The Structural Complexity of Matrix-Vector Multiplication

Emile Anand[*]
Georgia Institute of Technology
Atlanta, GA 30308
emile@gatech.edu

Jan van den Brand
Georgia Institute of Technology
Atlanta, GA 30308
vdbrand@gatech.edu

Rose McCarty[†]
Georgia Institute of Technology
Atlanta, GA 30308
rmccarty3@gatech.edu

Abstract

We consider the problem of preprocessing an $n \times n$ matrix $\mathbf{M}$, and supporting queries that, for any vector v, returns the matrix-vector product $\mathbf{M}v$. This problem has been extensively studied in both theory and practice: on one side, practitioners have developed algorithms that are highly efficient in practice, whereas on the other side, theoreticians have proven that the problem cannot be solved faster than naive multiplication in the worst-case. This lower bound holds even in the average-case, implying that existing average-case analyses cannot explain this gap between theory and practice. Hence, we study the problem for *structured* matrices. We show that for $n \times n$ Boolean matrices of VC-dimension d, the matrix-vector multiplication problem can be solved with $\widetilde{O}(n^2)$ preprocessing and $\widetilde{O}(n^{2-1/d})$ query time. Given the low constant VC-dimensions observed in most real-world data, our results posit an explanation for why the problem can be solved so much faster in practice. Furthermore, we show how to extend this result to the non-Boolean setting with the Pollard pseudodimension.

Our results yield the first non-trivial upper bounds for many applications. In previous works, the online matrix-vector (OMv) hypothesis (conjecturing that quadratic time is needed per query, even over the boolean semi-ring) was used to prove many conditional lower bounds, showing that it is impossible to compute and maintain high-accuracy estimates for effective resistance, Laplacian solvers, shortest paths, and triangle detection in graphs subject to node insertions and deletions in subquadratic time. Yet, via a reduction to our matrix-vector-multiplication result, we show we can maintain these problems efficiently if the input is structured, providing the first subquadratic upper bounds in the high-accuracy regime.

1 Introduction

Computing sequential matrix-vector products is a fundamental subroutine of many iterative algorithms in machine learning. In optimization [Renegar, 1996, Vempala et al., 2020, Lin et al., 2023a, Chaudhari et al., 2024], computational geometry [Har-Peled, 2009, Welzl, 1992, Fisikopoulos and Penaranda, 2016], online algorithms [Lin et al., 2024, Anand and Qu, 2024, Murray et al.,

[*]Work partially done while at Cognition AI
[†]Work done while at Princeton University.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

2021], and dynamic algorithms [Liu, 2024, Jiang et al., 2023, Anand et al., 2024b, Jin and Xu, 2022], sequential matrix-vector products are essential subroutines underlying performant algorithms. The current learning revolution is powered by hardware specifically designed to perform such products, as both neural network evaluation and back-propagation require repeated matrix-vector products [Sørensen, 2012, Rumelhart et al., 1986]. Hence, matrix-vector products are arguably one of the most important and fundamental subroutines, with any complexity improvement having wide-ranging implications, and is thus a prevalent research topic in both theory and practice.

The problem can be modeled via the following data structure task: construct a data structure that preprocesses a given $n \times n$ matrix $\mathbf{M}$. After preprocessing, we want to multiply vectors with $\mathbf{M}$ faster than naive matrix-vector multiplication. Depending on the application, this matrix $\mathbf{M}$ is fixed or the data structure may need to handle changes to $\mathbf{M}$. For instance, when calculating probabilities of a random walk (e.g., in the PageRank algorithm [Page et al., 1999]), performing the power-method, and evaluating a neural network, the matrix is fixed. Conversely, in convex optimization, such as when solving a linear or semi-definite program, applying Newton-Raphson's method, and training a neural network, the matrix is the inverse of a Hessian or is given by the network's weights, which changes from one iteration to the next [Arriaga and Vempala, 2006, Anand and Liaw, 2025].

Beyond speedups due to improved hardware, practitioners have made tremendous progress in accelerating the computation of matrix-vector products through heuristics that run in $\mathrm{nnz}(\mathbf{M})$ (the number of non-zero entries of $\mathbf{M}$) worst-case time, but which are much faster in practice. This has led to speedups in training GNNs [Alves et al., 2024] and other algorithms [Floros et al., 2024].

Conversely, from a theory perspective there are substantial lower bounds. Gronlund and Larsen [2015] showed that any $\mathrm{poly}(n)$-space data structure for matrix-vector multiplication over sufficiently large fields (for instance, $\mathbb{R}$) require $\Omega(n^2/\log n)$ time. This quadratic lower bound was also proven for arithmetic circuits [Frandsen et al., 2001]. While these are worst-case lower bounds, they also hold for the average case [Henzinger et al., 2022]. The only non-trivial upper bounds beating quadratic time are over the Boolean semi-ring (i.e., $\{0,1\}$ with $x \oplus y = \min(1, x+y)$). In a line of works that reduced the problem to finding neighborhoods in a graph and smaller algebraic products, this query complexity was improved to $O(n^2/\log^2 n)$ [Liberty and Zucker, 2006, Williams, 2007] and $O(n^{2-o(1)})$ [Larsen and Williams, 2017, Chakraborty et al., 2018, Abboud et al., 2024]. However, none of these algorithms are truly subquadratic, i.e., $O(n^{2-\epsilon})$ for some constant $\epsilon > 0$.

Moreover, while the $\Omega(n^2)$ lower bounds only hold over finite fields, even over the Boolean semi-ring it is conjectured that no truly subquadratic time algorithm exists [Henzinger et al., 2015], and this is commonly referred to as the OMv (online matrix-vector multiplication) conjecture. In fact, this has led to a recent line of works that use the conditional hardness of OMv to prove tight time lower bounds for many dynamic algorithms, such as dynamic matrix inversion [Brand et al., 2019a,b], dynamic subgraph connectivity [Henzinger and Neumann, 2016], dynamic regression [Jiang et al., 2023], dynamic range, Langerman's problem [Jin and Xu, 2022], and the generalized Klee's high-dimensional measure problem [Yildiz and Suri, 2012].

In summary, there is a notable difference between the perspectives of this problem in theory and practice: there are heuristics that are fast in practice, but from a theoretical perspective the problem is hard, and an entire subarea of fine-grained-complexity has been built on this hardness assumption [Hu and Polak, 2024]. Further, average-case analyses cannot explain this gap, as even the average-case is provably hard. In this work, we resolve this conflict: since the average-case (i.e., random non-structured inputs) is hard, the observed efficiency of practical algorithms must stem from some *inherent structure* in real-world data. Thus, we study the complexity for structured inputs.

A popular measure for structural complexity is the Vapnik-Chervonenkis (VC) dimension, which finds many applications in machine learning [Shalev-Shwartz and Ben-David, 2014, Bartlett et al., 2019], structural graph theory [Nguyen et al., 2024, Alon et al., 2016, Karczmarz and Zheng, 2024], and computational geometry [Har-Peled, 2009, B. Chazelle, 1989, Fisikopoulos and Penaranda, 2016] (see the preliminaries in Section 2 for a definition of the VC-dimension). For Boolean matrices, the VC-dimension has the following, more intuitive, description: it is the size of the largest subset $S \subseteq [n]$ of columns such that the rows of the sub-matrix whose columns are restricted to those in S contain every possible string in $\{0,1\}^{|S|}$. Many structured objects studied by theoreticians have inherently low VC dimension. For instance, star and interval graphs, and rank-one projection matrices, have VC-dimension 2, planar graphs have VC dimension 4, and any H-minor free graph has VC dimension at most $|V(H)| - 1$. Beyond being an established complexity measure in theory, it has also been

empirically observed that real-world data has low VC dimension [Coudert et al., 2024]. We explain this observation via the following structural characterization of matrices with low VC dimension:

Theorem 1.1 (Proof in Appendix D)**.** *Consider a hereditary class of $0/1$-matrices $\mathcal{M}$, meaning that $\mathcal{M}$ is closed under row/column deletion (i.e., each matrix $\mathbf{M} \in \mathcal{M}$ is still in $\mathcal{M}$ after deleting a row or column). If $\mathcal{M}$ is non-trivial, (i.e., does not contain every possible matrix), then there exists an absolute constant $c \in \mathbb{N}$ such that the VC-dimension of any matrix in $\mathcal{M}$ is at most c.*

Presumably, any unknown structure common to real-world matrices should not be lost when deleting rows/columns. Theorem 1.1 does not exclude other matrices (which do not satisfy this hereditary property) from having constant VC-dimension, e.g., adjacency matrices of minor-free graphs.

1.1 Our Results

While theoretical objects such as grid-graphs, intersection graphs, and kernel-matrices have a low VC-dimension, they are not perfect representations of real-world data. For instance, while city street networks are predominantly grid-graphs, there always exists exceptions in the form of bridges or few diagonal streets. Exceptions can also occur due to errors in measurements or other corruptions. To capture such *almost low VC-dimension objects*, we define the "corrupted VC-dimension."

Definition 1.2 (Corrupted VC-dimension d)**.** *The "corrupted VC-dimension" of a set system $\mathcal{F}$ with m sets is the smallest d such that there is another set system $\mathcal{F}'$ of VC-dimension $\leq d$ and $\mathcal{F}$ can be obtained from $\mathcal{F}'$ by adding and/or removing each element to/from at most $O(m^{1-1/d})$ sets.*

For matrices $\mathbf{M} \in \{0,1\}^{m \times n}$, a corrupted VC-dimension d implies the existence of a matrix $\mathbf{L} \in \{0,1\}^{m \times n}$ of VC-dimension $\leq d$ and a 'corruption' matrix $\mathbf{S} \in \{-1,0,1\}^{m \times n}$ with at most $O(m^{1-1/d})$ non-zero entries per column, such that $\mathbf{M} = \mathbf{L} + \mathbf{S}$. Note that the mere existence of such an $\mathcal{F}'$ (i.e., $\mathbf{L}$) of VC-dimension $\leq d$ suffices for $\mathcal{F}$ (i.e., $\mathbf{M}$) to have corrupted VC-dimension d. We do *not* need to know $\mathcal{F}'$ or which sets have been corrupted. Also, the VC-dimension of $\mathcal{F}$ upper bounds the corrupted VC-dimension of $\mathcal{F}$. Hence, while we state our results for matrices with corrupted VC-dimension d, the proven upper bounds also hold for matrices with VC-dimension d.

We now state the theoretical guarantees of our accelerated matrix-vector multiplication algorithm in Theorem 1.3, where the runtimes are parameterized by the corrupted VC dimension measure.

Theorem 1.3. *(Static Online Matrix-Vector Multiplication). If a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$ has corrupted VC-dimension d, then after an $\widetilde{O}(mn)$-time preprocessing, there is a data structure $\mathcal{D}$ that can compute $\mathbf{M}v$ for any $v \in \mathbb{R}^n$ in $\widetilde{O}(nm^{1-1/d} + m)$ time, with high probability.*

The problem of approximating the VC-dimension of any set system within a $(2-\epsilon)$ accuracy for any $\epsilon > 0$ is known to be Σ_3^P-hard [Mossel and Umans, 2002]; however, for our theorem, the algorithm does not need to know or compute or even approximate the VC-dimension to achieve this runtime.

Our data structure is based on Björklund and Lingas [2001] which proposed an algorithm for Boolean matrix multiplication. Their algorithm computes the matrix-matrix product via n matrix-vector products, thus implicitly providing a matrix-vector multiplication data structure as well. Their complexity depends on the weight of the minimum spanning tree (MST) defined with respect to the Hamming-distance between the rows of $\mathbf{M}$. However, the weight of the MST is $O(n^2)$ in the worst-case.

This algorithm was recently independently rediscovered in the graph neural networks community [Alves et al., 2024] which experimentally verified its efficiency on real-world inputs. Our Theorem 1.3 now gives a theoretical explanation as to why these practical algorithms are much more efficient in practice than the worst-case and average-case lower bounds of $\Omega(mn)$.

We obtain Theorem 1.3 via techniques from computational geometry. A line of works [Welzl, 1988, B. Chazelle, 1989, Matoušek, 1991, Har-Peled, 2009] consider the following *geometric intersection* data structure problem: given a set of points in $\mathbb{R}^d$, preprocess them. Then, given a convex polytope for each query, return whether any of the points intersect the polytope. However, prior work had exponential preprocessing time [Welzl, 1992] or unspecified polynomial time [Matoušek, 1991]. Matrix vector products can be interpreted as $O(m)$ intersection problems if we only care about Boolean outputs, because in the Boolean case $(\mathbf{M}v)_i \neq 0$ if and only if there is an intersection between the position of the non-zero elements in v and the non-zero elements in the i-th row of $\mathbf{M}$. We extend these computational geometry techniques to non-Boolean outputs, and to our more general notion of corrupted VC-dimension. We provide details of this proof in Section 3.2.

In addition to our novel complexity bound, we also provide alternative algorithms to those presented in Björklund and Lingas [2001], Alves et al. [2024]. We provide more details below.

Dynamic Setting. We extend Theorem 1.3 by allowing $\mathbf{M}$ to undergo row and column updates. The proof is given in Appendix B.3.

Theorem 1.4. *(Dynamic Online Matrix-Vector Multiplication). Given a matrix $\mathbf{M} \in \{0,1\}^{m\times n}$, there is a data structure $\mathcal{D}$ with $\widetilde{O}(mn)$ preprocessing time that supports row and column updates (insertions/deletions) to $\mathbf{M}$ in $\widetilde{O}(n)$ and $\widetilde{O}(m)$ time, respectively. Upon querying $\mathcal{D}$ with a vector $v \in \mathbb{R}^n$, it outputs $\mathbf{M}v$ in $\widetilde{O}(nm^{1-1/d^*} + m)$ time, with high probability, where d^* is the largest corrupted VC-dimension of $\mathbf{M}$ throughout the history of its updates.*

Transposed Matrices. If the VC-dimension of $\mathbf{M}^\top$ is d, then the VC-dimension of $\mathbf{M}$ is at most 2^{d+1} [Assouad, 1983], so Theorem 1.3 runs in $\widetilde{O}(nm^{1-1/2^d} + m)$ time. To address this slow-down, we extend the algorithm to bound its complexity by the VC-dimension of $\mathbf{M}^\top$.

If $\mathbf{M} \in \{0,1\}^{m\times n}$ and $\mathbf{M}^\top \in \{0,1\}^{n\times m}$ have corrupted VC-dimension d and d' (respectively), then after an $\widetilde{O}(mn)$-time preprocessing, we show we can compute $\mathbf{M}v$ in time $\widetilde{O}(\min\{nm^{1-1/d} + m, mn^{1-1/d'} + n\})$ with high probability. We do not need to know d or d', as the algorithm automatically achieves the minimum.

Non-Boolean Matrices. The Pollard pseudodimension extends the VC-dimension to non-binary thresholds. It finds many applications in theoretical machine learning [Shalev-Shwartz and Ben-David, 2014, Bartlett et al., 2019] and structural graph theory [Karczmarz and Zheng, 2024] (see Appendix B.4 for a definition of the *Pollard pseudodimension*). For $\mathbf{M} \in \mathbb{R}^{m\times n}$, if d bounds the Pollard pseudodimension of $\mathbf{M}$ and if T bounds the number of unique values per column of $\mathbf{M}$, then by Theorem B.9, $\mathbf{M}v$ can be computed in time $\widetilde{O}(Tnm^{1-1/d} + m)$. For $T \leq \widetilde{O}(1)$, this is subquadratic.

1.2 Applications

Random Walks and PageRank by Scaling. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be the adjacency matrix of an n-node graph and let $\mathbf{D} \in \mathbb{R}^{n\times n}$ be its degree matrix. The normalized Laplacian is $\widetilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Even though $\widetilde{\mathbf{L}}$ is not Boolean, we can still use Theorem 1.3 since $\widetilde{\mathbf{L}}v = \mathbf{D}^{-1/2}(w - \mathbf{A}w)$ for $w = \mathbf{D}^{-1/2}v$ and the product $\mathbf{A}w$ can be computed via Theorem 1.3. The same technique extends to random-walk probability matrices. In turn, this allows fast simulations of Markov chains to compute stationary distributions and iterative computation of the PageRank [Page et al., 1999] via Theorem 1.3.

Complexity-Theoretic Implications. The Boolean matrix multiplication (BMM) conjecture states that no algorithm can multiply two boolean $n \times n$ matrices in $O(n^{3-\epsilon})$ time for constant $\epsilon > 0$ [Basch et al., 1995, Williams and Williams, 2010]. This conjecture is used to prove lower bounds on combinatorial algorithms (i.e., algorithms that do not make use of Strassen-style fast matrix multiplication methods), motivated by the property that *fast matrix multiplication*, while being theoretically fast (currently $O(n^{2.372})$ Williams et al. [2024]), is very slow in practice due to large constants hidden in O-notation. Hence BMM gives a lower bound on more practical algorithms and can be used to prove that any further complexity improvement would require fast matrix multiplication.

