

Non-Boolean OMv: One More Reason to Believe Lower Bounds for Dynamic Problems

Bingbing Hu 

University of California San Diego, La Jolla, CA, USA

Adam Polak 

Bocconi University, Milan, Italy

Abstract

Most of the known tight lower bounds for dynamic problems are based on the Online Boolean Matrix-Vector Multiplication (OMv) Hypothesis, which is not as well studied and understood as some more popular hypotheses in fine-grained complexity. It would be desirable to base hardness of dynamic problems on a more believable hypothesis. We propose analogues of the OMv Hypothesis for variants of matrix multiplication that are known to be harder than Boolean product in the offline setting, namely: equality, dominance, min-witness, min-max, and bounded monotone min-plus products. These hypotheses are a priori weaker assumptions than the standard (Boolean) OMv Hypothesis and yet we show that they are actually equivalent to it. This establishes the first such fine-grained equivalence class for dynamic problems.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Complexity classes

Keywords and phrases Fine-grained complexity, OMv hypothesis, reductions, equivalence class

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.54

Funding Part of this work was done when both authors were at Max Planck Institute for Informatics.

Acknowledgements We thank anonymous reviewers for pointing us to the literature on the cell-probe model and for many useful hints on improving the manuscript.

1 Introduction

The job of a dynamic algorithm is to keep its output up to date whenever its input undergoes a local change, e.g., maintaining a shortest s - t path in a graph while it undergoes vertex deletions. Ideally each such update should take at most a polylogarithmic time, but at the very least it should be faster than it takes to recompute a solution from scratch. Despite great progress in the field, for many dynamic problems that goal is beyond the reach of current algorithmic techniques. Starting from the seminal paper by Pătraşcu [19], we often get to explain this hardness by fine-grained conditional lower bounds.

Most of the known *tight* lower bounds for dynamic problems are based on the OMv Hypothesis [11]. This hypothesis is not as widely studied and as well understood as some other hypotheses in fine-grained complexity, such as SETH, 3SUM Hypothesis, and APSP Hypothesis (see, e.g., [22]). It would be more desirable to base hardness of dynamic problems on these more popular (and hence also more believable) assumptions. Unfortunately, the existing lower bounds conditional on them are often not tight for dynamic problems. It seems likely that these hypotheses are not strong enough to explain the complexity of many dynamic problems. We may need to search for a different approach to the following glaring question:

*Can we have tight lower bounds for dynamic problems
based on a hypothesis that is more believable than OMv?*



© Bingbing Hu and Adam Polak;
licensed under Creative Commons License CC-BY 4.0
33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 54; pp. 54:1–54:16



Leibniz International Proceedings in Informatics

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

Recall that the OMv Hypothesis is about *Boolean product*; it asserts that computing the Boolean product of two $n \times n$ matrices requires cubic $n^{3-o(1)}$ time if the second matrix is given column by column in an online fashion. In the static (i.e., non-online) setting, Boolean product is arguably the easiest of the many studied variants of matrix multiplication. Indeed, it can be computed in time $O(n^\omega)$, where $\omega < 2.372$ [2] is the (integer) matrix multiplication exponent.¹

In the static matrix product world, if the $O(n^\omega)$ running time is on the “fast” end of the spectrum, then the min-plus product (related to distance computations in graphs) marks the other end: the fastest known algorithm shaves only a subpolynomial factor over the naive cubic running time [26], and the APSP Hypothesis from fine-grained complexity essentially says that no $n^{3-o(1)}$ -time algorithm is possible [23].

There are also numerous variants of matrix multiplication that seem to have an “intermediate” hardness on this spectrum. Examples include min-max product [21, 9], min-witness product [8], equality product (a.k.a. Hamming product [18]), dominance product [17], threshold product [12], plus-max product [20], ℓ_{2p+1} product [13], and many others. The fastest known algorithms for these problems have running times that are functions of the matrix multiplication exponent ω , and they converge to $O(n^{2.5})$ when $\omega = 2$. Although it is still an open problem whether this is necessarily the right complexity for all these problems, there are some partial results in the form of tight fine-grained reductions that suggest it might be the case [13, 15, 24].

1.1 Our contributions

The OMv Hypothesis (and the lower bounds it implies) would be a priori more believable if we could replace in its statement the Boolean product with some other product known to be harder in the static world. For instance, we can define in a similar way the Min-Max-OMv problem using the min-max product: Pre-process a matrix $M \in \mathbb{Z}^{n \times n}$, and then answer (one by one, in an online fashion) n queries, each of them asking, given a vector $v \in \mathbb{Z}^n$, to compute the min-max product of M and v , i.e., the vector $u \in \mathbb{Z}^n$ such that

$$u[i] := \min_{k \in [n]} \max\{M[i, k], v[k]\}.$$

Then we can state a corresponding hypothesis, let us call it the Min-Max-OMv Hypothesis, asserting that the Min-Max-OMv problem cannot be solved in truly subcubic time $O(n^{3-\varepsilon})$, for any $\varepsilon > 0$. This of course brings a question:

Can we still give tight reductions from Min-Max-OMv to those dynamic problems for which there are known reductions from (Boolean-)OMv?

It turns out, yes, we can! Somewhat surprisingly², we can even give a tight reduction from Min-Max-OMv to Boolean-OMv. This shows that the Min-Max-OMv Hypothesis and the standard (Boolean-)OMv Hypothesis are actually equivalent. Moreover, the min-max product is not a unique example of this phenomenon. We show more equivalent hypotheses based on several matrix products, which are harder than the Boolean product in the static setting (see Section 2 for the formal definitions).

¹ Moreover, the fastest known “combinatorial” algorithm for the Boolean product [1] does not give a similar improvement for the integer product.

² This is perhaps less surprising to those readers who are familiar with how the subcubic algebraic algorithms for the intermediate problems work; see Section 1.2.

