From Compression to Computation

A Dissertation
submitted to the Faculty of the
College of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Doctor of Philosophy
in Computer Science

By

Karthik Gajulapalli, M.S.

Washington, DC
March 25, 2026

From Compression to Computation

Karthik Gajulapalli, M.S.

Dissertation Advisor: Alexander Golovnev, Ph.D.

Abstract

We apply the powerful algorithmic method to uncover new connections between algorithms, lower bounds, and derandomization. Proving lower bounds in realistic computational models remains one of the central challenges in theoretical computer science. Unlike algorithm design, where one analyzes a specific strategy for a given task, proving lower bounds requires reasoning about all possible algorithms, including those that are not explicitly known or characterized.

Over the past decade, there has been significant progress linking proving lower bounds to the design of deterministic algorithms for highly structured problems such as Boolean Satisfiability and Range Avoidance. In this work, we substantially extend this paradigm in several new directions, as outlined below.

- We design new algorithms for Range Avoidance by uncovering a surprising connection to local search. We further show that even solving a very weak variant of Range Avoidance would imply breakthrough circuit lower bounds. Finally, we use the algorithms for Range Avoidance to prove arbitrary polynomial circuit lower bounds for the complexity class $\mathsf{O_2P}$. Thus, making it the smallest complexity class to have such lower bounds.

- We present a fast deterministic algorithm for multiplying two matrices when the product is promised to be sparse. As an application, we obtain the strongest known derandomization of celebrated randomized algorithm due to Freivalds'. Along the way, we identify concrete barriers to proving the hardness of matrix

multiplication under well-known hypotheses such as the Strong Exponential Time Hypothesis (SETH).

- Assuming the Non-Uniform Strong Exponential Hypothesis (NUSETH), we prove polynomial space lower bounds for any data structure solving problems such as Orthogonal Vectors, Approximate Nearest Neighbors, and 3-SUM. Complementing these lower bounds, we design state-of-the-art deterministic data structures for Orthogonal Vectors that match the performance of the best previously known randomized constructions.

A unifying theme throughout our results is the role of compression. Range Avoidance can be interpreted as a form of diagonalization against a collection of compressible objects. Our algorithm for output-sparse matrix multiplication fundamentally exploits the compressibility inherent in sparse matrices, drawing heavily on techniques from compressed sensing. More broadly, a data structure itself can be viewed as a compression of the joint space of inputs and all possible queries, encoding enough information to answer queries efficiently while using substantially less space than the full truth table.

Acknowledgments

The hardest part of writing this dissertation has been figuring out how to thank the many people who made it possible. I considered following Marcus Aurelius' example in Meditations and cataloging what I learned from each person I encountered. Unfortunately, Marcus was Emperor of Rome, and I am, on many days, someone who struggles to tie his own shoelaces. Nevertheless, I will do my best to acknowledge the support that carried me through this rollercoaster of a journey.

The biggest pillar of support in my academic life has been my advisor, Alexander Golovnev. I'm convinced that Sasha is actually a saint masquerading as a complexity theorist. He created an environment where I had no worries—I could work on whatever problem I wanted, with seemingly unlimited support to travel the world and develop my research program. Whenever I had a result, no matter how small, Sasha would tap deep into his network to arrange for me to give a full-hour long talk at a top school. What I will perhaps miss most is Sasha's uncanny ability to filter through incoherent ideas and extract something precise and meaningful. But above all, Sasha always put his students before himself, and I hope I can carry that same care forward.

I would also like to thank my committee—Huck, Jeremy, and Justin for their guidance throughout my time at Georgetown. My letter writers—Huck, Ilya, and Bala deserve special thanks for their infinite wisdom, patience and for never complaining about the gargantuan number of positions I applied to. Huck was also kind enough to host me for a summer in Boulder, which turned out to be one of the most enjoyable summers of my grad school years.

I was fortunate to collaborate with many brilliant people who gave me new perspectives on life and, occasionally, even research: Huck Bennett, Nikolai Chukhin, Surendra Ghentiyala, Sasha Golovnev, Samuel King, Zeyong Li, James Liu, Tung Mai, Ivan Mihajlin, Satyajeet Nagargoje, Sidhant Saraogi, Ilya Volkovich, Vijay Vazirani, and Evelyn Warton. A special thanks as well to the Simons Institute and DIMACS for hosting workshops that introduced me to so many amazing members of the community.

An underappreciated part of graduate school is the administrative support that makes everything else possible. I am very grateful to Ray, Djuana, Mari, Maria, and Venus for handling travel, courses, scheduling, and the countless bureaucratic details that allowed me to focus on research. A special thank you also to our wonderful chair, Lisa, who fostered a sense of community by encouraging regular interaction within the department, especially through tea time.

Before coming to Georgetown, I began as an undergraduate at UCI. Immaturity is something best understood in hindsight, and I am grateful to Philippe, who taught me how to write an email to a professor that highlights why I would be a good fit for their lab—rather than why overlooking me would be their greatest mistake. I started doing research in databases because Sharad was kind (or naive) enough to trust a clueless seventeen-year-old with his own office desk. That work led me to deeper questions about computation and eventually to an interest in zero-knowledge proofs.

Stas taught my first real theory course, and his advanced cryptography class with only three students remains my favorite course to this day. Working with him was my first exposure to theoretical computer science research. Later, Vijay joined UCI, and I had the opportunity to work with him for several years. He took me to the Simons Institute for a program on Matching Markets, and I published my first theory paper with Vijay on stable matching. From Vijay, I learned that presenting your

idea in the right way is almost as important as coming up with the idea itself. And most importantly, if you stare at any problem long enough you will eventually gain some structural insight about it. I would also like to thank Sandy Irani, who first introduced me to computational complexity, and Richert Wang, who supported every UROP proposal I submitted.

Life is far less lonely when shared with friends, many of whom visited me in D.C. I'm grateful to my summer-bridge friends Karla, Yaya, and Sunny for helping me transition to a new country. Ayushi and Vamsi for their constant support and long debates at Newport Beach. My basketball friends—Andre, Arjay, Austin, Cody, Grant, Heran, Madhav, and Rohan made school less monotonic. Madhav drove me everywhere during my first two years, and Rohan took over the next two. Andre accompanied me to many music concerts, Arjay passed down a much-appreciated couch and dining table, and Austin taught me that gambling is only a degenerate hobby when you lose.

During COVID, I discovered hiking, which became one of my most rewarding hobbies. Thanks to Astha, Siddharth, and Troy for trading sleep and comfort for the flimsy promise of a beautiful sunrise.

One of the best ways to make friends in college is through programming clubs. I'm grateful to my ACM friends: Arne, Bryon, Cindy, Jens, Junlin, Kaleo, Kevin, Linh, Meta, Pasha, Pooya, Taneisha, and Tim. Arne hosted me during many New York visits and tolerated endless debates about movies and music. Pooya's apartment in Irvine served as a winter home after I moved to D.C. Kevin for consistently blundering winning positions in chess, Bryon for his unwavering patriotism to this wonderful country, Jens who made Spartan weekends into a quarterly tradition, and Taneisha who taught me the power of bakeries and Cmd + Shift + 4.

At UCI, I also met many wonderful graduate students. The theory group was a supportive environment, and Daniel, James, Jiayu and Ofek were always open to discussing any topic I found interesting. James was extremely helpful in my transition from algorithmic game theory to complexity and was my first real collaborator. Joel explained to me that I was a frog in a well, and broadened my view of theoretical computer science beyond UCI. Esther provided me constantly with sage advice and encouragement, and continued to hire me as a programmer wherever she went to as a postdoc.

I was initially apprehensive about moving to Georgetown, knowing no one in the city. It turned out to be one of the best decisions I've made. I met many incredible friends: Ali, Chao, Devika, Dhiraj, Duong, Harshit, Hrishikesh, Jianan, Kenny, Laasya, Sam, Satyajeet, Shabnam, Sidhant, Shuchen, and Urshila. We shared countless dinners, celebrations, movie nights, and board game nights. Shuchen got me into running on my very first day, and Chao never declined a game of chess. I biked countless miles with Sidhant, who was also a steady source of support during the stressful job search even while competing for many of the same positions. Shabnam made sure we explored D.C., and Hrishikesh was always up for wandering without a plan. Laasya was a wonderful neighbor in D.C. and coincidentally Boulder, and made living in both those places much more fun.

Finally, I want to thank my family, who have supported me through everything. My grandfather, who made it out of rural India through education and who was still learning new languages into his eighties. My grandmother, for her wisdom and who taught me luck and belief were many times indistinguishable. My sister, Aparna, for her unwavering loyalty and at times overwhelming love. And my parents, Ravi and Anu—my father, the first mathematician in the family, and my mother, my first math

teacher. Above all, I am grateful to them for always putting my happiness before their own.

There is much more to be grateful for: the hill my house stood on, a physical metaphor for life; the hike up Old Rag; the food scientists at Trader Joe's; REI; the many restaurants and cafés in D.C.; the Warriors winning their fourth ring; My Mister; Siddhartha and the random stranger who told me about the magical powers of the anterior mid-cingulate cortex. The list goes on, and I have surely missed many.

If you've made it this far, I hope you keep reading—and enjoy the thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Complexity theory broadly seeks to characterize the amount of computational resources required to solve a given task. Different computational models capture the trade-offs between these resources—most notably time, space, and randomness. For example, how much time is necessary to solve a problem? Can we reduce the running time of our algorithm if we are allowed to use more space? If our machine is permitted to toss random coins, does this additional power enable it to compute more functions, or simply compute them more efficiently?

One of the most fundamental questions at the frontier of complexity theory is the following:

Open Problem 1. *Can every efficient randomized algorithm be simulated by an efficient deterministic algorithm?*

This is the core of the P vs BPP problem,[1] and has been an avenue of very beautiful and interesting research [36, 37, 69, 78, 86, 104]. Maybe counter-intuitively most evidence points in the direction that we do not really need randomness, and that randomness can be simulated by deterministic machines if it has access to a hard function. For example, in the seminal work of Impagliazzo and Wigderson [69], they show that if the class E^2 is not contained in circuits of size $2^{\epsilon n}$, then $\mathsf{P} = \mathsf{BPP}$.

[1] The class P refers to the class of languages decidable by a deterministic algorithm running in polynomial time, while the class BPP refers to the class of languages solvable by randomized algorithms running in polynomial time with bounded error.

[2] E is the class of languages that can be decided in deterministic time $2^{O(n)}$.

Under complexity theoretic assumptions (existence of a hard function in E), one can construct a Pseudorandom Generator (PRG), which one can think of as a deterministic procedure that outputs a sequence of bits, with the property that the output of a PRG will be indistinguishable from a sequence of truly random bits to any polynomial time algorithm. As a result replacing your randomness with the output of your PRG, lets you simulate any randomized algorithm deterministically in a black-box manner.

However, when translating this view to practice there are a couple of issues. For one, the existence of a PRG relies on the existence of a hard function which implicitly requires us to prove a lower bound, something that turns out to be an extremely difficult task. For example, in the [69] framework we would need to prove a lower bound of size $2^{\varepsilon n}$ against E, however the best known lower bound against E is that it cannot be computed by circuits of size $3.1n$ [95], an exponential gap. In fact, even for the larger complexity class E^{NP}, the strongest known lower bound shows only that it does not admit circuits of size $3.1n$. That is, we cannot even rule out the possibility that every language in E^{NP} has a linear-size circuit computing it. For comparison, one approach to resolving the famous P versus NP conjecture would be to prove that the much smaller class NP does not admit polynomial-size circuits, an even stronger lower bound.

One reason complexity theorists have struggled to prove even seemingly modest lower bounds is the fundamentally universal nature of such statements. To prove a circuit lower bound, one must show that no circuit of a given size computes the target function. This requires ruling out an enormous and poorly understood space of all possible circuits. On the other hand, when designing algorithms, the task is existential rather than universal: it suffices to exhibit a single strategy and analyze its properties. This asymmetry between constructing one good object and ruling out all bad ones lies

at the heart of why proving lower bounds is so much more challenging than designing algorithms.

So before we can construct a PRG we need to address the even simpler question.

Open Problem 2. *Is there a language in* E^{NP} *that cannot be computed by Boolean circuits of superlinear size?*

Moreover, a PRG is an exceptionally powerful tool: it derandomizes any randomized algorithm in a black-box manner. That is, the PRG is oblivious to the specific problem being solved, it simply replaces truly random bits with carefully constructed pseudorandom bits. At the same time, there are many natural problems like Polynomial Identity Testing (PIT), the Circuit Acceptance Probability Problem (CAPP), and Approximate Counting, for which we have fast randomized algorithms but no known efficient deterministic ones. Rather than seeking a universal black-box derandomization via PRGs, one can instead attempt to derandomize these problems individually in a white-box fashion, exploiting their internal structure.

This avenue of research is well motivated, and appears to bear many interesting consequences. For example, being able to derandomize PIT, will immediately yield circuit lower bounds beyond the grasp of current techniques [78]. In the case of CAPP, a pseudo-deterministic algorithm (a randomized algorithm that always outputs the same solution with high probability) for CAPP would yield a hierarchy theorem for BPP and a circuit lower bound against MA [46].

Although designing algorithms and proving lower bounds may appear to have opposing goals, it turns out that developing non-trivial algorithms for carefully chosen special problems can directly imply strong circuit lower bounds. As discussed above a polynomial time deterministic algorithm for PIT would imply one of either boolean or

arithmetic circuit lower bounds. Interestingly, even designing just better than brute-force algorithms for problems like Boolean Satisfiability (SAT) would yield many interesting lower bounds. That is, even an algorithm running in time $O(2^{0.99n})$ that solves SAT would imply strong circuit lower bounds [26, 76].[3] This perspective of reducing lower bound questions to the design of improved algorithms is known as the algorithmic method. Over the past two decades, it has emerged as a powerful paradigm, leading to several breakthrough lower bounds [35, 39, 40, 41, 82, 86, 96, 102, 109, 127, 129, 130].

In this thesis we will design algorithms for two kinds of structural questions, and as a consequence uncover more connections between algorithms, derandomization and lower bounds:

- **Hay in the Haystack Problems:**

  We refer to a problem as "hay in the haystack" if a uniformly random point is a good solution with high probability. In such problems, randomness succeeds not by carefully navigating structure, but simply because most points already work. For example, finding a prime number fits this paradigm: a randomly sampled large integer is prime with reasonably high probability. Similarly, testing whether a polynomial is identically zero can be reduced to checking whether it evaluates to zero on a randomly chosen point. What these problems share is a common structural feature: there exists an oblivious randomized algorithm that samples a random point and succeeds with high probability, whereas a deterministic algorithm, in the worst case, appears to require enumerating an exponentially large search space.

[3] SAT admits a trivial brute-force algorithm running in time $2^n \cdot \text{poly}(n)$.

- **Problems requiring Perebor:**

  "Perebor" is the Russian word for brute force, and it aptly describes our current state of knowledge for many fundamental problems. For tasks such as SAT, Hamiltonian Path, and Optimal Compression, we do not know algorithms that substantially improve over exhaustively enumerating all possible solutions. Strikingly, even a modest improvement over brute force for SAT would have major consequences: designing algorithms that slightly beat exhaustive search already implies new circuit lower bounds. In fact, improving over "perebor" even in stronger models, such as co-nondeterministic algorithms, would still yield lower bounds.

  Conversely, one can assume that a problem inherently requires "perebor" and use this as a foundation for conditional hardness results. The Strong Exponential Time Hypothesis (SETH), for example, essentially posits that SAT cannot be solved significantly faster than brute force. This assumption allows us to mechanize the algorithmic method, by designing fine-grained reductions from SAT to other problems, we ensure that any faster algorithm for the target problem would translate into a faster than brute-force algorithm for SAT, thereby refuting SETH.

1.1 Our Results

We briefly highlight our core results, but refer the reader to individual chapters for details and more context.

1.1.1 Range Avoidance

In Chapter 3, we investigate the *"hay in the haystack"* phenomenon through the lens of the Range Avoidance problem. The Range Avoidance problem (Avoid) takes as input a boolean circuit C with n input bits and $m > n$ output bits, and asks us to find a string y that does not lie in the range of C. Since $m > n$, the weak pigeonhole principle guarantees that such a y must exist.

Korten [86] showed that Avoid has a P^{NP} algorithm if and only if E^{NP} has maximum circuit complexity. Our first result strengthens this connection: even a P^{NP} algorithm for a highly restricted variant, called NC^3_0-Avoid, would already suffice to prover super linear circuit lower bounds and resolve Open Problem 2.

Here, NC^3_0-Avoid denotes the special case in which the input circuit belongs to the class NC^3_0; that is, each output bit depends on at most three input bits.

Theorem 1. *A P^{NP} algorithm solving NC^3_0-Avoid with stretch $m = n+O(n^{2/3})$ would result in super linear circuit lower bounds.*

We next design algorithms for the most general version of Avoid. Our approach exploits the recursive and downward self-reducible structure of a closely related problem known as the Linear Ordering Principle (LOP). Using this connection, we show that Avoid lies in the complexity class $\mathsf{UEOPL}^{\mathsf{NP}}$.

Here, UEOPL[4] is a subclass of TFNP, the class of total search problems solvable in non-deterministic polynomial time. Notably, UEOPL is among the smallest complexity classes widely conjectured to lie outside P. Placing Avoid in $\mathsf{UEOPL}^{\mathsf{NP}}$, therefore brings us closer to designing P^{NP} algorithms for Avoid. Achieving this would strongly resolve Open Problem 2, as it would imply that E^{NP} requires circuits of maximal complexity.

[4] UEOPL stands for Unique End of Potential Line, for a formal definition see Definition 3.1.6

Theorem 2. *AVOID* $\in \mathsf{UEOPL}^{\mathsf{NP}}$.

Finally, we give an application of current state-of-the-art AVOID algorithms to prove new unconditional arbitrary polynomial circuit lower bounds[5] for the computational complexity class $\mathsf{O_2P}$.[6] This makes $\mathsf{O_2P}$, the smallest complexity class to have such lower bounds.

Theorem 3. *For every $k \in \mathbb{N}$, there exists an explicit language $L_k \in \mathsf{O_2P}$, such that $L_k \notin \mathsf{SIZE}[n^k]$.*

Building on the above result, we establish a hierarchy theorem for $\mathsf{O_2TIME}$. Proving hierarchy theorems for uniform semantic complexity classes such as BPP, MA, and $\mathsf{S_2P}$ remains a major open problem in complexity theory. Although hierarchy theorems are known for these semantic classes when augmented with one bit of non-uniform advice, our result for $\mathsf{O_2TIME}$ provides the first hierarchy theorem for a uniform semantic class that contains BPP (see Section 3.4 for further discussion).

Corollary 1. *For all $c \geq 1$, $\varepsilon > 0$, $\mathsf{O_2TIME}[n^{c+\varepsilon}] \not\subseteq \mathsf{O_2TIME}[n^c]$.*

1.1.2 Matrix Multiplication Verification

In Chapter 4, we uncover the *"hay in haystack"* phenomena in a problem that, at first glance, does not seem to exhibit such structure. Specifically, we study the problem of Matrix Multiplication Verification (MMV), which takes as input three matrices A, B and C and outputs 1 if and only if $C = AB$.

By the celebrated result of Freivalds, there is an optimal randomized algorithm running in time $\widetilde{O}(n^2)$. However, the best known deterministic algorithms run in time

[5]We write $f \in \mathsf{SIZE}[t(n)]$ to denote that f can be computed by a Boolean circuit with at most $t(n)$ gates.

[6]The complexity class $\mathsf{O_2P}$ refers to oblivious version of the second level of the symmetric hierarchy. For a formal definition, see Definition 3.1.8.

$O(n^\omega)$, simply computing the product AB and then checking each entry of the result.[7] A major question in derandomization is whether one can design significantly faster deterministic algorithms for MMV, a problem highlighted in standard textbooks such as [101, 119].

We give fast deterministic algorithms for MMV, running in time $o(n^\omega)$ when the matrix C disagrees with AB on a sparse set of coordinates.[8] Our algorithm in fact serves as a corollary of a much more powerful algorithm that deterministically computes the product of two matrices when the output is promised to be sparse.

Prior to our work, no deterministic algorithm was known for computing output-sparse matrix products better than $o(n^\omega)$, even for matrices with output sparsity $O(n^{0.75})$ [89, 90]. We give an algorithm that is better than $o(n^\omega)$ even when the output sparsity is $O(n^{1.99})$, moreover the bounds on our algorithm scale with improved bounds on ω (see Section 4.2.4 for a plot comparing running times and dependence on sparsity).

Theorem 4. *Let $\omega > 2$, then for any two matrices A, B if their product AB has at most $O(n^{1.99})$ non-zero entries, then there is a deterministic algorithm that computes AB in time $o(n^\omega)$.*

As an application of Theorem 4 we are able to get the following result about MMV.