Theorem 1.3 implies that for matrices with corrupted VC-dimension d, BMM can be solved in $\widetilde{O}(n^{3-1/d})$ time, allowing for improved combinatorial upper bounds. The BMM problem was the original motivation for Björklund and Lingas [2001], Gasieniec and Lingas [2003]. Moreover, as demonstrated in Alves et al. [2024], this algorithm is also highly efficient in practice and does not encounter the large constant of fast matrix multiplication. Therefore, Theorem 1.3 implies that problems with lower bounds based on BMM can be solved more efficiently on structured inputs.

Corollary 1.5. *Suppose we are given matrices $\mathbf{M}_1, \mathbf{M}_2 \in \{0,1\}^{n\times n}$, where $\mathbf{M}_i$ has corrupted VC-dimension d_i and $\mathbf{M}_i^\top$ has corrupted VC dimension d_i' for $i \in \{1,2\}$. Let $s = \min\{d_1, d_2, d_1', d_2'\}$. Then, there is an algorithm for Boolean matrix multiplication which runs in $\widetilde{O}(n^{3-1/s})$ time.*

Another relevant conjecture is the OMv conjecture, which states that no algorithm can preprocess a Boolean matrix $M \in \{0,1\}^{n\times n}$ in polynomial time, and then answer n queries that compute $\mathbf{M}v^{(i)}$ for an online sequence of Boolean vectors $v^{(1)}, ..., v^{(n)} \in \{0,1\}^n$ in total time $O(n^{3-\epsilon})$. Observe

that here the vector $v^{(i+1)}$ is only given after the data structure returned $\mathbf{M}v^{(i)}$, hence the use of fast matrix multiplication to compute the matrix product $\mathbf{M}\,[v^{(1)}|...|v^{(n)}]$ is ruled out.

This conjecture is used to prove conditional lower bounds for many dynamic graph problems, such as maintaining shortest paths, effective resistances, reachability, bipartite matching, triangle detection, and many others in graphs undergoing vertex insertions and deletions [Henzinger et al., 2015]. Even using fast matrix multiplication, no algorithm can beat $O(n^2)$ time on dense graphs, which is typically equivalent to re-solving them from scratch whenever the graph changes. Since Theorem 1.3 shows the OMv conjecture does not hold for structured matrices, it leads to improved dynamic algorithms.

Implications to Dynamic Algorithms. We list some novel dynamic algorithms obtained via this technique, whose upper bounds are only achievable due to their structured nature, otherwise violating either the BMM or OMv conjecture. We defer the formal proofs for these results to Appendix C.

An important class of linear equations arising in practice have form $\mathbf{L}x = b$, where $\mathbf{L}$ is the Laplacian of an undirected graph $G = (V, E)$ given by $\mathbf{L} \coloneqq \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is a diagonal matrix such that $\mathbf{D}_{i,i} = \deg(i)$ for $i \in [n]$ where $|V| = n$, and $\mathbf{A}$ is the adjacency matrix of G. There are Laplacian solvers that run in time nearly linear in $|E|$ [Spielman and Teng, 2004]. We study the problem of constructing a dynamic Laplacian solver for $\mathbf{L}_t x = b_t$ where the Laplacian changes over time through vertex updates to the graph, and each b_t is an arbitrary new vector.

Previous upper bounds could only handle a $\mathrm{poly}(1/\epsilon)$ accuracy by maintaining spectral sparsifiers [Abraham et al., 2016] or vertex sparsifiers [Durfee et al., 2019]. Moreover, exact or $O(\log 1/\epsilon)$-time dependencies are ruled out by the OMv conjecture since no algorithm beats naive recomputation from scratch in $\widetilde{O}(|E|) = \widetilde{O}(n^2)$ time via nearly-linear time Laplacian solvers [Spielman and Teng, 2004]. We give the first result faster than naive recomputation in the high-accuracy $\log(1/\epsilon)$ regime.

Theorem 1.6. *(Dynamic Laplacian Solver). There is a dynamic algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension bounded by d, maintains a Laplacian system solver. The data structure supports queries that receive a vector $b \in \mathbb{R}^{|V|}$ and error parameter $\epsilon > 0$. Then, in $\widetilde{O}(n^{2-1/d} \log 1/\epsilon)$ time, the algorithm returns the (approximate) solution x to $\mathbf{L}x^* = b$ where $\|x - x^*\|_{\mathbf{L}} \leq \epsilon \|x^*\|_{\mathbf{L}}$. Each vertex update to G takes $\widetilde{O}(n)$ time.*

With the fast dynamic Laplacian solver, we maintain dynamic effective resistances, which is a critical subroutine in graph clustering [Alev et al., 2017], fault-tolerant computing [Ghosh et al., 2008], network analysis [Tizghadam and Leon-Garcia, 2008, Anand and Umans, 2023, Anand et al., 2024a], and biological systems [Klein et al., 2022], as it measures the connectivity between two vertices. The effective resistance $r_G(a, b)$ between vertices a and b in a graph G is a graph-analog of leverage scores that represents the energy needed to route one unit of electric flow from a to b. Formally, let $\mathbf{e}_i$ denote the i-th standard basis vector. Then, for all $a, b \in V$, $r_G(a, b) = (\mathbf{e}_a - \mathbf{e}_b)^\top \mathbf{L}^\dagger (\mathbf{e}_a - \mathbf{e}_b)$, where $\mathbf{L}$ is the Laplacian of G and $\mathbf{L}^\dagger$ is the Moore-Penrose pseudoinverse of $\mathbf{L}$.

Theorem 1.7. *(Dynamic Effective Resistance). There is a dynamic algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension bounded by d, maintains effective resistances in G. The data structure supports queries that receive a pair of vertices $u, v \in V$ and error parameter $\epsilon > 0$. Then, in $\widetilde{O}(n^{2-1/d} \log 1/\epsilon)$ time, the algorithm returns a $(1 \pm \epsilon)$-approximation of the effective resistance. Moreover, each node update to G in the dynamic data structure takes $\widetilde{O}(n)$ time.*

Our dynamic matrix-vector multiplication data structure also leads to faster dynamic triangle detection algorithms for graphs with corrupted VC-dimension d, which are critical components of fraud detection systems and community-detection algorithms. No subquadratic time algorithm exists for vertex updates, conditional on BMM [Abboud and Vassilevska Williams, 2014] or OMv [Henzinger et al., 2015]. We show that these lower bounds can be broken on structured graphs.

Theorem 1.8. *(Dynamic Triangle Detection). There is an algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension d, maintains whether G has a triangle or not. Each vertex update takes $\widetilde{O}(n^{2-1/d})$ time and returns a Boolean indicator for G containing a triangle.*

Our techniques lead to faster dynamic approximate algorithms for single-source distances, where we are tasked with finding the distances from a designated source vertex to every other vertex.

Theorem 1.9. *(Dynamic Approximate Single-Source Shortest Paths). There is a dynamic algorithm that maintains $(1 + \epsilon)$-approximate single-source distances on a dynamic unweighted graph*

$G = (V, E)$. *If the corrupted VC-dimension of G is bounded by d, each node update to G takes $\widetilde{O}(kn^{2-1/2d}/\epsilon)$ time, and querying the distances for any source node takes $\widetilde{O}(n^{2-1/2d}/\epsilon)$ time.*

Given a metric space $\mathcal{M}$ with n points and a positive integer $k \leq n$, the k-center problem asks one to select k points, referred to as centers, such that the maximum distance of any point in the metric space to its closest center is minimized. It is known that the k-center is NP-hard to $(2-\epsilon)$-approximate for any $\epsilon > 0$ [Hsu and Nemhauser, 1979]. The best existing dynamic algorithms [Cruciani et al., 2024] only work with edge updates and use fast matrix multiplication as a black-box. We present an algorithm without fast matrix multiplication that supports more powerful vertex updates.

Theorem 1.10. *(Dynamic Approximate k-center). Given an unweighted undirected graph $G = (V, E)$, there is a dynamic algorithm for $(2+\epsilon)$-approximate k-center with node update time $\widetilde{O}(kn^{2-1/2d}/\epsilon)$, where d is a bound on the corrupted VC-dimension of G.*

1.3 Further Related Work

Boolean matrix multiplication. A line of work shows that the complexity of matrix-vector multiplication over the Boolean semiring can be mildly subpolynomial. For instance, Williams [2007] showed that BMM can be computed in $O(n^3/\log n)$ time. In turn, Larsen and Williams [2017] and Abboud et al. [2024] extended this to $O(n^3/2^{\Omega(\sqrt[7]{\log n})})$.

Structured matrices. When the underlying matrix is Vandermonde, Toeplitz, Hankel, or Cauchy, a body of work shows that they admit fast matrix-vector multiplications through convolutional transformations [Motzkin, 1951, Olshevsky and Shokrollahi, 2000, Pan and Tsigaridas, 2014]. Orthogonal-vector (OV) matrices are structured matrices where each row and column receives a label $v \in \{0,1\}^d$ and $\mathbf{M}_{i,j} = 1$ if and only if the corresponding two labels are orthogonal. Similar to our corrupted VC-dimension results, Alman and Vassilevska Williams [2020] show that OV matrices allow fast matrix-vector products if there are few corruptions to the matrix. Matrix products can also be accelerated using geometric data structures Lingas [2002], Floderus et al. [2018]. Another approach to handle structured inputs is through the algorithms with predictions regime, where some learning algorithm extracts the structure and can predict information about future vectors, leading to faster matrix-vector multiplication algorithms Henzinger et al. [2024], Brand et al. [2024].

Dynamic structured graphs. Our dynamic algorithms hold for any structured graph, even if we do not know their structure. When the structure is known, there are specialized dynamic algorithms tailored to the specific graph. These include, for instance, interval graphs Crespelle [2019], Chen et al. [2024], geometric graphs defined from Kernels [Alman et al., 2020], dynamic planar graphs [Korhonen et al., 2024, Abboud and Dahlgaard, 2016], and minor free graphs [Dorn et al., 2012].

Matrix vector multiplication in optimization. In convex optimization, there is a long history on developing data structures to accelerate matrix-vector products. Here the matrices are generally projection matrices that require computation of some matrix inverse. While historically, most research was on accelerating the maintenance of the matrix inverse [Karmarkar, 1984, Vaidya, 1989, Lee and Sidford, 2015, Cohen et al., 2021, Lee et al., 2019, Brand, 2020, Jiang et al., 2021, 2020a, Huang et al., 2022, Jiang et al., 2022, 2020b], this research has progressed so far that the current bottleneck for faster linear program solvers are simple matrix-vector products [Brand et al., 2020, 2021].

2 Preliminaries

Notation. Let $[n] = \{1, \ldots, n\}$. We use $O(\cdot)$ to hide constant factors, and $\widetilde{O}(\cdot)$ to hide polylogarithmic factors. Let $\|\cdot\|_1$ denote the ℓ_1 (Manhattan distance). For a matrix $\mathbf{M}$, let $\mathbf{M}_i$ and $\mathbf{M}_{:,i}$ denote its i'th column/row (respectively). For a set V we write $\mathbf{M} \in \mathbb{R}^{m \times V}$ for the $m \times |V|$ matrix where we can index columns by $x \in V$, i.e., $\mathbf{M}_x \in \mathbb{R}^m$. For any vector $x \in \mathbb{R}^n$, the Hamming weight of x is the number of non-zero entries of x also denoted by $\mathrm{nnz}(x)$. For Boolean vectors $x, y \in \{0,1\}^n$, the Hamming distance between x and y is given by $\|x - y\|_1$. Finally, for any vector $x \in \mathbb{R}^n$ and positive definite matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$, define the $\mathbf{L}$-induced norm of x by $\|x\|_{\mathbf{L}} = \sqrt{x^\top \mathbf{L} x}$.

VC-Dimension. A range space (or a set system) is a pair $\mathfrak{R} = (X, R)$ where X is a set and R is a set of subsets of X. Let $\Pi_R(A) = \{A \cap r : r \in R\}$. The VC-dimension of $\mathfrak{R}$ is $\mathrm{VC}(\mathfrak{R}) = \max\{|S| : S \subseteq X \text{ and } |\Pi_R(S)| = 2^{|S|}\}$. The dual range space of $\mathfrak{R}$ is then given by $\mathfrak{R}^\top = (R, \{\{r | x \in r\} | x \in X\})$, and the dual VC-dimension of $\mathfrak{R}$ is the VC-dimension of $\mathfrak{R}^\top$. Any matrix $\mathbf{M} \in \{0,1\}^{m \times n}$

corresponds to a set family $\mathcal{F}_{\mathbf{M}} = \{\{j : \mathbf{M}_{i,j} = 1\} : i \in [m]\}$, i.e., each row of $\mathbf{M}$ is the indicator vector of some set. Through it, we can encode $\mathbf{M}$ by the range space $\mathfrak{R}_M = ([n], \mathcal{F}_{\mathbf{M}})$. Then, the VC-dimension of $\mathbf{M}$ is defined as the VC-dimension of $\mathfrak{R}_{\mathbf{M}}$, and the dual VC-dimension is the VC-dimension of $\mathbf{M}^\top$. Here, a subset S of the columns of $\mathbf{M}$ is said to be *shattered* if each of the $2^{|S|}$ many 0/1 strings appears in some row in the restriction of $\mathbf{M}$ to the columns in S. Then, the VC-dimension of $\mathbf{M}$ is the maximum size of a shattered subset of the columns of $\mathbf{M}$. Therefore, we see that the VC-dimension $d_{m,n}$ of a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$ satisfies $1 \leq d_{m,n} \leq \log m$.

3 The Structural Complexity of Matrix-Vector Multiplication

We present a technical overview of the algorithms and main theorems. We begin this section with an overview of Björklund and Lingas [2001], Alves et al. [2024] who established a connection between matrix vector products and minimum spanning trees on n points $\{0,1\}^m$, where the edge weights are given by the Hamming distance between two points (i.e., matrix $\mathbf{M}$ defines a collection of points).

Having established this connection, we prove one of our main results in Section 3.2. We prove via techniques from computational geometry that the weight of the minimum spanning tree is bounded if the matrix $\mathbf{M}$ has corrupted VC-dimension at most d. This yields our main result in Theorem 1.3.

3.1 Matrix Vector Products via Minimum Spanning Trees

We here recap the algorithm idea by Björklund and Lingas [2001], which was later independently rediscovered by Alves et al. [2024]. We also explain how to extend the results from $\mathbf{M}$ to $\mathbf{M}^\top$.

Differential Compression. Given a binary matrix $\mathbf{M} \in \{0,1\}^{m \times n}$, let $\mathbf{M}_i$ denote its i'th column. We compress $\mathbf{M}$ by writing each column $\mathbf{M}_x$ as the sum of another column $\mathbf{M}_y$ and a *change vector* $\Delta_{x,y}$ such that $\mathbf{M}_y = \mathbf{M}_x + \Delta_{x,y}$ and $\Delta_{x,y} = \mathbf{M}_x - \mathbf{M}_y$. If $\mathbf{M}_x$ and $\mathbf{M}_y$ are similar, then $\|\Delta_{x,y}\|_1$ is likely to be smaller than the number of nonzero elements of $\mathbf{M}_x$. So, it is more efficient to indirectly represent $\mathbf{M}_x$ with respect to $\mathbf{M}_y$, instead of through $\mathbf{M}$. This requires an algorithm that finds a chain of Δ's to represent all columns of $\mathbf{M}$: for each column $\mathbf{M}_x$, identify a similar column $\mathbf{M}_y$ such that the Hamming weight $\|\Delta_{x,y}\|_1$ required to represent $\mathbf{M}_x$ is minimized subject to $\mathbf{M}_y$. Since the algorithm measures the Hamming distance for each pair of columns in $\mathbf{M}$, we model this compression by an undirected weighted graph G on n vertices, where vertex i represents $\mathbf{M}_i$, and the weight of each edge (x, y) is $\|\Delta_{x,y}\|_1$. We can then find a Minimum Spanning Tree (MST) of G, which by definition spans G with the minimum sum of edge weights possible. Therefore, any MST of G rooted at vertex r defines a chain of Δ's that, starting at r, can represent all the columns of $\mathbf{M}$.

We formalize this intuition by defining the Δ-labeled spanning tree of a matrix and its weight.