► **Theorem 1.** *The following problems either all have truly subcubic algorithms or none of them do:*

- *Boolean-OMv;* $(\exists_k M[i, k] \wedge v[k])$
- *\exists Equality-OMv;* $(\exists_k M[i, k] = v[k])$
- *\exists Dominance-OMv;* $(\exists_k M[i, k] \leq v[k])$
- *Min-Witness-OMv;* $(\min \{k \mid M[i, k] \wedge v[k]\})$
- *Min-Max-OMv;* $(\min_k \max \{M[i, k], v[k]\})$
- *Bounded Monotone Min-Plus-OMv.* $(\min_k M[i, k] + v[k])$

For the Bounded Monotone Min-Plus-OMv problem the implied algorithm is randomized.

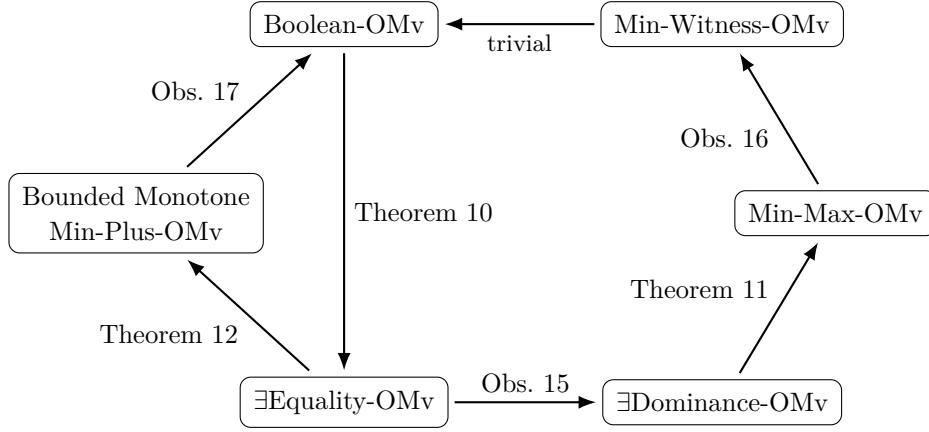
This conglomeration of equivalent problems can be interpreted as making the OMv Hypothesis more believable, and the conditional lower bounds based on it stronger. We recall two analogous conglomerations: the NP-complete problems and the problems equivalent to the All-Pairs Shortest Paths (APSP) problem under subcubic reductions. One of the reasons behind the great success of the theory of NP-completeness is its structural simplicity: many natural problems are NP-complete, and solving any of them efficiently would solve all of them efficiently, so they are all hard for the same underlying reason. For all the multi-faceted NP-complete problems, researchers from different areas have not managed to find a single efficient algorithm, so it seems very plausible that no such algorithm exists. The fine-grained complexity theory at large does not enjoy a similar simplicity: there are multiple independent hardness assumptions, and the reductions often go only in one way, establishing *hardness* but not *equivalence*. A notable exception³ is the APSP problem, which is conjectured to require cubic time and there are many other problems equivalent to it via subcubic reductions [23]. No truly subcubic algorithms have been found so far for any of these problems, which strengthens the APSP Hypothesis. Our Theorem 1 establishes another such class of problems equivalent under fine-grained subcubic reductions.

Subcubic algorithms in the cell-probe model. We remark that the above equivalences also hold in the cell-probe model, where the running time is measured solely by the number of memory cell probes, and computation is free of charge. To be more specific, from our proofs it follows that each of the above problems can be solved by solving $t^{O(1)}$ instances of any other of those problems and performing some additional computation that runs in time $O(n^3/t)$, in the word RAM model, where t is a parameter to be chosen. Since any algorithm that runs sequentially in time $O(n^{3-\epsilon})$ in the word RAM model can access only a truly subcubic number of memory cells, the subcubic equivalence in the cell-probe model follows. Larsen and Williams [14] showed that the OMv Hypothesis is actually *false* in the cell-probe model: Every query can be computed with $O(n^{7/4}w^{-1/2})$ probes, where w is the word size. This means that all the other problems listed above have truly subcubic algorithms in the cell-probe model as well.

1.2 Technical overview

We prove Theorem 1 by a series of fine-grained reductions, depicted in Figure 1. The reductions are inspired by known subcubic algebraic algorithms for the corresponding (static) matrix product problems. Such algorithms have running times that are functions of the (integer) matrix multiplication exponent ω . For example, the fastest known (static) min-max

³ Another such exception is a class of vector problems equivalent to the Orthogonal Vectors problem [7].



■ **Figure 1** Fine-grained reductions that together prove Theorem 1. An arrow from problem A to problem B means that a subcubic algorithm for A implies a subcubic algorithm for B.

product algorithm [9] runs in time $O(n^{(3+\omega)/2+o(1)})$, which is subcubic for any value of $\omega < 3$. In other words, this algorithm is an implicit subcubic reduction from the (static) min-max product to the (static, integer) matrix multiplication problem. The same is true about all the other intermediate problems we consider in this paper. Our technical contribution is adapting these implicit reductions so that (1) they work in the online setting, and (2) they require only Boolean (and not integer) product.

Detour into static combinatorial algorithms. Before we say a few words about each of the reductions in the paper, let us take a detour from online problems so that we can explicitly state the observation that we already hinted at in the previous paragraph.

► **Observation 2.** *The static variants of all the matrix products listed in Theorem 1 are equivalent to each other under subcubic reductions, and the reductions are “combinatorial”. That is, if any of the static problems admits a subcubic “combinatorial” algorithm, then all of them do.*

Why is this the case? In one direction, all those non-Boolean products can encode the Boolean product as a special case. In the other direction, all those non-Boolean products admit strongly subcubic algorithms with running times of the form $O(n^{f(\omega)})$ for a function f that happens to satisfy $f(x) < 3$ for every $x < 3$. Therefore, if we could replace in such an algorithm every use of fast algebraic (integer) matrix multiplication with a hypothesized $O(n^{3-\varepsilon})$ -time “combinatorial” Boolean matrix multiplication algorithm, we would get a “combinatorial” algorithm for the corresponding non-Boolean product with the running time exponent $f(3 - \varepsilon) < 3$. Such a replacement is possible whenever the non-Boolean algorithm does not use (or can be modified not to use) the full counting power of integer matrix multiplication but only checks which output entries are nonzero. This is explicitly the case for some of the non-Boolean products we consider (e.g., the min-witness product [8]). For some others, e.g., the bounded monotone min-plus product [25, 10], such a modification is nontrivial, but still possible.

