition is the sum of all weights of its triangles. The polygon triangulation problem asks to determine the minimal weight of all partitions.*

For an example, consider the polygon triangulation of a polygon P with 10 sides (see Figure 6). Starting from the side $(1, 10)$, we choose a node 6 and partitions P into 3 parts $P(1, 6)$, triangle $(1, 6, 10)$ and $P(6, 10)$, denoted by the black dashed lines. We further partitions $P(i, j)$ by choosing a node $i < k < j$ and partitions it to $P(i, k)$, triangle (i, j, k) and $P(k, j)$.

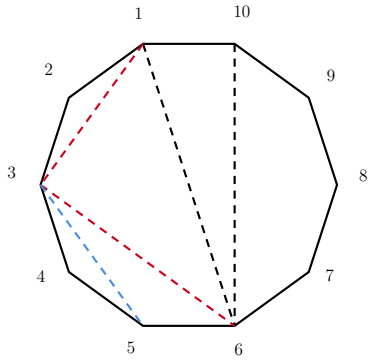


Figure 6 An example polygon triangulation problem.

The polygon triangulation problem can be solved via dynamic programing, motivating our 2D LWS$^{\text{PT}}$ problem.

▶ **Definition 34** (2D LWS$^{\text{PT}}$). *Fix a tensor w. The* 2D LWS$^{\text{PT}}$ *problem asks to compute the value of $T[n, n]$ given the dynamic programming recurrence relation:*

$$T[i,j] = \begin{cases} 0 & \text{if } i + j \leq n + 2 \\ \min\limits_{0<k<i+j-n} \left\{ T[n-j+k, j] + T[i, j-k] + w[i,j,k] \right\} & \text{otherwise.} \end{cases}$$

Under a change of variables/coordinates, this problem is equivalent to computing the value of $T[1,n]$ *given the dynamic programming recurrence relation:*

$$T[i,j] = \begin{cases} 0 & \text{if } j-i \leq 1 \\ \min\limits_{i<k<j} \left\{ T[i,k] + T[k,j] + w[i,j,k] \right\} & \text{otherwise.} \end{cases}$$

It is not hard to see that polygon triangulation and 2D LWS$^{\text{PT}}$ are the same problem: let $T[i,j]$ denote the weight of the sub-polygon containing nodes i to j and $w[i,j,k]$ be the weight of triangle (v_i, v_j, v_k). More generally, any problem which splits an interval $[i,j]$ at some point k where k is between i and j can be understood as a 2D LWS$^{\text{PT}}$ instance. Figure 7 captures the idea of 2D LWS$^{\text{PT}}$.

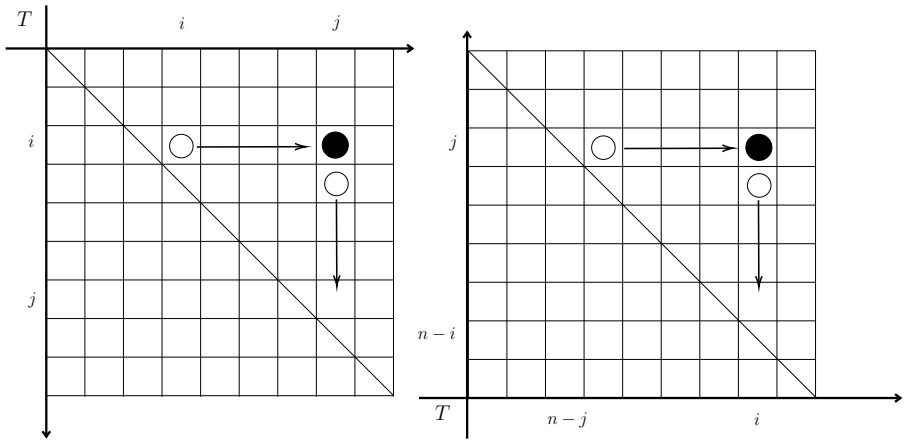
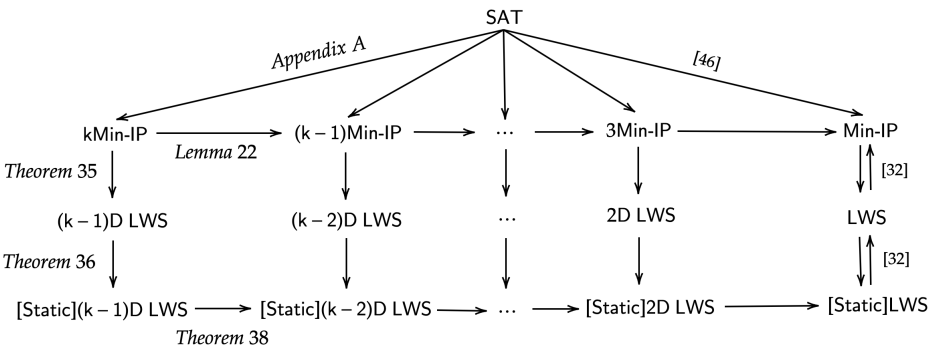