Definition 3.1 (Δ-labeled spanning tree of matrix $\mathbf{M}$)**.** *A Δ-labeled tree is a tree $T = (V, E)$ with some explicit root x, where each edge $e \in E$ is directed away from x and labeled by a vector $\Delta_e \in \mathbb{R}^m$. We say T is a Δ-labeled spanning tree for matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ if $\mathbf{M}_x = \vec{0}$ for root x, and for all edges $(i, j) \in E$ we have $\Delta_{i,j} \coloneqq \mathbf{M}_j - \mathbf{M}_i$.*

Definition 3.2 (Weight of a Δ-labeled spanning tree)**.** *For any Δ-labeled spanning tree of $\mathbf{M}$ denoted by T with directed edge-set E, define* $\mathrm{weight}(T) = \sum_{e \in T} \mathrm{nnz}(\Delta_e)$.

The minimum spanning tree of $\mathbf{M}$ is then the Δ-labeled spanning tree with the smallest possible weight. Given *any* spanning tree of $\mathbf{M}$, we use the weight of the Δ-labeled spanning tree to parameterize the runtime of the matrix-vector multiplication algorithm. Here, the lower the weight of the Δ-labeled spanning tree, the faster the runtime.

Lemma 3.3 (Björklund and Lingas [2001], Alves et al. [2024])**.** *Given a Δ-labeled spanning tree T of $\mathbf{M}^\top \in \{0,1\}^{n \times m}$ and a vector $v \in \mathbb{R}^n$, we can compute $\mathbf{M}v$ in $O(\mathrm{weight}(T) + n + m)$ time.*

We describe this procedure in Algorithm 1. As a quick recap, the algorithm computes $\mathbf{M}v$ by starting at a root node $r \in T$ of the spanning tree and performing depth-first search (DFS). The algorithm initializes an output vector and computes $(\mathbf{M}v)_r$ in $O(n)$ time. For each edge (x, y) traversed by DFS on the tree, the algorithm computes $(\mathbf{M}v)_y = (\mathbf{M}v)_x + \Delta_{y,x}^\top v$. The complexity of this iterative procedure is proportional to $\mathrm{weight}(T) = \sum_{e \in E(T)} \delta_e$ where for each edge $e = (x, y)$ in the Δ-labeled spanning tree, δ_e is the number of non-zeros in Δ_e, i.e., the Hamming distance between row x and row y of $\mathbf{M}$. For completeness, we prove Lemma 3.3 in Appendix B.1.

Since the weight of the Δ-labeled spanning tree of $\mathbf{M}^\top$ can be different from the weight of the Δ-labeled spanning tree of $\mathbf{M}$, we provide a new algorithm with an analogous guarantee in Lemma 3.4.

Lemma 3.4. *Given a Δ-labeled spanning tree T of $\mathbf{M} \in \{0,1\}^{m\times n}$ and a vector $v \in \mathbb{R}^n$, there is an algorithm that can compute $\mathbf{M}v$ in $O(\text{weight}(T) + n + m)$ time.*

To utilize the Δ-labeled spanning tree of $\mathbf{M}$, note that $\mathbf{M}v = \sum_{i=1}^{n} \mathbf{M}_i v_i$. In the special case where $\mathbf{M}$ has only two columns, $\mathbf{M}v = \mathbf{M}_1 v_1 + \mathbf{M}_2 v_2 = \mathbf{M}_1(v_1 + v_2) + \Delta_{1,2} v_2$. For more columns, this can be extended to require only a product with $\mathbf{M}_1$ and products with each Δ_e in T. Thus the complexity is bounded by $O(m + n + \text{weight}(T))$. We prove Lemma 3.4 in Appendix B.1.

To minimize the time complexity, we compute a MST. In Alves et al. [2024] this was done naïvely, whereas in Björklund and Lingas [2001] this is done via Theorem 4.2 of Indyk and Motwani [1998].

Lemma 3.5 (Theorem 4.2 of Indyk and Motwani [1998])**.** *For $\epsilon > 0$, we can compute a $(1+\epsilon)$-approximate MST with respect to the Hamming distance of n points in $\{0,1\}^m$ in time $\widetilde{O}(mn^{1+\frac{1}{1+\epsilon}})$.*

Theorem 3.6. *Given a Boolean $m \times n$ matrix $\mathbf{M}$, we can construct in $\widetilde{O}(nm)$ time a Δ-labeled spanning tree whose weight is at most a $O(\log n)$-factor larger than the MST of $\mathbf{M}$.*

Proof. Run Lemma 3.5 on the columns of $\mathbf{M}$ for $\epsilon = \log n$ to construct an $O(\log n)$-approximate MST in $\widetilde{O}(mn)$ time. Next, iterate over the edges of the MST and label each tree edge (x, y) by $\Delta_{x,y} = \mathbf{M}_y - \mathbf{M}_x$. This takes $\widetilde{O}(mn)$ time as there are only $n-1$ tree edges and $\Delta_{x,y} \in \mathbb{R}^m$. □

3.2 Existence of Low Weight Δ-labeled Minimum Spanning Trees

The Hamming distance between any two columns in $\mathbf{M}$ is at most m. Since there are $O(n)$ edges in the Δ-labeled MST, this gives a coarse bound of $O(mn)$ which is achieved for Hadamard matrices. One of our main technical contributions is to prove that Δ-labeled MST's with low weight exist for structure matrices. We bound the MST's weight parameterized by the corrupted VC-dimension of $\mathbf{M}$.

Theorem 3.7. *If a matrix $\mathbf{M}^\top \in \{0,1\}^{n\times m}$ has corrupted VC-dimension d, then the weight the minimum spanning tree of $\mathbf{M}$ is at most $O(mn^{1-1/d}\log^2 n)$.*

We show that for matrices $\mathbf{M}$ whose transpose $\mathbf{M}^\top$ has bounded corrupted VC-dimension, the MST has small weight. For this, we work directly with the definition of VC-dimension and use set terminology, which allows us to use relevant work from Welzl [1988]. This previous work studied bounded VC-dimensions in the context of computational geometry. We here use their results to bound the complexity of matrix vector products.

Definition 3.8 (Set crossings)**.** *Let A and r be sets. r* crosses *A if and only if neither $A \subseteq r$ nor $A \cap r = \emptyset$ holds. That is, there are $x, y \in A$ with $x \in r$ and $y \notin r$.*

Definition 3.9 (Spanning Tree)**.** *Let $A \subset X$ be a finite set of elements of a range space (X, R). A spanning tree on A is an undirected (unrooted) tree with node set A.*

Definition 3.10 (Weight of spanning tree T)**.** *A crossing of a range $r \in R$ in T is an edge in T that is crossed by r (i.e., an edge that has one endpoint in r and the other endpoint in $X \setminus r$). The crossing number $\kappa_r(T)$ of a range $r \in R$ is the number of crossings of r in T. The weight of tree T is defined as $w(T) = \sum_{r\in R} \kappa_r(T)$.*

This spanning tree corresponds to the Δ-labeled spanning tree for matrix $\mathbf{M}$, where the range space (X, R) induced by $\mathbf{M} \in \{0,1\}^{m\times n}$ has ground set $X = [n]$ (i.e., column indices) with the rows of $\mathbf{M}$ being the indicator vectors of m subsets of X. Thus, a spanning tree T_X for (X, R) and a spanning tree T_M for $\mathbf{M}$ both consist of edges connecting elements in $[n]$, and the number of crossings for an edge $(x, y) \in T_X$ equals the number of indices i where $(\mathbf{M}_x)_i$ and $(\mathbf{M}_y)_i$ differ. Hence, $w(T_X)$ is the weight of the corresponding Δ-labeled spanning tree of $\mathbf{M}$.

We show that when the dual of (X, R) has corrupted VC-dimension d, then (X, R) has spanning trees with small weight. The literature on finding spanning trees on X with few crossings stems from works in computational geometry from the 1900s [Welzl, 1988, B. Chazelle, 1989, Matoušek, 1991, Har-Peled, 2009] which used the trees to preprocess a set of points in $\mathbb{R}^d$ and answer queries where for any convex polytope, the query returns a point in the polytope. We transfer their techniques to our problem. The following lemma allows us to prove the existence of a Δ-labeled spanning tree for matrices whose transpose has bounded VC-dimension. We later extend this to corrupted VC-dimension.

Lemma 3.11 (Lemma 4.1 of Welzl [1988])**.** *Let (X, R) be a range space with dual VC-dimension at most d. For every $A \subseteq X, |A| = n \geq 1$ and every multiset Q of ranges in R with $|Q| = m$, there exists $x \neq y \in A$ such that the number of ranges in Q crossing (x, y) is at most $O(mn^{-1/d} \log^2 n)$.*

Remark 3.12. *The original proof of Lemma 4.1 of Welzl [1988] gives a bound of $O(mn^{-1/d} \log n)$ but hides some terms that depend on the VC-dimension: the original statement of Lemma 4.1 says that the constant c, hidden in the $O(\cdot)$ notation, depends on (X, R), but this is vacuous in our setting as (X, R) is the entire problem instance. We show that $c = O(d) \leq O(\log n)$ suffices, and provide a proof in B.2 for completeness.*

Lemma 3.13. *Let (X, R) be a range-space with $|X| = n, |R| = m$ with dual VC-dimension at most d. Then there exists a spanning tree T of weight $w(T) = O(mn^{1-1/d} \log^2 n)$.*

Proof. Let $A_0 = X$. Consider the following process initialized at $i = 0$. By Lemma 3.11, there exists distinct $x_i, y_i \in A_i$ where the number of ranges in R crossing (x_i, y_i) is $O(m|A_i|^{-1/d} \log^2 n)$. Iteratively letting $A_{i+1} \leftarrow A_i \setminus x_i$ constructs a spanning tree T with weight at most

$$\sum_{i=1}^{n} O(m(n-i)^{-\frac{1}{d}} \log^2(n-i)) \leq O\left(m \log^2 n \int_1^n x^{-\frac{1}{d}} \mathrm{d}x\right) = O(mn^{1-\frac{1}{d}} \log^2 n). \qquad \square$$

We extend Lemma 3.13 to corrupted VC-dimensions by converting the tree to a single line of vertices, so that any single corruption minimally affects the number of crossings.

For instance, suppose $X = [n]$, R is a multiset of m empty ranges (corresponding to an $m \times n$ zero matrix $\mathbf{M}$), and the spanning tree is a star with $1 \in X$ at the center. If the corruption adds 1 to each empty set in R (equivalently, turning the first column of $\mathbf{M}$ to an all-1-column), this fits into the definition of the dual having corrupted VC-dimension. This is because each set of (X, R) is corrupted by only one element, and thus in the dual set system each element is corrupted by adding it to one set.

After corruption, the number of crossings of each tree edge is $O(m)$ as 1 is in each of the m sets, but none of the other elements in X are in any set. As the tree has $n-1$ edges, each connecting 1 to another element of X, the tree has $O(mn)$ crossings in total. If, however, the tree was a single line, the corruption would affect at most 2 edges, and the total number of crossings would only increase by $O(m)$ rather than $O(mn)$. We convert the tree to a single line in Lemma 3.14, proven in Appendix B.2.

Lemma 3.14. *For a range space (X, R), $|X| = n$ with spanning tree T, there exists a permutation $\pi \in S_n$ such that the weight of the corresponding spanning tree T' (which is just the line $\pi(1), \pi(2), ..., \pi(n)$) is at most twice that of T.*

Proof of Theorem 3.7. Suppose $\mathbf{M}^\top = \mathbf{L}^\top + \mathbf{S}^\top \in \{0, 1\}^{n \times m}$ has corrupted VC-dimension $\leq d$, where $\mathbf{L}^\top \in \{0, 1\}^{n \times m}$ has VC-dimension d and $\mathbf{S}^\top \in \{-1, 0, 1\}^{n \times m}$ has at most $O(n^{1-1/d})$ non-zero entries per column. $\mathbf{L}$ and $\mathbf{S}$ exist by definition of corrupted VC-dimension.

Let (X, R) be the set system induced by $\mathbf{L}$ (i.e., each row of $\mathbf{L}$ is the indicator vector of a set in R with $|R| = m$), by $\mathbf{L}^\top$ having VC-dimension $\leq d$, (X, R) has dual VC-dimension $\leq d$. By Lemma 3.13, (X, R) has a spanning tree T with $w(T) \leq O(mn^{1-1/d} \log^2 n)$. Then, by Lemma 3.14, there exists a spanning tree T' which is a line with weight at most $O(mn^{1-1/d} \log^2 n)$. Let (X, R_c) be the range space of $\mathbf{M}$. T' has the same weight as (X, R_C) as on (X, R), excluding crossings due to the corruption. For range $r \in R_c$, we count the number of new crossings it incurs due to the corruption: an edge (x, y) only has a new crossing for a range $r \in R_c$ if x or y was corrupted in r. Since each $x \in X$ has at most 2 incident edges in T' (since it is a line) each corruption to r causes at most 2 new crossings. Since the total number of corruptions is bounded by $O(mn^{1-1/d})$, we have $w(T') \leq O(mn^{1-1/d} \log^2 n)$ and T' is a Δ-labeled spanning tree for $\mathbf{M}$, as $X = [n]$ are the column indices and the number of crossings for an edge (x, y) is the number of entries where $\mathbf{M}_x$ and $\mathbf{M}_y$ differ. Since the MST has lower weight, its weight is at most $O(mn^{1-1/d} \log^2 n)$. $\square$

We can now put everything together to prove Theorem 1.3.

Theorem 1.3. *(Static Online Matrix-Vector Multiplication). If a matrix $\mathbf{M} \in \{0, 1\}^{m \times n}$ has corrupted VC-dimension d, then after an $\widetilde{O}(mn)$-time preprocessing, there is a data structure $\mathcal{D}$ that can compute $\mathbf{M}v$ for any $v \in \mathbb{R}^n$ in $\widetilde{O}(nm^{1-1/d} + m)$ time, with high probability.*

Proof. Preprocess $\mathbf{M}$ by constructing a Δ-labeled spanning tree T for $\mathbf{M}^\top$ via Theorem 3.6 in $\widetilde{O}(mn)$ time. Given $v \in \mathbb{R}^n$, pass v and T to the algorithm of Lemma 3.3. It returns $\mathbf{M}v$ in time $\widetilde{O}(m+n+\text{weight}(T))$. By Theorem 3.7, the MST of $\mathbf{M}^\top$ has weight at most $\widetilde{O}(nm^{1-1/d})$ since $\mathbf{M}$ has corrupted VC-dimension at most d. Since T is a $O(\log n)$-approximation of the MST, answering a query takes $\widetilde{O}(nm^{1-1/d}+m)$ time. □

3.3 Dynamic Matrices

At last, we outline how to handle matrices $\mathbf{M}$ that receive updates over time, i.e., Theorem 1.4.

Theorem 1.4. *(Dynamic Online Matrix-Vector Multiplication). Given a matrix $\mathbf{M} \in \{0,1\}^{m\times n}$, there is a data structure $\mathcal{D}$ with $\widetilde{O}(mn)$ preprocessing time that supports row and column updates (insertions/deletions) to $\mathbf{M}$ in $\widetilde{O}(n)$ and $\widetilde{O}(m)$ time, respectively. Upon querying $\mathcal{D}$ with a vector $v \in \mathbb{R}^n$, it outputs $\mathbf{M}v$ in $\widetilde{O}(nm^{1-1/d^*}+m)$ time, with high probability, where d^* is the largest corrupted VC-dimension of $\mathbf{M}$ throughout the history of its updates.*

We sketch how to handle column insertions: Let $\mathbf{M}$ be the current matrix. We split it into smaller matrices $\mathbf{M}^{(0)}, \mathbf{M}^{(1)}, ..., \mathbf{M}^{(\log n)}$, where each $\mathbf{M}^{(i)}$ contains at most 2^i columns of $\mathbf{M}$. When a new column is inserted to $\mathbf{M}$, we insert that column to $\mathbf{M}^{(0)}$. When any $\mathbf{M}^{(i)}$ contains more than 2^i columns, then all its columns are inserted to $\mathbf{M}^{(i+1)}$ and $\mathbf{M}^{(i)}$ is reset to an empty matrix. Thus, any $\mathbf{M}^{(i)}$ receives new columns at most once every 2^i iterations to $\mathbf{M}$. When an $\mathbf{M}^{(i)}$ receives new columns, we initialize a new matrix-vector multiplication data structure (Theorem 1.3) on it. This takes $\widetilde{O}(m2^i)$ time, but amortizes to $\widetilde{O}(m)$ time per insertion to $\mathbf{M}$ since it only happens at most once every 2^i insertions. This yields an amortized time of $\sum_{i=0}^{\log n}\widetilde{O}(m) = \widetilde{O}(m)$ time to maintain all $\mathbf{M}^{(i)}$ for $i = 0, ..., \log n$. To answer a matrix-vector multiplication query $\mathbf{M}v$, we split v into vectors $v^{(0)}, ..., v^{(\log n)}$ such that $\mathbf{M}v = \sum_{i=0}^{\log n}\mathbf{M}^{(i)}v^{(i)}$. For the time complexity, note that each $\mathbf{M}^{(i)}$ is a submatrix of $\mathbf{M}$. Thus, the the time complexity needed to multiply a vector by $\mathbf{M}^{(i)}$ is bounded by $\widetilde{O}(nm^{1-1/d}+m)$.