While this observation is simple and in hindsight not really surprising, we are not aware of it being folklore in the field of fine-grained complexity, so we state it for completeness.

Subcubic reductions and where to find them. In Section 3 we show a subcubic reduction from \exists Equality-OMv to Boolean-OMv (Theorem 10), which can be seen as an adaptation to the online setting of the sparse matrix multiplication algorithm of Yuster and Zwick [27]. Specifically, our algorithm for \exists Equality-OMv handles the $n^{\varepsilon/2}$ most frequently appearing values with the help of $n^{\varepsilon/2}$ Boolean-OMv queries (about indicator vectors of those values). The remaining less frequent values can then be handled with a brute-force approach within the target running time bound.

In Section 4 we show how a subcubic algorithm for \exists Dominance-OMv would yield a subcubic algorithm for Min-Max-OMv (Theorem 11). The proof is inspired by the known (static) min-max product algorithms [21, 9], but it is at the same time simpler, because we do not have to optimize the dependence on ε in the running time of the resulting algorithm. In short, the algorithm splits each row of M into $n^{\varepsilon/2}$ buckets based on the values of the entries, and makes $n^{\varepsilon/2}$ many \exists Dominance-OMv queries in order to establish for each output entry to which bucket of the corresponding row it belongs; then, the algorithm exhaustively searches through the entire bucket.

In Section 5 we show a reduction from Bounded Monotone Min-Plus-OMv to \exists Equality-OMv. On a high level it follows some of the previous (static) algorithms for the bounded monotone min-plus product [25, 10]. However, it also gives a fresh perspective on the problem, because those previously known algorithms use a generalization of the min-witness and bounded (non-monotone) min-plus products (see [25, Theorem 1.2]), while ours deviates from this approach by using the equality product.

In Section 6 we show the remaining reductions (Observations 15, 16, 17). Each of them is either very simple or follows easily from folklore arguments.

1.3 Related work

Bringmann et al. [4] take a different approach at strengthening the OMv Hypothesis. They propose a hypothesis about the complexity of determining if a (nondeterministic) finite automaton accepts a word, and show that this hypothesis implies the OMv Hypothesis. While their new hypothesis is not as well supported as the three main fine-grained complexity hypotheses, it is remarkable that it is a statement about a *static* problem implying a tight lower bound for an online problem.

In a very recent work, Liu [16] shows that OMv is equivalent to the online problem of maintaining a $(1 + \epsilon)$ -approximate vertex cover in a fully dynamic bipartite graph.

To the best of our knowledge, the only other work that considers a variant of OMv for a non-Boolean product is by Chen et al. [6]. They use an assumption that the Min-Plus-OMv requires cubic time in order to separate partially retroactive from fully retroactive data structures. We note that this assumption seems too strong to be equivalent to the OMv Hypothesis. In particular, any “too simple” reduction from Min-Plus-OMv to Boolean-OMv would morally translate to a subcubic algorithm for the (static) Min-Plus Product problem, refuting the APSP Hypothesis.

1.4 Open problems

In this paper we manage to reduce to Boolean-OMv from OMv variants that *do not involve counting*. We leave it open whether a subcubic algorithm for Boolean-OMv would imply subcubic OMv algorithms for, e.g., the counting variants of the equality and dominance products (i.e., $u[i] := \#\{k \mid M[i, k] = v[k]\}$, and $u[i] := \#\{k \mid M[i, k] \leq v[k]\}$, respectively), or at least for the standard integer product ($u[i] := \sum_k M[i, k] \cdot v[k]$).

These open problems relate to the general quest for fine-grained *counting-to-decision* reductions. Chan, Vassilevska Williams, and Xu [5] gave such reductions for the Min-Plus Product, Exact Triangle, and 3SUM problems. Somewhat ironically, their reductions crucially rely on a fast algebraic algorithm for (static) integer matrix multiplication, so it seems unlikely that their techniques could be used to resolve the above open problems, which are about online problems.

2 Preliminaries

2.1 Notation

We use $[n] := \{1, 2, \dots, n\}$.

2.2 Problems

In this section we formally define all the problems that appear in Theorem 1. Since the definitions are similar to each other, we underline the differences between them.

► **Definition 3** (Boolean-OMv). We are first given for preprocessing a Boolean matrix $M \in \{0, 1\}^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in \{0, 1\}^n$, and we have to compute the Boolean product $Mv_j \in \{0, 1\}^n$ defined by

$$(Mv_j)[i] := \begin{cases} 1, & \text{if } \exists_{k \in [n]} M[i, k] = 1 \wedge v_j[k] = 1, \\ 0, & \text{otherwise.} \end{cases}$$

.....

We need to answer queries one by one in an online fashion, i.e., we have to output Mv_j before we can receive v_{j+1} .

► **Definition 4** (\exists Equality-OMv). We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the \exists equality product $M \ominus v_j \in \{0, 1\}^n$ defined by

$$(M \ominus v_j)[i] := \begin{cases} 1, & \text{if } \exists_{k \in [n]} M[i, k] = v_j[k], \\ 0, & \text{otherwise.} \end{cases}$$

.....

We need to answer queries one by one in an online fashion, i.e., we have to output $M \ominus v_j$ before we receive v_{j+1} .

► **Definition 5** (\exists Dominance-OMv). We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the \exists dominance product $M \odot v_j \in \{0, 1\}^n$ defined by

$$(M \odot v_j)[i] := \begin{cases} 1, & \text{if } \exists_{k \in [n]} M[i, k] \leq v_j[k], \\ 0, & \text{otherwise.} \end{cases}$$

.....

We need to answer queries one by one in an online fashion, i.e., we have to output $M \odot v_j$ before we receive v_{j+1} .