Corollary 2. *Let $\omega > 2$,[9] then for three matrices A, B, and C, if AB and C disagree on upto $n^{1.99}$ coordinates, then one can detect this in $o(n^\omega)$ time.*

This result is particularly counterintuitive because, when AB and C disagree on εn^2 fraction of entries, there is a black-box randomized algorithm running in $O(n)$

[7] ω denotes the matrix multiplication exponent; the current best bound is $\omega < 2.372$ [8].
[8] Here sparse means $O(n^{2-\varepsilon})$ coordinates, for some $\varepsilon > 0$.
[9] When $\omega = 2$, the naïve $O(n^\omega)$ algorithm is optimal.

time,[10] however our deterministic algorithms revert to running in the trivial $O(n^\omega)$ time on such instances. Indeed, this sharp contrast is exactly the *"hay in haystack"* phenomenon.

Our algorithms treat the fastest known matrix multiplication procedures as a black box. However, progress on improving the matrix multiplication exponent has been notoriously slow. This raises a natural question: can this slow progress be explained by a fine-grained complexity hypothesis such as SETH? In other words, could a substantially faster matrix multiplication algorithm break the "perebor" conjecture for satisfiability?

We show that any SETH-based hardness of matrix multiplication would yield either new breakthrough circuit lower bounds or the strongest known explicit construction of a rigid matrix.[11] Although both consequences are widely believed to be true, proving either appears well beyond the scope of current techniques. Thus, any proof establishing SETH-hardness of matrix multiplication would implicitly require overcoming one of these major barriers, thereby explaining the difficulty of such reductions.

Theorem 5. *If matrix multiplication is $n^{2.31}$-SETH-hard*[12] *then one of the following holds:*

- *A language E^{NP} cannot be computed by circuits of linear size (resolve Open Problem 2).*

[10] This algorithm just uniformly samples $O(1/\varepsilon) = O(1)$ entries of the matrix and checks if the product is correctly computed over these entries.

[11] A rigid matrix is a matrix that cannot be expressed as the sum of a low-rank matrix and a sparse matrix. A simple counting argument shows that a random matrix is rigid, however there is no efficient algorithm to deterministically find such a matrix (an explicit construction).

[12] A problem is $t(n)$ hard under SETH, if an algorithm for the problem running in time $t(n)^{1-\varepsilon}$ for a constant ε would result in a faster algorithm for SAT, thus refuting SETH (see Definition 2.3.6).

- *Explicit construction of a rigid matrix.*

1.1.3 Improved Data Structures and Lower Bounds

In Chapter 5, we present new deterministic data structures for the Orthogonal Vectors problem (OnlineOV). In $\text{OnlineOV}_{N,d}$, we are given a database of N boolean vectors in dimension d. An (S, T) data structure for OnlineOV first performs a preprocessing phase, storing S bits of information. Later, upon receiving a query vector in $\{0,1\}^d$, it determines in time T whether q is orthogonal to any vector in the database.

Designing fast data structures for OnlineOV leads to improved data structures for several fundamental problems, including Approximate Nearest Neighbor, Orthogonal Range Searching, Subset Query and Partial Match. Perhaps most fundamentally, the complexity of Orthogonal Vectors is closely tied to the complexity of SAT and the "perebor" conjecture (SETH). In particular, if there existed constants $\alpha, \varepsilon > 0$ such that, for every constant $c \geq 1$, one could construct an $(N^\alpha, N^{1-\varepsilon})$ data structure for $\text{OnlineOV}_{N,c\log N}$ with preprocessing time $O(N^\alpha)$, then this would refute SETH [60, 128].

In the regime where the dimension $d = c \log N$, our deterministic data structure for OnlineOV matches the performance of the state-of-the-art data structures for this problem, which are all randomized [11, 30]. Our result can be viewed both as progress toward resolving "perebor"-style questions surrounding SAT and as an instance of white-box derandomization in the context of data structures. While our data structures achieve sublinear query time $n^{1-\varepsilon}$, the parameter ε depends on the value of c in the dimension $d = c \log n$. In contrast, refuting SETH would require an ε that is independent of c. As an added feature, our data structures are combinatorial and have extremely efficient preprocessing.

Theorem 6. *Consider any $c > 1$, and any $\delta > \Omega(\log c/c)$. Then there is an (S, T) data structure with preprocessing time T_p for* $\text{OnlineOV}_{n,c\log n}$ *where:*

$$T = \tilde{O}\left(n^{1-\frac{1}{O(c\log c)}}\right), \qquad \text{and} \qquad S,\ T_p = \tilde{O}(n^{1+\delta}).$$

Our algorithms extend to arbitrary dimensions d. In the regime where $d = n^{\varepsilon}$, we obtain a substantial improvement over the best known data structures for moderate dimensions due to [34], which achieve space $S \leq n \cdot 2^{O(\sqrt{\varepsilon} d \log^2 d)}$ and query time $n^{1-\varepsilon} \cdot d$.

Theorem 7. *Consider any n, d and $\varepsilon < 1/2$, then there exists an (S, T)-data structure for* $\text{OnlineOV}_{n,d}$ *with preprocessing time T_p such that:*

$$T \leq n^{1-\varepsilon} d\,, \qquad S = n^{1-\varepsilon} d\, 2^{O(\varepsilon d \log(1/\varepsilon))}\,, \qquad T_p \leq n^{1+3\varepsilon} d\, 2^{O(\varepsilon d \log(1/\varepsilon))}\,.$$

As a consequence, we refute a strong version of the curse of dimensionality conjecture, which informally states that when $d = \omega(\log n)$, any data structure for $\text{OnlineOV}_{n,d}$ must use either $2^{\Omega(d)}$ space or $\Omega(n)$ query time. To see this, set $\varepsilon = \frac{2\log d}{\log n} = o(1)$ with $\varepsilon \log(1/\varepsilon) = o(1)$. Then in this regime, our data structure simultaneously achieves time $T \leq n/d = o(n)$, and space $S = 2^{o(d)}$. Thus, we simultaneously achieve sublinear query time and space sub-exponential in the dimension, contradicting the strong form of the curse of dimensionality.

Complementing our fast data structures, we also prove strong lower bounds for data structures with computationally unbounded preprocessing under two non-uniform "perebor" conjectures.

Obtaining unconditional lower bounds for data structure problems remains a major open challenge, and current techniques appear insufficient to prove any non-trivial bounds [48, 103, 126]. Conceptually, proving data structure lower bounds can be viewed as establishing a time–space tradeoff: if queries must be answered in time T,

then the data structure must use at least space S. However, when the preprocessing time is bounded, this interpretation becomes less natural, since the preprocessing itself may already dominate the computational cost.

To better capture the inherent tradeoff, we instead consider non-uniform algorithms,[13] which place no restriction on how the stored information is computed. In this model, one can view a data structure as a non-uniform algorithm, where the space used by the data structure is captured by the non-uniform advice.

Our first hypothesis is the non-uniform analogue of SETH, called NUSETH, which posits that even non-uniform algorithms cannot solve SAT substantially faster than exhaustive search ("perebor"). This assumption has been widely studied in the literature [4, 26, 42, 55] and has been used to establish several conditional lower bounds.

Assuming NUSETH, we show that even a highly restricted variant of OnlineOV, namely w-Sparse-OnlineOV, cannot be solved by arbitrary polynomial-size data structures. In this variant, every vector in the input database contains at most $w = O(1)$ nonzero entries.

Theorem 8. *Under NUSETH, for all constant $c > 0$, and $\delta, \gamma \in (0, 1)$, there exists a constant w, such that there is no $(n^c, n^{1-\gamma})$-data-structure for w-Sparse-*$\text{OnlineOV}_{n,n^\delta}$.

We next introduce a new "perebor" conjecture called Non-Uniform Hamiltonian Path Conjecture (NUHAMPATH), which informally states that any non-uniform algorithm solving Hamiltonian Path on n vertices, requires time at least 2^n (see Conjecture 5.1.8).

We use NUHAMPATH to prove a lower bound for the 3-SUM with preprocessing problem. In this problem, the preprocessing algorithm receives three sets: $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$ each consisting of N integers, and preprocesses them into a data structure $\sigma \in \{0, 1\}^S$

[13] A non-uniform algorithm has the additional power of accessing a fixed advice string that depends only on the input length (see Definition 5.1.6).

of size S. Then, the query algorithm receives a query $\mathcal{X}_1' \subseteq \mathcal{X}_1, \mathcal{X}_2' \subseteq \mathcal{X}_2, \mathcal{X}_3' \subseteq \mathcal{X}_3$, and decides in time T if there exist $x_1 \in \mathcal{X}_1', x_2 \in \mathcal{X}_2', x_3 \in \mathcal{X}_3'$ such that $x_1 + x_2 = x_3$.

A line of research [14, 31, 33, 81] culminated in an (S, T)-data structure for 3-SUM with preprocessing with $S = T = \widetilde{O}(N^{1.5})$. While this data structure is already very efficient, it is an open question if it can be further improved. We do not have any unconditional lower bounds better than the trivial ones requiring linear time or linear space.

Assuming NUHAMPATH, we now show that any data structure for 3-SUM with computationally unbounded preprocessing must require either super-linear space or super-linear query time.

Theorem 9. *Under NUHAMPATH, there is no* $(N^{1.087}, N^{1.087})$*-data structure for* 3-SUM *with computationally unbounded preprocessing.*

Chapter 2

Preliminaries

2.1 Models of Computation

2.1.1 Turing Machines

Definition 2.1.1 (Deterministic Time)**.** Let $t : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{DTIME}[t(n)]$, if there exists a deterministic time multi-tape Turing machine that decides L, in at most $O(t(n))$ steps.

2.1.2 Boolean Circuits

Definition 2.1.2. A boolean circuit C with n inputs and size s, is a Directed Acyclic Graph (DAG) with $(s+n)$ nodes. There are n source nodes corresponding to the inputs labelled $1, \ldots, n$ and one sink node labelled $(n+s)$ corresponding to the output. Each node, labelled $(n+i)$, for $1 \leq i \leq s$ has an in-degree of 2 and corresponds to a gate computing a binary operation over its two incoming edges.

Definition 2.1.3. Let $s : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{SIZE}[s(n)]$ if L can be computed by circuit families of size $s(n)$ for all sufficiently large input size n.

We now state the hierarchy theorem for circuit size. The standard proof of this result is existential and goes through a counting argument (see e.g. [13]). However, we comment that using the framework of Avoid, we can now actually get a constructive size hierarchy theorem, albeit with worse parameters.

Theorem 2.1.4 (Circuit Size Hierarchy Theorem [13]). *For all functions $s : \mathbb{N} \to \mathbb{N}$ with $n \leq s(n) < o(2^n/n)$:*

$$\mathsf{SIZE}[s(n)] \subsetneq \mathsf{SIZE}[10s(n)] \ .$$

For our applications, it will be essential to have a tight encoding scheme for circuits. In fact, we will also need the fine-grained complexity of the Turing machine computing Circuit-Eval (i.e. given as input a description of a circuit C and a point x, computes $C(x)$).

Lemma 2.1.5. *For $n, s \in \mathbb{N}$, and $s \geq n \geq 12$, any n-input, s-size circuit C, there exists an encoding scheme $E_{n,s}$ which encodes C using $5s \log s$ bits. Moreover, there exists a multi-tape Turing machine M such that $M(E_{n,s}, x) = C(x)$ and it runs in $O(s^2 \log s)$ time.*

Finally, we recall the famous Cook-Levin theorem that lets us convert a machine $M \in \mathsf{TIME}[t(n)]$ into a circuit $C \in \mathsf{SIZE}[t(n) \log t(n)]$.

Theorem 2.1.6 (Cook-Levin Theorem [13]). *Let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function. Then any multi-tape Turing machine running in $\mathsf{TIME}[t(n)]$ time can be simulated by a circuit-family of $\mathsf{SIZE}[t(n) \log t(n)]$.*

2.1.3 Complexity Classes

We assume familiarity with basic complexity theory and notion of non-uniform circuit families. For definitions of complexity classes $\mathsf{P}, \mathsf{NP}, \mathsf{BPP}, \mathsf{ZPP}, \mathsf{MA}, \mathsf{E}, \mathsf{P/Poly}$ see, e.g., [13, 57].

The complexity classes $\mathsf{FP}, \mathsf{FP}^{\mathsf{NP}}, \mathsf{FE}$, and $\mathsf{FE}^{\mathsf{NP}}$ are classes of search problems analogous to the classes of decision problems $\mathsf{P}, \mathsf{P}^{\mathsf{NP}}, \mathsf{E}$, and E^{NP}. For example, the class FP contains all relations that can be computed by deterministic polynomial-time Turing machines.

2.2 Log Depth Circuits and Matrix Rigidity

Here we introduce special circuit classes NC^0_k and NC^1, which we define below.

Definition 2.2.1. (NC Circuits) The circuit class NC^i contains multi-output Boolean circuits on n inputs of depth $O(\log^i(n))$ where each gate has fan-in 2. We are particularly concerned with the following classes of circuits:

- For every constant $k \geq 1$, NC^0_k is the class of circuits where each output depends on at most k inputs.
- NC^1 is the class of circuits of depth $O(\log(n))$ where all gates have fan-in 2.
- Linear NC^1 circuits are circuits of depth $O(\log(n))$ where every gate has fan-in 2 and computes an affine function, i.e., the XOR of its two inputs or its negation.

It is a long-standing open problem in circuit complexity to prove super-linear lower bounds on the size of (linear) NC^1 circuits computing an n-output function from FP or even $\mathsf{FE}^{\mathsf{NP}}$ [13, 120, Frontier 3]. Valiant [120] suggested an approach for proving super-linear lower bounds for linear NC^1 circuits using the notion of matrix rigidity.

We begin by introducing the notion of global rigidity, which may be viewed as an average-case analogue of rank. Intuitively, a matrix is globally rigid if its rank remains high even after modifying an arbitrary sparse subset of its entries.

Definition 2.2.2 (Global Rigidity)**.** Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n \times n}$ be a matrix, and $0 \leq r \leq n$. The rigidity of A over $\mathbb{F}$, denoted by $\mathcal{R}^{\mathbb{F}}_r(A)$, is the Hamming distance between A and the set of matrices of rank at most r. Formally,

$$\mathcal{R}^{\mathbb{F}}_r(A) := \min_{S:\ \mathrm{rank}(A-S) \leq r} \|S\|_0 \ .$$

In other words, a matrix A has rigidity $\mathcal{R}_r^{\mathbb{F}}(A) \geq s$ if and only if $A \in \mathbb{F}^{n\times n}$ *cannot* be written as a sum

$$A = S + L\,,$$

where $S \in \mathbb{F}^{n\times n}$ is an $(s-1)$-sparse matrix, and $L \in \mathbb{F}^{n\times n}$ has low rank $\text{rank}(L) \leq r$.

However, for applications such as Valiant's program on proving super-linear circuit lower bounds, a potentially weaker notion of rigidity (row-rigidity) suffices. In this variant, it is enough that the rank remains high even after modifying up to s entries in each row.

Definition 2.2.3 (Row Rigidity)**.** For $r, s \in \mathbb{Z}^+$, a matrix $M \in \mathbb{F}_2^{n\times n}$ is (r, s)-*rigid* if M cannot be written as a sum

$$M = L + S\,,$$

where $L, S \in \mathbb{F}_2^{n\times n}$, L is low rank, i.e., $\text{rank}(L) \leq r$, and S is row sparse, i.e., every row of S has at most s non-zero entries.

Valiant [120] proved that a linear operator given by a sufficiently rigid matrix requires linear NC^1 circuits of size at least $\Omega(n \log\log(n))$, but there are still no known constructions of such rigid matrices even in $\mathsf{FE}^{\mathsf{NP}}$.

Theorem 2.2.4 ([120])**.** *If a family of matrices $(M_n)_{n\geq 1}$, $M_n \in \mathbb{F}^{n\times n}$, is $(\epsilon n, n^{\delta})$-rigid for constant $\epsilon, \delta > 0$, then the linear map $x \mapsto Mx$ requires linear NC^1 circuits of size $\Omega(n \log\log(n))$.*

2.3 Fine-Grained Complexity

In the k-SAT problem, given a k-CNF formula φ, the task is to check if φ has a satisfying assignment. The complement of k-SAT, the k-TAUT problem, is to decide

if all assignments to the variables of a given k-DNF formula satisfy it. The celebrated Strong Exponential Time Hypothesis (SETH) is now central to the field of fine-grained complexity.

Definition 2.3.1 (Strong Exponential Time Hypothesis (SETH), [68])**.** For every constant $\varepsilon > 0$ there exists $k \in \mathbb{Z}^+$ such that no deterministic algorithm solves k-SAT on formulas with n variables in time $2^{(1-\varepsilon)n}$.

The field of fine-grained complexity has leveraged SETH to prove tight bounds on the complexity of a large host of problems (see the excellent surveys by Vassilevska Williams [122, 123]). We will also use the following stronger hypothesis.

Definition 2.3.2 (Nondeterministic Strong Exponential Time Hypothesis (NSETH), [26])**.** For every constant $\varepsilon > 0$ there exists $k \in \mathbb{Z}^+$ such that no nondeterministic algorithm solves k-TAUT on formulas with n variables in time $2^{(1-\varepsilon)n}$.

Turns out that even refuting the powerful NSETH conjecture would still result in super-linear circuit lower bounds.

Theorem 2.3.3 ([26, 76])**.** *If NSETH is False, then there exists an explicit function in* E^{NP} *that cannot be computed by series-parallel circuits of size* $\omega(n)$*.*

A classical tool used in studies of algorithms for k-SAT is the Sparsification Lemma.

Theorem 2.3.4 (Sparsification Lemma [70])**.** *For every* $k \geq 3$ *and* $\lambda > 0$ *there exists a constant* $c = c(k, \lambda)$ *such that every* k*-SAT formula* φ *with* n *variables can be expressed as* $\varphi = \vee_{i=1}^{r} \psi_i$ *where* $r \leq 2^{\lambda n}$ *and each* ψ_i *is a* k*-SAT formula with at most* cn *clauses. Moreover, all* ψ_i *can be computed in* $2^{\lambda n}$*-time.*

Now we formally define the notions of fine-grained reductions and SETH-hardness. We start with the definition of fine-grained reductions from [122, Definition 6] with a small modification that specifies the dependence of δ on ε.

Definition 2.3.5 (Fine-grained reductions)**.** Let P, Q be problems, $p, q\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be non-decreasing functions and $\delta\colon \mathbb{R}_{>0} \to \mathbb{R}_{>0}$. We say that $(P, p(n))$ δ-*fine-grained reduces* to $(Q, q(n))$ and write $(P, p(n)) \leq_\delta (Q, q(n))$, if for every $\varepsilon > 0$ and $\delta = \delta(\varepsilon) > 0$, there exists an algorithm $\mathcal{A}$ for P with oracle access to Q, a constant d, a function $t(n)\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, such that on any instance of P of size n, the algorithm $\mathcal{A}$

- runs in time at most $d(p(n))^{1-\delta}$;
- produces at most $t(n)$ instances of Q adaptively: every instance depends on the previously produced instances as well as the answers of the oracle Q;
- the sizes n_i of the produced instances satisfy the inequality

$$\sum_{i=1}^{t(n)} q(n_i)^{1-\varepsilon} \leq d(p(n))^{1-\delta}\,.$$

It is easy to see that if $(P, p(n)) \leq_\delta (Q, q(n))$, then for every $\varepsilon > 0$, an algorithm for Q running in time $O(q(n)^{1-\varepsilon})$ implies an algorithm for P running in time $O(p(n)^{1-\delta})$. Equipped with this definition, we are ready to define SETH-hardness of a problem. SETH-hardness of a problem is a sequence of fine-grained reductions from k-SAT for every value of k.

Definition 2.3.6 (SETH-hardness)**.** Let $p\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a non-decreasing function. We say that a problem P is $p(n)$-*SETH-hard* if there exists a function $\delta\colon \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ and for every $k \in \mathbb{N}$,

$$(k\text{-SAT}, 2^n) \leq_\delta (P, p(n))\,.$$

It is now easy to see that if a problem P is $p(n)$-SETH-hard, then any algorithm solving P in time $p(n)^{(1-\varepsilon)}$ implies an algorithm solving k-SAT in time $2^{(1-\delta(\varepsilon))n}$ for all k, thus, breaking SETH.