We ignore column deletions to $\mathbf{M}^{(i)}$ until $\mathbf{M}^{(i)}$ is reinitialized: simply set $v_j = 0$ for the columns that should have been deleted. We can handle row updates to $\mathbf{M}$ in a similar way, by splitting the matrix again into $\log m$ matrices, each containing at most 2^j rows. The full proof is given in Appendix B.3.

4 Conclusion

We study the structural complexity of matrix-vector multiplication. We propose a theoretical framework to study heuristic algorithms that have achieved tremendous practical success. Building on the VC-dimension literature, we propose the notion of a corrupted VC-dimension d and provide structural characterizations. We show that if a matrix $\mathbf{M} \in \{0,1\}^{m\times n}$ has corrupted VC-dimension d, then its matrix-vector product $\mathbf{M}v$ can be computed in $\widetilde{O}(nm^{1-1/d}+m)$ time, providing polynomial speed-ups over existing methods. Moreover, our algorithm maintains updates to $\mathbf{M}$ in $\widetilde{O}(m)$ and $\widetilde{O}(n)$ time, leading to the first $\widetilde{O}(n^{2-\epsilon})$-time algorithms for maintaining high-accuracy estimates for Laplacian solvers, effective resistance, and triangle detection in dynamic graphs.

Limitations. While we show subquadratic runtimes in non-Boolean matrices with bounded Pollard pseudodimension, we believe that the linear dependence of the number of distinct values T in the runtime is suboptimal and should instead be logarithmic. Finally, without matching lower bounds, we cannot verify whether the $\widetilde{O}(n^{2-1/d})$ scaling with respect to the VC-dimension is optimal.

Societal Impacts. This work is foundational in nature. As such, while it enables more efficient computation in iterative algorithms, it is not tied to any specific applications or deployments.

Acknowledgements

This work was supported by NSF Grants DMS 2202961 and CCF 2338816. We also express our thanks to Ce Jin, Andrzej Lingas, Albert Weng, Jesper Jansson, Shunhua Jiang, Xiaobai Sun, Elia Gorokhovsky, and Jingtong Sun for insightful discussions.

References

Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. *IEEE 57th Annual Symposium on Foundations of Computer Science*, 2016.

Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi: 10.1109/FOCS.2014.53. URL https://doi.org/10.1109/FOCS.2014.53.

Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms. *STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, 2024.

Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 335–344, 2016. doi: 10.1109/FOCS.2016.44.

Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, 2017. URL https://arxiv.org/abs/1711.06530.

Josh Alman and Virginia Vassilevska Williams. OV graphs are (probably) hard instances. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 83:1–83:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPICS.ITCS.2020.83.

Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 541–552. IEEE, 2020. doi: 10.1109/FOCS46700.2020.00057.

Noga Alon, Shay Moran, and Amir Yehudayoff. Sign rank versus VC dimension. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 47–80, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

João N. F. Alves, Samir Moustafa, Siegfried Benkner, Alexandre P. Francisco, Wilfried N. Gansterer, and Luís M. S. Russo. Accelerating graph neural networks with a novel matrix compression format, 2024.

Emile Anand and Sarah Liaw. Feel-good thompson sampling for contextual bandits: a markov chain monte carlo showdown, 2025. URL https://arxiv.org/abs/2507.15290.

Emile Anand and Guannan Qu. Efficient reinforcement learning for global decision making in the presence of local agents at scale. *arXiv preprint arXiv:2403.00222*, 2024.

Emile Anand and Chris Umans. Pseudorandomness of the sticky random walk. *Caltech Thesis Archives*, 2023. URL https://arxiv.org/abs/2307.11104.

Emile Anand, Ishani Karmarkar, and Guannan Qu. Mean-field sampling for cooperative multi-agent reinforcement learning. *ArXiv*, abs/2412.00661, 2024a.

Emile Anand, Jan van den Brand, Mehrdad Ghadiri, and Daniel J. Zhang. The Bit Complexity of Dynamic Algebraic Formulas and Their Determinants. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2024b. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-322-5. doi: 10.4230/LIPIcs.ICALP.2024.10. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2024.10.

Rosa I Arriaga and Santosh Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Machine learning*, 63(2):161–182, 2006.

P. Assouad. Densité et dimension. *Annales de l'Institut Fourier*, 33:233–282, 1983.

Emo Welzl B. Chazelle. Quasi-optimal range searching in spaces of finite vc-dimension. *Discrete & computational geometry*, 4(5):467–490, 1989. URL http://eudml.org/doc/131092.

Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019. URL http://jmlr.org/papers/v20/17-612.html.

Julien Basch, Sanjeev Khanna, and Rajeev Motwani. On diameter verification and boolean matrix multiplication. Technical report, Stanford University, Stanford, CA, USA, 1995.

Andreas Björklund and Andrzej Lingas. Fast boolean matrix multiplication for highly clustered data. In Frank Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures*, pages 258–263, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44634-7.

Nicolas Bousquet, Aurélie Lagoutte, Zhentao Li, Aline Parreau, and Stéphan Thomassé. Identifying codes in hereditary classes of graphs and vc-dimension. *SIAM Journal on Discrete Mathematics*, 29(4):2047–2064, January 2015. ISSN 1095-7146. doi: 10.1137/14097879x.

Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *SODA*, pages 259–278. SIAM, 2020.

Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *FOCS*, pages 456–480. IEEE Computer Society, 2019a. URL https://arxiv.org/pdf/1905.05067.pdf.

Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 456–480. IEEE, 2019b.

Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 775–788. ACM, 2020. doi: 10.1145/3357713.3384309. URL https://dl.acm.org/doi/abs/10.1145/3357713.3384309.

Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1)-regression in nearly linear time for dense instances. In *STOC*, pages 859–869. ACM, 2021.

Jan van den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1011–1022. IEEE, 2022. doi: 10.1109/FOCS54457.2022.00099. URL https://doi.org/10.1109/FOCS54457.2022.00099.

Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3534–3557. SIAM, 2024. doi: 10.1137/1.9781611977912.126.

Diptarka Chakraborty, Lior Kamma, and Kasper Green Larsen. Tight cell probe bounds for succinct boolean matrix-vector multiplication. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1297–1306. ACM, 2018. doi: 10.1145/3188745.3188830.

Shreyas Chaudhari, Srinivasa Pranav, Emile Anand, and José M. F. Moura. Peer-to-peer learning dynamics of wide neural networks. *IEEE International Conference on Acoustics, Speech, and Signal Processing 2025*, 2024. URL https://arxiv.org/abs/2409.15267.

Jingbang Chen, Meng He, J. Ian Munro, Richard Peng, Kaiyu Wu, and Daniel J. Zhang. Distance queries over dynamic interval graphs. *Comput. Geom.*, 122:102103, 2024. doi: 10.1016/J.COMGEO.2024.102103. URL https://doi.org/10.1016/j.comgeo.2024.102103.

Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1):3:1–3:39, 2021. Announced at STOC'19.

David Coudert, Mónika Csikós, Guillaume Ducoffe, and Laurent Viennot. Practical Computation of Graph VC-Dimension. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms (SEA 2024)*, volume 301 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-325-6. doi: 10.4230/LIPIcs.SEA.2024.8.

Christophe Crespelle. Fully dynamic representations of interval graphs. *Theor. Comput. Sci.*, 759: 14–49, 2019. doi: 10.1016/J.TCS.2019.01.007.

Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for-center on graphs. In *SODA*, pages 3441–3462, 2024.

Mohsen Dehghankar, Mahdi Erfanian, and Abolfazl Asudeh. Optimized inference for 1.58-bit llms: A time and memory-efficient algorithm for binary and ternary matrix multiplication. *arXiv preprint arXiv:2411.06360*, 2024.

Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. *Journal of Computer and System Sciences*, 78(5):1606–1622, 2012. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2012.02.004. JCSS Special Issue: Cloud Computing 2011.

David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 914–925. ACM, 2019. doi: 10.1145/3313276.3316379.

David Eppstein. Graphs with bounded radial order are small. In *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA)*, volume 979 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1995. doi: 10.1007/3-540-60313-1_165.

Vissarion Fisikopoulos and Luis Penaranda. Faster geometric algorithms via dynamic determinant computation. *Computational Geometry*, 54:1–16, 2016.

Peter Floderus, Jesper Jansson, Christos Levcopoulos, Andrzej Lingas, and Dzmitry Sledneu. 3d rectangulations and geometric matrix multiplication. *Algorithmica*, 80(1):136–154, January 2018. ISSN 0178-4617. doi: 10.1007/s00453-016-0247-3. URL `https://doi.org/10.1007/s00453-016-0247-3`.

Dimitris Floros, Nikos Pitsianis, and Xiaobai Sun. Algebraic vertex ordering of a sparse graph for adjacency access locality and graph compression, 2024.

Gudmund Skovbjerg Frandsen, Johan P. Hansen, and Peter Bro Miltersen. Lower bounds for dynamic algebraic problems. *Inf. Comput.*, 171(2):333–349, 2001. doi: 10.1006/INCO.2001.3046.

Leszek Gasieniec and Andrzej Lingas. An improved bound on boolean matrix multiplication for highly clustered data. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Michiel H. M. Smid, editors, *Algorithms and Data Structures, 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 329–339. Springer, 2003. doi: 10.1007/978-3-540-45078-8_29.

Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM Review*, 50(1):37–66, 2008. doi: 10.1137/050645452.

Raphael Clifford. Allan Gronlund and Kasper Green Larsen. New unconditional hardness results for dynamic and online problems. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, page 1089–1107, USA, 2015. IEEE Computer Society. ISBN 9781467381918. doi: 10.1109/FOCS.2015.71.

Sariel Har-Peled. Approximating spanning trees with low crossing number. *ArXiv*, abs/0907.1131, 2009. URL `https://api.semanticscholar.org/CorpusID:15811439`.

D Haussler and E Welzl. Epsilon-nets and simplex range queries. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, page 61–71, New York, NY, USA, 1986. Association for Computing Machinery. ISBN 0897911946. doi: 10.1145/10515.10522. URL https://doi.org/10.1145/10515.10522.

Monika Henzinger and Stefan Neumann. Incremental and Fully Dynamic Subgraph Connectivity For Emergency Planning. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-015-6. doi: 10.4230/LIPIcs.ESA.2016.48.

Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 21–30, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335362. doi: 10.1145/2746539.2746609.

Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 459–498. SIAM, 2022. doi: 10.1137/1.9781611977073.23.

Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. On the complexity of algorithms with predictions for dynamic graph problems. *15th Innovations in Theoretical Computer Science Conference, ITCS 2024*, 287:62:1–62:25, 2024. doi: 10.4230/LIPICS.ITCS.2024.62.

Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. ISSN 0166-218X. doi: https://doi.org/10.1016/0166-218X(79)90044-1.

Bingbing Hu and Adam Polak. Non-boolean omv: One more reason to believe lower bounds for dynamic problems. *arXiv preprint arXiv:2409.15970*, 2024.

Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving SDP faster: A robust IPM framework and efficient implementation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 233–244. IEEE, 2022. doi: 10.1109/FOCS54457.2022.00029.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276876.

Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *FOCS*, pages 910–918. IEEE, 2020a.

Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In *STOC*, pages 944–953. ACM, 2020b.

Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *STOC*, pages 823–832. ACM, 2021.

Shunhua Jiang, Bento Natura, and Omri Weinstein. A faster interior-point method for sum-of-squares optimization. In *ICALP*, volume 229 of *LIPIcs*, pages 79:1–79:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

Shunhua Jiang, Binghui Peng, and Omri Weinstein. The complexity of dynamic least-squares regression. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1605–1627. IEEE, 2023. doi: 10.1109/FOCS57990.2023.00097.

Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized bmm and omv. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1515–1528, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520036.

Adam Karczmarz and Da Wei Zheng. Subquadratic algorithms in minor-free digraphs: (weighted) distance oracles, decremental reachability, and more. *arXiV*, 2024.

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4): 373–396, 1984.

Toni Maria Klein, Christine Blome, C. Elise Kleyn, Curdin Conrad, Paul G. Sator, Mona Ståhle, Kilian Eyerich, Marc Alexander Radtke, Christine Bundy, Myriam Cordey, Christopher E. M. Griffiths, and Matthias Augustin. Real-world experience of patient-relevant benefits and treatment satisfaction with apremilast in patients with psoriasis: An analysis of the appreciate study. *Dermatology and Therapy*, 12(1):81–95, 2022. doi: 10.1007/s13555-021-00628-3.

Tuukka Korhonen, Wojciech Nadara, Michal Pilipczuk, and Marek Sokolowski. Fully dynamic approximation schemes on planar and apex-minor-free graphs. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 296–313. SIAM, 2024. doi: 10.1137/1.9781611977912.12.

Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *FOCS*, pages 573–582. IEEE Computer Society, 2016.

Kasper Green Larsen and Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 2182–2189, USA, 2017. Society for Industrial and Applied Mathematics. URL `https://epubs.siam.org/doi/10.1137/1.9781611974782.142`.

Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *FOCS*, pages 230–249. IEEE Computer Society, 2015.

Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, volume 99 of *Proceedings of Machine Learning Research*, pages 2140–2157. PMLR, 2019.

Edo Liberty and Steven W. Zucker. The mailman algorithm: A note on matrix–vector multiplication. *Information Processing Letters*, 109(3):179–182, 2006. ISSN 0020-0190. doi: https://doi.org/10.1016/j.ipl.2008.09.028.

Yiheng Lin, James A Preiss, Emile Timothy Anand, Yingying Li, Yisong Yue, and Adam Wierman. Online adaptive policy selection in time-varying systems: No-regret via contractive perturbations. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/a7a7180fe7f82ff98eee0827c5e9c141-Paper-Conference.pdf`.

Yiheng Lin, James A Preiss, Emile Timothy Anand, Yingying Li, Yisong Yue, and Adam Wierman. Learning-augmented control via online adaptive policy selection: No regret via contractive perturbations. In *ACM SIGMETRICS, Workshop on Learning-augmented Algorithms: Theory and Applications 2023*, 2023b.

Yiheng Lin, James A. Preiss, Fengze Xie, Emile Anand, Soon-Jo Chung, Yisong Yue, and Adam Wierman. Online policy optimization in unknown nonlinear systems. In Shipra Agrawal and Aaron Roth, editors, *Proceedings of Thirty Seventh Conference on Learning Theory*, volume 247 of *Proceedings of Machine Learning Research*, pages 3475–3522. PMLR, 30 Jun–03 Jul 2024. URL `https://proceedings.mlr.press/v247/lin24a.html`.

Andrzej Lingas. A geometric approach to boolean matrix multiplication. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, ISAAC '02, page 501–510, Berlin, Heidelberg, 2002. Springer-Verlag. ISBN 3540001425.

Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication, 2024.

Alane Marie de Lima, Murilo da Silva, and André Vignatti. A range space with constant vc dimension for all-pairs shortest paths in graphs. *Journal of Graph Algorithms and Applications*, 27:603–619, 08 2023. doi: 10.7155/jgaa.00636.

Jiří Matoušek. Spanning trees with low crossing number. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 25(2):103–123, 1991.

Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer, 2002. ISBN 978-0-387-95373-0. doi: 10.1007/978-1-4613-0039-7.

Elchanan Mossel and Christopher Umans. On the complexity of approximating the vc dimension. *Journal of Computer and System Sciences*, 65(4):660–671, 2002. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(02)00022-3. URL http://ieeexplore.ieee.org/document/933889. Special Issue on Complexity 2001.

Theodore S. Motzkin. Evaluation of polynomials and evaluation of rational functions. *Bulletin of the American Mathematical Society*, 57(2):163–169, 1951.

Riley Murray, Venkat Chandrasekaran, and Adam Wierman. Newton polytopes and relative entropy optimization. *Foundations of Computational Mathematics*, 21(6):1703–1737, 2021. doi: 10.1007/s10208-021-09497-w.

Tung Nguyen, Alex Scott, and Paul Seymour. Induced subgraph density. vi. bounded vc-dimension. *arXiV*, 2024. URL https://arxiv.org/abs/2312.15572.