► **Definition 6 (Min-Witness-OMv).** We are first given for preprocessing a Boolean matrix $M \in \{0,1\}^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in \{0,1\}^n$, and we have to compute the min-witness product $M \otimes v_j \in ([n] \cup \{\infty\})^n$ defined by

$$(M \otimes v_j)[i] := \min(\{k \in [n] \mid M[i, k] = 1 \wedge v_j[k] = 1\} \cup \{\infty\}).$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \otimes v_j$ before we can receive v_{j+1} .

► **Definition 7 (Min-Max-OMv).** We are first given for preprocessing an integer matrix $M \in \mathbb{Z}^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in \mathbb{Z}^n$, and we have to compute the min-max product $M \odot v_j \in \mathbb{Z}^n$ defined by

$$(M \odot v_j)[i] := \min_{k \in [n]} \max\{M[i, k], v_j[k]\}.$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \odot v_j$ before we receive v_{j+1} .

► **Definition 8 (Bounded Monotone Min-Plus-OMv).** We are first given for preprocessing an integer matrix $M \in [n]^{n \times n}$, and then we need to answer n queries: In the j -th query, we are given a column vector $v_j \in [n]^n$, and we have to compute the min-plus product $M \oplus v_j \in \mathbb{Z}^n$ defined by

$$(M \oplus v_j)[i] := \min_{k \in [n]} (M[i, k] + v_j[k]).$$

We need to answer queries one by one in an online fashion, i.e., we have to output $M \oplus v_j$ before we receive v_{j+1} . We are guaranteed that at least one of the following conditions holds:

- each row of M is nondecreasing, i.e., $M[i, k] \leq M[i, k+1]$;
 - each column of M is nondecreasing, i.e., $M[i, k] \leq M[i+1, k]$;
 - each v_j is nondecreasing, i.e., $v_j[k] \leq v_j[k+1]$;
 - for every k , $v_j[k]$ is a nondecreasing function of j , i.e., $v_j[k] \leq v_{j+1}[k]$;
- or it holds with “nondecreasing” replaced by “nonincreasing”.

2.3 Hypotheses

Each of the problems defined above admits a naive cubic time algorithm, and for each of them we can conjecture that it is optimal up to subpolynomial factors.

► **Definition 9 (*-OMv Hypotheses).** For $x \in \{\text{Boolean}, \exists\text{Equality}, \exists\text{Dominance}, \text{Min-Witness}, \text{Min-Max}, \text{Bounded Monotone Min-Plus}\}$, the x -OMv Hypothesis is the statement that there is no algorithm for the x -OMv problem running in time $O(n^{3-\varepsilon})$, for any $\varepsilon > 0$.

In other words, Theorem 1 says that all the hypotheses stated in Definition 9 are equivalent.

3 Reduction from \exists Equality-OMv to Boolean-OMv

In this section we give a subcubic reduction that can be seen as an easy adaptation to the online setting of the sparse matrix multiplication algorithm of Yuster and Zwick [27].

► **Theorem 10.** *If Boolean-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then \exists Equality-OMv can be solved in time $O(n^{3-(\varepsilon/2)})$.*

Proof. Recall that M denotes the input matrix given for preprocessing in the \exists Equality-OMv problem. Let $t := \lceil n^{\varepsilon/2} \rceil$ be a parameter. For every $k \in [n]$ and every $\ell \in [t]$, let $f_k^{(\ell)}$ be the ℓ -th most frequent value appearing in the k -th column of matrix M (if there are less than ℓ distinct values in the column, let $f_k^{(\ell)}$ be some other arbitrary integer). Note that for any value x not in $\{f_k^{(1)}, f_k^{(2)}, \dots, f_k^{(t)}\}$, x appears in the k -th column of M at most n/t times; we call such values *rare*. In the preprocessing phase, the algorithm prepares t Boolean matrices $M^{(1)}, M^{(2)}, \dots, M^{(t)} \in \{0, 1\}^{n \times n}$ defined as follows:

$$M^{(\ell)}[i, k] := \begin{cases} 1, & \text{if } M[i, k] = f_k^{(\ell)}, \\ 0, & \text{otherwise.} \end{cases}$$

Then, it instantiates the hypothesized Boolean-OMv algorithm for each of these matrices separately. Finally, for each column of M , the algorithm prepares a dictionary mapping each rare value in that column to a list of indices under which that value appears in the column. This ends the preprocessing phase.

Upon receiving a query $v \in \mathbb{Z}^n$, the algorithm first initializes the output vector to all zeros. Then, for every $\ell = 1, \dots, t$, it creates the vector $v^{(\ell)}$ defined by

$$v^{(\ell)}[k] := \begin{cases} 1, & \text{if } v[k] = f_k^{(\ell)}, \\ 0, & \text{otherwise,} \end{cases}$$

and computes the Boolean product $M^{(\ell)}v^{(\ell)}$, using the ℓ -th instantiation of the hypothesized Boolean-OMv algorithm. Each such product gets then element-wise OR-ed to the output vector. Finally, for every $k = 1, \dots, n$, if $v[k]$ is a rare value in the k -th column of matrix M , the algorithm goes through the list of all indices i such that $M[i][k] = v[k]$ (recall that there are at most n/t of them) and for each of them sets the corresponding i -th entry of the output vector to 1.

It is easy to see that whenever the algorithm sets an output entry to 1, it is because of some pair of entries $M[i][k]$ and $v[k]$ that have the same value. Conversely, if some pair of entries $M[i][k]$ and $v[k]$ have the same value, then either it is a frequent value and some $M^{(\ell)}v^{(\ell)}$ contributes a 1, or it is a rare value and gets manually matched.

Let us analyze the running of our \exists Equality-OMv algorithm. There are t instantiations of the hypothesized Boolean-OMv algorithm, which require $O(tn^{3-\varepsilon})$ time in total. Then, going through all rare values takes at most $O(n^2/t)$ time per v_j , and thus $O(n^3/t)$ time for all n queries. This adds up to total time $O(tn^{3-\varepsilon} + n^3/t)$. By choosing $t := \lceil n^{\varepsilon/2} \rceil$ we get the claimed running time $O(n^{3-(\varepsilon/2)})$. ◀

4 Reduction from Min-Max-OMv to \exists Dominance-OMv

In this we show how a subcubic algorithm for \exists Dominance-OMv would yield a subcubic algorithm for Min-Max-OMv. Our proof is inspired by the static min-max product algorithms [21, 9], but it is at the same time simpler, because we do not optimize the dependence on ε in the running time of the resulting algorithm.