Chapter 3

Range Avoidance

The probabilistic method has been a very powerful tool used to show the existence of many combinatorial objects. For example, a simple counting argument by Shannon [116] tells us that most functions (encoded by their truth table) cannot be computed by small circuits. The argument follows since there are 2^{2^n} different truth tables for functions on n variables, while there are at most $2^{O(s \log s)}$ circuits of any given size s. As a result, for any $s < 2^n/10n$ the fraction of functions that can be computed by circuits of size s, is negligible. In other words, a random function cannot be computed by circuits of size $2^n/10n$ with high probability. So if most functions are hard, what is the complexity of finding such a hard truth table?

Open Problem 3.0.1 (ϵ-Hard Truth Tables)**.** *Input:* 1^N*, Output a truth table* T *of length* N *such that no circuits of size* N^ε *can decide* T*.*

While there is a very simple randomized algorithm (just sample a uniformly random string) to solve Open Problem 3.0.1, we will care about quantifying the complexity of a deterministic procedure that finds such a combinatorial object. We call this restriction, an explicit construction problem. A problem has an explicit construction, if given a unary string 1^N as input, there is a deterministic procedure running in polynomial time that outputs a desired combinatorial object of length N. In many cases we will relax the notion of polynomial time, and allow the procedure to run even in E^{NP}.

The notion of explicit constructions is well studied and motivated [87, 115]. As a concrete application, a polynomial time algorithm for Open Problem 3.0.1 would yield a PRG [104] that fully derandomizes BPP. In fact, one can use the explicit construction template to understand the complexity of finding hard truth tables, but also a host of combinatorial objects like rigid matrices, two source extractors, Ramsey graphs, high Kolmogorov strings and linear codes just to name a few.

To encapsulate explicit constructions, and give a general framework for explicit constructions [82] introduced the problem of Range Avoidance (Avoid) which we define below:

Problem: AVOID

Input: A polynomial size circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$.

Output: Find an element in $\{0,1\}^{n+1}$ not in the range of the circuit C.

Avoid is a total problem, since it always has a solution via the dual pigeonhole principle. Moreover, Avoid has an abundance of solutions, since at least half the elements in the co-domain are not in the range of C. A solution y, to Avoid can be verified with an NP oracle via the following coNP predicate: $\forall x : C(x) \neq y$. Combined with the abundance, this immediately, yields a randomized algorithm in $\mathsf{ZPP}^{\mathsf{NP}}$ that solves Avoid by just sampling random strings and using the NP oracle to check if it is a valid solution.

The intuition between connecting Avoid and constructing combinatorial objects whose existence follows from the probabilistic method is as follows: there is a succinct way to encode all "easy" objects (objects without combinatorial property) in the input space of a polynomial size circuit C, that acts as a decoder for the object. A solution to Avoid, then yields a "hard" object (with the combinatorial property we cared about)

since all objects not having this property were by construction mapped somewhere in the range of C.

For a concrete example, let us focus on the simple problem of finding a truth table which cannot be computed by circuits of size n^2. Let C be the circuit that takes as input an encoding of a circuit and outputs its corresponding truth table by evaluating on all 2^n inputs. Any circuit of size s, can be encoded into $O(s \log s)$ bits so all circuits of size n^2 can be encoded by less than n^3 bits. So if $\mathsf{C} : \{0,1\}^{n^3} \to 2^n$, is a circuit who takes as input a circuit description and outputs a truth-table, then AVOID(C) is a truth table that cannot be computed by circuits of size n^2, since by construction any truth table that admitted circuits of size less than n^2 would have been mapped into the range of C. Thus, a natural way to view an algorithm for AVOID is a form of constructive diagonalization against the objects that do not possess the desired combinatorial property.

However to get explicit constructions of these combinatorial objects, it is not sufficient to just have an oracle to AVOID, we also require that our oracle isolate a single solution for a given instance (we call this problem single-value AVOID). Consider again for instance the task of constructing a language that cannot be computed by circuits of size n^2. If one tries to define a language based on the output of the hard truth table generated from an AVOID oracle that is not single-valued, then the language would not be well-defined since different hard truth tables will define different languages.

We now show the power of single-value AVOID as a tool for capturing most explicit constructions via the following theorem that strongly motivates the design of polynomial time algorithms for Single-Value AVOID.

Theorem ([62, 86])**.** *If Single-Value AVOID* $\in \mathfrak{T}$ *for some complexity class* $\mathfrak{T}$,[1] *then we can output the explicit construction of the following objects in* $\mathfrak{T}$*: Ramsey Graphs,*

[1]We only consider classes $\mathfrak{T} \supseteq \mathsf{P}$.

Two Source Extractors, Rigid Matrices, Linear Codes, Hard Truth Tables for any Fixed Polynomial Sized Circuit, $\mathsf{K}^{\mathrm{poly}}$*-random strings.*

It appears, the framework of AVOID while being very powerful for explicit constructions, also encodes hard problems. As a result, under reasonable cryptographic assumptions we do not expect either a polynomial time algorithm [67] or even a non-deterministic algorithm [43] for Single-Value AVOID. This leaves the possibility of designing a P^{NP} algorithm for AVOID, which would still be very interesting and resolve a host of long-standing open problems. In fact, [86] shows the surprising equivalence between designing P^{NP} algorithms for AVOID, and the existence of a language in E^{NP} with circuit complexity $2^{\Omega(n)}$. This reduces the problem of proving lower bounds, to an algorithm design question, something researchers have found more success with. Moreover, this approach could yield lower bounds exponentially better than our current lower bound of $\sim 3.1n$ [95].

Open Problem 3.0.2. *Is there a* P^{NP} *algorithm for AVOID?*

In Section 3.2 we will see that even if we restrict our decoder in the AVOID instance to a very weak model of computation $\mathfrak{C}$, algorithms for $\mathfrak{C}$-AVOID will still imply breakthrough explicit constructions. In Section 3.3 we will design new algorithms for AVOID, and in Section 3.4 we will apply known upper bounds on AVOID to get new unconditional circuit lower bounds.

This chapter is based on a series of joint papers co-authored with Alexander Golovnev, Surendra Ghentiyala, Zeyong Li, Satyajeet Nagargoje, and Sidhant Saraogi [51, 52, 54]. Much of the technical sections in this chapter is taken verbatim from these works.

3.1 Avoid Preliminaries

Let $L \subseteq \{0,1\}^*$ be a language. For $n \geq 1$ we define the *n-th slice of* L, $L|_n := L \cap \{0,1\}^n$ as all the strings in L of length n. The characteristic string of $L|_n$, denoted by $\mathcal{X}_{L|_n}$, is the binary string of length 2^n which represents the truth table defined by $L|_n$.

3.1.1 The Total Function Hierarchy

We formally define a search problem and problems in the total function polynomial hierarchy, i.e. $\mathsf{TF\Sigma_i^P}$ (the total function analog of the polynomial hierarchy [13, Chapter 5]).

Definition 3.1.1. A search problem is a binary relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ where we say that a y is a solution to x if and only if $(x,y) \in \mathcal{R}$.

Definition 3.1.2 ([82, Definition 23])**.** A relation $\mathcal{R}$ is in $\mathsf{TF\Sigma_i^P}$ if the following conditions hold:

1. $\mathcal{R}$ is total: for all $x \in \{0,1\}^*$, there exists $y \in \{0,1\}^*$ such that $(x,y) \in \mathcal{R}$.
2. $\mathcal{R}$ is polynomial: For all $(x,y) \in \mathcal{R}$, $|y| \leq \text{poly}(|x|)$.
3. There exists a polynomial time Turing machine M and polynomials $p_1(n), \ldots, p_{i-1}(n)$ such that

   $(x,y) \in R \iff \forall z_1 \in \{0,1\}^{p_1(|x|)} \exists z_2 \in \{0,1\}^{p_2(|x|)} \ldots V(x,y,z_1,\ldots,z_{i-1})$ accepts.

Definition 3.1.3. A relation $\mathcal{R}$ is in $\mathsf{TFU\Sigma_i^P}$ if it is in $\mathsf{TF\Sigma_i^P}$ and for all $x \in \{0,1\}^*$, there exists exactly one y such that $(x,y) \in \mathcal{R}$.

$\mathsf{TF\Sigma}_1^{\mathsf{P}}$ is also referred to as TFNP . Informally, TFNP is the set of search problems where a solution always exists and is efficiently verifiable (is in FNP, the search analogue of NP). $\mathsf{TF\Sigma}_{\mathsf{i}}^{\mathsf{P}}$ is a generalization of this notion higher in the polynomial hierarchy, where the verifier need not be polynomial time.

With the notion of search problem pinned down, one can begin to talk about reductions between search problems. The norm when reducing between search problems is to restrict oneself to many-to-one reductions. The following definition formalizes the intuitive idea that a reduction from $\mathcal{R}$ to $\mathcal{Q}$ should take as input an instance x of $\mathcal{R}$, transform it to an instance $f(x)$ of $\mathcal{Q}$, get back an answer y to that instance, and transform that to $g(y)$, an answer for instance x of $\mathcal{R}$.

Definition 3.1.4. Let $\mathcal{R}, \mathcal{Q}$ be two search problems. A many-to-one reduction from $\mathcal{R}$ to $\mathcal{Q}$ is defined as two polynomial time computable functions f, g such that for all $x \in \pi_1(\mathcal{R}), y \in \{0,1\}^*$, the following holds,

$$(x, g(y)) \in \mathcal{R} \iff (f(x), y) \in \mathcal{Q}.$$

TFNP Sub-problems. Finally, we will make use of the following TFNP subclasses.

Definition 3.1.5 (PLS and Sink-of-DAG)**.** PLS is defined as all problems which are many-to-one reducible to Sink-of-DAG, which is defined as follows. Given poly(n) size circuits $S : [2^n] \to [2^n]$ (successor circuit) and $V : [2^n] \to [2^n]$ (value circuit), find v such that $S(v) \neq v$ and either $S(S(v)) = S(v)$ or $V(S(v)) \leq V(v)$.

Definition 3.1.6 (Unique-End-of-Potential-Line)**.** UEOPL is defined as follows. Given two poly(n) size circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, such that $P(0^n) = 0^n \neq S(0^n)$, and a value circuit $V : \{0,1\}^n \to \{0, 1, \ldots, 2^{m-1}\}$ where $V(0^n) = 0$. Find one of the following:

- A point $x \in \{0,1\}^n$ such that $P(S(x)) \neq x$.

- A point $x \in \{0,1\}^n$ such that $x \neq S(x), P(S(X)) = x$ and $V(S(x)) - V(x) \leq 0$.
- A point $x \in \{0,1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.
- Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < S(x)$.

UEOPL is defined as all problems which are many-to-one reducible to Unique-End-of-Potential-Line.

3.1.2 The Symetric Hierarchy

Symmetric polynomial time, denoted by $\mathsf{S_2P}$, was introduced independently by Canetti [25], and Russell and Sundaram [113]. Intuitively speaking, this class captures the interaction between an efficient (polynomial-time) verifier V and two all-powerful provers: the "YES"-prover Y and the "NO"-prover Z, exhibiting the following behaviour:

- If x is a yes-instance, then the "YES"-prover Y can send an irrefutable proof/certificate y to V that will make V *accept*, **regardless** of the communication from Z.
- Likewise, if x is a no-instance, then the "NO"-prover can send an irrefutable proof/certificate proof z to V that will make V *reject*, **regardless** of the communication from Y.

Definition 3.1.7 (Symmetric Time)**.** We say that a language $L \in \mathsf{S_2P}$, if there exists a polynomial $p(n)$, such that the predicate $P(x, y, z)$ that takes $x \in \{0,1\}^n$ and $y, z \in \{0,1\}^{p(n)}$ as input, satisfying that:

1. If $x \in L$, then there exists a y such that for all z, $P(x, y, z) = 1$.

2. If $x \notin L$, then there exists a z such that for all y, $P(x, y, z) = 0$.

We stress that in both cases the irrefutable certificates can depend on x itself. A seminal result of [23] provides the best known upper bound $\mathsf{S_2P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. At the same time, $\mathsf{S_2P}$ appears to be a very powerful class as it contains MA and P^{NP}.

The corresponding *oblivious* version of $\mathsf{S_2P}$ is obtained by stipulating that the for every $n \in \mathbb{N}$ there exists a "common" witness w_n for **all** the "respective" inputs of length n. Thus, in a similar manner, one can define the class $\mathsf{O_2P}$ — the oblivious version of $\mathsf{S_2P}$, that is referred to as "oblivious symmetric polynomial time" in the literature. $\mathsf{O_2P}$ has the additional requirement that for every $n \in \mathbb{N}$ there exist an irrefutable yes-certificate y^* and an irrefutable no-certificate z^* for all the yes-instances and the no-instances of length n, respectively.

Definition 3.1.8 (Oblivious Symmetric Time)**.** We say that a language $L \in \mathsf{O_2P}$, if there exists a polynomial $p(n)$ and predicate $P(x, y, z)$, such that for every $n \in \mathbb{N}$, there exist $\mathbf{y}^*$ and $\mathbf{z}^*$ of length $O(p(n))$ satisfying the following for every input $x \in \{0,1\}^n$:

1. If $x \in L$, then for all z, $P(x, \mathbf{y}^*, z) = 1$.

2. If $x \notin L$, then for all y, $P(x, y, \mathbf{z}^*) = 0$.

While the oblivious classes seem to be more restricted than their non-oblivious counterparts ($\mathsf{O_2P} \subseteq \mathsf{S_2P}$), proving any non-trivial upper bounds could still be challenging. In the case of $\mathsf{O_2P}$, since one can hard code the irrefutable yes and no witnesses as advice in a circuit, we have that $\mathsf{O_2P} \subseteq \mathsf{P/Poly}$. In terms of lower bounds, the best known containment of a non-oblivious class is $\mathsf{BPP} \subseteq \mathsf{O_2P}$ [27].

3.2 Range Avoidance on Restricted Circuit Classes

One of the reasons why solving Avoid appears to be a hard problem, is that we allow any polynomial size circuit to be a "decoder" mapping the inputs into the co-domain. Can we learn anything insightful if we only allow our circuit to come from a very restricted class $\mathfrak{C}$. In this section we will consider exactly this question, in the case of local circuits.

Definition 3.2.1 ($\mathfrak{C}$-Avoid)**.** Given a polynomial sized circuit $C : \{0,1\}^n \to \{0,1\}^m$ where $m > n$, and $C \in \mathfrak{C}$ find an element y not in the range of C.

Let NC^1 be the class of boolean circuits of log-depth, and let NC^0_k be the class of circuits where each output bit depends on at most k input bits. The study of $\mathfrak{C}$-Avoid was first initiated in [109] who showed that even solving the much easier NC^0_4-Avoid would imply an algorithm for NC^1-Avoid. Subsequently, [62] improved the reductions for most explicit constructions to Avoid by showing that there exists a NC^1 decoder that computes the same function. Applying the framework from [109] this showed that even designing algorithms for the very weak problem of NC^0_4-Avoid would yield breakthrough explicit constructions.

Designing algorithms for $\mathfrak{C}$-Avoid appears to be an important intermediate step in understanding the complexity of Avoid. [41] design P^{NP} algorithms for ACC^0-Avoid and are able to recover the best known lower bounds against ACC^0 circuits. In the case of NC^0_k-Avoid, [62] give a polynomial time algorithm for NC^0_2-Avoid for any stretch $m \geq n + 1$. For the more generic case of NC^k_0-Avoid, they give a deterministic algorithm that uses an NP oracle and is able to find a solution for any stretch $m \geq n^{k-1}$. This leaves us with the following natural question asked first in [62].

Open Problem 3.2.2 ([62]). *Can we reduce explicit construction problems to solving* NC^0_3-*Avoid?*

We answer Open Problem 3.2.2 by showing that we can find an explicit construction of a rigid matrix that is rigid enough to carry Valiant's program of proving super linear circuit lower bounds.

Theorem 3.2.3. *An* FP^2 *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) algorithm for* NC^0_3-*Avoid with stretch* $m = n + O(n^{2/3})$ *implies an explicit construction of a linear map in* FP *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) that cannot be computed by linear* NC^1 *circuits of size* $o(n \log\log(n))$.

We prove Theorem 3.2.3 by constructing a special NC^3_0 circuit C with the property that given any element outside the range of C, one can compute a rigid matrix in polynomial time.

To build C, we first design a degree-2 decoder $\mathfrak{D}$ with $\ll n^2$ input bits and n^2 output bits, such that each output bit is given by a degree-2 polynomial in the input bits. The decoder $\mathfrak{D}$ maps its input space onto the set of all non-rigid matrices. Consequently, any matrix outside the range of $\mathfrak{D}$ must be rigid.

We then apply the perfect randomized encoding scheme due to [12] to transform $\mathfrak{D}$ from a degree-2 polynomial map into an NC^3_0 circuit C. Moreover, as a property of the perfect randomized encodings we are guaranteed that an element outside the Range of C can be transformed in polynomial time into an element outside the Range of $\mathfrak{D}$.

Lemma 3.2.4 ([12]). *Let* $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ *be a multi-output function where every output computes a sum of* k *monomials of degree* 2*. Then there exists a function* $\widehat{f}\colon \{0,1\}^{n+(2k-1)m} \to \{0,1\}^{2km}$ *computed by an* NC^0_3 *circuit and a polynomial time*

[2] FP is the functional version of the decisional class P.

algorithm $\mathrm{Dec}\colon \{0,1\}^{2km} \to \{0,1\}^m$ *such that for all* x, y, *if* $\mathrm{Dec}(y) = f(x)$, *there exists an* $r \in \{0,1\}^{(2k-1)m}$ *such that* $\widehat{f}(x,r) = y$.

To construct our degree-2 map for non-rigid matrices, we rely on the following lemma, which provides a degree-2 polynomial map whose range contains all sparse vectors.

Lemma 3.2.5 ([51, Lemma 3.1])**.** *For every polynomial-time computable* $s := s(n) < \frac{\sqrt{n}}{2}$, *there exists a polynomial-time computable function* $f\colon \mathbb{F}_2^{2s\sqrt{n}} \to \mathbb{F}_2^n$ *whose range contains all vectors of sparsity at most* s, *and each output of* f *is a degree-2 polynomial.*

Equipped with Lemma 3.2.5 we can now prove Theorem 3.2.3.

Proof of Theorem 3.2.3. Let $r = n^\delta/10$ and $s = n^{\delta-1/2}/10$. First, we reduce (in deterministic polynomial time) the problem of finding an (r,s)-rigid matrix to solving degree-2-Avoid for a function $g\colon \mathbb{F}_2^{4rn} \to \mathbb{F}_2^{n^2}$.

Suppose $M \in \mathbb{F}_2^{n\times n}$ is not (r,s)-rigid. Then, M can be written as a sum $M = Q+S$, where $\mathrm{rank}(Q) \le r$ and S is s-row sparse. Furthermore, $Q = L \cdot R$ for some matrices $L, R^T \in \mathbb{F}_2^{n\times r}$.

We view the input of g as $2rn$ entries of the matrices L and R, and $2sn^{3/2}$ inputs of n copies of the degree-2 function f from Lemma 3.2.5 needed to encode the entries of n sparse rows of S. Then the function g simply outputs all n^2 entries of $M = L \cdot R + S$. Note that each output of g computes a dot-product of a row of L and a column of R, and adds a degree-2 output of f. Therefore, we constructed a degree-2 function g whose range contains all non-rigid matrices. A solution to degree-2-Avoid on input g would therefore give an (r,s)-rigid matrix. The number of inputs of g is $n' = 2rn + 2sn^{3/2} = 4rn = 2n^{1+\delta}/5$, and the stretch of the function g is at least $m'(n') \ge 2(n')^{2/(1+\delta)}$. This concludes the proof of the first part of the theorem.

For the second part, we use the polynomial-time reduction from degree-2-Avoid to NC^0_3-Avoid from Lemma 3.2.4. By the construction above, we have a degree-2 function $g\colon \mathbb{F}_2^{4rn} \to \mathbb{F}_2^{n^2}$ where each output bit is the sum of at most $t = r + s \leq 2r$ degree-2 monomials. We apply Lemma 3.2.4 to reduce Avoid for g to Avoid for an NC^0_3 function $\widehat{g}\colon \{0,1\}^{\widehat{n}} \to \{0,1\}^{\widehat{m}}$, where $\widehat{n} = n' + (2t-1)n^2$ and $\widehat{m} = 2tn^2$. This yields a stretch of $\widehat{m}(\widehat{n}) = \widehat{n} + O(\widehat{n}^{2/(2+\delta)})$ for the function $\widehat{g}$. Therefore, an algorithm for NC^0_3-Avoid for stretch $\widehat{m}(n)$ yields an (r,s)-rigid matrix.