Vadim Olshevsky and Amin Shokrollahi. Matrix-vector product for confluent cauchy-like matrices with application to confluent rational interpolation. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 573–581, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581131844. doi: 10.1145/335305.335380. URL https://dl.acm.org/doi/10.1145/335305.335380.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999. URL http://ilpubs.stanford.edu/422/.

Victor Y. Pan and Elias P. Tsigaridas. Nearly optimal computations with structured matrices. In *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation*, SNC '14, page 21–30, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329637. doi: 10.1145/2631948.2631954. URL https://doi.org/10.1145/2631948.2631954.

James Renegar. Condition numbers, the barrier method, and the conjugate-gradient method. *SIAM Journal on Optimization*, 6(4):879–912, 1996. doi: 10.1137/S105262349427532X. URL https://doi.org/10.1137/S105262349427532X.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, Cambridge, UK, 2014. ISBN 978-1-107-05713-5.

Hans Henrik Brandenborg Sørensen. High-performance matrix-vector multiplication on the gpu. In Michael Alexander, Pasqua D'Ambra, Adam Belloum, George Bosilca, Mario Cannataro, Marco Danelutto, Beniamino Di Martino, Michael Gerndt, Emmanuel Jeannot, Raymond Namyst, Jean Roman, Stephen L. Scott, Jesper Larsson Traff, Geoffroy Vallée, and Josef Weidendorfer, editors, *Euro-Par 2011: Parallel Processing Workshops*, pages 377–386, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29737-3. URL https://link.springer.com/chapter/10.1007/978-3-642-29737-3_42.

Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, pages 81–90. ACM, 2004. URL https://dl.acm.org/doi/10.1145/1007352.1007372.

Ali Tizghadam and Alberto Leon-Garcia. On robust traffic engineering in transport networks. In *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*, pages 1–6, 2008. doi: 10.1109/GLOCOM.2008.ECP.456.

Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication (extended abstract). In *FOCS*, pages 332–337. IEEE Computer Society, 1989. URL https://ieeexplore.ieee.org/document/63499/.

Santosh S Vempala, Ruosong Wang, and David P Woodruff. The communication complexity of optimization. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1733–1752. SIAM, 2020.

E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, page 23–33, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912705. doi: 10.1145/73393.73397.

Emo Welzl. On spanning trees with low crossing numbers. *Lecture Notes in Computer Science*, 1992. URL https://www.numdam.org/article/ITA_1991__25_2_103_0.pdf.

Ryan Williams. Matrix-vector multiplication in sub-quadratic time: (some preprocessing required). In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 995–1001, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi: 10.1109/FOCS.2010.67. URL https://doi.org/10.1109/FOCS.2010.67.

Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, pages 3792–3835. SIAM, 2024. URL https://epubs.siam.org/doi/10.1137/1.9781611977912.134.

Hakan Yildiz and Subhash Suri. On klee's measure problem for grounded boxes. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, SoCG '12, page 111–120, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312998. doi: 10.1145/2261250.2261267. URL https://doi.org/10.1145/2261250.2261267.

Contents

A Preliminaries for Supplementary Material

Notation. Let $[n] = \{1, \ldots, n\}$. We use $O(\cdot)$ to hide constant factors, and $\widetilde{O}(\cdot)$ to hide polylogarithmic factors. Let $\|\cdot\|_1$ denote the ℓ_1 (Manhattan distance). For any matrix $\mathbf{M}$, let $\mathbf{M}_i$ and $\mathbf{M}_{:,i}$ denote its i'th column and i'th row (respectively). For a set V we write $\mathbf{M} \in \mathbb{R}^{m \times V}$ for the $m \times |V|$ matrix where we can index columns by $x \in V$, i.e., $\mathbf{M}_x \in \mathbb{R}^m$. For any vector $x \in \mathbb{R}^n$, the Hamming weight of x is the number of non-zero entries of x also denoted by $\mathrm{nnz}(x)$. For Boolean vectors $x, y \in \{0,1\}^n$, the Hamming distance between x and y is given by $\|x - y\|_1$. Finally, for any vector $x \in \mathbb{R}^n$ and positive definite matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$, define the $\mathbf{L}$-induced norm of x by $\|x\|_{\mathbf{L}} = \sqrt{x^\top \mathbf{L} x}$.

VC-Dimension of Range Spaces. A range space (also called a set system) is a pair $\mathfrak{R} = (X, R)$ where X is a set and R is a set of subsets of X. Let $\Pi_R(A) = \{A \cap r : r \in R\}$. The VC-dimension of $\mathfrak{R}$ is given by $\mathrm{VC}(\mathfrak{R}) = \max\{|S| : S \subseteq X \text{ and } |\Pi_R(S)| = 2^{|S|}\}$. The primal shatter function π of $\mathfrak{R}$ is given by $\pi(m) = \max\{|\Pi_R(A)| : A \subseteq X, |A| = m\}$ for $m \geq 0$. Let B be a set of ranges in R. Then, let $\Pi^*_X(B) = \{C : C \subseteq X \text{and } C \text{ is not crossed by any range in } Q\}$ and let $\Pi^*_X(B)$ be maximal. Then, for $m \geq 0$, the dual shatter function π^* of the range space is given by $\pi^*(m) = \max\{|\Pi^*_X(B)| : B \subseteq R, |B| = m\}$.

VC-Dimension of Matrices. Any matrix $\mathbf{M} \in \{0,1\}^{m \times n}$ corresponds to a set family given by $\mathcal{F}_{\mathbf{M}} = \{\{j : \mathbf{M}_{i,j} = 1\} : i \in [m]\}$, i.e., consider each row of $\mathbf{M}$ as the indicator vector of some set. Through it, we can encode $\mathbf{M}$ by the range space $\mathfrak{R}_M = ([n], \mathcal{F}_{\mathbf{M}})$. Then, the VC-dimension of $\mathbf{M}$ is defined as the VC-dimension of $\mathfrak{R}_{\mathbf{M}}$. Here, a subset S of the columns of $\mathbf{M}$ is said to be *shattered* if each of the $2^{|S|}$ many 0/1 strings appears in some row in the restriction of $\mathbf{M}$ to the columns in S. Then, the VC-dimension of $\mathbf{M}$ is the maximum size of a shattered subset of the columns of $\mathbf{M}$. Moreover, the VC-dimension $d_{m,n}$ of a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$ satisfies $1 \leq d_{m,n} \leq \log m$.

Pollard Pseudodimension of Matrices. For a family of functions $\mathcal{F}$ mapping X to Y, the Pollard pseudodimension of $\mathcal{F}$, denoted by $\mathrm{Pdim}(\mathcal{F})$, is the VC dimension of the set system $(X \times Y, \{\{(x,y) \mid f(x) \geq y\} \mid f \in \mathcal{F}\})$.

For a real matrix $\mathbf{M}$, define the pseudo-dimension of $\mathbf{M}$, denoted by $\mathrm{Pdim}(\mathbf{M})$ by thinking of the rows of $\mathbf{M}$ as functions and taking the pseudo-dimension of the resulting class of functions. Specifically, if $\mathbf{M}$ is an $m \times n$ matrix, for each $i \in \{1, \ldots, m\}$, define $f_{\mathbf{M},i} : \{1, \ldots, n\} \to \mathbb{R}$ by $f_{\mathbf{M},i}(j) = \mathbf{M}_{i,j}$, and let $\mathrm{Pdim}(\mathbf{M}) = \mathrm{Pdim}(\{f_{\mathbf{M},i} : i \in \{1, \ldots, m\}\})$.

B Online Matrix Vector Data Structure

In this section we prove all the intermediate lemmas required for our first two main results on subquadratic online matrix-vector multiplication. The proof that combined the technical lemmas was already given in Section 3.

Theorem 1.3. *(Static Online Matrix-Vector Multiplication). If a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$ has corrupted VC-dimension d, then after an $\widetilde{O}(mn)$-time preprocessing, there is a data structure $\mathcal{D}$ that can compute $\mathbf{M}v$ for any $v \in \mathbb{R}^n$ in $\widetilde{O}(nm^{1-1/d} + m)$ time, with high probability.*

Theorem 1.4. *(Dynamic Online Matrix-Vector Multiplication). Given a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$, there is a data structure $\mathcal{D}$ with $\widetilde{O}(mn)$ preprocessing time that supports row and column updates (insertions/deletions) to $\mathbf{M}$ in $\widetilde{O}(n)$ and $\widetilde{O}(m)$ time, respectively. Upon querying $\mathcal{D}$ with a vector $v \in \mathbb{R}^n$, it outputs $\mathbf{M}v$ in $\widetilde{O}(nm^{1-1/d^*} + m)$ time, with high probability, where d^* is the largest corrupted VC-dimension of $\mathbf{M}$ throughout the history of its updates.*

B.1 Static Online Matrix Vector Multiplication Data Structure

Definition 3.1 (Δ-labeled spanning tree of matrix $\mathbf{M}$)**.** *A Δ-labeled tree is a tree $T = (V, E)$ with some explicit root x, where each edge $e \in E$ is directed away from x and labeled by a vector $\Delta_e \in \mathbb{R}^m$. We say T is a Δ-labeled spanning tree for matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ if $\mathbf{M}_x = \vec{0}$ for root x, and for all edges $(i,j) \in E$ we have $\Delta_{i,j} \coloneqq \mathbf{M}_j - \mathbf{M}_i$.*

Definition 3.2 (Weight of a Δ-labeled spanning tree)**.** *For any Δ-labeled spanning tree of $\mathbf{M}$ denoted by T with directed edge-set E, define* $\mathrm{weight}(T) = \sum_{e \in T} \mathrm{nnz}(\Delta_e)$.

Algorithm 1 Online Matrix-vector multiplication (given a Δ-labeled tree of $\mathbf{M}^\top$)

Require: Input matrix $\mathbf{M} \in \{0,1\}^{m \times n}$, and a Δ-labeled tree of $\mathbf{M}^\top$ denoted by T and rooted at r.
Require: Input vector $v \in \mathbb{R}^n$.
Initialize: $\Phi = 0^m$
$\Phi_r = \mathbf{M}_r^\top v$
Run DFS on T from root r. Whenever DFS visits a vertex $x \in V$ via some edge (y, x), then set $\Phi_x = \Phi_y + \Delta_{x,y} v$
Return Φ

Lemma 3.3 (Björklund and Lingas [2001], Alves et al. [2024])**.** *Given a Δ-labeled spanning tree T of $\mathbf{M}^\top \in \{0,1\}^{n \times m}$ and a vector $v \in \mathbb{R}^n$, we can compute $\mathbf{M}v$ in $O(\text{weight}(T) + n + m)$ time.*

Proof. Consider Algorithm 1 which takes as input $\mathbf{M} \in \{0,1\}^{m \times n}$, a Δ-labeled spanning tree of $\mathbf{M}^\top$ (denoted by T), and a vector $v \in \mathbb{R}^n$.

We inductively prove that $\Phi = \mathbf{M}v$ over the number of visited vertices. This is true when only one vertex is visited, as that is the root r and by definition $\Phi_r = (\mathbf{M}v)_r$. For the inductive step, assume DFS visits the $(k+1)$st vertex x. Let y be the parent of the vertex, then the algorithm assigns

$$\begin{aligned}\Phi_x &= \Phi_y + \Delta_{y,x}^\top v \\ &= (\mathbf{M}v)_y + (\mathbf{M}_x - \mathbf{M}_y)v \\ &= (\mathbf{M}v)_u.\end{aligned}$$

Here Φ_y is already computed since it was among the first k visited vertices.

Runtime analysis. Let $\delta_{i,j}$ be the number of non-zero entries in $\Delta_{i,j}$ for all $(i,j) \in E(T)$. Initializing the vector Φ and populating its first element Φ_r takes $O(n+m)$ time. The cost of populating the remaining entries is proportional to the number of non-zero entries in each $\Delta_{x,y}$, since we can store each Δ via a sparser representation of only its non-zero entries and use this to compute $\Delta_{y,x}^\top v$ in time $O(\delta_{y,x})$. Thus, the total time complexity is $O(\sum_{(x,y)\in E(T)} \delta_{x,y}) = O(\text{weight}(T))$. Hence, the runtime of Algorithm 1 is $O(\text{weight}(T) + n + m)$, proving the lemma. □

When the spanning tree is constructed over the columns of $\mathbf{M}$, we run a different algorithm. The following matrix $\mathbf{N}$ is defined based on the spanning tree. We will prove (i) that multiplying vectors with $\mathbf{N}$ can be done efficiently (Lemma B.2), and (ii) that $\mathbf{N} = \mathbf{M}$ (Lemma B.3). Combining the two then yields Lemma 3.4.

Definition B.1 (Path-Δ matrix $\mathbf{N}^{(T)}$)**.** *Given a Δ-labeled subtree $T = (V, E)$ with root r, define the path-Δ matrix $\mathbf{N}^{(T)} \in \mathbb{R}^{m \times V}$ such that for $u \in V$,*

$$\mathbf{N}_u^{(T)} \coloneqq \sum_{e \in P(r \to u)} \Delta_e.$$

Here, $P(r \to u)$ denotes the edges of the unique (since T is acyclic) directed path from r to u.

Lemma B.2. *For Δ-labeled tree $T = (V, E)$, and a vector $v \in \mathbb{R}^V$, let $\vec{\sigma} \in \mathbb{R}^E$ with $\vec{\sigma}_{(y,c)} = \sum_{z \in V(T_c)} v_z$, where T_c is the subtree rooted at $c \in V$. Let $\vec{\Delta} \in \mathbb{R}^{m \times E}$ be the matrix with columns being the labels of tree T. Then $\vec{\Delta}\vec{\sigma} = \mathbf{N}^{(T)}v$.*

Proof. By the definition of $\mathbf{N}^{(T)}$, we have

$$\begin{aligned}\mathbf{N}^{(T)}v &= \sum_{y \in V} \mathbf{N}_y^{(T)} \cdot v_y \\ &= \sum_{y \in V} \sum_{e \in P(r \to y)} \Delta_e \cdot v_y\end{aligned}$$

Now consider how often any one edge e shows up in this sum: an edge $e = (x, z)$ is in $P(r \to y)$ if and only if z is an ancestor of y (or $y = z$) (or, in other words, if y is a descendant of z (or $y = z$)).

This condition can be written as $y \in T_z$. Thus we write the sum as

$$\begin{aligned}\sum_{y \in V} \sum_{e \in P(r \to y)} \Delta_e \cdot v_y &= \sum_{(x,z) \in E} \sum_{y \in T_z} \Delta_{x,z} \cdot v_y \\ &= \sum_{(x,z) \in E} \Delta_{x,z} \sum_{y \in T_z} v_y \\ &= \sum_{(x,y) \in E} \Delta_{x,y} \vec{\sigma}_y \\ &= \vec{\Delta} \vec{\sigma},\end{aligned}$$

which proves the claim. □

Lemma B.3. *Let T be a Δ-labeled spanning tree of $\mathbf{M}$ where the root r of T has $\mathbf{M}_r = \vec{0}$, then $\mathbf{N}^{(T)} = \mathbf{M}$.*

Proof. We prove this by induction over the width of $\mathbf{M}$.

Base Case. For width 1, T is just the root r, so $\mathbf{N}^{(T)} = \vec{0} = \mathbf{M}$.

Inductive Step. Suppose the statement holds for matrices of width $< n$ and we now have a matrix $\mathbf{M}$ of width n. Let $L \subset V$ be the leaves of T. Let T' be the tree T with these leafs removed, and let $\mathbf{M}' \in \{0,1\}^{m \times (V \setminus L)}$ be matrix $\mathbf{M}$ with all columns corresponding to L removed. Then for any $y \in V \setminus L$

$$\mathbf{M}_y = \mathbf{M}'_y \overset{(1)}{=} \mathbf{N}^{(T')}_y \overset{(2)}{=} \mathbf{N}^{(T)}_y$$

where equality (1) uses the induction hypothesis and equality (2) uses the fact that any path in the leaf-removed tree T' also exists in T.

For any $y \in L$ let z be its ancestor in T. Therefore, $z \notin L$. Then, we have

$$\begin{aligned}\mathbf{N}^{(T)}_y &= \mathbf{N}^{(T)}_z + \Delta_{z,y} && (1)\\ &= \mathbf{N}^{(T')}_z + \Delta_{z,y} && (2)\\ &= \mathbf{M}'_z + \Delta_{z,y} && (3)\\ &= \mathbf{M}_z + (\mathbf{M}_y - \mathbf{M}_z) = \mathbf{M}_y,\end{aligned}$$

where equation 1 uses the fact that the path from root r to y is a path from root r to z with one extra edge (z, y). Equation 2 uses the fact the path from r to z in T also exists in T', and finally equation 3 follows by the induction hypothesis. Together, they prove the lemma. □

Lemma 3.4. *Given a Δ-labeled spanning tree T of $\mathbf{M} \in \{0,1\}^{m \times n}$ and a vector $v \in \mathbb{R}^n$, there is an algorithm that can compute $\mathbf{M}v$ in $O(\text{weight}(T) + n + m)$ time.*

Proof. **Algorithm:**

We compute $\mathbf{N}^{(T)}v$ via Lemma B.2 which requires us to compute the vector $\vec{\sigma} \in \mathbb{R}^E$ with $\vec{\sigma}_{(y,c)} = \sum_{z \in V(T_c)} v_z$ for all $(y, c) \in E$ where T_c is the subtree of T rooted at c. This can be done in $O(n)$ time via a simple "dynamic programming on trees" algorithm: For any $c \in V$, let C be its children.