► **Theorem 11.** *If \exists Dominance-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then Min-Max-OMv can be solved in time $O(n^{3-(\varepsilon/2)})$.*

Proof. Let $t := \lceil n^{\varepsilon/2} \rceil$ be a parameter. For every $i \in [n]$, let R_i be the sorted i -th row of the input matrix M . Consider partitioning each R_i into t buckets of consecutive elements, with at most $\lceil n/t \rceil$ elements per bucket. For every $\ell \in [t]$, let $M^{(\ell)} \in (\mathbb{Z} \cup \{\infty\})^{n \times n}$ be the matrix defined as follows:

$$M^{(\ell)}[i, k] := \begin{cases} -M[i, k], & \text{if } M[i, k] \text{ lands in the } \ell\text{-th bucket of } R_i, \\ \infty, & \text{otherwise.} \end{cases}$$

Note that each row of $M^{(\ell)}$ contains $\Theta(n/t)$ finite entries.⁴

In the preprocessing phase, the algorithm instantiates the hypothesised \exists Dominance-OMv algorithm for each of the matrices $M^{(1)}, M^{(2)}, \dots, M^{(\ell)}$, and also for the matrix M .⁵

Upon receiving a query $v \in \mathbb{Z}^n$, the algorithm proceeds to compute the product $M \odot v$ in two steps. First, for every $i \in [n]$, it computes the minimum $M[i, k]$ such that $M[i, k] \geq v[k]$, and stores the results in a column vector u . Second, for every $i \in [n]$, it computes the minimum $v[k]$ such that $v[k] \geq M[i, k]$, and stores the results in a column vector w . At the very end the algorithm computes $(M \odot v)[i] = \min\{u[i], w[i]\}$, for every $i \in [n]$.

In order to compute u , the algorithm first asks for the dominance products $M^{(\ell)} \odot (-v)$, for all $\ell \in [t]$. Then, for each $i = 1, \dots, n$, the algorithm finds the smallest ℓ such that $(M^{(\ell)} \odot (-v))[i] = 1$, which corresponds to finding the first bucket in R_i containing an element *greater*⁶ than or equal to the corresponding element in v . Hence, the algorithm can scan the elements in this bucket and pick the smallest one that is larger than or equal to the corresponding element in v ; this element is then stored in $u[i]$.

Let us analyze the cost of computing u 's over the span of n queries. The t dominance products require time $O(tn^{3-\varepsilon})$ in total. On top of that, for each of the n queries and for each of the n output coordinates, the algorithm scans one bucket of size $\Theta(n/t)$, which takes time $O(n^3/t)$ in total. All together, the algorithm spends time $O(tn^{3-\varepsilon} + n^3/t) = O(n^{3-(\varepsilon/2)})$ on computing u 's.

Next, it is almost symmetric to calculate w . The algorithm sorts the entries of v into an ordered list S , and partitions S into t buckets, with at most $\lceil n/t \rceil$ elements per bucket. For each bucket $\ell \in [t]$, the algorithm computes the dominance product $M \odot v^{(\ell)}$, where $v^{(\ell)} \in (\mathbb{Z} \cup \{-\infty\})^n$ is the column vector such that

$$v^{(\ell)}[k] = \begin{cases} v[k], & \text{if } v[k] \text{ lands in the } \ell\text{-th bucket of } S, \\ -\infty, & \text{otherwise.} \end{cases}$$

Then, for each $i = 1, \dots, n$, the algorithm looks for the smallest ℓ such that $(M \odot v^{(\ell)})[i] = 1$, and scans the elements in the ℓ -th bucket looking for the smallest $v[k]$ that is greater than or equal to the corresponding $M[i, k]$. By the same argument as before, computing all w 's takes time $O(n^{3-(\varepsilon/2)})$. ◀

⁴ If there are multiple entries with the same value, they may land in different buckets.

⁵ Formally, the \exists Dominance-OMv algorithm may not accept infinite entries in the input, but we can replace each ∞ with $W + 2$, where W denotes the largest absolute value of any entry in M , and each entry greater than W in any query vector with $W + 1$.

⁶ This is because the entries in $M^{(\ell)}$ and $-v$ are negated.

5 Reduction from Bounded Monotone Min-Plus-OMv to \exists Equality-OMv

In this section we show a reduction from Bounded Monotone Min-Plus-OMv to \exists Equality-OMv. We follow the high-level approach of some of the previous static algorithms for the bounded monotone min-plus product [25, 10]. However, we deviate from that approach by using the equality product where the known algorithms use a generalization of the min-witness and bounded (non-monotone) min-plus products (see [25, Theorem 1.2]).

► **Theorem 12.** *If \exists Equality-OMv can be solved in time $O(n^{3-\varepsilon})$, for some $\varepsilon > 0$, then Bounded Monotone Min-Plus-OMv can be solved in time $O(n^{3-(\varepsilon/3)} \log n)$ by a randomized algorithm that succeeds with probability⁷ at least $1 - 1/n$.*

Before we present the algorithm itself let us introduce some notation and prove some preliminary facts. Let $\Delta := \lceil n^{\varepsilon/3} \rceil$ be a parameter. For a fixed query vector $v \in \mathbb{Z}^n$, let

$$u := M \oplus v, \quad \widehat{M} := \lfloor M/\Delta \rfloor, \quad \widehat{v} := \lfloor v/\Delta \rfloor, \quad \text{and} \quad \widehat{u} := \widehat{M} \oplus \widehat{v}.$$

Be mindful that it is *not* necessarily the case that $\widehat{u} = \lfloor u/\Delta \rfloor$. Finally, for every $i \in [n]$, let us define the set of *candidates* for $u[i]$ to be

$$C_i := \{k \in [n] \mid \widehat{M}[i, k] + \widehat{v}[k] \in \{\widehat{u}[i], \widehat{u}[i] + 1\}\}.$$