Concretely, setting $\delta = 1$, we get that an algorithm for NC^3_0-Avoid would give an explicit construction of a matrix that is $(n/10, \sqrt{n}/10)$-row-rigid. This is sufficient for proving super-linear circuit lower bounds via Valiant's program Theorem 2.2.4. □

3.3 New Algorithms for Avoid

In the previous section, we showed that even algorithms for the restricted case of NC^0_3-Avoid would yield breakthrough explicit constructions. Here, we instead give a new upper bound on the complexity of Avoid within the Total Function Polynomial Hierarchy (TFPH).

As discussed above, Avoid is total by the dual pigeonhole principle, and a candidate solution y can be verified using the following coNP predicate:

$$\forall x : C(x) \neq y$$

This places Avoid in $\mathsf{TF\Sigma}^{\mathsf{P}}_2$. Our new containment strengthens this classification by showing that Avoid $\in \mathsf{UEOPL}^{\mathsf{NP}}$, a significantly smaller subclass of $\mathsf{TF\Sigma}^{\mathsf{P}}_2$ (see Section 3.1.1 for defintions of relevant total function classes.).

Our algorithm for Avoid follows as a corollary of a more general phenomenon: total problems that are downward self-reducible admit improved upper bounds. This

phenomenon was observed for problems in TFNP by [65], who showed that any downward self-reducible problem in TFNP in fact lies in PLS, a known subclass of TFNP. We show that this result relativizes. As a consequence, we obtain a general meta-theorem characterizing the complexity of downward self-reducible total search problems in the polynomial hierarchy.

Theorem 3.3.1 ([54, Theorem 1])**.** *Let $\mathcal{R}$ be a search problem in $\mathsf{F}\Sigma_i^{\mathsf{P}}$ which is downward self reducible, then there is a reduction from $\mathcal{R}$ to $\mathsf{PLS}^{\Sigma_{i-1}^{\mathsf{P}}}$. Furthermore, if $\mathcal{R}$ has unique solutions, then there is a reduction from $\mathcal{R}$ to $\mathsf{UEOPL}^{\Sigma_{i-1}^{\mathsf{P}}}$.*

At first glance, it is not clear why Theorem 3.3.1 would be helpful for Avoid. While Avoid is in $\mathsf{TF}\Sigma_2^{\mathsf{P}}$, it does not appear to have any structure that makes it amenable to being downward self-reducible. To prove our result, we use an adjacent problem called the Linear Ordering Principle for which it was shown Avoid was many-one reducible to in polynomial time.

Problem: Linear Ordering Principle (LOP)

Input: A polynomial sized circuit $C : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, encoding an order $\prec$.

Output:

1. Find a witness that $\prec$ does not define a total order i.e. $x, y, z \in \{0,1\}^n$ such that one of the following holds: (i) $x \prec x$, (ii) $x \neq y$, $x \not\prec y$ and $y \not\prec x$ or (iii) $x \prec y \prec z$ and $x \not\prec z$.

2. Find the minimal element of the total order defined by $\prec$.

Theorem 3.3.2 ([88, Theorem 1])**.** *Avoid is polynomial time reducible to LOP.*

By definition, LOP is a total problem. Verifying a witness of the violation of a total order can be done in polynomial time. To verify that a solution y is actually the minimal element, we can use an NP oracle since it corresponds to checking the following coNP predicate $\forall x : \prec\ (y, x) = 1$. As a consequence we have that LOP is in $\mathsf{TF}\Sigma_2^{\mathsf{P}}$, and LOP has essentially unique solutions.

We now show that LOP is downward self reducible.

Theorem 3.3.3. *LOP is downward self reducible with access to an* NP *oracle.*

Proof. Let $\prec$: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be our LOP instance. We define $\mu(\prec) = n$. Clearly, $\mu(\prec) \leq \mid \prec \mid$. If $\prec$ is not a total order, we can find the lexicographically smallest witness violating the total order using an NP oracle. We can ensure that the three types of violations have some lexicographic preference over each other.

Otherwise, $\prec$ must be a total order. For $i \in \{0,1\}$, define $\prec_i : \{0,1\}^{n-1} \times \{0,1\}^{n-1} \to \{0,1\}$ such that $\prec_i (x, y) = \prec (i||x, i||y)$. We query the oracle on $\prec_0$ and $\prec_1$, each has a unique solution a_0 and a_1 respectively. We output $\min(a_0, a_1)$ with respect to $\prec$.

$\prec_0$ and $\prec_1$ are total since $\prec$ is total. Therefore, $a_i \prec x$ for all $x \in i||\{0,1\}^{n-1}$. Therefore $\min(a_0, a_1) \prec x$ for all $x \in \{0,1\}^n$, as required. □

As a corollary, combining Theorems 3.3.1 to 3.3.3, we get the following.

Corollary 3.3.4. *Avoid* $\in \mathsf{UEOPL}^{\mathsf{NP}}$.

We now compare the new containment in Corollary 3.3.4 to previous results. [40] and [96] show that Avoid is in $\mathsf{FS_2P}$. In [88] they define $\mathsf{L_2P}$ as a new syntactic subclass of $\mathsf{S_2P}$ and show that $\mathsf{MA} \subseteq \mathsf{L_2P}$. Moreover, they show that a language $L \in \mathsf{L_2P}$ is equivalent to L being polynomial time many-one reducible to LOP. Thus, placing LOP in a class of search problems also implies that $\mathsf{L_2P}$ reduces to this class. While

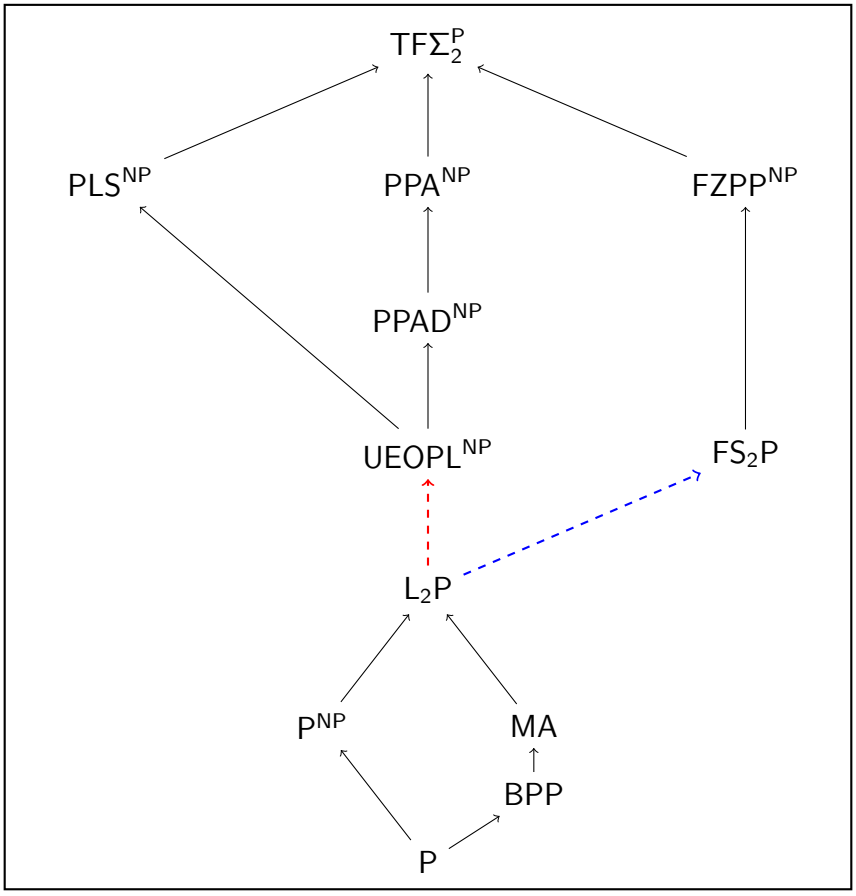

Figure 3.1 A Fine-Grained $\mathsf{TF\Sigma_2^P}$ Version of the Sipser-Gács-Lautemann Theorem [91, 118]. Solid arrows denote inclusions and dotted arrows denote decision to search reductions. The inclusion $\mathsf{FS_2P} \subseteq \mathsf{FZPP^{NP}}$ is due to [23]. $\mathsf{L_2P}$ which is a class of decision problems was shown to be equivalent to class of problems that are polynomially many-one reducible to LOP [88]. The blue dotted arrow indicates the containment of LOP $\in \mathsf{FS_2P}$ as shown in [88]. The red dotted arrow indicates our new containment of LOP $\in \mathsf{UEOPL^{NP}}$.

[88] places LOP in the class $\mathsf{FS_2P}$, our result places both LOP and AVOID in the class $\mathsf{UEOPL}^{\mathsf{NP}}$. In fact, before Corollary 3.3.4 it was open if AVOID or LOP was even in $\mathsf{PLS}^{\mathsf{NP}}$ or $\mathsf{PPA}^{\mathsf{NP}}$. A diagram of our new containment is given in Section 3.3.

We now highlight why our new upper bound of LOP and AVOID is interesting.

1. Since AVOID is in $\mathsf{TF\Sigma_2^P}$, it is interesting to see what is the smallest standard TFNP sub-class with access to an NP oracle that can solve AVOID. UEOPL is one of the smallest TFNP sub-classes, so getting explicit constructions in $\mathsf{UEOPL}^{\mathsf{NP}}$ would bring us a step closer to the goal of providing explicit constructions in P^{NP}. Moreover, a sub-exponential simulation of UEOPL could yield new approaches to getting an E^{NP} construction of rigid matrices and hard truth tables for circuits of size cn for some fixed constant $c > 0$, another approach for resolving Open Problem 2.

2. The original motivation when defining $\mathsf{S_2P}$ by [113] and [25] was to find the smallest class within the polynomial hierarchy that captures probabilistic classes like BPP and MA. Such a class, inherently captures an abundance of solutions (most random strings are good). And while, AVOID seems to share this property of abundance of solutions, the results of [88] indicate that the algorithms for AVOID [40, 96] isolate a single solution and implicitly solve the more restrictive LOP problem, which has a unique solution. In this regard, it seems that the class $\mathsf{UEOPL}^{\mathsf{NP}}$ which doesn't correspond to any notion of abundance, and has a unique solution captures the behavior of LOP better than $\mathsf{FS_2P}$.

3. We do not understand how the complexity classes $\mathsf{UEOPL}^{\mathsf{NP}}$ and $\mathsf{FS_2P}$ relate to each other. Under very strong hardness assumptions for derandomization [83] we have that $\mathsf{FS_2P} \subseteq \mathsf{FZPP}^{\mathsf{NP}} \subseteq \mathsf{FP}^{\mathsf{NP}} \subseteq \mathsf{UEOPL}^{\mathsf{NP}}$. However, unconditionally they seem incomparable.

3.4 Application of Range Avoidance for Unconditional Circuit Lower Bounds

In this section we will see an application of range avoidance framework to prove new "fixed-polynomial" circuit lower bounds. That is, the goal is to show that for every $k \in \mathbb{N}$ there is a language L_k (that may depend on k) which cannot be computed by circuits of size n^k. The previous smallest complexity class for which we could prove such fixed-polynomial lower bounds was $\mathsf{S_2P}$ [23]. We will build on this and prove a fixed-polynomial lower bound for the smaller oblivious counter part of $\mathsf{S_2P}$ called $\mathsf{O_2P}$.

We first give a short overview of the Karp-Lipton framework for proving fixed polynomial circuit lower bounds. The first lower bound was obtained by Kannan [79] via diagonalization. In particular, it was shown that for every $k \in \mathbb{N}$ there exists a language $L_k \in \mathsf{\Sigma_4P}$ that cannot be computed by circuits of size n^k. This result was then improved to $\mathsf{\Sigma_2P}$. The key idea behind this and, in fact, the vast majority of subsequent improvements is a 'win-win' argument that relies on the *Karp-Lipton* collapse theorem [80]: if NP has polynomial-size circuits (i.e $\mathsf{NP} \subseteq \mathsf{P/Poly}$) then the (whole) polynomial hierarchy collapses to $\mathsf{\Sigma_2P}$. More specifically, the argument proceeds by a two-pronged approach:

- Suppose $\mathsf{NP} \not\subseteq \mathsf{P/Poly}$. Then the claim follows as $\mathsf{NP} \subseteq \mathsf{\Sigma_2P}$.
- On the other hand, suppose $\mathsf{NP} \subseteq \mathsf{P/Poly}$. Then by the Karp-Lipton theorem: $\mathsf{\Sigma_4P} = \mathsf{\Sigma_2P}$ and in particular for all $k \in \mathbb{N} : L_k \in \mathsf{\Sigma_2P}$.

Indeed, by deepening the collapse, the result was further improved to $\mathsf{ZPP^{NP}}$ [22, 85], $\mathsf{P^{prMA}}$ [28] and $\mathsf{S_2P}$ [23]. By using different versions of the Karp-Lipton theorem, the result has also been extended to PP [1, 125] and $\mathsf{MA/1}$ [114].

Yet, despite the success of the "win-win" argument, the obtained lower bounds are often non-explicit due to the non-constructive nature of the argument. Different results [24, 114] were required to exhibit explicit "hard" languages in $\Sigma_2\mathsf{P}, \mathsf{PP}$ and $\mathsf{MA}/1$. Nonetheless, the last word about $\mathsf{S}_2\mathsf{P}$ is yet to be said. For instance, we know that there is a language in $\mathsf{S}_2\mathsf{P}$ that requires circuits of size, say, n^2 from such arguments. However, prior to the results of [40, 96] where they give an $\mathsf{S}_2\mathsf{P}$ algorithm for Range Avoidance, one could not prove any super-linear lower bound for **any** explicit language in $\mathsf{S}_2\mathsf{P}$. Another limitation of the "win-win" argument stems from the fact that it only applies to complexity classes which (provably) contain NP. In particular, in [27] it was actually shown that if $\mathsf{NP} \subseteq \mathsf{P/Poly}$ then the polynomial hierarchy collapses all the way to $\mathsf{O}_2\mathsf{P}$! Unfortunately, this result does not immediately imply fixed-polynomial lower bounds for $\mathsf{O}_2\mathsf{P}$[3] as it is unknown and, in fact, *unlikely* that $\mathsf{O}_2\mathsf{P}$ contains NP. Furthermore, such a containment will be "self-defeating". Recall that $\mathsf{O}_2\mathsf{P} \subseteq \mathsf{P/Poly}$. Hence, if $\mathsf{NP} \subseteq \mathsf{O}_2\mathsf{P}$ then $\mathsf{NP} \subseteq \mathsf{P/Poly}$ which in and of itself already implies the collapse of the whole polynomial hierarchy!

In the main result of this section, we extend circuit lower bounds known for $\mathsf{S}_2\mathsf{P}$ to its weaker oblivious counterpart $\mathsf{O}_2\mathsf{P}$. Our result builds on a recent line of work that obtains circuit lower bounds by *deterministically* solving (that is, derandomizing) the Avoid problem [40, 96].

To prove our result, we exploit the fact that the language arising from the Avoid instance used to establish the circuit lower bound is extremely sparse. In Section 3.4.1, we introduce new notions of sparsity and use them to establish connections between the classes $\mathsf{S}_2\mathsf{P}$ and $\mathsf{O}_2\mathsf{P}$. Building on this framework, in Section 3.4.2 we prove arbi-

[3]Indeed, the authors in [27] could only obtain fixed-polynomial lower bounds for $\mathsf{NP}^{\mathsf{O}_2\mathsf{P}}$ which was later subsumed by the results of [114].

trary polynomial circuit lower bounds for $\mathsf{O_2P}$, and also establish a uniform hierarchy theorem for $\mathsf{O_2P}$.

3.4.1 Sparse Extensions

We begin by defining the class of SPARSE languages.

Definition 3.4.1. A language $L \in \mathsf{SPARSE}$ if for all n, $|L \cap \{0,1\}^n| \leq \mathsf{poly}(n)$. Moreover, L is called *uniformly-sparse* if $L \in \mathsf{P} \cap \mathsf{SPARSE}$.

We will use the notation $t(n)$-*uniformly-sparse* if $L \in \mathsf{TIME}[t(n)] \cap \mathsf{SPARSE}$. It is easy to see that $\mathsf{SPARSE} \subseteq \mathsf{P/Poly}$. That is, one can identify the yes-instances efficiently, albeit in non-uniform fashion. The purpose of introducing the *uniform-sparsity* is to be able to identify these inputs efficiently in a uniform fashion. Unfortunately, we cannot expect any such language L to lie even in a modestly hard class as, by definition, $L \in \mathsf{P}$. The purpose of the *uniformly-sparse extensions*, on the other hand, is to bridge this gap. One can observe that unlike the *uniformly-sparse* languages, which are contained in P, languages with uniformly-sparse extensions can even be undecidable! In particular, any unary language has a uniformly-sparse extension in form of 1^*.

Definition 3.4.2. A language L has a *uniformly-sparse extension*, if there exists an L' s.t. :

1. $L \subseteq L'$,
2. L' is *uniformly-sparse*.

From these definitions we make the following structural observation about sparse $\mathsf{S_2P}$ languages.

Theorem 3.4.3. *If $L \in \mathsf{S_2TIME}[t(n)]$ and L has a uniformly-sparse extension, then $L \in \mathsf{O_2TIME}[t(n)^2]$.*

Proof. For any n, let L' be the $t(n)$-*uniformly-sparse extension* of L, and let $\mathcal{F}$ be the $\mathsf{TIME}[t(n)]$ predicate that decides membership in L'. We will now design an $\mathsf{O_2TIME}[t(n)^2]$ verifier V for $L|_n$. Since both L and L' are $t(n)$-SPARSE, we have that for most $x \in \{0,1\}^n$: $L|_n(x) = L'|_n(x) = 0$. V will first use $\mathcal{F}$ to efficiently filter out most non-membership in $L'|_n$, and hence $L|_n$ in $\mathsf{TIME}[t(n)]$. Now V only has to decide membership in L over $t(n)$ many inputs $X = \{x \in \{0,1\}^n : \mathcal{F}(x) = 1\}$. We will use the fact that since $L \in \mathsf{S_2TIME}[t(n)]$, for all $x \in X$, if $x \in L$ there is an irrefutable YES certificate y_x and if $x \notin L$ there is an irrefutable NO certificate z_x and a verifier V^*, running in $\mathsf{TIME}[t(n)]$ s.t.

- if $x \in L$, $\exists y_x$, $\forall z$ s.t. $V^*(x, y_x, z) = 1$
- if $x \notin L$, $\exists z_x$, $\forall y$ s.t. $V^*(x, y, z_x) = 0$

$V(x, Y^*, Z^*)$:

(1) Set output = 1.

(2) If $\mathcal{F}(x) = 0$, return 0.

(3) Parse Y^* to get y_x^*.

(4) For $z_i \in Z^*$, do:

  (a) output = output $\wedge\, V^*(x, y_x^*, z_i)$.

(5) Return output.

Figure 3.2 $\mathsf{O_2TIME}[t(n)^2]$ Verifier for Language in $\mathsf{S_2TIME}[t(n)]$ with $t(n)$-*uniformly-sparse Extension.*

Consider the string Y^* which encodes a table of YES witnesses y_x^* for every input $x \in X$. When $x \in L$ we set $y_x^* = y_x$, and when $x \notin L$ we will set $y_x^* = 0^{t(n)}$. The size of Y^* is $O(t(n)^2)$, since there are at most $t(n)$ entries in the table each of length $t(n) + n$. For every $x \in X \cap \overline{L}$, let z_x be the irrefutable NO-certificate corresponding to x for V^*. We set Z^* to be the concatenation of all such z_x. The size of Z^* is also at most $t(n)^2$.

We now show that Y^* and Z^* will serve as oblivious irrefutable "YES" and "NO" certificates respectively for V. On input (x, Y^*, Z^*), V first parses Y^* to find the corresponding y_x^* in time $\mathsf{TIME}[t(n)^2]$. Then for each $z_i \in Z^*$ we run $V^*(x, y_x^*, z_i)$. If for all z_i, $V^*(x, y_x^*, z_i) = 1$ then V outputs 1, otherwise V will output 0. Since we are making at most $t(n)$ calls that each cost $\mathsf{TIME}[t(n)]$, V runs in $\mathsf{TIME}[t(n)^2]$.

To see correctness, we first analyze the case when $x \in L$, then by construction Y^* includes $y_x^* = y_x$ and V will output 1. On the other hand if $x \notin L$ then there is an irrefutable no-certificate z_x in Z^* so there is no y_i such that $V(x, y_i, z_x) = 1$. Hence V outputs 0. □

Setting $t(n)$ to be an arbitrary polynomial we can make the structural connection between $\mathsf{S_2P}$ and $\mathsf{O_2P}$.

Corollary 3.4.4. *If $L \in \mathsf{S_2P}$, and L has a uniformly-sparse-extension, then $L \in \mathsf{O_2P}$.*

3.4.2 Lower Bounds and Hierarchies

An important object that connects Avoid and circuit lower bound is the truth table generator circuit. That is an element outside the range of $\mathsf{PTT_{n,s}}$ would constitute a prefix of a truth table that cannot be computed by circuits of size s.