Then,

$$\begin{aligned}\vec{\sigma}_{(y,c)} &= \sum_{z \in V(T_c)} v_z \\ &= v_c + \sum_{x \in C} \sum_{z \in V(T_x)} v_z \\ &= v_c + \sum_{x \in C} \vec{\sigma}_{(c,x)}.\end{aligned}$$

So we can compute $\vec{\sigma}$ in $O(n)$ by starting at the leaf-edges, then propagating the sums up towards the root. At last, we multiply $\vec{\Delta}\vec{\sigma}$ in $O(\text{weight}(T))$ time as that is the number of non-zeros in $\vec{\Delta}$ (which is the matrix consisting of the tree labels as column-vectors).

Correctness. Without loss of generality, assume that for root r of tree T we have $\mathbf{M}_r = \vec{0}$. Otherwise, append a $\vec{0}$-column to $\mathbf{M}$, and add edge $(n+1, r)$ to T with label $\Delta_{n+1,r} = \mathbf{M}_r - \mathbf{M}_{n+1} = \mathbf{M}_r$, and make $n+1$ the new root of T. This increases the tree weight by at most m, resulting in an additive $O(m)$ in the time complexity.

Append another entry to v so that the dimensions match and $\mathbf{M}v$ is well-defined. By Lemma B.3, we thus have $\mathbf{N}^{(T)}v = \mathbf{M}v$. □

B.2 Existence of Low Weight Δ-labeled Minimum Spanning Trees

We first provide a proof of Lemma 3.11 (Lemma 4.1 of Welzl [1988]), restated below for completeness.

Lemma 3.11 (Lemma 4.1 of Welzl [1988])**.** *Let (X, R) be a range space with dual VC-dimension at most d. For every $A \subseteq X, |A| = n \geq 1$ and every multiset Q of ranges in R with $|Q| = m$, there exists $x \neq y \in A$ such that the number of ranges in Q crossing (x, y) is at most $O(mn^{-1/d}\log^2 n)$.*

For this, we introduce the notion of ϵ-nets and a crucial proposition of Welzl [1988] and Haussler and Welzl [1986].

Definition B.4 (ϵ-net)**.** *Let (X, R) be a range space and $0 \leq \epsilon \leq 1$. Let A be a finite multiset of elements in X. Then, $N \subseteq A$ is an ϵ-net of A for R if $\frac{|A \cap r|}{|A|} > \epsilon$ implies $r \cap N \neq \emptyset$ for all $r \in R$.*

Proposition B.5 ([Welzl, 1988, Proposition 2.4])**.** *Let (X, R) be a range space of VC-dimension $d \geq 1$ and let $0 < \epsilon \leq 1$ be a real number. For every finite multiset A of elements in X, there is an ϵ-net N of A for R such that $|N| \leq \lceil \frac{8d}{\epsilon} \log \frac{8d}{\epsilon} \rceil$.*

Lemma B.6 (Lemma 2.2 of Welzl [1988])**.** *If (X, R) is a range space of VC-dimension d, then $(X, \widetilde{R})$, where $\widetilde{R} = \{(r' \cup r'') - (r' \cap r'') | r', r'' \in R\}$ has VC-dimension at most $O(d \log d)$.*

We are now ready to prove Lemma 3.11, which we adapt from Welzl [1988].

Proof of Lemma 3.11. By the pigeonhole principle, it is sufficient to show the existence of a subset N of Q such that (i) $\pi^*(|N|) < n$ and (ii) if a pair of elements $\{x, y\}$ is crossed by more than $dmn^{-1/d}\log n$ ranges in Q, then $\{x, y\}$ is crossed by a range in N.

For $x \in X$, let $R_x = \{r \in R : x \in r\}$ and for $x, y \in X$ with $x \neq y$, let R_{xy} be the set of ranges in R that cross $\{x, y\}$. Since $(R, \{R_x | d \in X\})$ is the dual of (X, R) it has VC-dimension at most d by assumption. So by Lemma B.6, $(R, \{R_{xy} | x, y \in X, x \neq y\})$ has VC-dimension $O(d \log d)$.

Therefore, by Proposition B.5, for every $\epsilon > 0$ and every multiset Q of ranges in R, there is an ϵ-net of Q for $\{R_{xy} | x, y \in X, x \neq y\}$ with $|N| \leq \lceil \frac{O(d \log d)}{\epsilon} \log \frac{O(d \log d)}{\epsilon} \rceil$. Now, let $\epsilon = cdn^{-1/d}\log n$, where c is some constant. Then, there is an ϵ-net of Q of size at most

$$O(d \log d) \cdot \frac{n^{1/d}}{cd \log n} \log \left(O(d \log d) \frac{n^{1/d}}{cd \log n} \right) \leq c_1 n^{1/d}$$

where $c_1 \leq O(1)$. For large enough c, we can get small enough c_1 such that $\pi^*(\lfloor c_1 n^{1/d} \rfloor) < n$. For some constant c, we thereby get a $(cdn^{-1/d}\log n)$-net of Q for the ranges R_{xy} with $\pi^*(|N|) < n$. This implies that there is a pair of elements $\{x, y\}$ crossed by more than $cd|Q|n^{-1/d}\log n = cdmn^{-1/d}\log n$ ranges in Q. Finally, applying $d \leq \log n$ concludes the argument. □

Lemma 3.14. *For a range space (X, R), $|X| = n$ with spanning tree T, there exists a permutation $\pi \in S_n$ such that the weight of the corresponding spanning tree T' (which is just the line $\pi(1), \pi(2), ..., \pi(n)$) is at most twice that of T.*

Proof. Enumerate all the ranges in R by $R_1, \ldots, R_m$, where $R_i := \{c : \mathbf{M}_{i,c} = 1\}$ for $i \in [m]$.

The crossing number of R_i in T counts the number of edges in T such that one endpoint of the edge is in R_i and the other endpoint is in $[n] \setminus R_i$. Suppose the spanning tree T on X has a crossing number of κ_i for just R_i and let $w(T) = \sum_{i=1}^{m} \kappa_i$ be the weight of T for (X, R).

Perform a DFS traversal on T and write the in-order traversal $\mathcal{I}$. We use the permutation $\pi(i) = \mathcal{I}_i$ for $i \in [n]$. Let T' be the corresponding linear tree. In the worst case, this vertex path crosses R_i $2\kappa_i$ times, that is because DFS uses each edge (x, y) of T twice: once when visiting y (going down one level in the tree) and once when it is done processing y (DFS pops the current level from the stack).

Thus the total number of crossings $w(T')$ for the linear tree T' is $w(T') \leq \sum_{i=1}^{m} 2\kappa_i \leq 2\kappa(T)$. □

B.3 Dynamic OMv Data Structure

In this section we give a black-box reduction from dynamic OMv to static OMv. In the static version, we are given a matrix $\mathbf{M}$ to preprocess. This matrix stays fixed, while we receive an online sequence of vectors and must repeatedly compute $\mathbf{M}v$. In the dynamic version, we can inform the data structure about changes to $\mathbf{M}$. Specifically, we are allowed to add/remove/change rows/columns of the matrix.

Using the reduction from dynamic to static, we obtain Theorem 1.4.

Theorem 1.4. *(Dynamic Online Matrix-Vector Multiplication). Given a matrix $\mathbf{M} \in \{0,1\}^{m \times n}$, there is a data structure $\mathcal{D}$ with $\widetilde{O}(mn)$ preprocessing time that supports row and column updates (insertions/deletions) to $\mathbf{M}$ in $\widetilde{O}(n)$ and $\widetilde{O}(m)$ time, respectively. Upon querying $\mathcal{D}$ with a vector $v \in \mathbb{R}^n$, it outputs $\mathbf{M}v$ in $\widetilde{O}(nm^{1-1/d^*} + m)$ time, with high probability, where d^* is the largest corrupted VC-dimension of $\mathbf{M}$ throughout the history of its updates.*

The idea of this reduction is to split the columns of $\mathbf{M}$ into $\log n$ matrices $\mathbf{M}_0, ..., \mathbf{M}_{\log n}$ where $\mathbf{M}_i$ contains at most 2^i columns. Each $\mathbf{M}_i$ is used as input to a static OMv data structure which is reinitialized whenever $\mathbf{M}_i$ changes which occurs at most once every 2^i updates to $\mathbf{M}$. This allows to amortize the reinitialization cost over multiple updates.

Lemma B.7. *Assume there is a static OMv data structure for matrices of size $m \times n$ with preprocessing time $P(m, n)$ and query time $Q(m, n)$. Then there exists a dynamic OMv data structure supporting row and column insertions and deletions. The preprocessing time is $\widetilde{O}(P(m, n))$, the amortized row update time is*

$$O\left(\sum_{j=0}^{\log m} \frac{P(2^j, n)}{2^j}\right)$$

and for column updates

$$O\left(\sum_{i=0}^{\log n} \frac{P(m, 2^i)}{2^j}\right)$$

and query time is $\widetilde{O}(Q(m, n))$. This reduction runs several static OMv data structures, each receiving a submatrix of the dynamic input matrix.

We start by proving column updates only, then extend the reduction to support row updates as well.

Lemma B.8. *Assume there is a static OMv data structure for matrices of size $m \times n$ with preprocessing time $P(m, n)$ and query time $Q(m, n)$. Then there exists a dynamic OMv data structure supporting column insertions and deletions. The preprocessing time is $\widetilde{O}(P(m, n))$, the query time is $\widetilde{O}(Q(m, n))$, and the amortized update time is*

$$O\left(\sum_{i=0}^{\log m} \frac{P(m, 2^i)}{2^i}\right)$$

This reduction runs several static OMv data structures, each receiving a submatrix of the dynamic input matrix.

Proof. Let $\mathbf{M}$ be the input matrix that changes over time.

Initialization. We run $\log n$ instances of the static OMv data structure, where the ith instance is guaranteed to run on a matrix with at most 2^i columns. Let $\mathbf{M}^{(0)}, ..., \mathbf{M}^{(\log n)}$ be the respective input matrices to these instances. Initially, only the last ($i = \log n$) data structure receives the initial matrix $\mathbf{M}$ as input, and all other instances of the data structure initialize on an empty matrix. Hence initialization takes $O(P(m, n))$ time.

Insertions. Whenever there is a column insertion to $\mathbf{M}^{(i)}$ we reinitialize the ith static OMv data structure on that new matrix. When $\mathbf{M}^{(i)}$ has more than 2^i columns, then all columns are removed (i.e., we set $\mathbf{M}^{(i)} = 0$) and inserted to matrix $\mathbf{M}^{(i+1)}$. All column insertions to $\mathbf{M}$ are passed to $\mathbf{M}^{(0)}$. Observe that any $\mathbf{M}^{(i)}$ is updated every 2^i updates to $\mathbf{M}$, hence the amortized update time is

$$O\left(\sum_{i=0}^{\log n} P(m, 2^i)/2^i\right).$$

Queries. We can answer queries by observing that

$$\mathbf{M} = [\mathbf{M}^{(\log m)}|...|\mathbf{M}^{(1)}|\mathbf{M}^{(0)}].$$

So, for a query vector v we split it into corresponding pieces $v^{(0)}, ...v^{(\log m)}$ and query each $\mathbf{M}^{(i)}v^{(i)}$. The time for this is

$$O(Q(m, n) \log n)$$

because each $\mathbf{M}^{(i)}$ is a submatrix of $\mathbf{M}$ and we have $\log n$ such matrices.

Deletions. When deleting a column of $\mathbf{M}$, we do not delete the column from the respective $\mathbf{M}^{(i)}$. Instead, all future query vectors v will have an extra 0 entry for the deleted column.

When all columns from $\mathbf{M}^{(i)}$ are appended to $\mathbf{M}^{(i+1)}$ because of some insertions, then the deleted columns are not appended to $\mathbf{M}^{(i+1)}$. Hence whenever a static OMv data structure is initialized, it is a submatrix of the input matrix $\mathbf{M}$.

When there have been $2^i/2$ postponed deletions to any $\mathbf{M}^{(i)}$, then the columns are actually deleted and the respective static OMv data structure is reinitialized on the new $\mathbf{M}^{(i)}$. The remaining columns in $\mathbf{M}^{(i)}$ all exist in the current dynamic input $\mathbf{M}$, so again, the static OMv data structure receives a submatrix of $\mathbf{M}$ as input. For any one i, this takes amortized $O(P(n, 2^i)/2^i)$ time which is subsumed by the complexity of handling insertions.

Observe that internally, our matrices $\mathbf{M}^{(i)}$ are always at most twice their intended size because of the yet to be removed columns. Thus the query complexity increases by at most a constant factor. □

We now prove Lemma B.7 (Row & Column Updates)

Proof. Here, we extend Lemma B.8 to support row updates as well. The reduction follows the same idea as Lemma B.8.

Let $\mathbf{N}$ be the dynamic input matrix. We run $\log n$ copies of the column update data structure Lemma B.8 on matrices $\mathbf{N}^{(0)}, ..., \mathbf{N}^{(\log n)}$, with the i'th matrix having at most 2^i rows. When inserting new rows to $\mathbf{N}$, they are passed to $\mathbf{N}^{(0)}$. When $\mathbf{N}^{(i)}$ has more than 2^i rows, all rows are appended to $\mathbf{N}^{(i+1)}$. When rows are appended to any $\mathbf{N}^{(i)}$, the respective data structure is reinitialized. To handle deletions, we remove the respective row from the output until $2^i/2$ rows have been deleted from $\mathbf{N}^{(i)}$, then the rows are removed from $\mathbf{N}^{(i)}$ and the respective data structure is reinitialized. The amortized row update time thus becomes

$$O\left(\sum_{j=0}^{\log n} \frac{\text{Time to initialize Lemma B.8 on } 2^j \times n \text{ matrix}}{2^j}\right) = O\left(\sum_{j=0}^{\log m} \frac{P(2^j, n)}{2^j}\right).$$

For column updates, each of the $\mathbf{N}^{(i)}$ receives a new column, so each of the $\log m$ column update data structures is updated which takes:

$$O\left(\sum_{j=0}^{\log m}\sum_{i=0}^{\log n} \frac{P(2^j, 2^i)}{2^i}\right) = O\left(\sum_{i=0}^{\log n} \frac{P(m, 2^i)}{2^i}\right)$$

because geometric sums are bounded by their largest term in O-notation.

Finally, the query time is

$$\widetilde{O}\left(\sum_{j=0}^{\log m} Q(2^j, n)\right) = \widetilde{O}(Q(m, n)),$$

which proves the lemma. □

We now prove Theorem 1.4 (Dynamic OMv for corrupted VC-dimension d).

Proof. The initialization cost of static OMv is $P(m, n) = \widetilde{O}(mn)$, and the query complexity is $Q(m, n) = \widetilde{O}(nm^{1-1/d} + m)$ by Theorem 1.3. This complexity also holds for submatrices, i.e., $Q(a, b) = \widetilde{O}(nm^{2-1/d} + m)$ for any $a \times b$ sized submatrix.

Thus we have update time for column updates:

$$O\left(\sum_{i=0}^{\log n} \frac{P(m, 2^i)}{2^i}\right) = \widetilde{O}\left(\sum_{i=0}^{\log n} \frac{m2^i}{2^i}\right) = \widetilde{O}(m)$$

and for row updates we have

$$O\left(\sum_{j=0}^{\log m} \frac{P(2^j, n)}{2^j}\right) = \widetilde{O}\left(\sum_{j=0}^{\log m} \frac{2^j n}{2^j}\right) = \widetilde{O}(n)$$

and for query time we have

$$\widetilde{O}(Q(m, n)) \leq \widetilde{O}(nm^{1-\frac{1}{d}} + m),$$

which proves the theorem. □

B.4 Extension to Structured Non-Boolean Matrices

In this section, we prove a result for a subquadratic-time accelerated matrix-vector multiplication for non-Boolean matrices, when the analogous *Pollard pseudodimension* is bounded. We also list further applications of this result.