► **Lemma 13.** *It suffices to check only $k \in C_i$ in order to compute $u[i]$, i.e.,*

$$\min_{k \in [n]} M[i, k] + v[k] = \min_{k \in C_i} M[i, k] + v[k].$$

Proof. First, for any pair $(i, j) \in [n] \times [n]$, due to rounding down we have

$$M[i, j] + v[j] - \Delta \cdot (\widehat{M}[i, j] + \widehat{v}[j]) \in [0, 2\Delta).$$

Now, suppose that k is a witness for $u[i]$, and l is a witness for $\widehat{u}[i]$, i.e., $M[i, k] + v[k] = u[i]$, and $\widehat{M}[i, l] + \widehat{v}[l] = \widehat{u}[i]$. We derive that

$$\begin{aligned} \Delta \cdot \widehat{u}[i] + 2\Delta &= \Delta \cdot (\widehat{M}[i, l] + \widehat{v}[l]) + 2\Delta \\ &> M[i, l] + v[l] \\ &\geq M[i, k] + v[k] \\ &\geq \Delta \cdot (\widehat{M}[i, k] + \widehat{v}[k]). \end{aligned}$$

Therefore, we have $\widehat{M}[i, k] + \widehat{v}[k] < \widehat{u}[i] + 2$. Since the matrix entries all take integer values, we have that if $k \in [n]$ is a witness for $u[i]$, then it must satisfy that $\widehat{M}[i, k] + \widehat{v}[k] \in \{\widehat{u}[i], \widehat{u}[i] + 1\}$, i.e., $k \in C_i$. ◀

Now we argue that small sets of candidates can be enumerated efficiently.

► **Lemma 14.** *For a fixed query vector $v \in \mathbb{Z}^n$, there is an algorithm that runs in time $O(n^2 \log n / \Delta)$ and lists all elements of all sets C_i such that $|C_i| \leq n/\Delta$. In the case that $v_j[k]$ is a (weakly) monotone function of j (i.e., the 4-th case in Definition 8) this running time is amortized over n query vectors.*

⁷ Note that the success probability can be amplified to $1 - 1/\text{poly}(n)$ by running in parallel a constant number of copies of the algorithm and taking the majority vote.

Proof. We consider four cases, based on the direction of the monotonicity:

(1) Each column of M is monotone. In this case also the columns of \widehat{M} are monotone, and their entries are bounded by $\lfloor n/\Delta \rfloor$. The algorithm uses a self-balancing binary search tree (BST) to maintain, while i iterates from 1 to n , the set of pairs

$$\{(\widehat{M}[i, k] + \widehat{v}[k], k) \mid k \in [n]\}.$$

Computing $\widehat{u}[i]$ is the standard tree operation of querying for the minimum. Moreover, the BST can report the number of elements smaller than a certain value in time $O(\log n)$, and enumerate them in time proportional to that number. This allows the algorithm to determine the size of C_i quickly, and enumerate it if it is small. As i iterates from 1 to n , the algorithm only needs to update the elements where there is an increase (or decrease) from $\widehat{M}[i, k]$ to $\widehat{M}[i+1, k]$. In each column of M there are at most n/Δ such changes, thanks to the boundedness and monotonicity.⁸ Therefore the total number of updates over the n iterations is at most n^2/Δ , and each update takes time $O(\log n)$. The time spent on listing elements of C_i (for all i) is $O(n \log n + n^2/\Delta)$.

(2) For each k , $v_j[k]$ is a monotone function of j . This case is very similar to the previous one. The algorithm maintains (over the span of n queries) a separate BST for each i , and uses it to compute $(\widehat{M} \oplus \widehat{v}_j)[i]$ for all j 's. When there is an increase (or decrease) from $\widehat{v}_j[k]$ to $\widehat{v}_{j+1}[k]$, the algorithm has to update an element in all n trees, but this happens at most n/Δ times for each k , so n^2/Δ times for all k 's. Hence, the total time spent on such updates over the course of n queries is $O(n^3 \log n/\Delta)$, and the amortized time per query is $O(n^2 \log n/\Delta)$.

(3) Each row of M is monotone. Due to the monotonicity, we can think of the i -th row of \widehat{M} , for each $i = 1, \dots, n$, as consisting of $\lfloor n/\Delta \rfloor + 1$ contiguous blocks $K_i^{(0)}, K_i^{(1)}, \dots, K_i^{(\lfloor n/\Delta \rfloor)} \subseteq [n]$ of identical entries, i.e., $\forall k \in K_i^{(x)} M[i, k] = x$. Upon receiving a query vector v , the algorithm builds (in linear time) a range minimum query (RMQ) data structure (see, e.g., [3]) in order to compute in constant time the minimum entry of \widehat{v} in each of the $O(n^2/\Delta)$ blocks, i.e., $\widehat{v}[K_i^{(x)}] := \min\{\widehat{v}[k] \mid k \in K_i^{(x)}\}$. Adding each of these minima to their corresponding values from \widehat{M} gives a list of candidate values for $\widehat{u}[i]$'s, i.e.,

$$\widehat{u}[i] = \min \{0 + \widehat{v}[K_i^{(0)}], 1 + \widehat{v}[K_i^{(1)}], \dots, \lfloor n/\Delta \rfloor + \widehat{v}[K_i^{(\lfloor n/\Delta \rfloor)}]\}.$$

Thus, we already know how to compute \widehat{u} in time $O(n^2/\Delta)$. Now let us explain how to extend this idea to also list elements of all small enough C_i 's. For each value that appears in \widehat{v} , the algorithm calculates the sorted sequence of indices under which this value appears in \widehat{v} . This allows computing in time $O(\log n)$ how many times a given value appears in a given range of indices in \widehat{v} ; indeed, it boils down to performing two binary searches of the two endpoints of the range in the sequence corresponding to the given value. Furthermore, all these appearances can be enumerated in time proportional to their count. For each block $K_i^{(x)}$ such that $x + \widehat{v}[K_i^{(x)}] = \widehat{u}[i]$ the algorithm enumerates all appearances of $\widehat{v}[K_i^{(x)}]$ and $\widehat{v}[K_i^{(x)}] + 1$ in the range $K_i^{(x)}$ in \widehat{v} , and adds them to C_i . If the total size of C_i would exceed n/Δ , the algorithm stops the enumeration and proceeds to the next block. Similarly, for each block such that $x + \widehat{v}[K_i^{(x)}] = \widehat{u}[i] + 1$ the algorithm enumerates all appearances of $\widehat{v}[K_i^{(x)}]$.