Figure 7 2D LWS$^{\text{PT}}$. To compute $T[i,j]$ (black circle), we are taking the minimum over the sum of all possible **pairs** of white circles (plus their respective weight values w). The solution to 2D LWS$^{\text{PT}}$ is found at $T[1,n]$ in the left figure and at $T[n,n]$ in the right figure due to a change of variables.

3 k-dimensional Least Weight Subsequence (kD LWS)

In this section, we discuss kD LWS with rank and slice rank respectively. For kD LWS with rank d, we prove the reductions in the following diagram. Due to page limit, we omit the proofs of our statements. Our full version [4] contains all the proofs needed.



All our reductions preserve the rank of the problem, so this diagram shows that there exists truly sub-cubic algorithm for kD LWS and [Static]kD LWS when the rank is constant (because Min-IP does [48]). In addition, we show that kMin-IP reduces to kD LWS and [Static]kD LWS, so their hardness are established. Our proof of SAT reducing to kMin-IP mimics the proof of SAT reducing to Min-IP in [14].

In addition, we show that 2D LWS, [Static]2D LWS with slice rank 3 or above is APSP-hard, and give truly sub-cubic algorithms for 2D LWS, [Static]2D LWS with slice rank 1.

3.1 Rank d [Static]kD LWS Hierarchy

In this section we establish a hierarchy for kD LWS and [Static]kD LWS with rank d.

▶ **Theorem 35.** (kMin-IP $\rightarrow$ (k $-$ 1)D LWS) *Suppose there exists an algorithm for* (k $-$ 1)D LWS *with rank d with running time $O(n^{k-\varepsilon})$ for some $\varepsilon > 0$, then there exists an algorithm for* kMin-IP *with rank d with running time $O(n^{k-\delta})$ for some $\delta > 0$.*

▶ **Theorem 36.** (kD LWS $\rightarrow$ [Static]kD LWS) *Suppose there exists an algorithm for* [Static]kD $\mathsf{LWS}_{n,N,d}$ *with running time $O(N^{2-\varepsilon} \cdot n^{k-1})$ for some $\varepsilon > 0$, then there exists an algorithm for* kD $\mathsf{LWS}_{n,d}$ *with running time $O(n^{k+1-\delta})$ for some $\delta > 0$.*

Since SAT reduces to $\mathsf{kMin\text{-}IP}_{n,2^{O(\log^* n)}}$, we immediately get hardness results of kD LWS and [Static]kD LWS from SETH:

▶ **Corollary 37.** *Assuming* SETH*, there is no $O(n^{k+1-\varepsilon})$ time algorithm for* kD LWS *or* [Static]kD LWS *with rank at least $2^{O(\log^* n)}$ for any $\varepsilon > 0$.*

Finally, we show that just like kMin-IP, [Static]kD LWS also exhibits a hierarchy.

▶ **Theorem 38.** ([Static]kD LWS $\rightarrow$ [Static](k $-$ 1)D LWS) *Suppose there exists an algorithm for* [Static](k $-$ 1)D $\mathsf{LWS}_{n,N,d}$ *with running time $O(N^{2-\varepsilon} \cdot n^{k-2})$ for some $\varepsilon > 0$, then there exists an algorithm for* [Static]kD $\mathsf{LWS}_{n,N,d}$ *with running time $O(N^{2-\delta} \cdot n^{k-1})$ for some $\delta > 0$.*

3.2 Slice Rank 2D LWS

In this section, we show that 2D LWS, [Static]2D LWS with even slice rank 3 is APSP-hard, but 2D LWS, [Static]2D LWS with slice rank 1 is truly sub-cubic.