Definition 3.4.5 (Prefix Truth Table Generator [52, Definition 2.14])**.** For $n, s \in \mathbb{N}$ where $12 \leq n \leq s \leq 2^n$ and $|E_{n,s}| = 5s \log s < 2^n$, the prefix truth table generator

circuit $\mathsf{PTT}_{n,s} : \{0,1\}^{|E_{n,s}|} \to \{0,1\}^{|E_{n,s}|+1}$ maps a n-input circuit of size s described with $|E_{n,s}|$ bits into the lexicographically first $|E_{n,s}|+1$ entries of its truth table.

It is not hard to see that one can construct the prefix-truth table generator very efficiently.

Lemma 3.4.6 ([52, Lemma 2.15])**.** *The prefix truth table generator circuit* $\mathsf{PTT}_{n,s} : \{0,1\}^{|E_{n,s}|} \to \{0,1\}^{|E_{n,s}|+1}$ *has size* $O(|E_{n,s}|^3)$ *and can be uniformly constructed in time* $O(|E_{n,s}|^3)$*.*

Finally, we state the breakthrough $\mathsf{S_2P}$ algorithms for AVOID.

Theorem 3.4.7 ([40, 96])**.** *There exists a single-valued* $\mathsf{FS_2P}$ *algorithm for* Avoid*. Moreover, on input circuit* $C : \{0,1\}^n \to \{0,1\}^{n+1}$*, the algorithm runs in time* $O(n|C|)$*.*

We can now prove our lower bound for $\mathsf{O_2P}$. At a high level we first construct a language in $\mathsf{S_2P}$, that cannot be computed by circuits of $\mathsf{SIZE}[n^k]$. We will then see that this language admits a uniformly-sparse-extension, and hence must reside in $\mathsf{O_2P}$, completing the argument.

Lemma 3.4.8. *For every* $k \geq 1$*, there exists an explicit language* $L_k \in \mathsf{S_2P}$*, such that* $L_k \not\subseteq \mathsf{SIZE}[n^k]$*.*

Proof. Set $s = n^c \log n$, and construct the circuit $\mathsf{PTT}_{n,s}$ corresponding to the prefix truth table generator (Definition 3.4.5) of size $O(5s^3 \log^3 s)$ via Lemma 3.4.6. Let $f_n \in \{0,1\}^{|E_{n,s}|+1}$ be the canonical solution to $\mathsf{Avoid}(\mathsf{PTT}_{n,s})$ as outputted by the single-valued algorithm from Theorem 3.4.7.

The hard language L_k is defined as follows: for any $n \in \mathbb{Z}$, the characteristic string of $L_k|_n$ is set to be $\mathcal{X}_{L_k|_n} := f_n||0^{2^n-|E_{n,s}|-1}$.

By definition of $\mathsf{PTT}_{n,s}$ and the fact that $f_n \notin \mathsf{Image}(\mathsf{PTT}_{n,s})$, we have that $L_k \notin \mathsf{SIZE}[s]$. Since $s = \text{poly}(n)$, the single-valued algorithm for finding f_n also runs in poly(n). Hence $L_k \in \mathsf{S_2P}$. □

We now show that the language we defined above, is in fact in $\mathsf{O_2P}$.

Theorem 3.4.9. *For every $k \in \mathbb{N}$, there exists a $L_k \in \mathsf{O_2P}$, such that $L_k \notin \mathsf{SIZE}[n^k]$.*

Proof. Let L_k be the $\mathsf{S_2P}$ language defined in the proof of Lemma 3.4.8 with the characteristic string $\mathcal{X}_{L_k|_n} := f_n||0^{2^n-|E_{n,s}|-1}$. We now define the language L'_k whose characteristic string $\mathcal{X}_{L'_k|_n} := 1^{|E_{n,s}|+1}||0^{2^n-|E_{n,s}|-1}$. To see that this L'_k is a *uniformly-sparse extension* of L_k, clearly $L_k \subseteq L'_k$. Moreover membership of $x \in L'_k$ can be decided by checking if the binary value of x is less than or equal to $|E_{n,s}| + 1$ which can be done in linear time. □

Combining Theorem 3.4.9 along with the circuit size hierarchy theorem (Theorem 2.1.4) we are able to get a hiearchy theorem for $\mathsf{O_2TIME}$.

Corollary 3.4.10 ([52, Theorem 3.6])**.** *For any time constructible function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n) \geq n$ and any $\varepsilon > 0$ it holds that: $\mathsf{O_2TIME}[t(n)] \subsetneq \mathsf{O_2TIME}[t(n)^{1+\varepsilon}]$.*

We conclude, by highlighting why the results in this section are interesting:

1. Our lower bound does not follow the framework of "win-win" style Karp-Lipton collapses. As was mentioned above, since already $\mathsf{O_2P} \subseteq \mathsf{P/Poly}$ the pre-requisite for proving the bound via the "win-win" argument will be self-defeating.

2. Our proof is constructive and for every $k \in \mathbb{N}$ we define an explicit language $L_k \in \mathsf{O_2P}$ for which $L_k \not\subseteq \mathsf{SIZE}[n^k]$.

3. $\mathsf{O_2P}$ becomes the smallest uniform complexity class known for which fixed-polynomial lower bounds are known. Moreover, after more than 15 years, this

class coincides again with the deepest known collapse result of the Karp-Lipton Theorem.[4]

4. Our hierarchy theorem for $\mathsf{O}_2\mathsf{P}$ is the first hierarchy theorem of a uniform sematic class known to contain BPP. Previous to this result the only way to prove a hierarchy theorems for a semantic class required one bit of non-uniform advice [100].

[4]Indeed, in the universe of [23] and [27] prior to our work, the smallest class has been $\mathsf{S}_2\mathsf{P}$, while the deepest known collapse was to $\mathsf{O}_2\mathsf{P}$.

Chapter 4

Matrix Multiplication and Verification

The verification of matrix multiplication lies at the frontier of derandomization. In the Matrix Multiplication Verification (MMV) problem, we are given three matrices A, B, and C, and asked to determine whether $C = AB$. Beyond its practical applications,[1] the problem has played a central role in theoretical computer science.

In 1979, Freivalds [49] gave a randomized algorithm running in $O(n^2)$ time for MMV, which is optimal since simply reading the input requires $\Theta(n^2)$ time.[2] In contrast, the best known deterministic approach is the naïve one: explicitly compute AB and compare the result to C. Using the fastest known matrix multiplication algorithms, this takes $O(n^\omega)$ time, where the current best bound is $\omega \leq 2.372$.

Perhaps most intriguingly, we obtain an improved deterministic algorithm running in time $O(n^{\omega-\varepsilon})$ for instances in which AB and C disagree on at most $O(n^{1.99})$ entries.[3] In contrast, when AB and C differ on $\Omega(n^2)$ entries, our algorithm runs in time $O(n^\omega)$, offering no improvement over the naïve approach.

At first glance, this behavior seems counterintuitive. In the regime where the matrices disagree on many entries, there is a simple $O(n)$-time randomized algorithm that just samples a constant number of coordinates and checks whether AB, and

[1] Reed–Solomon fingerprinting can be viewed as a direct application of this algorithmic paradigm, and it has had substantial practical impact, particularly in the design of more efficient cryptographic proof systems [119].

[2] The input consists of three $n \times n$ matrices.

[3] Here $\varepsilon > 0$ is a constant depending on the exponent 1.99.

C agree on these positions. On the other hand, when the matrices disagree on say just one position (and in fact up to $n^{0.64}$ positions) we can deterministically detect each of these discrepancies in the optimal $O(n^2)$ time. Our result can therefore be interpreted as isolating a "hay in a haystack" phenomenon in MMV, the true obstacle to derandomization does not arise when discrepancies are sparse, but rather when disagreements are widespread.

We establish our results by designing an algorithm for a harder problem: given two matrices A and B whose prouct AB has at most $O(n^{1.99})$ non-zero entries, we can compute AB deterministically in time $O(n^{\omega-\varepsilon})$ for some constant $\varepsilon > 0$. Using a reduction from MMV to sparse matrix multiplication due to Künnemann (see [89] and Lemma 4.1.4), we can use this algorithm to obtain improved deterministic algorithms for Matrix Multiplication Verification. As a consequence, our algorithm not only deterministically verifies whether $AB = C$, but in essentially the same running time, it also identifies all entries on which AB and C disagree. One can alternatively view our results as an equivalence between the search and decision versions of MMV in a fine-grained setting.

Finally, we present evidence suggesting that attempts to explain the limitations of progress in matrix multiplication through the lens of the Strong Exponential Time Hypothesis (SETH) may have to overcome significant challenges. Specifically, we show that if matrix multiplication were hard under SETH, this would imply either superlinear circuit lower bounds or the explicit construction of a rigid matrix.

While both superlinear circuit lower bounds and explicit rigid matrices are widely believed to exist, proving either would require techniques far beyond what is currently known. Thus, establishing SETH-hardness for matrix multiplication would not merely highlight the complexity of matrix multiplication itself, but also implicitly constitute major progress on longstanding lower bound questions.

This chapter is based on a couple of joint papers co-authored with Huck Bennett, Alexander Golovnev, and Evelyn Warton [16, 17]. Much of the technical sections in this chapter is taken verbatim from these works.

4.1 Matrix Multiplication Preliminaries

4.1.1 Matrix Multiplication Problems

Definition 4.1.1 (MM_R)**.** The Matrix Multiplication Problem over a ring R (MM_R) is defined as follows. Given matrices $A, B \in R^{n\times n}$ for some $n \in \mathbb{Z}^+$, compute their product AB.

Definition 4.1.2 (Output-Sparse Matrix Multiplication)**.** Let R be a ring, and let $\delta_{\mathrm{in}}, \delta_{\mathrm{out}} \in [0, 2]$. We define the (δ_{out})-*Output-Sparse Matrix Multiplication Problem* over R ((δ_{out})-OSMM_R) as follows. The input consists of $A, B \in R^{n\times n}$ for some $n \in \mathbb{Z}^+$ satisfying the promise that $\|AB\|_0 \leq O(n^{\delta_{\mathrm{out}}})$. The goal is to compute AB.

Definition 4.1.3 (MMV_R^t)**.** The Matrix Multiplication Verification Problem over a ring R with sparsity parameter $\delta \in [0, 2]$ (MMV_R^δ) is defined as follows. Given matrices $A, B, C \in R^{n\times n}$ for some $n \in \mathbb{Z}^+$ such that $\|AB - C\|_0 \leq n^\delta$ as input, decide whether $AB = C$.

We use the following reduction from [89] with the additional observations that it is sparsity-preserving and works over arbitrary rings.

Lemma 4.1.4 ([89, Proposition 3.1])**.** *For all rings R, $n \in \mathbb{Z}^+$, and $0 \leq \delta \leq 2$, there is a reduction from MMV_R^δ on $n \times n$ matrices to δ-OSMM_R^t on $2n \times 2n$ matrices that uses $O(n^2)$ arithmetic operations on R.*

4.1.2 Fast Rectangular MM

We next define rectangular matrix multiplication exponents and the dual matrix multiplication exponent.

Definition 4.1.5 (Rectangular Multiplication Exponents)**.** For constants $\alpha, \beta, \gamma \geq 0$, the *rectangular matrix multiplication exponent* $\omega(\alpha, \beta, \gamma)$ is the infimum over all $\omega' \geq 0$ such that the product of $A \in \mathbb{Z}^{n^\alpha \times n^\beta}$ and $B \in \mathbb{Z}^{n^\beta \times n^\gamma}$ can be computed using $O(n^{\omega'})$ arithmetic operations.

We also define the (square) *matrix multiplication exponent* as $\omega := \omega(1, 1, 1)$.

Definition 4.1.6 (Dual Matrix Multiplication Exponent)**.** The *dual matrix multiplication exponent* $\omega^\perp$ is defined as

$$\omega^\perp := \sup\{\omega' \geq 0 : \omega(1, 1, \omega') = 2\} \ .$$

A beautiful line of work [6, 8, 47, 93, 94, 124] has established strong bounds on multiplication exponents, in particular showing that

$$\omega < 2.371339 \quad [8] \ ,$$
$$\omega^\perp \geq 0.321334 \quad [124] \ .$$

Next we list several properties of rectangular matrix multiplication exponents $\omega(\cdot, \cdot, \cdot)$.

We will make use of the following upper bound on $\omega(\alpha, 1, 1)$.

Theorem 4.1.7 ([92, 99])**.** *Let $\alpha \in [0, 1]$. Then*

$$\omega(\alpha, 1, 1) \leq \begin{cases} 2 & \text{if } 0 \leq \alpha \leq \omega^\perp \ , \\ 2 + \dfrac{\omega - 2}{1 - \omega^\perp} \cdot (\alpha - \omega^\perp) \leq 1.825 + 0.548\alpha & \text{if } \omega^\perp < \alpha \leq 1 \ . \end{cases}$$

We note that the second case of Theorem 4.1.7 implies that if $\omega > 2$ then $\omega(1,1,\beta) < \omega$ for $\beta < 1$.

Finally, we state some properties about rectangular MM, that we will use in our analysis.

Proposition 4.1.8 ([99]). *Let $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \lambda \geq 0$. The following hold:*

1. $\max(\alpha_1 + \alpha_2, \alpha_1 + \alpha_3, \alpha_2 + \alpha_3) \leq \omega(\alpha_1, \alpha_2, \alpha_3) \leq \alpha_1 + \alpha_2 + \alpha_3$.

2. $\omega(\alpha_1 + \beta_1, \alpha_2 + \beta_2, \alpha_3 + \beta_3) \leq \omega(\alpha_1, \alpha_2, \alpha_3) + \omega(\beta_1, \beta_2, \beta_3)$.

4.2 MMV via OSMM

We now present our deterministic algorithm for MMV_R^δ and OSMM. In Section 4.2.1, we formally define compressed sensing schemes and state our main theorem about a concrete such scheme. In Section 4.2.3, we give a reduction from δ-OSMM to compressed sensing and instantiate the reduction to give our roughly $n^{\omega(\delta/2,1,1)}$-time deterministic algorithm for δ-OSMM, proving Theorem 4.

4.2.1 Compressed Sensing

We will use the following formalization of a compressed sensing scheme.

Definition 4.2.1. Let R be a ring and let $m = m(n,t)$ be a non-decreasing, integer-valued function. An *m-compressed sensing scheme* over R is a pair of algorithms $(\mathcal{M}, \mathcal{R})$ that work as follows. On inputs n and t in unary with $1 \leq t \leq n$, $\mathcal{M}$ outputs a matrix $H \in R^{m \times n}$. On input $H\boldsymbol{x}$ for a vector $\boldsymbol{x} \in R^n$ satisfying $\|\boldsymbol{x}\|_0 \leq t$, $\mathcal{R}$ outputs $\boldsymbol{x}$.[4]

[4]In fact, the sparse recovery algorithm $\mathcal{R}$ needs to know the value of n, and often it also needs t and H as well. This is usually handled by having the algorithm $\mathcal{M}$ also compute a string σ as preprocessing that is passed as auxiliary input to $\mathcal{R}$. Without loss of generality,

Here the matrix H output by $\mathcal{M}$ is called a *measurement matrix*, and $\mathcal{R}$ is called a *sparse recovery algorithm*. We will crucially use the following theorem about an explicit compressed sensing scheme from [18] (which we instantiate with a construction of expanders from [61]) in both of our main algorithms. The claim in Theorem 4.2.2 follows implicitly by combining [18, 61].[5]

Theorem 4.2.2 ([18, 61])**.** *Let R be a ring and let $\alpha > 0$ be a constant. There is a deterministic m-compressed sensing scheme $(\mathcal{M}, \mathcal{R})$ over R with $m = m(n,t) = t^{1+\alpha} \cdot \mathrm{poly}(\log n)$. The algorithm $\mathcal{M}$ runs in time $\widetilde{O}(n)$, and the algorithm $\mathcal{R}$ performs $t^{1+\alpha} \cdot \mathrm{poly}(\log n)$ arithmetic operations over R.*

Furthermore, the measurement matrix $H := \mathcal{M}(1^n, 1^t)$ is binary and d-column-sparse for $d = \mathrm{poly}(\log n)$, and therefore can be multiplied by an arbitrary vector $\boldsymbol{x} \in R^n$ using $\|\boldsymbol{x}\|_0 \cdot \mathrm{poly}(\log n)$ addition operations over R.[6]

4.2.2 Algorithmic Overview Using Compressed Sensing

In this section we give an overview of how our algorithms use compressed sensing to compute output-sparse matrix products.

Output-column-sparse MM using compressed sensing. We first consider the simple case where every column $\boldsymbol{c}_i$ of the output matrix $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) := AB$ is promised to be t-sparse. Let H be the measurement matrix from your compressed sensing scheme. We can now compute HAB parenthesized as $(HA)B$ using fast rectangular matrix multiplication, and then recover the columns $\boldsymbol{c}_i$ of $C = AB$ from HC using

we assume that σ encodes H, n, and t, but it could also encode more. So, formally we should write $\mathcal{R}(H\boldsymbol{x}, \sigma)$ for running $\mathcal{R}$ on $H\boldsymbol{x}$, but for conciseness we often abuse notation and simply write $\mathcal{R}(H\boldsymbol{x})$.

[5] For a formal proof combining the two constructions we refer the reader to[17, Appendix].

[6] Assuming that H and $\boldsymbol{x}$ are given in sparse representation by lists of non-zero entries (as in the output of the algorithm $\mathcal{M}$).

$\mathcal{R}$ as $\boldsymbol{c}_i = \mathcal{R}(H\boldsymbol{c}_i)$ for $i = 1, \ldots, n$. If computing H takes $T_{\mathcal{M}}(n)$ time, $\mathcal{R}$ takes $T_{\mathcal{R}}(n)$ time, and $m = O(n^{\beta})$ for some $\beta \in [0, 1]$, then for any constant $\varepsilon > 0$ the overall running time of this algorithm is

$$O(T_{\mathcal{M}}(n) + n^{\omega(\beta,1,1)+\varepsilon} + n \cdot T_{\mathcal{R}}(n)) \ . \tag{4.1}$$

Reducing OSMM to compressed sensing. We now show how to replace the assumption in the above compressed-sensing-based algorithm for MM that every column of $C = AB$ is t-sparse with the "global" assumption that C is t^2-sparse. If $t^2 = O(n^{\delta})$, this is exactly the assumption in δ-OSMM. To do this, we use a two-pass reduction, which roughly speaking first computes a matrix C' that agrees with C on sparse columns and then computes all rows of $C - C'$. This reduction appears formally as Algorithm 1.

Assume without loss of generality that $\mathcal{R}$ always outputs *some* vector of length n even when its input is not t-sparse. More specifically, assume that $\boldsymbol{c}'_i := \mathcal{R}(H\boldsymbol{c}_i)$ is always a vector of length n with the guarantee that $\boldsymbol{c}'_i = \boldsymbol{c}_i$ if $\boldsymbol{c}_i$ is t-sparse. Let $C' := (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_n)$. The key observation is that if $C = AB$ is t^2-sparse, then *every* row of $C - C'$ must be t-sparse. To see this, note that if C is t^2-sparse, then it has fewer than t columns that are *not* t-sparse, and that these are the only (potentially) non-zero columns in $C - C'$. We can then recover the entries of $(C - C')^T$ (and therefore also C, since we know C') by computing $H(C - C')^T$ as $(HB^T)A^T - H(C')^T$ and then running $\mathcal{R}$ on each column of the result.

4.2.3 Deterministic OSMM via a Reduction to Compressed Sensing

In this section, we start by giving our two-pass reduction from OSMM to compressed sensing in Algorithm 1. One may also view this reduction as a meta-algorithm. Making

it into an explicit algorithm requires using an explicit compressed sensing scheme (we note this in the "Subroutine" heading in the specification of Algorithm 1).

We next analyze Algorithm 1.

Theorem 4.2.3. *Let R be a ring, and let $A, B \in R^{n\times n}$ be matrices satisfying $\|AB\|_0 \leq t^2$ for $t \in \mathbb{Z}^+$ with $t^2 \leq O(n^\delta)$ for some $\delta \in [0,2]$. Let $(\mathcal{M}, \mathcal{R})$ be an m-compressed sensing scheme over R with $m = m(n,t) \leq O(t \cdot n^\alpha)$ for some $\alpha > 0$. Then on input A, B, Algorithm 1 outputs the matrix product $C := AB$. Furthermore, if $\mathcal{M}$ runs in time $T_{\mathcal{M}}(n,t)$ and $\mathcal{R}$ runs in time $T_{\mathcal{R}}(n,t)$ then for any constant $\varepsilon > 0$, Algorithm 1 performs*

$$O(T_{\mathcal{M}}(n,t) + n \cdot T_{\mathcal{R}}(n,t) + n^{\omega(\delta/2+\alpha,1,1)+\varepsilon})$$

arithmetic operations over R.