Theorem B.9. *Suppose that the Pollard pseudodimension of the matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ is d, and that the number of distinct values per column is $\leq T$. Then, after an $\widetilde{O}(Tmn)$ time preprocessing, there is a data structure that, upon receiving a vector $v \in \mathbb{R}^n$, returns $\mathbf{M}v$ in time $O(Tnm^{1-1/d} + m)$.*

Remark B.10. *When $\mathbf{M}$ is the weight matrix of a neural network and v is the activation output from the previous layer of the neural network, the matrix-vector product $\mathbf{M}v$ corresponds to running inference on v with the neural network $\mathbf{M}$. This is a key step in a number of applications, including online optimization in unknown no-linear systems [Lin et al., 2023b] and inference in large language models. Dehghankar et al. [2024] studies the latter application with ternary matrices $\mathbf{M} \in \{-1, 0, 1\}^{n \times m}$ and shows that they are amenable to $O(n^2/\log n)$ multiplication. If $\mathbf{M}$ has bounded Pollard pseudodimension d, then as $T = 3$, we obtain a polynomial acceleration on this result: by Theorem B.9, the matrix-vector product can be computed in $\widetilde{O}(mn^{1-1/d} + n)$ time.*

Recall the definition of the Pollard pseudodimension:

Definition B.11 (Pollard Pseudodimension)**.** *For a family of functions $\mathcal{F}$ mapping X to Y, the Pollard pseudodimension of $\mathcal{F}$, denoted by $\mathrm{Pdim}(\mathcal{F})$, is the VC dimension of the set system $(X \times Y, \{\{(x, y) \mid f(x) \geq y\} \mid f \in \mathcal{F}\})$.*

In the setting of matrices $\mathbf{M} \in \mathbb{R}^{m \times n}$, we interpret each row as a function $f_{\mathbf{M},i} : x \mapsto \mathbf{M}_{i,x}$. In particular, when $\mathbf{M}$ has Pollard pseudodimension d, then that means the set system $([n] \times \mathbb{R}, \{(j, y) \mid \mathbf{M}_{i,j} \geq y\} \mid i \in [m]\})$ has VC-dimension d. Restricting the groundset $[n] \times \mathbb{R}$ to $\{(j, y) \mid \exists i : \mathbf{M}_{i,j} = y\}$ does not increase the VC-dimension. This latter set system can be interpreted as the following Boolean matrix $\mathbf{N}$:

Given a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ where each column has at most T distinct values, consider the following construction of an $\mathbf{N} \in \{0, 1\}^{m \times nT}$ matrix. Let $\mathbf{Y} \in \mathbb{R}^{T \times n}$ be the distinct values per column of $\mathbf{M}$

(we may duplicate values if a column has $< T$ distinct values). Assume the entries in each column of $\mathbf{Y}$ are sorted in increasing order. Let $\mathbf{N}_{i,jT+k} = \mathbb{1}_{\mathbf{M}_{i,j} \geq \mathbf{Y}_{i,k}}$.

For example, the following $\mathbf{M}$ has at most $T = 4$ distinct values per column:

$$\mathbf{M} := \left[\begin{array}{c|c} 1 & -1 \\ 4 & -2 \\ 2 & 4 \\ 5 & 3 \\ 4 & 3 \end{array}\right] \to \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right] =: \mathbf{N}, \quad \mathbf{Y} := \begin{bmatrix} 1 & -2 \\ 2 & -1 \\ 4 & 3 \\ 5 & 4 \end{bmatrix}$$

Since matrix $\mathbf{N}$ is Boolean and of VC-dimension at most d, we can run our fast matrix-vector product data structures on it (e.g., Theorem 1.3).

Proof of Theorem B.9. Given $\mathbf{M} \in \mathbb{R}^{m \times n}$ we construct the Boolean matrix $\mathbf{N} \in \{0,1\}^{m \times nT}$ as previously described.

We initialize Theorem 1.3 on $\mathbf{N}$ which takes $\widetilde{O}(Tmn)$ time. Multiplying a vector $v' \in \mathbb{R}^{nT}$ with $\mathbf{N}$ takes $\widetilde{O}(Tnm^{1-1/d} + m)$ time. We are left with explaining how to compute $\mathbf{M}v$ via $\mathbf{N}v'$.

We have

$$\begin{aligned} \mathbf{M}_{i,j} &= \mathbf{Y}_{1,j} + \sum_{k>1 : \mathbf{Y}_{k,j} \leq \mathbf{M}_{i,j}} \mathbf{Y}_{k,j} - \mathbf{Y}_{k-1,j} \\ &= \mathbf{N}_{i,jT}\mathbf{Y}_{1,j} + \sum_{k=2}^{T} \mathbf{N}_{i,j\cdot T+k}(\mathbf{Y}_{k,j} - \mathbf{Y}_{k-1,j}) \end{aligned}$$

In particular, we can compute $\mathbf{M}v = \mathbf{N}v'$where

$$v'_{j\cdot T} = v_j \cdot \mathbf{Y}_{1,j},\ v'_{j\cdot T+k} = v_j \cdot (\mathbf{Y}_{k,j} - \mathbf{Y}_{k-1,j}) \text{ for } k \geq 2.$$

Constructing v' takes $O(Tn)$ time per query. □

Matrices with small constant Pollard pseudodimensions are also prevalent in real-world data, and capture a similar notion of structure as matrices with low constant VC-dimensions. For instance, the following are examples of matrices with small Pollard pseudodimension.

Definition B.12 (Multiball vectors)**.** *Let $v \in V$ be a vertex of the graph G. Let r be a real number and Δ be a set of distances $-\infty = \delta_0 < \delta_1 < \cdots < \delta_{\ell-1} < \delta_\ell = \infty$. Define the multiball vectors as follows:*

$$\vec{\mathrm{MB}}(v, r, \Delta) = (y_u)_{u \in V} \text{ where } y_u \in [\ell] \text{ is the smallest integer such that } u \in \vec{B}(v, r + \delta_{yu}).$$

Let $\vec{\mathrm{MB}}_{G,\Delta} = \{\vec{MB}(v, r, \Delta) | v \in V, r \in \mathbb{R}\}$ be the set of all multiball vectors in G with shifts Δ.

Theorem B.13 (Karczmarz and Zheng [2024])**.** *For a K_h-minor-free graph G, for any set Δ of ℓ distances, the set of multiball vectors $\vec{\mathrm{MB}}_{G,\Delta}$ has pseudodimension at most $h - 1$.*

C Applications

In this section, we consider some applications of our OMv result in Theorem 1.4. These applications have quadratic $\Omega(n^2)$-time lower bounds, conditional on the OMv conjecture. We show that on structured graph this lower bound can be beaten.

C.1 High-accuracy Dynamic Laplacian Solver

Theorem 1.6. *(Dynamic Laplacian Solver). There is a dynamic algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension bounded by d, maintains a Laplacian system solver. The data structure supports queries that receive a vector $b \in \mathbb{R}^{|V|}$ and error parameter $\epsilon > 0$. Then, in $\widetilde{O}(n^{2-1/d} \log 1/\epsilon)$ time, the algorithm returns the (approximate) solution x to $\mathbf{L}x^* = b$ where $\|x - x^*\|_{\mathbf{L}} \leq \epsilon \|x^*\|_{\mathbf{L}}$. Each vertex update to G takes $\widetilde{O}(n)$ time.*

We consider the setting where the Laplacian $\mathbf{L}$ corresponds to a graph with bounded VC-dimension. First, recall the notion of spectral sparsifiers.

Definition C.1 ($(1+\epsilon)$-spectral sparsifiers)**.** *Let $\mathbf{L}_X$ denote the Laplacian of any undirected graph X. Then, a $(1\pm\epsilon)$-spectral sparsifier H of a graph G is a subgraph of G such that for every vector $x \in \mathbb{R}^n$,*

$$(1-\epsilon)x^\top \mathbf{L}_H x \leq x^\top \mathbf{L}_G x \leq (1+\epsilon)x^\top \mathbf{L}_H x,$$

Then, a result from Abraham et al. [2016] provides a construction for a dynamic spectral sparsifier under edge deletions and insertions with polylogarithmic amortized update time:

Theorem C.2 (Theorem 4.1 of Abraham et al. [2016])**.** *There exists a fully dynamic randomized algorithm with polylogarithmic update time for maintaining a $(1\pm\epsilon)$-spectral sparsifier H of a graph G, with probability at least $1-1/n^c$ for any $0<\epsilon\leq 1$ and $c\geq 1$. Specifically, the amortized update time of the algorithm is $O(c\epsilon^{-2}\log^3\rho\log^6 n)$ and the size of H is $O(cn\epsilon^{-2}\log^3\rho\log^5 n\log W + m\rho^{-1})$, where $1\leq\rho\leq m$ is a parameter of choice. Here, W is the ratio between the largest and the smallest edge weight in G.*

We use the following result for solving Laplacian systems in the static setting.

Lemma C.3 (Kyng and Sachdeva [2016])**.** *There is a randomized procedure that given any n-vertex m-edge graph G with Laplacian matrix $\mathbf{L}_G$, and vector $b\in\mathbb{R}^V$ such that there exists an $x\in\mathbb{R}^V$ with $\mathbf{L}_G x = b$ computes $\overline{x}\in\mathbb{R}^V$ with $\|\overline{x}-x\|_{\mathbf{L}_G}\leq\epsilon\|x\|_{\mathbf{L}_G}$ in $\widetilde{O}(m\log\epsilon^{-1})$ with high probability.*

Lemma C.4. *Let G be a dynamic graph and let $\mathbf{A}$ be its dynamic adjacency matrix. Assume there is a dynamic OMv data structure for this $\mathbf{A}$ with update time $U(n)$ and query time $Q(n)$.*

Then there exists a dynamic Laplacian System solver for the same dynamic graph G, supporting updates to G in $O(U(n)+n)$ time. The data structure supports queries which for any given $b\in\mathbb{R}^V$ and $\epsilon>0$ in $\widetilde{O}(Q(n)\log 1/\epsilon)$ time return $x\in\mathbb{R}^V$ with $\|x-x^*\|_{\mathbf{L}_G}\leq\epsilon\|x^*\|_{\mathbf{L}_G}$ where x^* is the exact solution for $\mathbf{L}_G x = b$.

Via Theorem 1.4 we have $U(n)=\widetilde{O}(n)$ and $Q(n)=\widetilde{O}(n^{2-1/d})$, thus obtaining Theorem 1.6.

Proof. We run two separate data structures. The first is a dynamic spectral sparsifier Theorem C.2 which maintains a spectral sparsifier $H\approx\mathbf{L}_G$ with error $\epsilon=1/2$. We also run the dynamic OMv data structure for matrix $\mathbf{A}$ (adjacency matrix).

The update time is thus $\widetilde{O}(n+U(n))$ as the spectral sparsifier needs polylog time per edge updates, thus $\widetilde{O}(n)$ time for node updates.

Queries When given a vector $b\in\mathbb{R}^V$ and error parameter $\epsilon>0$, we use $\mathbf{H}^{-1}$ as preconditioner for the linear system $\mathbf{L}_G x = b$. We can multiply with $\mathbf{H}^{-1}$ by running a static Laplacian system solver Lemma C.3 which takes $\widetilde{O}(n)$ time per product as $\mathbf{H}$ has $\widetilde{O}(n)$ edges. In particular, we let

$$x^0 = \mathbf{H}^{-1}b$$

and then perform iterative refinement (Richardson iteration)

$$x^{t+1}\leftarrow x^t - \mathbf{H}^{-1}(\mathbf{L}_G x^t - b).$$

After $\widetilde{O}(\log 1/\epsilon)$ iterations we have $\|x^t-x^*\|_{\mathbf{L}_G}\leq\epsilon\|x^*\|_{\mathbf{L}_G}$.

Each iteration takes $\widetilde{O}(n+Q(n))$ time, where $\widetilde{O}(n)$ is the time for multiplying by $\mathbf{H}^{-1}$, and $Q(n)$ the time for multiplying by $\mathbf{L}_G$. Observe that $\mathbf{L}_G v = \mathbf{D}v - \mathbf{A}v$ where $\mathbf{D}$ is a diagonal matrix and $\mathbf{A}$ is an adjacency matrix. So the first product takes $O(n)$ time and the second product is handled by the dynamic OMv data structure in $O(Q(n))$ time. Together, this proves the lemma.

□

C.2 Effective Resistance

The following Theorem 1.7 is a corollary of the dynamic Laplacian solver from Theorem 1.6.

Theorem 1.7. *(Dynamic Effective Resistance). There is a dynamic algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension bounded by d, maintains effective resistances in G. The data structure supports queries that receive a pair of vertices $u, v \in V$ and error parameter $\epsilon > 0$. Then, in $\widetilde{O}(n^{2-1/d} \log 1/\epsilon)$ time, the algorithm returns a $(1 \pm \epsilon)$-approximation of the effective resistance. Moreover, each node update to G in the dynamic data structure takes $\widetilde{O}(n)$ time.*

Proof. The effective resistance of a pair $u, v \in V$ is the energy of an electric flow, routing one unit of flow between u and v. The energy of an electric flow $f \in \mathbb{R}^E$ is $\sum_e f_e^2$ and the electric flow is given by $f = \mathbf{B}\mathbf{L}^\dagger(e_u - e_v)$ where $\mathbf{B} \in \{-1, 0, 1\}^{E \times V}$ is the incidence matrix of G with arbitrary directions. Thus the effective resistance is

$$\begin{aligned}\|\mathbf{B}\mathbf{L}^\dagger(e_u - e_v)\|_2^2 &= (e_u - e_v)^\top \mathbf{L}^\dagger \mathbf{B}^\top \mathbf{B}\mathbf{L}^\dagger(e_u - e_v) \\ &= (e_u - e_v)^\top \mathbf{L}^\dagger(e_u - e_v).\end{aligned}$$

This can be computed via the dynamic Laplacian system solver in Theorem 1.6. □

C.3 Triangle Detection

Theorem 1.8. *(Dynamic Triangle Detection). There is an algorithm that, given a dynamic graph $G = (V, E)$ with corrupted VC-dimension d, maintains whether G has a triangle or not. Each vertex update takes $\widetilde{O}(n^{2-1/d})$ time and returns a Boolean indicator for G containing a triangle.*

The result follows form the following lemma for $U(n) = \widetilde{O}(n)$ and $Q(n) = \widetilde{O}(n^{2-1/d})$ from Theorem 1.4.

Lemma C.5. *Let G be a dynamic graph and let $\mathbf{A}$ be its dynamic adjacency matrix. Assume there is a dynamic OMv data structure for this $\mathbf{A}$ with update time $U(n)$ and query time $Q(n)$.*

Then there exists a dynamic triangle detection algorithm for the same dynamic graph G, supporting updates in $O(U(n) + Q(n))$ time.

Proof. Let $\mathbf{A}$ be the adjacency matrix of the graph, then the diagonal entries $(\mathbf{A}^3)_{v,v}$ are non-zero if and only if v participates in a triangle. Thus by maintaining the sum $\sum_i (\mathbf{A}^3)_i$ we can detect if there is any triangle in the graph.

We maintain the sum as follows: Let $\mathbf{A}$ be the incidence matrix before, and $\mathbf{A}$ after a node v is updated.

Then

$$\sum_i (\mathbf{A}'^3)_i = (\sum_i (\mathbf{A}^3)_i) + 3(\underbrace{(\mathbf{A}'^3)_{v,v} - (\mathbf{A}^3)_{v,v}}_{\text{difference in \# of triangles } v \text{ participates in}}).$$

This is because each triangle v participates in is counted thrice in the sum (once for each vertex of the triangle). Thus we can maintain the sum by computing $(\mathbf{A}'^3)_{v,v}$ and $(\mathbf{A}^3)_{v,v}$. Observe that this is the vector-matrix-vector product of (i) the vth row of $\mathbf{A}$, (ii) matrix $\mathbf{A}$, (iii) and v-th column vector of $\mathbf{A}$. Thus it can be computed in $O(Q(n))$ time, with an extra $U(n)$ time to update $\mathbf{A}$ to $\mathbf{A}'$.

□

C.4 Single-Source Shortest Paths and k-Center

Lemma C.6 (Brand et al. [2022])**.** *Assume there is a dynamic distance oracle for unweighted undirected graphs with the following operations.*

An initialization procedure that is given $G = (V, E)$ and a threshold $1 \leq d \leq n$.

A node update operation with update time $U(n, d)$.

A query operation which for any source node, given during the query, returns the d-bounded single source distances in $O(Q(n, d))$. That is, return for each $v \in V$ the distance if it is at most d, or ∞ if the distance is $> d$.