▶ **Theorem 39.** *Assuming the* APSP *conjecture, there is no truly sub-cubic algorithm for* 2D LWS *with slice rank* 3*.*

Now we show that [Static]2D LWS with slice rank 1 can be solved in truly sub-cubic time (which implies 2D LWS with slice rank 1 is also truly sub-cubic) but there is no truly sub-cubic algorithm for 2D LWS with slice rank 3 or above assuming APSP conjecture.

▶ **Theorem 40.** [Static]2D $\mathsf{LWS}_{n,N}$ *with slice rank* 1 *can be solved in $O(n^2 \cdot N^{1-\varepsilon})$ for some $\varepsilon > 0$.*

▶ **Theorem 41.** 2D LWS *with slice rank 1 is truly sub-cubic.*

Proof. Immediately follows from Theorem 40 and the fact that our reduction from 2D LWS to [Static]2D LWS in Theorem 36 preserves slice rank. ◀

4 Polygon Triangulation (2D LWS$^{\mathsf{PT}}$)

In this section, we discuss the polygon triangulation problem 2D LWS$^{\mathsf{PT}}$ and its connections with 2D LWS. It was shown in [25, 26, 27, 34], that if $w[i,j,k] = x_i \cdot x_j \cdot x_k$ for all i,j,k with $x_i, x_j, x_k > 0$ for all i, then 2D LWS$^{\mathsf{PT}}$ can be solved in $O(n^2)$ time. We establish several conditional hardness results of 2D LWS$^{\mathsf{PT}}$ based on SETH and APSP conjecture. Namely, 2D LWS where $w_1 = w_2$ can be reduced to 2D LWS$^{\mathsf{PT}}$ with rank/slice rank unchanged, and thus finding the optimal triangulation for certain weight functions (rank $2^{O(\log^* n)}$ or slice rank 3) is hard under SETH.

Due to page limit we are unable to include the proofs. Our full version [4] contains all the proofs needed.

4.1 Low Rank Polygon Triangulation is SETH-hard

▶ **Theorem 42.** (2D LWS $\to$ 2D LWS$^{\mathsf{PT}}$). *There exists an $O(nd)$ time reduction from* 2D LWS$_n$ *with rank d, $w_1 = w_2 = w$, to* 2D LWS$^{\mathsf{PT}}_{2n}$ *with rank d.*

▶ **Corollary 43.** *Under* SETH*, there is no truly sub-cubic algorithm for* 2D LWS$^{\mathsf{PT}}$ *with weight function whose rank is $2^{O(\log^* n)}$ or above.*

Proof. When we reduce 3Min-IP to 2D LWS in Theorem 35, the tensors w_ℓ that we use are all the same, so Theorem 35 immediately gives a reduction from 3Min-IP to 2D LWS with $w_1 = w_2$ (preserving rank), which further reduces to 2D LWS$^{\mathsf{PT}}$ (preserving rank) by Theorem 42. ◀

4.2 Constant Slice Rank Polygon Triangulation is APSP-hard

In fact, the we can modify the reduction in Theorem 42 such that it preserved slice rank as well.

▶ **Theorem 44.** (2D LWS $\to$ 2D LWS$^{\mathsf{PT}}$*, slice rank version*) *There exists an $O(nd)$ time reduction from* 2D LWS$_n$ *with slice rank d, $w_1 = w_2 = w$, to* 2D LWS$^{\mathsf{PT}}_{2n}$ *with slice rank d.*

▶ **Corollary 45.** *Under* APSP *conjecture, there is no truly sub-cubic algorithm for* 2D LWS$^{\mathsf{PT}}$ *with weight function whose slice rank is* 3 *or above.*

Proof. When we reduce APSP to 2D LWS in Theorem 39, the tensors α that we use are the same, so Theorem 39 immediately gives a reduction from APSP to 2D LWS with $w_1 = w_2$, which further reduces to 2D LWS$^{\mathsf{PT}}$ (preserving slice rank) by Theorem 42. ◀

5 Applications of kD LWS

In Section 3, we have shown that kD LWS can solve kMin-IP and APSP with different tensors. In this section we discuss more applications of kD LWS.