Algorithm 1: Deterministic output-sparse matrix multiplication over a ring R.

Input: Matrices $A, B \in R^{n\times n}$ over a ring R, and $t \in \mathbb{Z}^+$ such that $\|AB\|_0 \leq t^2$.
Subroutine: An m-compressed sensing scheme $(\mathcal{M}, \mathcal{R})$ over R.
Output: The product $C = AB$.

Compute a measurement matrix $H \in R^{m\times n} := \mathcal{M}(1^n, 1^t)$.
Compute $D = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n) := HAB$ using fast rectangular MM.
Compute $D' := (\mathcal{R}(\boldsymbol{d}_1), \ldots, \mathcal{R}(\boldsymbol{d}_n))$. // t-sparse columns of $C = AB$ agree with D'.
Compute $F = (\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n) := (H(AB - D')^T)$ using fast rectangular MM.
Compute $F' := (\mathcal{R}(\boldsymbol{f}_1), \ldots, \mathcal{R}(\boldsymbol{f}_n))$. // $F' = (AB - D')^T$.
return $D' + (F')^T$.

Table 4.1: Deterministic Algorithm for Output-sparse Matrix Multiplication

Proof. We start by proving correctness. Let $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$, $D' = (\boldsymbol{d}'_1, \ldots, \boldsymbol{d}'_n)$, and $F' = (\boldsymbol{f}'_1, \ldots, \boldsymbol{f}'_n)$. We assume without loss of generality that $\mathcal{R}$ outputs *some* vector in R^n on every input $\boldsymbol{y} \in R^m$ (including when $\boldsymbol{y} \neq H\boldsymbol{x}$ for any t-sparse $\boldsymbol{x} \in R^n$).

We note that C has at most t columns $\boldsymbol{c}_i$ that are not t-sparse, since otherwise we would have $\|C\|_0 > t^2$. Furthermore, by the specification of $\mathcal{R}$, $\boldsymbol{d}'_i = \mathcal{R}(H\boldsymbol{c}_i) = \boldsymbol{c}_i$ for each $i \in [n]$ such that $\boldsymbol{c}_i$ is t-sparse. It follows that every row of $C - D'$ is t-sparse, and therefore that every column of $(AB - D')^T$ is t-sparse. From this and again using the specification of $\mathcal{R}$, we get that $F' = (AB - D')^T$. Correctness then follows by noting that $C = AB = D' + (F')^T$.

We next analyze the time complexity of Algorithm 1. It makes one call to $\mathcal{M}$ on input $(1^n, 1^t)$, which takes time at most $T_{\mathcal{M}}(n, t)$, and $2n$ calls to $\mathcal{R}$, which takes time at most $2n \cdot T_{\mathcal{R}}(n, t)$. Furthermore, because $H \in R^{m \times n}$ for $t = O(n^{\delta/2})$ and $m \leq O(t \cdot n^{\alpha}) = O(n^{\delta/2+\alpha})$, HAB and $H(AB - D')^T$ can be computed in time $O(n^{\omega(\delta/2+\alpha,1,1)+\varepsilon})$ for any constant $\varepsilon > 0$ using fast rectangular matrix multiplication. All other operations run in $O(n^2) \leq O(n^{\omega(\delta/2+\alpha,1,1)})$ time. The result follows by summing the running time bounds for each of these components. □

Our main theorem about solving OSMM deterministically follows as a corollary.

Corollary 4.2.4. *Let $\delta \in [0, 2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a deterministic algorithm for solving δ-OSMM$_R$ on $n \times n$ matrices that performs $O(n^{\omega(\delta/2,1,1)+\varepsilon})$ operations over R.*

Proof. We instantiate the compressed sensing scheme subroutine needed in Algorithm 1 with the scheme described in Theorem 4.2.2. Let $\alpha > 0$ be a constant, which we will set later, and let α' be a constant satisfying $0 < \alpha' < \alpha$. From Theorem 4.2.2, we have that there exists a deterministic m-compressed sensing scheme where $m = m(n, t) = t^{1+\alpha'} \cdot \text{poly}(\log n) \leq O(tn^{\alpha})$, with $T_{\mathcal{M}}(n, t) \leq T_{\mathcal{M}}(n, n) \leq \widetilde{O}(n) \leq O(n^{1+\alpha})$

and $T_{\mathcal{R}}(n,t) \leq T_{\mathcal{R}}(n,n) \leq O(n^{1+\alpha})$. Furthermore, we have that $\omega(\delta/2+\alpha,1,1) \leq \omega(\delta/2,1,1)+\omega(\alpha,0,0) = \omega(\delta/2,1,1)+\alpha$ by Proposition 4.1.8, Items 1 and 2. Using the fact that $\omega(\delta/2,1,1) \geq 2$ for all $\delta \in [0,2]$, we then have that for any constant $\varepsilon' > 0$ the algorithm performs at most

$$O(T_{\mathcal{M}}(n,t)+n\cdot T_{\mathcal{R}}(n,t)+n^{\omega(\delta/2+\alpha,1,1)+\varepsilon'}) \leq O(n^{1+\alpha}+n^{2+\alpha}+n^{\omega(\delta/2,1,1)+\alpha+\varepsilon'}) \leq O(n^{\omega(\delta/2,1,1)+\alpha+\varepsilon'})$$

operations over R. To prove the claim, we need to show that the algorithm performs at most $O(n^{\omega(\delta/2,1,1)+\varepsilon})$ operations. This follows by choosing $\varepsilon', \alpha > 0$ such that $\varepsilon'+\alpha \leq \varepsilon$. □

Now applying Lemma 4.1.4 we get a deterministic algorithm for MMV_R^δ.

Corollary 4.2.5. *For every $\varepsilon > 0$ and $\delta \in [0,2]$, MMV_R^δ can be solved in $O(n^{\omega(1,1,\delta/2)+\varepsilon})$.*

4.2.4 Comparison to Past Work

Deterministic MMV algorithms. We note that $\omega(1,1,\beta) < \omega$ for all $\beta < 1$ (assuming $\omega > 2$; see Theorem 4.1.7), and so our algorithm is faster than matrix multiplication when $AB-C$ is promised to be $O(n^\delta)$-sparse for constant $\delta < 2$. Furthermore, it is faster than Künnemann's algorithm [89], which is also for MMV in the regime where $AB-C$ is sparse, when $\omega(1,1,\delta/2) < 1+\delta$. The equation $\omega(1,1,\delta/2) = 1+\delta$ whose unique solution corresponds to the crossover point at which our algorithm becomes faster than Künnemann's turns out to be relevant in other contexts too [131], and [93, 124] both provide bounds on its solution. Specifically, [124] shows that the solution δ to this equation satisfies $\delta \leq 1.056$, and so our algorithm in Corollary 4.2.5 is (strictly) faster than any previously known deterministic algorithm for MMV when $1.056 \leq \delta < 2$. See Section 4.2.4.

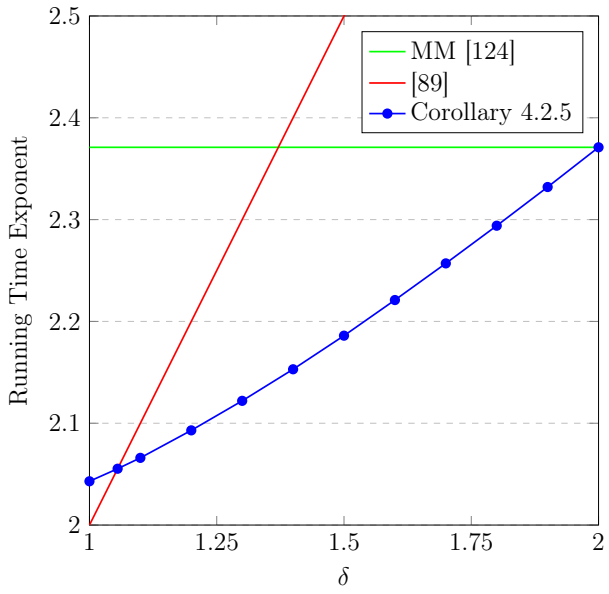

Figure 4.1 Running Times of Deterministic Algorithms for MMV when $AB - C$ is $O(n^\delta)$-sparse for $1 \leq \delta \leq 2$. Our algorithm from Corollary 4.2.5 is faster than the best known algorithms for matrix multiplication [124] and faster than Künnemann's algorithm [89] for all $1.056 \leq \delta < 2$. The plotted blue points corresponding to the running time of the algorithm in Corollary 4.2.5 are derived from the bounds on $\omega(1, 1, \delta/2)$ in [124, Table 1]. The line segments connecting them are justified by the fact that $\omega(1, 1, \cdot)$ is a convex function.

Additional bounds on $\omega(1, \delta/2, 1) = \omega(1, 1, \delta/2)$—and hence the running time of the algorithm in Corollary 4.2.5—appear in [124, Table 1]. For example, that table shows that $\omega(1, 1, 0.55) < 2.067$ and $\omega(1, 1, 0.95) < 2.333$ (which correspond to $\delta = 1.1$ and $\delta = 1.9$, respectively). We again refer the reader to Section 4.2.4. We also note that our algorithm runs in essentially optimal $\widetilde{O}(n^2)$ time when $\delta \leq 0.642 \leq 2\omega^\perp$, where $\omega^\perp := \sup\{\omega' > 0 : \omega(1, 1, \omega') = 2\} \geq 0.321$ is the dual matrix multiplication exponent [124], but that Künnemann's algorithm [89] runs in $\widetilde{O}(n^2)$ time for any $\delta \leq 1$.

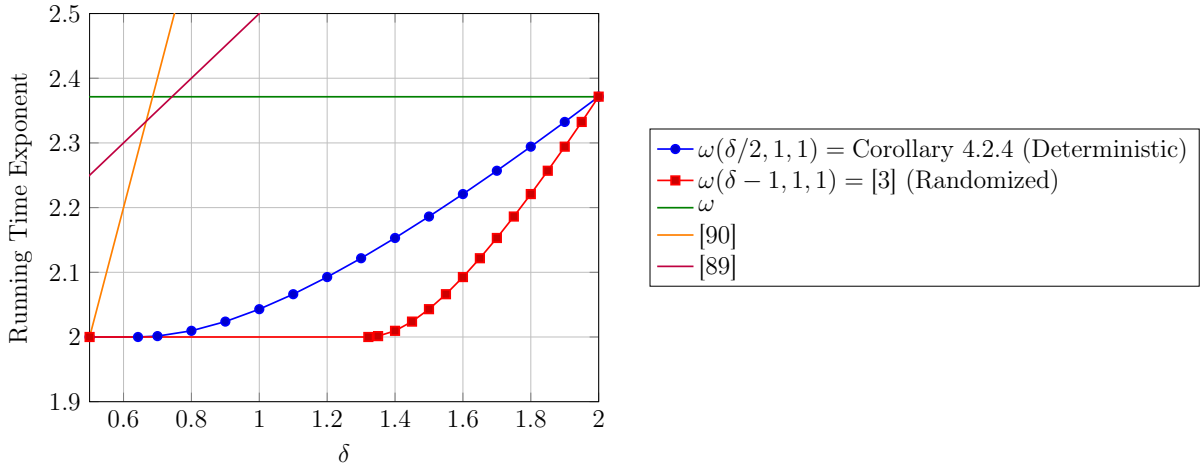

Figure 4.2 A Plot of the Running Time Exponents of Several Algorithms for δ-OSMM, with Arbitrarily Small Polynomial Factors Suppressed. The blue curve gives the running time of our deterministic algorithm from Corollary 4.2.4, which runs in $n^{\omega(\delta/2,1,1)}$ time. The red curve gives the running time of the randomized algorithm from [3]. These algorithms both run in roughly $n^{\omega(\delta-1,1,1)}$ time. The points on the curves come from upper bounds on rectangular MM exponents in [124], and the line segments connecting them are justified by the convexity of $\omega(\cdot, 1, 1)$. We also plot the running time exponents of the $O(n^{\max\{2,2\delta+1\}})$-time algorithm of Kutzkov [90] and the $O(n^{\max\{2+\delta/2,2\delta\}})$-time algorithm of Künnemann [89]—which are the main prior works giving *deterministic* algorithms for OSMM—as well as the MM exponent ω.

Deterministic OSMM algorithms. A long line of work has studied both input- and output-sparse matrix multiplication [3, 9, 45, 63, 74, 75, 89, 90, 97, 105, 112, 121, 131]. Our deterministic algorithm in Corollary 4.2.4 substantially improves on prior deterministic δ-OSMM algorithms for most values of δ, and is never slower than them. See Section 4.2.4. As noted above, our algorithm runs in essentially optimal quadratic time for $\delta \leq 0.642$, and for every constant $\delta < 2$ it runs in $O(n^{\omega-\varepsilon})$ time for some $\varepsilon = \varepsilon(\delta) > 0$.

The main prior works on deterministic OSMM are Kutzkov [90], which gives a $O(n^2 + n^{2\delta+1})$-time algorithm for OSMM over the reals, and Künnemann [89], which gives a $O(n^{2+\delta/2} + n^{2\delta})$-time algorithm for OSMM over the integers. Their algorithms only run in $O(n^{\omega-\varepsilon})$ time for $\delta < (\omega - 1)/2 \approx 0.686$ and $\delta < 2(\omega - 2) \approx 0.744$, respectively. (One can also show analytically that Corollary 4.2.4 is at least as fast as [89, 90] for all δ.[7])

4.3 On the non-SETH-hardness of Matrix Multiplication

In this section, we show a barrier for proving a lower bound for Matrix Multiplication (and, thus, for MMV) under SETH. More specifically, we show that if Matrix Multiplication is $n^{2.31}$-SETH-hard (recall Definition 2.3.6), then such a result would imply either super-linear circuit lower bounds, or an explicit construction of rigid matrices.

We now instantiate a lemma from [16] highlighting that if the input matrices are sufficiently non-rigid, then there is a simple *non-deterministic* algorithm that efficiently multiplies non-rigid matrices. In [16], we provide a general proof against n^γ-SETH-hardness for any $\gamma > 2$, however for simplicity we focus only on the case when $\gamma \geq 2.31$.

Lemma 4.3.1 ([16, Lemma 4.3])**.** *Let $\mathbb{F}_q$ be a finite field, and let $A, B \in \mathbb{F}_q^{n\times n}$ be two matrices satisfying*

$$\mathcal{R}^{\mathbb{F}_q}_{n^{0.9}}(A) \leq n^{1.5}\,, \quad \mathcal{R}^{\mathbb{F}_q}_{n^{0.9}}(B) \leq n^{1.5}\,.$$

Then, there is a non-deterministic algorithm that computes AB in time $O(n^{2.3})$.

[7] By Theorem 4.1.7 we have that $\omega(1, 1, \delta/2) \leq 1.825 + 0.548\delta/2$ from which it follows that $\omega(1, 1, \delta/2) \leq 2\delta + 1$ for all $\delta \geq 0.5$. So, our algorithm is always at least as fast as the algorithm in [90]. Furthermore, by Proposition 4.1.8, Item 1, $\omega(\delta/2, 1, 1) \leq 2 + \delta/2$ and so our algorithm is always at least as fast as the algorithm in [89].

We are now ready to present the main result of this section: a barrier to proving SETH-hardness of Matrix Multiplication (and hence also of Matrix Multiplication Verification). Below, by $\mathsf{QP} = \mathsf{DTIME}[n^{\mathrm{poly}(\log n)}]$ we denote the class of deterministic algorithms running in quasi-polynomial time.

Theorem 4.3.2. *Let q be a prime power. If Matrix Multiplication over $\mathbb{F}_q$ is $n^{2.31}$-SETH-hard then one of the following holds:*

- *NSETH is false, and hence E^{NP} has super linear circuit lower bounds (Theorem 2.3.3).*
- *There is a $\mathsf{QP}^{\mathsf{NP}}$ algorithm $\mathcal{A}$ such that for infinitely many values of N, $\mathcal{A}(1^N)$ outputs a matrix $A \in \mathbb{F}_q^{M \times M}$ where $M = N^{\Theta(1)}$ satisfying*

$$\mathcal{R}^{\mathbb{F}_q}_{M^{0.9}}(A) \geq M^{1.5}.$$

Specifically, the running time of the algorithm $\mathcal{A}$ on input 1^N is $N^{O(\log \log N)}$.

Proof. The high-level idea of the proof is the following. Assume there is a fine-grained reduction R_k from k-SAT to $\mathrm{MM}_{\mathbb{F}_q}$ for every k. For every k-SAT formula on n variables, R_k adaptively produces a (possibly exponentially large) sequence of MM instances. If for all large enough n, all the produced matrices are non-rigid, then from Lemma 4.3.1 we have an efficient non-deterministic algorithm for all encountered instances of MM, and, thus, an efficient non-deterministic algorithm for k-TAUT. This refutes NSETH. On the other hand, if for infinitely many values of n, at least some of the produced instances of MM are rigid, then we have an algorithm that brute forces all k-SAT instances, applies R to them, and then uses the NP oracle to verify if they are rigid. We will show that this algorithm finds rigid matrices $M \in \mathbb{F}_q^{N \times N}$ in time $N^{O(\log \log N)}$ with an NP oracle. Below we formalize this argument.

Assume that $\mathrm{MM}_{\mathbb{F}_q}$ is $n^{2.31}$-SETH-hard (see Definition 2.3.6). Then there exists a function $\delta\colon \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ such that for every $k \in \mathbb{N}$, $(k\text{-SAT}, 2^n) \leq_\delta (\mathrm{MM}_{\mathbb{F}_q}, n^\gamma)$. Let us fix $\varepsilon_0 = 0.01/2.31$, so that $n^{2.31(1-\varepsilon_0)} = n^{2.3}$, and $\delta_0 = \delta(\varepsilon_0) \in (0,1)$, where δ is the function from the definition of SETH-hardness. For every k, let R_k be the fine-grained reduction from k-SAT to $\mathrm{MM}_{\mathbb{F}_q}$ guaranteed by the $n^{2.31}$-SETH-hardness of $\mathrm{MM}_{\mathbb{F}_q}$. An $n^{2.3}$-time algorithm for $\mathrm{MM}_{\mathbb{F}_q}$ would imply (via R_k) a $2^{n(1-\delta_0)}$-time algorithm for k-SAT. In particular, the reduction R_k runs in time $O(2^{n(1-\delta_0)})$.

For every k, let us consider the following algorithm $\mathcal{A}_k$ equipped with an NP oracle. On input 1^N, if N is not a power of two, the algorithm halts. Otherwise, the algorithm sets $n = \log_2 N$ and $\lambda = \delta_0/4$. Let $c = c(k,\lambda)$ be the constant from the Sparsification Lemma (Theorem 2.3.4). The algorithm $\mathcal{A}_k$ enumerates all k-SAT formulas with n variables and at most cn clauses, applies the fine-grained reduction R_k to each of them, solves all queried instances of $\mathrm{MM}_{\mathbb{F}_q}$ by the trivial cubic-time algorithm, and this way adaptively obtains all MM instances produced by R_k. Let us denote these instances of MM by $(A_1, B_1), \ldots, (A_T, B_T)$. For each instance (A_i, B_i) where $A_i, B_i \in \mathbb{F}_q^{M \times M}$, if $M > 2^{\delta_0 n/12}$ the algorithm checks if it satisfies

$$\mathcal{R}^{\mathbb{F}_q}_{M^{0.9}}(A_i), \mathcal{R}^{\mathbb{F}_q}_{M^{0.9}}(B_i) \geq M^{1.5}\,.$$

Since the problem of checking rigidity is trivially in coNP, $\mathcal{A}_k$ checks this using the NP oracle. If $\mathcal{A}_k$ finds a rigid matrix, it outputs the first such found matrix, otherwise it outputs nothing.

Now we bound the running time of the algorithms $\mathcal{A}_k$. Enumeration of all k-SAT instances with n variables and cn clauses takes time $\binom{O(n^k)}{\leq cn} = 2^{O(n \log n)}$. The running time of the Sparsification Lemma is $2^{\lambda n} = 2^{\delta_0 n/4}$. The running time of each execution of the fine-grained reduction is $O(2^{n(1-\delta_0)})$, and the time needed to perform each

matrix multiplication is at most $O(2^{3n(1-\delta_0)})$. This leads to the total running time of $2^{O(n \log n)} = N^{O(\log \log N)}$.