Then for any $1 \leq k \leq n$ and $\epsilon > 0$ there exists a dynamic $(1+\epsilon)$-approximate single source distance data structure on unweighted undirected graphs. It supports node updates in

$$O\left(U\left(n, \frac{4}{\epsilon}\right) + \frac{k}{\epsilon} \cdot Q\left(n, \frac{4}{\epsilon}\right) + \frac{n^2}{k}\right)$$

time.

Theorem 1.9. *(Dynamic Approximate Single-Source Shortest Paths). There is a dynamic algorithm that maintains $(1+\epsilon)$-approximate single-source distances on a dynamic unweighted graph $G = (V, E)$. If the corrupted VC-dimension of G is bounded by d, each node update to G takes $\widetilde{O}(kn^{2-1/2d}/\epsilon)$ time, and querying the distances for any source node takes $\widetilde{O}(n^{2-1/2d}/\epsilon)$ time.*

Proof. We first describe how to obtain a dynamic distance oracle that yields distances up to $1/\epsilon$. This is done by running the dynamic OMv data structure of Theorem 1.4 on the adjacency matrix of G. Then single source distances up to $1/\epsilon$ from source vertex u can be computed by repeatedly multiplying $w \leftarrow \mathbf{A}w$ for initial $w = e_u$, i.e., we compute $\mathbf{A}^i e_u$ for $i = 1, 2, ..., 1/\epsilon$. The smallest i with $(\mathbf{A}^i e_u)_v \neq 0$ is the smallest number of steps to reach from u to v, i.e., the distance between the vertices.

This data structure internally constructs the a Δ-labeled spanning tree of the boolean matrix $\mathbf{A}$, where the weight of the tree gives us the time complexity per query. Hence, while we do not know the corrupted VC-dimension d, we do know how much time each query is going to take.

So when running Lemma C.6 on the above distance data structure, we can pick $k = n/\sqrt{T}$ where $T \geq n$ is the weight of the spanning tree. Thus, we get the update time

$$\begin{aligned}\widetilde{O}\left(n + \frac{k}{\epsilon}T + nk + \frac{n^2}{k}\right) &= \widetilde{O}\left(n + \frac{n\sqrt{T}}{\epsilon} + \frac{n}{\sqrt{T}}\right) \\ &= \widetilde{O}(n^{2-\frac{1}{2d}}),\end{aligned}$$

which proves the claim. □

Theorem 1.10. *(Dynamic Approximate k-center). Given an unweighted undirected graph $G = (V, E)$, there is a dynamic algorithm for $(2+\epsilon)$-approximate k-center with node update time $\widetilde{O}(kn^{2-1/2d}/\epsilon)$, where d is a bound on the corrupted VC-dimension of G.*

This result is a corollary of Theorem 1.9, as k-center can be reduced to computing k-single source distances.

Lemma C.7 (Theorem 7.3 by Cruciani et al. [2024])**.** *Given a graph $G = (V, E)$, a positive parameter $\epsilon \leq 1/2$, and a fully-dynamic data structure that maintains $(1+\epsilon)$-approximate single source distances with update time $T(n, m, \epsilon)$ and $Q(n, m, \epsilon)$ query time, there is a dynamic algorithm that maintains a $2(1+4\epsilon)$-approximate solution to fully-dynamic k-center in time $O(T(n, m, \epsilon) + k \cdot (Q(n, m, \epsilon) + n))$.*

D Characterization of matrices with constant VC-dimension

In this section, we provide a characterization of matrices with constant VC-dimension. Particularly, we prove Theorem 1.1 and state a number of examples of other constant VC-dimension matrices.

Fact 1. *Consider a hereditary class of $0/1$-matrices $\mathcal{M}$, meaning that $\mathcal{M}$ is closed under row/column deletion (i.e., each matrix $\mathbf{M} \in \mathcal{M}$ is still in $\mathcal{M}$ after deleting a row or column). If $\mathcal{M}$ is non-trivial, (i.e., does not contain every possible matrix), then there exists an absolute constant $c \in \mathbb{N}$ such that the VC-dimension of any matrix in $\mathcal{M}$ is at most c.*

Fact 1.1 provides a structural characterization of some matrices with constant VC-dimension, and it is an analogous result to the fact that non-trivial hereditary classes of graphs have low VC-dimension. To prove Fact 1.1, we introduce the notion of a bipartisation of a graph.

Definition D.1. *A bipartite graph $H = (A \cup B, E)$ is a bipartisation of graph G if removing all edges in A and B in G yields H for some partition A, B of $V(G)$. A graph class $\mathcal{G}$ is said to be hereditary if it is closed under induced subgraphs.*

Lemma D.2 (Lemma 3.4 of Bousquet et al. [2015])**.** *For any hereditary class $\mathcal{C}$ of graphs and for any bipartite graph H, if the graphs in $\mathcal{C}$ have infinite VC-dimension, then $\mathcal{C}$ contains a graph G whose bipartisation is H.*

By transposition, we get the following corollary:

Corollary D.3. *The equivalent contrapositive statement is that for any hereditary class $\mathcal{C}$ of graphs and for any bipartite graph H, if C does not contain a graph G whose bipartisation is $H = (A \cup B, E)$ where $A \cup B = V(G)$, then the graphs in C have finite* VC*-dimension.*

Lemma D.4. *The* VC*-dimension of any matrix in $\mathcal{M}$ is upper bounded by a finite constant.*

Proof. Since $\mathbf{M} \in \mathcal{M}$ is Boolean, we can treat it as a bipartite graph G, with rows representing left vertices and columns representing right vertices. In particular, we can treat $\mathcal{M}$ as a class of bipartite graphs closed under vertex deletion (row deletion = deleting a left vertex, column deletion = deleting a right vertex). Thus we can apply Lemma D.2 to $\mathcal{M}$.

Since $\mathcal{M}$ is non-trivial, there must be some graph H not contained in it. For this excluded graph H, define H' as the same graph but add one extra left vertex and one extra right vertex. Connect this new left vertex to every vertex on the right, and connect the new right vertex to every vertex on the left. Note that since H was not contained in $\mathcal{M}$ and it is closed under vertex deletion, H' also cannot be contained in $\mathcal{M}$.

Now assume there is $G \in \mathcal{M}$ that contains H' as bipartization. Then there is a partition A, B of V, such that keeping only the edges between A and B of G, results in H'. Because of the left vertex in H' that connects to all right vertices, and the right vertex connecting to all left vertices, we must have that A are all left and B are all the right vertices of G (or the other way around). But that means no edges were removed from G when restricting to edges between A and B, since a bipartite graph by definition only has edges connecting left and right vertices. So $G = H'$ and $H' \in \mathcal{M}$. This is a contradiction because by construction of H', it does not exist in $\mathcal{M}$.

We conclude, that set of bipartite graphs corresponding to $\mathcal{M}$ does not contain any graph whose bipartisation is H'. Hence by Corollary D.3 the VC-dimension must be a finite constant. Note that this is equivalent to the VC-dimension of the adjacency matrices being bounded, but $\mathcal{M}$ are not adjacency matrices. The adjacency matrices are of form

$$\begin{bmatrix} 0 & \mathbf{M} \\ \mathbf{M}^\top & 0 \end{bmatrix}$$

for matrices $\mathbf{M} \in \mathcal{M}$. We already argued that each adjacency matrix has constant VC-dimension, and since removing rows does not increase the VC-dimension (it is equivalent to deleting sets), this implies $\mathbf{M}$ also has constant VC-dimension. In conclusion, each matrix in $\mathcal{M}$ has constant VC-dimension. □

D.1 Examples of low VC-dimension matrices

We provide some broad families of matrices with constant VC dimension.

Adjacency matrices of H-minor free graphs. Eppstein [1995] showed that H-minor free directed graphs (where the minor can be obtained through vertex deletion, edge deletion, and edge contraction) has VC-dimension at most $|V(H)| - 1$. As an immediate consequence, since bipartite-graphs are triangle (K_3) free, the VC-dimension of any bipartite graph is 2. As a consequence, given any adjacency matrix $\mathbf{A}$ of a H-minor free graph G, for any $k \geq 1$, $\mathbf{A}^k$ is the adjacency matrix of a H-minor free graph G' (since $\mathbf{A}^k$ corresponds to the reachability graph where an edge $(u, v) \in E(G')$ corresponds to the existence of a k-length walk between u and v in G). Therefore, the VC-dimension of $\mathbf{A}^k$ is also at most $|V(H)| - 1$.

Adjacency matrices of interval graphs. Interval graphs are the intersection graphs of a family of intervals on the real line. Then, since the hypothesis class of intervals on the real line $\mathcal{H} = \{\mathbb{1}_{a,b} : a \leq x \leq b\}$ has VC-dimension 2 (any two points can be shattered by choosing an interval that

includes one or both points, but no set of three points can always be shattered), every interval graph has VC-dimension at most 2.

Boolean kernel matrices. A matrix where each column i (and each row) receives some label $\ell_i \in L$ from some (possibly infinite) set of possible labels L. Then, suppose each entry of $\mathbf{M}$ satisfies $\mathbf{M}_{i,j} = f(\ell_i, \ell_j)$ for some function $f : L \to \{0, 1\}$. For example, the labels could be points in $L = \mathbb{R}^d$ with f being indicator that the distance is at most some threshold. If f, L is fixed (i.e., independent of the number of rows/columns) then these graphs are closed under row/column deletion and thus have bounded VC-dimension by Theorem 1.1.

Shortest-path structures. Let G be an undirected graph with non-negative edge weights and let S be a subset of its shortest paths such that, for every pair (u, v) of distinct vertices, S contains exactly one shortest path between u and v. Marie de Lima et al. [2023] defines a range space associated with S and proves that its VC dimension is 2.

Adjacency matrices of planar graphs. Let G be a planar graph. Then, there is a planar drawing of G, and deleting vertices and edges retains this property. Therefore, these graphs are closed under row/column deletion and thus have bounded VC-dimension by Theorem 1.1.

Adjacency matrices of semi-algebraic graphs of bounded description complexity. By the Milnor-Thom theorem in real algebraic geometry [Matoušek, 2002] and Assouad's theorem [Assouad, 1983], semi-algebraic graphs of bounded description complexity have constant VC-dimensions. In turn, this extends to adjacency matrices of string graphs where every two curves intersect.

Practical matrices. Coudert et al. [2024] computes the VC-dimension of families of graphs that arise in practice, from protein interaction networks to autonomous Internet systems. Even for such graphs with millions of nodes, the VC-dimension of the graphs are typically between 3 and 8.

E Numerical Simulations

The algorithm from Theorem 1.3 has previously been described in Björklund and Lingas [2001] and Alves et al. [2024]. The latter already experimentally verified the efficiency of the algorithm on real-world data sets and observed that it beats the naive quadratic time matrix vector multiplication. Our work proves theoretic guarantees for this speed-up parameterized by the (corrupted) VC-dimension. While the main contribution of this work is to have theoretic guarantees and explanation for why the algorithm is efficient in practice, we here complement the result with a few experiments[3]. As we give theoretic bounds parameterized by d, our experiments focus on the d-dependence too.

Matrix Generation. Since we want to study the complexity dependence on d, we fix the matrix dimension to $n = 2^{12} = 4096$ and run experiments for $d = 2, ..., 12$. The maximum d is 12 because the VC-dimension $\leq \log(n) = 12$ so we do not need to consider $d > \log(n)$. The matrix size n was picked due to execution time constraints. The $n \times n$ Binary matrices are generated by sampling i.i.d uniform $a^{(i)} \in [0, 1]^d, b^{(i)} \in [0, d/2]$ for $i = 1, ..., n$ and $x^{(j)} \in [0, 1]^d$ for $j = 1, ..., n$, and letting $\mathbf{M}_{i,j} = \mathbb{1}_{\{a^{(i)\top} x^{(j)} \geq b^{(i)}\}}$. The range for the $b^{(i)}$ was chosen so that neither $\mathbf{M}$ nor $1 - \mathbf{M}$ are sparse, which otherwise would allow for naive matrix multiplication to already beat $O(n^2)$ time. The set system of linear classifiers has VC-dimension d, so the corrupted VC-dimension of $\mathbf{M}$ is upper bounded by d. This implies that the respective MST has weight at most $\widetilde{O}(n^{2-1/d})$. Figure 1 shows the observed weight of the MST relative to the ambient dimension d. The MST was computed exactly since easier to implement and the preprocessing time complexity is not focus of this work.

Complexity. For the matrix vector products we generate i.i.d. uniform vectors $v \in [0, 1]^n$. For each d, we compute 1000 matrix vector products and measure the run time. Figure 2 shows the mean and standard deviation of the observed time. We also compare the time to the naive numpy matrix vector multiplication. Figure 2 shows that, since matrix size n is fixed, the time complexity of numpy is constant for different d, whereas the MST-based algorithm is substantially faster for small d. As d increases, the gap gets smaller as one would expect, given the $\widetilde{O}(n^{2-1/d})$ theoretic upper bound. The experiments were conducted on an Apple M4 Pro macOS model with 12 physical cores / threads, and 48 GB unified memory RAM, and an Apple Accelerate Framework BLAS backend. The total

[3]The code for the experiments is available at `https://github.com/emiletimothy/structural_complexity_of_matrix_vector_multiplication`

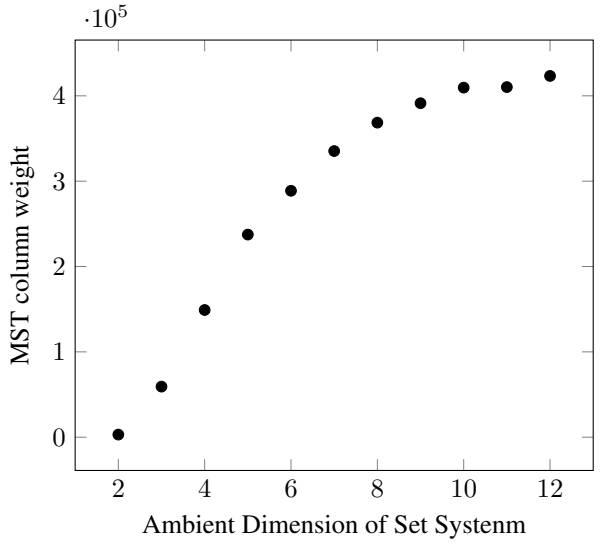

Figure 1: Weight of the Δ-labeled spanning tree for $2^{12} \times 2^{12}$ matrices generated from halfspaces in d dimensions.

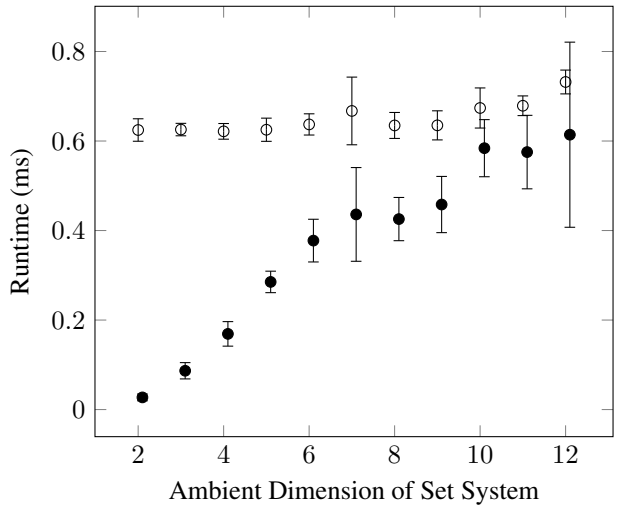

Figure 2: Average run time of the MST-based matrix vector algorithm (bold) and naive numpy matrix vector multiplication (blank), with 1-sigma error bars. Horizontal axis is the dimension of the points and halfspaces used to construct the matrix.

amount of time to run the experiments was ≈ 30 minutes. No GPU acceleration was used; all matrix operations ran on CPU.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We provide theoretical proofs for each claim made in the abstract and introduction. We do not make any other simplifying assumptions, and we explicitly discuss the limitations of this work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explicitly discuss the limitations of our results in the paper: the lack of matching lower bounds to verify if the scaling with the VC-dimension is optimal, and the sub-optimality of the non-Boolean setting.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We provide clear and formal mathematical proofs to support each claim.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide a full description of the algorithms and experimental setup.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our simulations are on synthetic matrices. We do provide open source code with sufficient instructions to reproduce the main experimental results.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The test details such as matrix size and dimension and how they were chosen have been described in both appendix and in the code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The observed time complexities are accompanied by 1-sigma error bars.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Our numerical simulations section in the Appendix lists the hardware specifications used, and the total time to run the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in this work conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This work is theoretical and foundational in nature. As such, it is not tied to any specific applications or deployments.

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: The paper does not use existing assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve any crowd-sourcing nor any research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve any crowd-sourcing nor any research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.