5.1 Higher Dimension Airplane Refueling

The airplane refueling problem was brought by [24] as an example of LWS.

▶ **Definition 46** (Airplane Refueling)**.** *Suppose an airplane needs to fly between 2 given airports which are distance R apart. Suppose there are $n-1$ different refueling stops at distance $x_1, \ldots, x_{n-1}$ from the departure point and all stops lie on the segment between departure*

and destination points. We can let $0 = x_0 < x_1 < \ldots < x_n = R$. *The cost of flying* ℓ *miles is* $(k - \ell)^2$ *for some* $k > 0$ *(we prefer flying close to* k *miles), and the goal is to fly from departure point to arrival point while minimizing the cost.*

It is not hard to see setting $w[i, j] = (x_j - x_i - k)^2$ in LWS will solve the problem since $T[j]$ is always the minimum cost of flying from x_0 to x_j. w has rank 4 because

$$w[i, j] = x_j^2 \cdot 1 + 1 \cdot x_i^2 + (-2x_j) \cdot (x_i + 2k) + k \cdot (2x_i + k),$$

and [24] shows that airplane refueling can be solved in linear time.

In the real world, it is usually unlikely that all refueling stops are located on a single line. In addition, the plane can move in multiple directions. The higher dimension airplane refueling problem is motivated by these observations.

▶ **Definition 47** (Higher Dimension Airplane Refueling). *Suppose an airplane needs to fly between two given airports on a* k*-dimensional grid with* n *points at each dimension. Each point in the grid represents a refueling stop, and the cost of flying from stop* $(i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, \ldots, i_k)$ *to* $(i_1, \ldots, i_k)$ *to is* $c(i_1, \ldots, i_k, j_\ell)$. *The problem asks the minimum cost of flying from* $(1, \ldots, 1)$ *to* $(n, \ldots, n)$.

Notice that this is closer to real-world scenario where trains need to travel on railways, or we are driving in a city with well-organized roads.

Setting $w[i_1, \ldots, i_{k+1}] = c[i_1, \ldots, i_{k+1}]$ in kD LWS will solve the problem because $T[i_1, \ldots, i_k]$ will always be the minimum cost of flying from $(1, \ldots, 1)$ to $(i_1, \ldots, i_k)$. If we were to follow the cost function suggested in [24], then we have $c[i_1, \ldots, i_k, j_\ell] = (L - (i_\ell - j_\ell))^2$, which has constant rank and thus it can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$.

Another natural scenario is that the cost of flying from $(i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, i_k)$ to $(i_1, \ldots, i_k)$ only depends on $(i_1, \ldots, i_k)$. It mimics the scenario that the airplane is charged a fee upon arrival.

▶ **Definition 48** (Arrival Fee Airplane Refueling). *Suppose an airplane needs to fly between two given airports on a* k*-dimensional grid with* n *points at each dimension. Each point in the grid represents a refueling stop, and the cost of flying from stop* $(i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, \ldots, i_k)$ *to* $(i_1, \ldots, i_k)$ *to is* $c(i_1, \ldots, i_k)$. *The problem asks the minimum cost of flying from* $(1, \ldots, 1)$ *to* $(n, \ldots, n)$.

In the arrival fee airplane refueling problem with dimension k, the tensor has slice rank 1, so by Theorem 40 it can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$.

5.2 Multiple Nested Boxes

Nested boxes problem, or box stacking problem, is a famous example with a DP solution.

▶ **Definition 49** (Nested Boxes). *Given* n *boxes in* d *dimension of size* $(b_1, \ldots, b_d)$, *find the longest chain such that each box fits into the next (without rotation). We say that box* a *of size* $(a_1, \ldots, a_d)$ *fits into box* b *of size* $(b_1, \ldots, b_d)$ *if* $a_i \le b_i$ *for all* $1 \le i \le d$.

[33] proves that nested boxes is sub-quadratic equivalent to the vector domination problem defined in [28] and both can be solved by LWS: sort the boxes by volumn in increasing order as $B_1, \ldots, B_n$ and set w_{ij} to be -1 if B_j contains B_i and 0 otherwise.

It is natural to consider the case where we are allowed to have multiple locations to put the boxes, which motivates our multiple nested boxes problem.