If one of the constructed algorithms $\mathcal{A}_k$ outputs matrices on inputs 1^N infinitely often, then we are done, as we have a $\mathsf{QP}^{\mathsf{NP}}$ algorithm outputting rigid matrices infinitely often. Indeed, the algorithms $\mathcal{A}_k$ output only rigid matrices, and the dimension of each such matrix is $M > 2^{\delta_0 n/12} = N^{\Theta(1)}$.

In the following, we assume that for each k, there exists N_k such that the algorithm $\mathcal{A}_k$ does not output matrices on inputs 1^N for all $N \geq N_k$. Equivalently, there exists n_k such that all k-SAT formulas with $n \geq n_k$ variables and cn clauses, when fed to the reduction R_k, result in instances (A_i, B_i) of $\mathrm{MM}_{\mathbb{F}_q}$ with $A_i, B_i \in \mathbb{F}_q^{M \times M}$ either of size $M \leq 2^{\delta_0 n/12}$ or satisfying

$$\mathcal{R}^{\mathbb{F}_q}_{M^{0.9}}(A_i) < M^{1.5} \quad \text{and} \quad \mathcal{R}^{\mathbb{F}_q}_{M^{0.9}}(B_i) < M^{1.5} \,. \tag{4.2}$$

We will use this to design a non-deterministic algorithm that solves k-TAUT in time $O(2^{n(1-\delta_0)})$ for every k, which refutes NSETH.

Given a k-DNF formula φ, we first apply the Sparsification Lemma to (the negation of) φ with $\lambda = \delta_0/4$. This gives us $2^{\lambda n}$ k-CNF formulas φ_i such that φ is a tautology if and only each φ_i is unsatisfiable. Therefore, solving k-TAUT for the negation of each φ_i is sufficient for solving the original instance φ. Now we apply the reduction R_k to each φ_i. Recall that the fine-grained reduction runs in time $2^{(1-\delta_0)n}$, and, in particular, creates at most $2^{n(1-\delta_0)}$ instances of MM, each of size at most $2^{n(1-\delta_0)}$. Therefore, the total number T of instances of $\mathrm{MM}_{\mathbb{F}_q}$ we obtain is at most $T \leq 2^{\lambda n} \cdot 2^{n(1-\delta_0)} = 2^{(1-3\delta_0/4)n}$. We solve each instance of size $M \leq 2^{\delta n/12}$ by a trivial cubic-time algorithm. The overall running time to solve all such small instances is at most $T \cdot 2^{3\delta_0 n/12} \leq 2^{n(1-\delta_0/2)}$.

Now we use the assumption that the algorithm $\mathcal{A}_k$ does not find rigid matrices for all but finitely many inputs. Therefore, for every $n \geq n_k$, every instance (A_i, B_i), $A_i, B_i \in \mathbb{F}^{M \times M}$ where $M > 2^{\delta_0 n/12}$ satisfies Equation (4.2). We apply Lemma 4.3.1 to compute $A_i \cdot B_i$ in non-deterministic time $O(M^{2.3})$. By the guarantee of the SETH-hardness reduction R_k, we have that a non-deterministic algorithm solving $\mathrm{MM}_{\mathbb{F}_q}$ in time $O(M^{2.3})$ implies a non-deterministic algorithm for k-TAUT with running time $2^{n(1-\delta_0)}$. This refutes NSETH, and finishes the proof of the theorem. □

Comparison with Known Rigidity Results. While the rigidity parameters that come out of n^γ-SETH-hardness of matrix multiplication via Theorem 4.3.2 are not strong enough for Valiant's program for proving circuit lower bounds, they would improve on all currently known constructions of rigid matrices. The main problem in this area is to find an *explicit* matrix (i.e., a matrix from a complexity class where we cannot prove circuit lower bounds: P, NP, or even E^{NP}) of high rigidity for linear rank $r = \Theta(n)$.

The best known rigidity lower bound for matrices constructible in polynomial time (i.e., from the class P) is $\mathcal{R}^{\mathbb{F}}_r(A) \geq \Omega\left(\frac{n^2}{r} \cdot \log \frac{n}{r}\right)$ [50, 108, 117], and it falls short of achieving the rigidity parameters needed for Valiant's program (that requires rigidity $\Omega(n^{1+\varepsilon})$ for $r = \Theta(n)$ for a constant $\varepsilon > 0$). Goldreich and Tal [58] give a matrix $A \in \mathbb{F}^{n \times n}$ constructible in time $2^{O(n)}$ (i.e., in the class E) of rigidity $\mathcal{R}^{\mathbb{F}}_r(A) \geq \Omega\left(\frac{n^3}{r^2 \log(n)}\right)$ for any $r \geq \sqrt{n}$, which improves on the previous constructions for $r = o\left(\frac{n}{\log(n) \log\log(n)}\right)$. Alman and Chen, and Bhangale et al. [7, 19] give a matrix $A \in \mathbb{F}_2^{n \times n}$ achieving very high rigidity $\mathcal{R}^{\mathbb{F}_2}_r(A) \geq \Omega(n^2)$ for sub-polynomial rank $r = 2^{\varepsilon \log(n)/\log\log(n)}$ in P^{NP}.

We now compare the rigidity bounds implied by Theorem 4.3.2 from n^γ-SETH-hardness for various values of $\gamma > 2$ to the known lower bounds on rigidity. As seen

above when $\gamma = 2.3$ we get a matrix A of rigidity $\mathcal{R}^{\mathbb{F}}_{n^{0.9}}(A) \geq n^{1.5}$. The rigidity bound obtained from $\gamma \geq 2.24$ would improve upon the best known construction in E [58] for every $r \geq n^{0.751}$. The rigidity bound obtained from any $\gamma \geq 2.17$ would already improve upon the best known constructions in P [50, 108, 117] for every $r \geq n^{0.63}$. Finally, the bound obtained from any $\gamma \geq 2$ would give us rigidity for higher rank than the known P^{NP} constructions [7, 19].

Chapter 5

Improved Data Structures and Lower Bounds

Data structures model a natural form of computation that takes place in two phases. First, given an input database, we perform a preprocessing step that produces a data structure using S bits of space. Later, we must answer queries using only T time and access to this stored information. One reason we study data structures is because they naturally capture time-space tradeoff: how much space is required to answer queries in time T.

For most data structure problems there are two trivial solutions. One approach is to simply store the input database ($S \approx N$) and answer each query recomputing from scratch, yielding $T \approx N$. At the other extreme, we can precompute and store the answer to every possible query in a lookup table ($S \approx M$, where M is the size of the query space), allowing any query to be answered in constant time $T = O(1)$. The goal of designing non-trivial data structures is to find meaningful tradeoffs between these extremes achieving both sublinear query time and using less space then a complete look up table.

We study the complexity of designing data structures for the Orthogonal Vectors problem. In the *Orthogonal Vectors* problem ($\mathrm{OV}_{n,d}$), the goal is to determine whether a set of n vectors $x_1, \ldots, x_n \in \{0,1\}^d$ contains a pair that are orthogonal. The problem plays a central role in fine-grained complexity, and its associated hardness assumptions have been used to obtain tight bounds for a wide range of problems [38, 56, 122, 123, 128]. The most interesting regimes for d is the low-dimensional regime ($d = c \log n$

where c is a large constant) and the moderate-dimensional regime ($d = n^{\varepsilon}$, for a small constant $\varepsilon > 0$). The best algorithms for OV run in time $n^{2-\Omega(1/\log c)}$ when $d = c \log n$ [2, 29, 32, 71].

Here we will consider the data-structure variant of OV, known as the *Online Orthogonal Vectors* problem (OnlineOV). The problem $\text{OnlineOV}_{n,d}$ is solved in two phases by a pair of algorithms $(\mathcal{P}, \mathcal{A})$. In the *preprocessing phase*, the algorithm $\mathcal{P}$ receives n vectors $x_1, \ldots, x_n \in \{0,1\}^d$ and constructs a data structure $\sigma \in \{0,1\}^S$ of size S. In the *query phase*, the algorithm $\mathcal{A}$ is given σ and a query vector $q \in \{0,1\}^d$, and must determine whether q is orthogonal to any input vector x_i. The query algorithm runs in time T. Such a pair $(\mathcal{P}, \mathcal{A})$ is called an (S,T)-data structure for $\text{OnlineOV}_{n,d}$, and the preprocessing time $T_{\mathcal{P}}$ is the running time of $\mathcal{P}$.

The study of OnlineOV dates back to the 1970s [84, 110], and the problem has since appeared in several areas including information retrieval, databases, and networking (see [5, 34, 64, 84, Section 6.5]). Moreover, OnlineOV is essentially equivalent to several well-studied problems, most notably the *Partial Match* problem (PartialMatch) [11, 30, 34, 44, 84, 110, 111]. It is also closely related to other fundamental problems such as Orthogonal Range Searching, Subset Query, DNF evaluation, Maximum Inner Product, and variants of Nearest Neighbor search [2, 30, 34, 38, 72].

The best known deterministic algorithms for $\text{OnlineOV}_{n,d}$ are those by [34], which run in (i) space $S = n \cdot 2^{O(d \log^2(d)\sqrt{t/\log n})}$ and time $T = n/2^t$ for any t; and (ii) space $S = nd^t$ and time $T = O(nd/t)$ for $t \leq n$. Notably, when $d = c \log n$ for a large enough constant $c > 1$, a more efficient *randomized* algorithm [30] achieves space $S = n^{1.1}$ and query time $T = n^{1-1/(c\log(c))}$.

In Theorem 5.2.1 we design a deterministic data structure that matches the guarantees of the best known randomized data structure in the low-dimensional

regime [30]. Moreover our data structure generalizes to arbitrary dimension d. As a consequence, in Theorem 5.2.2, we further obtain the first improvements since [34] for data structures in the moderate dimension regime.

On the other hand, one can study the hardness of OnlineOV in both the worst-case and average-case settings [21, 59, 60, 73, 77, 98, 111]. Proving nontrivial data structure lower bounds, however, is notoriously difficult. Nevertheless, there has been substantial progress on lower bounds in restricted computational models [5, 10, 11, 72, 106, 107]. For example, [106, 107] showed that any decision tree solving $\text{OnlineOV}_{n,d}$ must either have size $2^{\Omega(d)}$ or depth $\Omega(n^{1-2\delta}/d)$.

When the preprocessing time is required to be efficient, conditional lower bounds for OnlineOV can be obtained from standard fine-grained assumptions such as the *Strong Exponential Time Hypothesis* (SETH). Using Williams' reduction from k-SAT to OV [128], one can show that under SETH, for every pair of constants $\alpha, \delta > 0$ there exists $c > 1$ such that $\text{OnlineOV}_{n,c\log n}$ cannot be solved with query time $n^{1-\delta}$ and preprocessing time n^{α}.

However, to prove lower bounds on the space used by a data structure, we must strengthen our assumptions even further. In [6], the authors establish similar lower bounds even under unbounded preprocessing, albeit at the cost of a stronger assumption, namely the Non-Uniform Strong Exponential Time Hypothesis (NUSETH) which roughly posits that SAT requires "perebor" even for non-uniform algorithms.

We push this paradigm further. In Theorem 5.3.3, we show that under NUSETH there is no $(N^{1.1}, N^{0.99})$ data structure for $\text{OnlineOV}_{N,N^{\delta}}$, even when every vector in the input database has constant sparsity. Using similar ideas, we are also able to connect two seemingly unrelated problems. In Theorem 5.3.6, we show that if non-uniform algorithms solving the Hamiltonian path require 2^n time, then any data structure for 3-SUM must require super-linear space.

This chapter is based on a paper co-authored with Alexander Golovnev, Sam King and Sidhant Saraogi [53]. Much of the technical sections in this chapter is taken verbatim from this work.

5.1 Data Structures Preliminaries

Let H denote the binary entropy function where $H(p) = -p \log p - (1-p)\log(1-p)$ for $0 < p < 1$. Then,

Fact 5.1.1. *For all* $1 \leq k \leq n$,

$$\binom{n}{k} \leq 2^{H(k/n)n} .$$

5.1.1 Data Structure Problems

We define a data structure problem as a triplet $(\mathcal{D}, \mathcal{Q}, \mathcal{F})$, which corresponds to possible inputs $D \in \mathcal{D}$, $D \in \{0,1\}^N$, a collection of queries $\mathcal{Q}$, and a function $\mathcal{F} : \mathcal{D} \times \mathcal{Q} \to \{0,1\}$ which encodes the answers to all the queries $q \in \mathcal{Q}$ for inputs from $\mathcal{D}$.

Definition 5.1.2 (Data Structures)**.** An (S,T)-data structure for a problem $(\mathcal{D}, \mathcal{Q}, \mathcal{F})$ is a pair of deterministic algorithms $(\mathcal{P}, \mathcal{A})$ that solve the problem in two phases.

Preprocessing phase. In the first phase, the preprocessing algorithm P receives an input D, and preprocesses it into a data structure $\sigma \in \{0,1\}^S$ consisting of S bits.

Query phase. In the second phase, the query algorithm $\mathcal{A}$ runs in time T, and receives a query $q \in \mathcal{Q}$ and (query access to) the data structure σ, and outputs 1 if and only if $\mathcal{F}(D, q) = 1$.

In addition, we say that the data structure $(\mathcal{P}, \mathcal{A})$ has preprocessing time $T_{\mathcal{P}}$ if $\mathcal{P}$ runs in time at most $T_{\mathcal{P}}$.

We generally wish to simultaneously minimize all three computational parameters: the space S, the query time T, and the preprocessing time $T_{\mathcal{P}}$. However, in our lower bounds we will allow the preprocessing algorithm to be computationally unbounded, which only makes our results stronger. In the upper bounds, we will present algorithms with efficient preprocessing.

5.1.2 OnlineOV and Related Data Structures

Definition 5.1.3 (Online Orthogonal Vectors (OnlineOV$_{n,d}$))**.** For a dimension parameter $d := d(n)$, OnlineOV$_{n,d}$ is a data structure problem $(\mathcal{D}, \mathcal{Q}, \mathcal{F})$ where:

- $\mathcal{D} = \{D \colon D \text{ is a set of vectors } \{x_1, \ldots, x_n\}, \text{ where } x_i \in \{0,1\}^d\}$.
- $\mathcal{Q}$ is the set of all vectors in $\{0,1\}^d$.
- $\mathcal{F}(D, q) = 1$ if and only if there exists $x_i \in D$ such that $x_i \perp q$.

In the special case where for every $x_i \in \mathcal{D}$, $\|x_i\|_0 = c$, we call this problem c-*Sparse*-OnlineOV$_{n,d}$.

We will be primarily interested in two parameter regimes—when $d = c \log n$ for a constant $c > 0$ (OnlineOV$_{n,c\log n}$) and when $d = n^{\varepsilon}$ for a small constant $0 < \varepsilon < 1$ (OnlineOV$_{n,n^{\varepsilon}}$).

We will work with the following related data structure problems.

- PartialMatch$_{n,d}$: Preprocess n vectors $(x_1, \ldots, x_n)$ from $\{0,1\}^d$, and for a query vector $q \in \{0,1,*\}^d$, output 1 if and only if there exists i such that for all j: $q[j] = *$ or $q[j] = x_i[j]$.

- SubsetQuery$_{n,d}$: Preprocess n sets $S_1, \ldots S_n \subseteq [d]$, and for a query $q \subseteq [d]$ output 1 if and only if there exists an i such that $q \subseteq S_i$.

- ContainmentQuery$_{n,d}$: Preprocess n sets $S_1, \ldots S_n \subseteq [d]$, and for a query $q \subseteq [d]$ output 1 if and only if there exists an i such that $S_i \subseteq q$.

- OrthogonalRangeSearch$_{n,d,u}$: Let u be an integer. Preprocess n points $(x_1, \ldots, x_n)$ from $[u]^d$, and for a query vector $q = (\ell_1, r_1, \ldots, \ell_d, r_d)$ that represents a rectangle in $[u]^d$ output 1 if there is an x_i contained in the rectangle, i.e., if there exists i such that for all $j \in [d]$: $\ell_j \leq x_i[j] \leq r_j$.

- $(1+\varepsilon)$-OrthogonalRangeSearch$_{n,d,u}$: A generalization of OrthogonalRangeSearch$_{n,d,u}$ where every query rectangle is approximated by a larger rectangle containing the query rectangle where each side is blown up by a factor of $(1+\varepsilon)$.

- DNF-Evaluation$_{n,d}$: Let φ be a DNF formula with n clauses and d variables. Preprocess φ, and for a query $q \in \{0,1\}^d$, output 1 if and only if $\varphi(q) = 1$.

Lemma 5.1.4 ([2, 34, 38, 72])**.** *If there is an (S,T)-data structure with preprocessing time $T_{\mathcal{P}}$ for any of the following problems, then there is an $(O(S), O(T))$-data structure with preprocessing time $O(T_{\mathcal{P}})$ for* OnlineOV$_{n,d}$*.*

$$\text{PartialMatch}_{n,d},\ \text{SubsetQuery}_{n,d},\ \text{ContainmentQuery}_{n,d}\ ,$$
$$\text{OrthogonalRangeSearch}_{n,d,1},\ \text{DNF-Evaluation}_{n,d}\ .$$

Lemma 5.1.5 ([2, 34])**.** *If there is an (S,T)-data structure with preprocessing time $T_{\mathcal{P}}$ for* OnlineOV$_{n,d}$*, then there is an $(O(S), O(T))$-data structure with preprocessing time $O(T_{\mathcal{P}})$ for each of the following problems.*

$$\text{PartialMatch}_{n,d/2},\ \text{DNF-Evaluation}_{n,d/2},\ \text{SubsetQuery}_{n,d},$$
$$\text{ContainmentQuery}_{n,d},\ (1+\varepsilon)\text{-OrthogonalRangeSearch}_{n,O(d\varepsilon/\log u),u}\ .$$

5.1.3 Non-Uniform algorithms and Conjectures

In this work, we study the non-uniform version of SETH. Therefore, we begin with the definition of non-uniform algorithms.

Definition 5.1.6 (Non-Uniform RAM)**.** A non-uniform RAM algorithm is a RAM algorithm $\mathcal{A}$ and an infinite family of advice strings $\{a_i\}_{i\in\mathbb{Z}^+}$. Such an algorithm solves a problem $\mathcal{P}$ in time T and using S bits of non-uniform advice if for all sufficiently large n, (i) the length of $|a_i| \leq S$, and (ii) for every input x of length n, $\mathcal{A}(x, a_n)$ solves $\mathcal{P}$ on input x in time T.

Now, equipped with the definition of non-uniform algorithms, we are ready to state the Non-Uniform SETH.

Conjecture 5.1.7 (Non-Uniform SETH (NUSETH))**.** *For every constant $\varepsilon \in (0, 1)$ there exists $k \in \mathbb{Z}^+$ such that no non-uniform RAM algorithm solves k-SAT on formulas with n variables in time $2^{(1-\varepsilon)n}$ using $2^{(1-\varepsilon)n}$ bits of advice.*

On the choice of the computational model in NUSETH. Various non-uniform versions of SETH and related conjectures have been studied in previous works (see, e.g., [4, 26, 42, 55]). There are two natural choices for the computational model in the conjecture: non-uniform circuits and non-uniform RAM algorithms, and the former model would lead to a weaker conjecture.[1] For simplicity, in this paper, we choose to adapt the stronger version of the conjecture: no non-uniform RAM algorithm solves k-SAT in $2^{(1-\varepsilon)n}$ for all k. The point of this work is to show that when we construct a non-trivial data structure for OnlineOV (or a number of related problems), we will

[1]Note that even for the extremely well-studied SETH conjecture, there is no clear agreement on the computational model. In particular, a $2^{n/2}$ RAM algorithm for SAT would not necessarily imply a Turing machine solving SAT in fewer than 2^n steps. Therefore, such an algorithm would refute SETH stated for RAM algorithms but not the Turing machines version of SETH.

obtain a faster-than-2^n non-uniform algorithm for SAT. This will be great regardless of the definition of NUSETH!

Finally, we state the non-uniform version of the conjecture regarding the hardness of Hamiltonian Path. We say that a path in a graph is simple if it does not repeat any vertices. In the Hamiltonian Path problem (HamPath), given a directed graph G on n vertices, the goal is to check if G has a simple path that visits every vertex. While many special cases of HamPath admit efficient algorithms (including Björklund's celebrated algorithm for Undirected HamPath [20]), the best known algorithm for the general case of the problem is due to Bellman, and Held and Karp [15, 66], and dates back 60 years. This algorithm runs in time $O(2^n \cdot \text{poly}(n))$, and despite much effort, no faster algorithms are known even in the non-deterministic and non-uniform settings. Since progress has been hard on this problem we state the following non-uniform conjecture for Hamiltonian Path.