⁸ The locations of all such changes can be, e.g., precomputed before processing any query, or computed during each query using binary search in time $O(\log n)$ per each location.

(4) Each v is monotone. This case is symmetric to the previous one. The difference is that now the algorithm splits \widehat{v} into $O(n/\Delta)$ blocks, and prepares an RMQ data structure for each row of \widehat{M} . ◀

Now we are ready to present our subcubic algorithm for Bounded Monotone Min-Plus OMv, assuming a subcubic algorithm for \exists Equality-OMv.

Proof of Theorem 12. In the preprocessing, the algorithm samples uniformly and independently at random a set $R \subseteq [n]$ of columns of M , of size $|R| := \lceil 3\Delta \ln n \rceil$. For each $r \in R$, the algorithm prepares an \exists Equality-OMv data structure for matrix $M^{(r)}$ obtained from M by subtracting the r -th column from all the columns, i.e.,

$$M^{(r)}[i, k] := M[i, k] - M[i, r].$$

The algorithm handles each query in two independent steps. The goal of the first step is to compute $u[i]$ for those i that have $|C_i| \leq n/\Delta$, and the goal of the second step is to compute $u[i]$ for i with $|C_i| > n/\Delta$.

First step. For each $i \in [n]$, the algorithm either finds out that $|C_i| > n/\Delta$, or lists all elements of C_i and then computes $u[i] = \min_{k \in C_i} (M[i, k] + v[k])$. By Lemma 14, this takes time $O(n^2 \log n/\Delta)$, for all i 's in total. The correctness of this step follows from Lemma 13.

Second step. In the second step, the algorithm must compute the remaining $u[i]$'s, i.e., those for which C_i 's contain too many elements to be handled in the first step. To this end, for every $r \in R$ and every $\delta \in \{0, 1, \dots, 3\Delta - 2\}$ the algorithm computes the equality product $M^{(r)} \ominus -(v - v[r] + \delta)$. For every $i \in [n]$, if $(M^{(r)} \ominus -(v - v[r] + \delta))[i] = 1$, then there must exist $k \in [n]$ such that

$$M[i, k] - M[i, r] = -(v[k] - v[r] + \delta)$$

and hence

$$M[i, k] + v[k] = M[i, r] + v[r] - \delta.$$

The algorithm therefore adds $M[i, r] + v[r] - \delta$ to the list of possible values for $u[i]$, and at the end of the process it sets each $u[i]$ to the minimum over those values.

Analysis of the second step. We now argue that if $R \cap C_i \neq \emptyset$ (which holds with high probability when $|C_i| > n/\Delta$ via a standard hitting set argument, see below), then the algorithm correctly computes $u[i]$ in the second step. Indeed, pick $r \in R \cap C_i$ and let $k \in [n]$ be a witness for $(M \oplus v)[i]$, i.e., $M[i, k] + v[k] = (M \oplus v)[i]$. Let $\delta := (M[i, r] + v[r]) - (M[i, k] + v[k])$. Clearly, $(M^{(r)} \ominus -(v - v[r] + \delta))[i] = 1$, so it only remains to show that $\delta \in \{0, 1, \dots, 3\Delta - 2\}$. Obviously, $\delta \geq 0$, because k minimizes $M[i, k] + v[k]$. Now let us upper bound the offset δ . Since $r \in C_i$, we have $\widehat{M}[i, r] + \widehat{v}[r] \leq \widehat{u}[i] + 1$, and hence

$$M[i, r] + v[r] \leq (\Delta \widehat{M}[i, r] + \Delta - 1) + (\Delta \widehat{v}[r] + \Delta - 1) \leq \Delta \widehat{u}[i] + 3\Delta - 2.$$

Moreover, $\widehat{M}[i, k] + \widehat{v}[k] \geq \widehat{u}[i]$, and therefore

$$M[i, k] + v[k] \geq \Delta \widehat{M}[i, k] + \Delta \widehat{v}[k] \geq \Delta \widehat{u}[i].$$

We conclude that $\delta \leq (\Delta \widehat{u}[i] + 3\Delta - 2) - \Delta \widehat{u}[i] = 3\Delta - 2$, as required.

It remains to analyze the success probability of the whole algorithm. For a fixed output index $i \in [n]$ such that $|C_i| > n/\Delta$, the probability that the algorithm failed to sample an element r from C_i in all $|R| = \lceil 3\Delta \ln n \rceil$ rounds is at most $(1 - 1/\Delta)^{3\Delta \ln n} < (1/e)^{3 \ln n} = 1/n^3$. By a union bound over all n output indices for each of the n queries, the algorithm succeeds to correctly compute all n^2 output entries with probability at least $1 - n^2/n^3 = 1 - 1/n$.

Running time. The first step of each query (Lemma 14) takes time $O(n^2 \log n/\Delta)$, summing up to $O(n^3 \log n/\Delta)$ for all n queries. Regarding the second step, for each query the algorithm computes $O(|R|\Delta) = O(\Delta^2 \log n)$ equality matrix-vector products, and over the course of n queries this takes time $O(n^{3-\varepsilon} \Delta^2 \log n)$. The total running time is therefore $O(n^3 \log n/\Delta + n^{3-\varepsilon} \Delta^2 \log n) = O(n^{3-(\varepsilon/3)} \log n)$. ◀

6 Remaining reductions

In this section we state the remaining easy reductions that complete the proof of Theorem 1.