▶ **Definition 50** (Multiple Nested Boxes). *Given n boxes in d dimension of size $(b_1, \ldots, b_d)$ and k piles, find the maximum number of boxes we can use such that each in each pile, each box fits into the next (without rotation). We say that box a of size $(a_1, \ldots, a_d)$ fits into box b of size $(b_1, \ldots, b_d)$ if $a_i \leq b_i$ for all $1 \leq i \leq d$.*

▶ **Theorem 51.** *Multiple nested boxes with k dimension can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$.*

Proof. We first sort all the boxes by their volume in increasing order $B_1, \ldots, B_n$, and let

$$w_\ell[i_1, \ldots, i_k, j] = \begin{cases} -1 & \text{if } B_j \text{ fits into } B_{i_\ell} \\ 0 & \text{otherwise.} \end{cases}$$

To see that this indeed solves multiple nested boxes, we claim that $-T[i_1, \ldots, i_k]$ is the maximum number of boxes over all assignments such that the rank of the outer box in t-th pile is at most i_t. This will solve the multiple nested boxes when $i_1 = \ldots = i_k = n + 1$.

We proceed by induction. When $i_1 = \ldots = i_k = 1$, we have $T[i_1, \ldots, i_k] = 0$. There is no box with rank 0 so we cannot put any boxes. Now for general $(i_1, \ldots, i_k)$, by recurrence we have

$$T\Big[i_1, \ldots, i_k\Big] = \min_{1 \leq \ell \leq k} \Big\{ \min_{1 \leq j_\ell < i_\ell} \Big\{ T\Big[i_1, \ldots, i_{\ell-1}, j_\ell, i_{\ell+1}, i_k\Big] + w_\ell\Big[i_1, \ldots, i_k, j_\ell\Big] \Big\} \Big\}.$$

For any assignment $(u_1, \ldots, u_k)$ to the piles such that the t-th pile outer box has rank $u_t \leq i_t$, it can be achieved from adding B_{u_ℓ} to the assignment $(u_1, \ldots, u_{\ell-1}, u'_\ell, u_{\ell+1}, \ldots, u_k)$ with the guarantee that $B_{u'_\ell}$ fits inside B_{u_ℓ}. This case is covered by the right-hand-side of the equation because by induction hypothesis,

$$-T\Big[u_1, \ldots, u_{\ell-1}, u'_\ell, u_{\ell+1}, \ldots, u_k\Big] - w_\ell\Big[u_1, \ldots, u_k, u'_\ell\Big]$$

is the maximum over the assignments under this procedure.

Therefore, right-hand-side of the equation is exactly all possible ways to achieve the assignment $(u_1, \ldots, u_k)$ such that $u_t \leq i_t$ for all t, so our kD LWS instance indeed solves the multiple nested boxes with k dimension. In addition, notice that w_ℓ only depends on its ℓ-th and last coordinate, so it can be expressed as a matrix. The same reasoning in Theorem 40 shows that it can be reduced to [Static](k − 1)D LWS with rank 1, which implies that it can be solved in time $O(n^{k+1-\varepsilon})$ for some $\varepsilon > 0$. ◀

6 Applications of 2D LWS^{PT}

In section 4, we showed we can solve polygon triangulation and 2D LWS^{PT} with different tensors. In this section, we discuss more applications of 2D LWS^{PT}.

6.1 Matrix-Chain Multiplication

The matrix-chain multiplication problem was introduced in [22] and is defined as follows:

▶ **Definition 52** (Matrix-Chain Multiplication). *Given a chain of n matrices $A_1, \ldots, A_n$ where matrix A_i has dimension $d_{i-1} \times d_i$, find the order of matrix multiplications which minimizes the number of scalar multiplications using the straightforward matrix multiplication algorithm.*

Recall that multiplying an $n \times m$ matrix by an $m \times p$ matrix using the straightforward algorithm uses $n \cdot m \cdot p$ scalar multiplications. Moreover, the order in which you multiply a chain of matrices determines the number of scalar multiplications performed. For instance, consider three matrices A_1, A_2, A_3 with dimensions $(10, 20)$, $(20, 30)$, and $(30, 40)$ respectively. Multiplying $(A_1, A_2)A_3$ takes $(10 \cdot 20 \cdot 30) + (10 \cdot 30 \cdot 40) = 18000$ scalar multiplications while multiplying $A_1(A_2A_3)$ takes $(20 \cdot 30 \cdot 40) + (10 \cdot 20 \cdot 40) = 32000$ scalar multiplications.

Matrix-chain multiplication is a 2D LWS$^{\mathsf{PT}}$ problem where $T[i, j]$ is the cost of multiplying matrices $A_i, \ldots, A_j$ and we want to find the k which minimizes the cost of multiplying $(A_i, \ldots, A_k)$ by $(A_{k+1}, \ldots, A_j)$. Multiplying matrices $(A_i, \ldots, A_k)$ would result in a matrix of dimension (d_{i-1}, d_k) and multiplying $(A_{k+1}, \ldots, A_j)$ would result in a matrix of dimension (d_k, d_j). Thus $T[i, j]$ equals the cost of multiplying all matrices $A_i, \ldots, A_k$ (i.e. $T[i, k]$) and all matrices $A_{k+1}, \ldots, A_j$ (i.e. $T[k, j]$) plus the cost of multiplying those two resultant matrices together (i.e. $d_{i-1}d_kd_j$). Setting $w_1[i, j, k] = w_2[i, j, k] = d_{i-1}d_kd_j$ would solve this problem.

Let us construct a vector $d = [d_0, d_1, \ldots, d_n]$ with the dimensions of our matrices $A_1, \ldots, A_n$. Then w has a tensor rank of 1 because it can be represented as the product of different entries of d, namely $w[i, j, k] = d[i - 1] \cdot d[j] \cdot d[k]$. Moreover, there exists an $O(n \log n)$ time algorithm for this problem [26, 27]. Corollary 43 helps explain why this speedup is possible.

6.2 Optimal Binary Search Tree

The optimal binary search tree construction problem was introduced in [30, 21] and is defined as follows:

▶ **Definition 53** (Optimal Binary Search Tree). *Given a sequence of n distinct keys $h_1, \ldots, h_n$ in sorted order where the probability of accessing key h_i is p_i, construct a binary search tree from these keys which minimizes the expected access time.*

This problem is a 2D LWS$^{\mathsf{PT}}$ instance where $T[i, j]$ is the minimum cost binary search tree with keys $h_i, \ldots, h_j$. We want to chose a key h_k to be the root of the sub-tree containing keys $h_i, \ldots, h_j$ which minimizes the expected access time. The expected access time for a key h_t is $p_t \cdot (d_t + 1)$, the key's probability times its depth in the tree (i.e. the number of times this item has been accessed). We can compute this quantity incrementally, adding the probability p_t of key h_t once at every level it appears in the tree, summing up p_t a total of d_t times. Thus the expected cost of accessing keys $h_i, \ldots, h_j$ is $w[i, j, k] = \sum_{t=i}^{j} p_t$.

w has a slice rank of 1 because it can be written as $w[i, j, k] = a[k] \cdot b[i, j]$ where $a[k] = 1$ and $b[i, j] = \sum_{t=i}^{j} p_t$. This observation, together with Corollary 45, recovers the known $O(n^2)$ time algorithm for this problem [51, 50].

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78. IEEE, 2015.

2 Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 203–210, 1990.

3 Josh Alman. Limits on the universal method for matrix multiplication. *Theory Of Computing*, 17(1):1–30, 2021.

4 Josh Alman, Ethan Turok, Hantao Yu, and Hengzhi Zhang. Tensor ranks and the fine-grained complexity of dynamic programming, 2023. arXiv:2309.04683.

5 Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. *SIAM Journal on Computing*, 0(0):FOCS18–285, 2021.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58, 2015.

7 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE, 2016.

8 Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean geometry*, pages 47–123. World Scientific, 1995.

9 Markus Bläser, Christian Ikenmeyer, Vladimir Lysikov, Anurag Pandey, and Frank-Olaf Schreyer. On the orbit closure containment problem and slice rank of tensors. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2565–2584. SIAM, 2021.