6.1 Reduction from \exists Dominance-OMv to \exists Equality-OMv

► **Observation 15.** *If \exists Equality-OMv can be solved in time $T(n)$, then \exists Dominance-OMv can be solved in time $O(T(n) \log n)$.*

Proof. The proof follows a folklore argument, see, e.g., [13]. It uses the fact that, for any two non-negative integers a and b , it holds that $a < b$ if and only if there exists $\ell \geq 0$ such that

- the ℓ -th least significant bit of a is 0; and
- the ℓ -th least significant bit of b is 1; and
- a agrees with b on all bits higher than the ℓ -th least significant, i.e., $\lfloor a/2^{\ell+1} \rfloor = \lfloor b/2^{\ell+1} \rfloor$.

Moreover, without loss of generality, all the input numbers are integers between 0 and $n^2 - 1$. Indeed, in the preprocessing, each entry of M can be replaced by its rank in the sorted order of all entries of M ; then, during a query, each entry of v can be replaced by the rank of the smallest entry of M greater than or equal to it. Last but not least, $M[i, k] \leq v[k]$ if and only if $M[i, k] < v[k] + 1$, because the input numbers are integers. Hence, the algorithm sets, for $\ell = 0, 1, \dots, \lceil \log(n^2) \rceil$,

$$M^{(\ell)}[i, k] := \begin{cases} \left\lfloor \frac{M[i, k]}{2^{\ell+1}} \right\rfloor, & \text{if the } \ell\text{-th least significant bit of } M[i, k] \text{ is 0} \\ -1, & \text{otherwise,} \end{cases}$$

$$v^{(\ell)}[k] := \begin{cases} \left\lfloor \frac{v[k]+1}{2^{\ell+1}} \right\rfloor, & \text{if the } \ell\text{-th least significant bit of } v[k] + 1 \text{ is 1} \\ -2, & \text{otherwise,} \end{cases}$$

and uses the fact that $(M \odot v)[i] = 1$ if and only if $\exists \ell \ (M^{(\ell)} \ominus v^{(\ell)})[i] = 1$. ◀

6.2 Reduction from Min-Witness-OMv to Min-Max-OMv

► **Observation 16.** *If Min-Max-OMv can be solved in time $T(n)$, then Min-Witness-OMv can be solved in time $T(n) + O(n^2)$.*

Proof. The proof follows another folklore argument, see, e.g., [15]. The algorithm sets

$$M'[i, k] := \begin{cases} k, & \text{if } M[i, k] = 1 \\ \infty, & \text{otherwise,} \end{cases} \quad \text{and} \quad v'[k] := \begin{cases} k, & \text{if } v[k] = 1 \\ \infty, & \text{otherwise,} \end{cases}$$

and uses the fact $M \otimes v = M' \odot v'$. ◀

6.3 Reduction from Boolean-OMv to Bounded Monotone Min-Plus-OMv

► **Observation 17.** *If Bounded Monotone Min-Plus-OMv can be solved in time $T(n)$, then Boolean-OMv can be solved in time $T(n) + O(n^2)$.*

Proof. The algorithm sets

$$M'[i, k] := 2 \cdot (i + k) - M[i, k], \quad \text{and} \quad v'_j[k] := 2 \cdot (j - k) - v_j[k],$$

and uses the fact that $(Mv_j)[i] = 1$ if and only if $(M' \oplus v'_j)[i] = 2 \cdot (i + j) - 2$. ◀

We remark that the above reduction produces Min-Plus-OMv instances that are monotone in all four directions simultaneously, while our Bounded Monotone Min-Plus-OMv algorithm of Theorem 12 works already for instances with monotonicity in one (arbitrarily chosen) direction.

References

- 1 Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 935–943. ACM, 2024. doi:10.1145/3618260.3649696.
- 2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.
- 3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
- 4 Karl Bringmann, Allan Grønlund, Marvin Künnemann, and Kasper Green Larsen. The NFA acceptance hypothesis: Non-combinatorial and dynamic lower bounds. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024*, volume 287 of *LIPICs*, pages 22:1–22:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ITCS.2024.22.
- 5 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman’s trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM counting, and more. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 419–432. ACM, 2023. doi:10.1145/3564246.3585237.
- 6 Lijie Chen, Erik D. Demaine, Yuzhou Gu, Virginia Vassilevska Williams, Yinzhan Xu, and Yuancheng Yu. Nearly optimal separation between partially and fully retroactive data structures. In *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018*, volume 101 of *LIPICs*, pages 33:1–33:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.SWAT.2018.33.
- 7 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.
- 8 Artur Czumaj, Mirosław Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007. doi:10.1016/J.TCS.2007.02.053.
- 9 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 384–391. SIAM, 2009. doi:10.1137/1.9781611973068.43.

- 10 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.75.
- 11 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 12 Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*, volume 3142 of *Lecture Notes in Computer Science*, pages 782–792. Springer, 2004. doi:10.1007/978-3-540-27836-8_66.
- 13 Karim Labib, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Hamming distance completeness. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CPM.2019.14.
- 14 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142.
- 15 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.
- 16 Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 228–243. IEEE, 2024. doi:10.1109/FOCS61266.2024.00006.
- 17 Jirí Matousek. Computing dominances in e^n . *Inf. Process. Lett.*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-0.
- 18 Kerui Min, Ming-Yang Kao, and Hong Zhu. The closest pair problem under the Hamming metric. In *Computing and Combinatorics, 15th Annual International Conference, COCOON 2009*, volume 5609 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 2009. doi:10.1007/978-3-642-02882-3_21.
- 19 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.
- 20 Virginia Vassilevska. *Efficient algorithms for path problems in weighted graphs*. PhD thesis, Carnegie Mellon University, USA, 2008.
- 21 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory Comput.*, 5(1):173–189, 2009. Announced at STOC 2007. doi:10.4086/T0C.2009.V005A009.
- 22 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians, ICM 2018*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.
- 23 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. Announced at FOCS 2010. doi:10.1145/3186893.
- 24 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.

- 25 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.
- 26 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. Announced at STOC 2014. doi:10.1137/15M1024524.
- 27 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. Announced at ESA 2004. doi:10.1145/1077464.1077466.