10 Markus Bläser and Vladimir Lysikov. Slice rank of block tensors and irreversibility of structure tensors of algebras. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

11 Jonah Blasiak, Thomas Church, Henry Cohn, Joshua A Grochow, Eric Naslund, William F Sawin, and Chris Umans. On cap sets and the group-theoretic approach to matrix multiplication. *Discrete Analysis*, 3, 2017.

12 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97, 2015.

14 Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. *Theory Of Computing*, 16(4):1–50, 2020.

15 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–40. SIAM, 2019.

16 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

17 Ernie Croot, Vsevolod F Lev, and Péter Pál Pach. Progression-free sets in are exponentially small. *Annals of Mathematics*, pages 331–337, 2017.

18 Jordan S Ellenberg and Dion Gijswijt. On large subsets of with no three-term arithmetic progression. *Annals of Mathematics*, pages 339–343, 2017.

19 Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

20 Zvi Galil and Raffaele Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science*, 64(1):107–118, 1989.

21 Edgar N Gilbert and Edward F Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

22 Sadashiva S Godbole. On efficient computation of matrix chain products. *IEEE Transactions on Computers*, 100(9):864–866, 1973.

23 John Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4(1-4):141–155, 1989.

24 D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987. doi:10.1137/0216043.

25 T.C. Hu and M. Shing. *Combinatorial Algorithms: T.C. Hu and M.T. Shing*. Dover Books on Computer Science Series. Dover Publications, 2002.

26 TC Hu and MT Shing. Computation of matrix chain products. part i. *SIAM Journal on Computing*, 11(2):362–373, 1982.

27 TC Hu and MT Shing. Computation of matrix chain products. part ii. *SIAM Journal on Computing*, 13(2):228–251, 1984.

28 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *CoRR*, abs/1401.5512, 2014. `arXiv:1401.5512`.

29 Maria M. Klawe and Daniel J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM Journal on Discrete Mathematics*, 3(1):81–97, 1990. `doi:10.1137/0403009`.

30 DE Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971.

31 Donald E Knuth and Michael F Plass. Breaking paragraphs into lines. *Software: Practice and Experience*, 11(11):1119–1184, 1981.

32 Swastik Kopparty, Guy Moshkovitz, and Jeroen Zuiddam. Geometric rank of tensors and subrank of matrix multiplication. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

33 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

34 Thong Le and Dan Gusfield. Matrix chain multiplication and polygon triangulation revisited and generalized. *arXiv preprint*, 2021. `arXiv:2104.01777`.

35 László Miklós Lovász and Lisa Sauermann. A lower bound for the k-multicolored sum-free problem in z mn. *Proceedings of the London Mathematical Society*, 119(1):55–103, 2019.

36 Jiří Matoušek. Efficient partition trees. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 1–9, New York, NY, USA, 1991. Association for Computing Machinery.

37 Webb Miller and Eugene W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50(2):97–120, 1988.

38 Atul Narkhede and Dinesh Manocha. Fast polygon triangulation based on seidel's algorithm. In *Graphics Gems V*, pages 394–397. Elsevier, 1995.

39 Eric Naslund and Will Sawin. Upper bounds for sunflower-free sets. In *Forum of Mathematics, Sigma*, volume 5, page e15. Cambridge University Press, 2017.

40 Giorgio Ottaviani and Philipp Reichenbach. Tensor rank and complexity. *arXiv preprint*, 2020. `arXiv:2004.01492`.

41 Muzafer Saracevic, Predrag Stanimirovic, Sead Mašovic, and Enver Biševac. Implementation of the convex polygon triangulation algorithm. *Facta Universitatis, series: Mathematics and Informatics*, 27(2):213–228, 2012.

42 Will Sawin. Bounds for matchings in nonabelian groups. *The Electronic Journal of Combinatorics*, 25(4):4–23, 2018.

43 Richard P Stanley. *Catalan numbers*. Cambridge University Press, 2015.

44 Terence Tao. A symmetric formulation of the crootlev-pach-ellenberg-gijswijt capset bound. *Tao's blog post*, 2016.

45 Terence Tao and Will Sawin. Notes on the "slice rank" of tensors. *Tao's blog post*, 2016.

46 Robert Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.

47 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004).

48 Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1207–1215. SIAM, 2018.

49 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5), August 2018.

50 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, November 1982.

51 F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Symposium on the Theory of Computing*, 1980. URL: `https://api.semanticscholar.org/CorpusID:3154717`.