Conjecture 5.1.8 (Non-Uniform HamPath Conjecture)**.** *For every constant $\varepsilon \in (0,1)$, no non-uniform RAM algorithm solves* HamPath *on graphs with n vertices in time $2^{(1-\varepsilon)n}$ using $2^{(1-\varepsilon)n}$ bits of advice.*

We will use the following version of the dynamic programming algorithm for HamPath from [15, 66].

Proposition 5.1.9 (Implicit in [15, 66])**.** *There is a deterministic algorithm that for every $k := k(n)$ outputs all simple paths of length at most k in a given n-vertex graph in time $\binom{n}{\leq k} \cdot \text{poly}(n)$.*

Proof. For a set of vertices $S \subseteq [n]$ and a pair of distinct vertices $u, v \in S$, let $D[S, u, v] = 1$ if there exists a simple path starting at the vertex u, passing through all vertices of S, and ending at the vertex v, and 0 otherwise. Each value $D[S, u, v]$ can be computed in linear time given the values of $D[S', u, v']$ for smaller sets S'.

Therefore, the naïve dynamic programming algorithm computing $D[S, u, v]$ for all sets S of size $|S| \leq k$ runs in time $\binom{n}{\leq k} \cdot \text{poly}(n)$. □

5.2 Improved Data Structures

In this section we sketch the construction of our fast data structures for the OnlineOV problem. We will focus on the case where the input $X \subseteq \{0,1\}^d$ contains $|X| = n$ vectors of dimension $d := c \log n$. First, we describe a simple algorithm which is efficient on average when the input vectors are picked independently and uniformly at random.

5.2.1 Average-case algorithm

We begin with the simple observation that, when X is chosen at random, it is unlikely that there exists a large subset $X' \subseteq X$ and a large set of coordinates $S \subseteq [c \log n]$ such that every vector $x \in X'$ satisfies $x|_S = \mathbf{0}$. Let $Y_S := \{x \in X : \ x|_S = \mathbf{0}\}$. To quantify the observation, we fix some appropriate $\varepsilon > 0$ such that, with high probability, we expect that $|Y_S| < n^{1-\varepsilon}$ for an arbitrary S of certain size t. Note that $\mathbb{E}_X[|Y_S|] = n \cdot \frac{1}{2^{|S|}}$. Using the Chernoff and union bounds, $|Y_S| < n^{1-\varepsilon}$ for all $|S| = t \approx \varepsilon \log n$ with high probability.

Armed with this observation, our average-case algorithm is simple to state. In the discussion below, we will focus on the space complexity of the data structure and the query time of the online phase. In the preprocessing phase, `AverageOVPre` uses $\binom{d}{\leq t}$ space to store the answers to all possible sparse queries of Hamming weight $\leq t$. Furthermore, it stores the lists of relevant inputs Y_S for all $S \subseteq [c \log n]$ with $|S| = t$. This takes space at most $\binom{c \log n}{t} \cdot n^{1-\varepsilon}$ with high probability. In the online phase, `AverageOVOnl`, if the query q has Hamming weight $\|q\|_0 < t$, it takes constant time

to check the answer among the stored answers. Otherwise, the algorithm picks the first set S of t 1's of q. These coordinates must be 0 in any orthogonal input vector and therefore belong to Y_S. Now, in time $|Y_S| < n^{1-\varepsilon}$ it is easy to check if one of the vectors in Y_S is orthogonal to q.

To summarize, the space required by `AverageOVPre` is at most $\binom{c \log n}{\varepsilon \log n} \cdot n^{1-\varepsilon} \leq n^{1+\varepsilon \log(ec/\varepsilon)}$, and the query time of `AverageOVOnl` is at most $O(n^{1-\varepsilon})$.

5.2.2 Pseudorandomness

For an arbitrary set of input vectors X, there is no guarantee that X behaves like a random input. For example, every vector in X could have its first $\frac{c \log n}{2}$ coordinates be 0. Then one of our stored lists has size $|Y_{[c \log n/2]}| = n$, leading to our average-case algorithm having poor query time. However, violating the assumption that Y_S is small for a particular S leads to finding some structure in our problem. For example, in the above case, we could reduce the dimension of the problem by half. We formalize this intuition by defining a simple notion of pseudorandomness. For any $\varepsilon > 0$, we say that X is $(n^{1-\varepsilon}, t)$-pseudorandom if for every $S \subseteq [c \log n]$ with $|S| = t$, we have that $|Y_S| \leq n^{1-\varepsilon}$. While we cannot expect X itself to be pseudorandom, we find a decomposition of $X = X' \sqcup X_1 \sqcup \cdots \sqcup X_{n^\varepsilon}$ such that:

- For each X_i, we have $|X_i| = n^{1-\varepsilon}$. Additionally, we find a corresponding $S_i \subseteq [c \log n]$, $|S_i| = t$, such that each $x \in X_i$ satisfies $x|_{S_i} = \mathbf{0}$. For the purpose of finding orthogonal vectors in X_i, we can ignore the coordinates in S_i, i.e. X_i has some structure.

- The remaining part X' of X is $(n^{1-\varepsilon}, t)$-pseudorandom.

5.2.3 Worst-case algorithm

We now describe our worst-case algorithm. First, using a greedy procedure, we find an appropriate decomposition of $X = X' \sqcup X_1 \sqcup \cdots \sqcup X_{n^\varepsilon}$ as above. As in the average-case algorithm, in the preprocessing phase, `OVPre` (computes and) stores:

1. the answers to all possible sparse queries q with $\|q\|_0 < t$ in space $\binom{c \log n}{<t}$, and
2. for each $S \subseteq [c \log n]$ with $|S| = t$, a list of candidate orthogonal vectors Y_S from X' with $|Y_S| \leq n^{1-\varepsilon}$ in total space $\binom{c \log n}{t} \cdot n^{1-\varepsilon}$.

From this information, as in the average-case algorithm, we can answer all sparse queries and check any queries against the candidate vectors in the pseudorandom part X'. However, this leaves us with all the remaining candidate vectors in the structured parts $X_1, \ldots, X_{n^\varepsilon}$. For each set of vectors X_i, we can drop the coordinates in S_i as they are always 0. In other words, each X_i defines a subproblem of OnlineOV with $n^{1-\varepsilon}$ input vectors in $c \log n - t$ dimensions. Therefore, `OVPre` recursively creates a data structure for each of the n^ε sets X_i. This recursion continues until (A) the subproblem contains just one vector, in which case the algorithm just stores that vector, or (B) the dimension gets small enough, in which case the algorithm stores the answers to all possible queries. We can then use induction to argue that each recursive data structure requires space $\approx \binom{c \log n}{t} \cdot n^{1-2\varepsilon}$ for $\binom{c \log n}{t} \cdot n^{1-\varepsilon}$ space total across all recursively-constructed data structures, about matching the amount of space needed for the pseudorandom part.

During query time, if $\|q\|_0 < t$, `OVOnl` simply looks up the answer. Otherwise, it finds the first t non-zero coordinates of q and checks the corresponding list Y_S of candidates from X', which takes time at most $n^{1-\varepsilon}$. Finally, it determines whether there are any orthogonal vectors in each X_i using the recursively-constructed data

structures. By induction, we argue that each of these subproblems takes time at most $\approx n^{1-2\varepsilon}$ for $n^{1-\varepsilon}$ time total across all of the structured parts.

For technical reasons, we need to vary the parameters ε and t at different levels of recursion. To do this, we introduce a parameter i to track how many levels of recursion you are above the base case ($i = 1$ is at the base case), and we set $\varepsilon = 1/i$ and $t = c \log n/i$. Thus, setting the top level of recursion to be at $i \approx c \log c$, we get query time $T \leq n^{1-\Omega(\delta/c \log c)}$ and space $S \leq O(n^{1+\delta})$ where $\delta > 0$ is an arbitrarily small constant. Moreover, `OVPre` is time efficient, with preprocessing time $T_p \leq O(n^{1+\delta})$.

We now formally state our results. In the regime where the dimension $d = c \log n$, we give the state-of-the-art data strucutre for $\text{OnlineOV}_{n,d}$. Our data structures are deterministic and combinatorial.

Theorem 5.2.1 ([54, Corollary 3.9])**.** *Consider any $c > 1$, sufficiently large n and any $\delta > \Omega(\log c/c)$. There is an (S, T)-data structure with preprocessing time T_p for* $\text{OnlineOV}_{n,c \log n}$ *where:*

$$T = \tilde{O}\left(n^{1-\frac{1}{O(c \log c)}}\right) , \qquad \text{and} \qquad S,\ T_p = \tilde{O}(n^{1+\delta}) .$$

Our data structures have the added feature of working in arbitrary dimensions. As a result we get better data structures in the moderate dimension regime with $d = n^{\varepsilon}$.

Theorem 5.2.2 ([53, Corollary 3.8])**.** *Consider any n, d and $\epsilon < 1/2$, then there exists an (S, T)-data structure for* $\text{OnlineOV}_{n,d}$ *with preprocessing time T_p such that:*

$$T \leq n^{1-\varepsilon} d , \qquad S = n^{1-\varepsilon} d\, 2^{O(\varepsilon d \log(1/\varepsilon))} , \qquad T_p \leq n^{1+3\varepsilon} d\, 2^{O(\varepsilon d \log(1/\varepsilon))} .$$

Finally, we can also combine our data structures with known reductions from various online problems to OnlineOV to obtain new deterministic data structures for these problems (Corollary 5.2.3). These problems include Partial Match, DNF-Evaluation, Subset Query, Containment Query, and Approximate Orthogonal Range

Searching (See Section 5.1.2 for definitions). All of our data structures can be easily modified to work for *reporting* versions of the problem, i.e., returning all the orthogonal vectors in the input as opposed to identifying if one exists, with minimal overhead.

Corollary 5.2.3. *For all large enough n, c and $\delta \geq \Omega(\log c/c)$, there are (S, T)-data structures with preprocessing time T_p for each of the following problems:*

$$\text{PartialMatch}_{n,c\log n},\ \text{DNF-Evaluation}_{n,c\log n},\ \text{SubsetQuery}_{n,c\log n},$$
$$\text{ContainmentQuery}_{n,c\log n},\ (1+\varepsilon)\text{-OrthogonalRangeSearch}_{n,O(\varepsilon c\log n/\log u),u},$$

where

$$T = n^{1-\Omega\left(\frac{1}{c\log c}\right)} \quad \textit{and} \quad S,\ T_p = \tilde{O}(n^{1+\delta})\ .$$

5.3 Lower Bounds

In this section, we use hardness conjectures against non-uniform algorithms for k-SAT (Conjecture 5.1.7) and HamPath (Conjecture 5.1.8) to prove polynomial lower bounds on the space and query complexities of data structures (with computationally unbounded preprocessing) for Sparse-OnlineOV and 3-SUM.

5.3.1 Space Lower Bounds from Non-Uniform SETH

We begin by defining the hardest OnlineOV instance: a family of sparse OnlineOV instances for which efficiently checking orthogonality against the vectors in the instance would enable us to solve SAT for any input formula. We then show that there exists an explicit hardest OnlineOV instance, and we use it to prove space lower bounds for a range of data structure problems.

Definition 5.3.1 (Hardest OnlineOV Instance)**.** For a function $w := w(k)$ and a constant $\delta \in (0, 1]$, a family $(\mathcal{D}_{N,w})_{N,w\in\mathbb{Z}^+}$ of OnlineOV instances is a (w, δ)-*hardest* OnlineOV *instance* if it satisfies the following properties.

- For every N, w, $\mathcal{D}_{N,w}$ is an OnlineOV instance whose input contains N w-sparse vectors of dimension $O(N^\delta)$.
- Efficiently computable: there is a deterministic algorithm that, given N and w, outputs $\mathcal{D}_{N,w}$ in time $\widetilde{O}(Nw)$.
- Evaluating $(\mathcal{D}_{N,w})_{N,w\in\mathbb{Z}^+}$ solves k-SAT: There is a deterministic algorithm $\mathcal{A}$ that, given a k-SAT formula φ on n variables, outputs a vector a_φ of dimension N^δ for $N = 2^n$ and $w = w(k)$ such that φ is satisfiable if and only if there is a vector in $\mathcal{D}_{N,w}$ that is orthogonal to a_φ. The running time of $\mathcal{D}$ is $\widetilde{O}(N^\delta)$.

Lemma 5.3.2 ([53, Lemma 4.2])**.** *For every constant $\delta \in (0, 1]$, there is a (w, δ)-hardest* OnlineOV *instance with $w = \binom{k/\delta}{k}$.*

We now use Lemma 5.3.2 to show that under Non-Uniform SETH, computationally unbounded data structures even for the *batch* version of Sparse OnlineOV require either super-polynomial space or nearly linear time per query.

Theorem 5.3.3. *Under* NUSETH, for all constants $c, \alpha > 0$ and $\delta, \gamma \in (0, 1)$, there exists constant w such that no batch data structure with computationally unbounded preprocessing can answer a set of n^α queries of w-Sparse-OnlineOV$_{n,d}$ for $d = n^\delta$ in space n^c and time $n^{1-\gamma}$ per query.

Proof. Assume that there exist constant $c, \delta, \alpha, \gamma$ and a data structure that solves n^α queries of w-Sparse-OnlineOV$_{n,d}$ for $d = n^\delta$ in space n^c and time $n^{1-\gamma}$ per query. Let

$$\beta = \min(1/(2c),\ 1/(\alpha + 1)) ,$$

$$\varepsilon = \min(1/2,\ \beta(1 - \delta),\ \beta\gamma) .$$

We will construct a non-uniform algorithm solving k-SAT on m variables in time $2^{(1-\varepsilon)m}$ for every constant k, which refutes NUSETH.

Let φ be a k-CNF formula on m variables. We apply the standard branching argument based on the downward self-reducibility of k-SAT. Namely, for every assignment $\mathsf{u} \in \{0,1\}^{(1-\beta)m}$ to the first $(1-\beta)m$ variables of φ, let φ_{u} be the remaining k-CNF formula on βm variables. Note that φ is satisfiable if and only if at least one φ_{u} is satisfiable.

For each $\mathsf{u} \in \{0,1\}^{(1-\beta)m}$, we wish to determine if φ_{u} is satisfiable. By Definition 5.3.1, we obtain a vector of dimension N^{δ}, a_{u}, such that there exists $v \in \mathcal{D}_{N,w}$ with $v \perp a_{\mathsf{u}}$ if and only if φ_{u} is satisfiable. Let $\mathcal{Q}$ be the set of the vectors for all formulas φ_{u}. Note that size of $|\mathcal{Q}| = 2^{(1-\beta)m}$, and that by Definition 5.3.1 it can be computed in time $\widetilde{O}(|\mathcal{Q}| \cdot n^{\delta}) = 2^{(1-\beta+\beta\delta)m} = 2^{(1-\beta(1-\delta))m}$.

The constructed non-uniform algorithm for k-SAT uses the advice string σ and the query algorithm for the w-Sparse-OnlineOV$_{n,d}$ problem to determine if there is an orthogonal pair of vectors in $\mathcal{Q}$ and D.

First we argue the correctness of the designed non-uniform algorithm for k-SAT. First, by self-reduction of k-SAT, φ is satisfiable if and only if one of φ_{u} is satisfiable. By Definition 5.3.1, φ_{u} is satisfiable if and only if there exists $v \in \mathcal{D}_{N,w}$ with $v \perp a_{\mathsf{u}}$.

Now we analyze the running time of the designed algorithm for k-SAT. The length of the advice string σ is $S = n^c = 2^{\beta c m} \leq 2^{m/2} \leq 2^{(1-\varepsilon)m}$ by the choice of $\beta \leq 1/(2c)$ and $\varepsilon \leq 1/2$. The time needed to construct all queries $\mathcal{Q}$ is $2^{(1-\beta(1-\delta))m} \leq 2^{(1-\varepsilon)m}$ by the choice of $\varepsilon \leq \beta(1-\delta)$. Finally, since the number of queries $|K| = 2^{(1-\beta)m} \geq 2^{(\alpha\beta)m} = n^{\alpha}$ is no less than the number of queries in the assumed batch data structure, the running time per query is $n^{1-\gamma}$, and the total running time of the OnlineOV query algorithm for all queries is $|\mathcal{Q}| \cdot n^{1-\gamma} = 2^{(1-\beta\gamma)m} \leq 2^{(1-\varepsilon)m}$ by $\varepsilon \leq \beta\gamma$. □

Finally, we observe that since some problems admit sparsity-preserving reductions from OnlineOV, we can establish hardness results for their sparse variants in the sub-linear dimension regime from Theorem 5.3.3.

Corollary 5.3.4. *Under* NUSETH, for all constant $c, \alpha > 0$ and $\delta, \gamma \in (0, 1)$, there exists constant w such that no $(n^c, n^{1-\gamma})$-data structure with computationally unbounded preprocessing for any of the following problems with $d = n^\delta$.

w-Sparse-SubsetQuery$_{n,d}$, w-Sparse-ContainmentQuery$_{n,d}$, Monotone-w-DNF-Evaluation$_{n,d}$.

5.3.2 Space Lower Bounds from Non-Uniform HamPath

In this section, we show that an efficient data structure for 3-SUM with preprocessing would imply a non-uniform algorithm for HamPath with running time exponentially smaller than 2^n. First, we define the 3-SUM problem with preprocessing.

Definition 5.3.5. 3-SUM with preprocessing is a problem to be solved by a pair of deterministic algorithms $(\mathcal{P}, \mathcal{A})$ in two phases.

Preprocessing phase. In the first phase, the algorithm P receives k sets: $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$ each consisting of N integers, and preprocesses them into a data structure $\sigma \in \{0,1\}^S$ consisting of S bits.

Query phase. In the second phase, the algorithm $\mathcal{A}$ receives a query $\mathcal{X}'_1 \subseteq \mathcal{X}_1, \mathcal{X}'_2 \subseteq \mathcal{X}_2, \mathcal{X}'_3 \subseteq \mathcal{X}_3$, (query access to) the data structure σ, and runs in time T. $\mathcal{A}$ outputs 1 if there exist $x_1 \in \mathcal{X}'_1, x_2 \in \mathcal{X}'_2, x_3 \in \mathcal{X}'_3$ such that $x_1 + x_2 = x_k$, and it outputs 0 otherwise.

Such a data structure is called an (S, T)-data structure for k-SUM with preprocessing.

Theorem 5.3.6. *Under the Non-Uniform HamPath Conjecture, there is no* $(N^{1.088}, N^{1.088})$*-data structure with computationally unbounded preprocessing solving* 3-SUM.

Proof. Let G be an instance of HamPath on n vertices. If n is not a multiple of 3, assuming $r = n \bmod k$, we add to the graph a path on $(3 - r)$ vertices, and add an edge from each vertex of the original graph to the first vertex of the path. The new graph has a Hamiltonian path if and only if the original graph does. Thus, in the following we will assume that n is a multiple of 3.

Assume, towards a contradiction, that there is an $(N^{1.088}, N^{1.088})$-data structure $(\mathcal{P}, \mathcal{A})$ for 3-SUM with preprocessing. Then on input sets $\mathcal{X}_1, \dots \mathcal{X}_3$ of size N, the computationally unbounded $\mathcal{P}$ outputs a data structure $\sigma \in \{0,1\}^{N^\delta}$, and for every query $q = \{\mathcal{X'}_1 \subseteq \mathcal{X}_1, \mathcal{X'}_2 \subseteq \mathcal{X}_2, \mathcal{X'}_3 \subseteq \mathcal{X}_3\}$, $\mathcal{A}^\sigma(q)$ runs in time $T = N^\delta$ and outputs 1 if and only if there exist $x'_1 \in \mathcal{X'}_1, x'_2 \in \mathcal{X'}_2, x'_3 \in \mathcal{X'}_3$ such that $x'_1 + x'_2 = x'_3$.

We describe a (non-uniform) reduction from HamPath to 3-SUM with preprocessing. For a set $S \subseteq [n]$, let $\mathsf{Int}(S)$ denote the integer from $\{0, \dots, 2^n - 1\}$ whose binary representation is the indicator vector of S. Let $\mathcal{S}_{n/3}$ be the collection of all subsets of $[n]$ of size $n/3$. We run the preprocessing algorithm P for 3-SUM on the following sets of size $N = |\mathcal{S}_{n/3}|$.

$$\mathcal{X}_1, \mathcal{X}_2 = \{\mathsf{Int}(S) \colon S \in \mathcal{S}_{n/3}\} ,$$
$$\mathcal{X}_3 = \{2^n - 1 - \mathsf{Int}(S) \colon S \in \mathcal{S}_{n/3}\} .$$

We store the returned data structure $\sigma \in \{0,1\}^{N^\delta}$ as the non-uniform advice of our algorithm for HamPath. (We note that the sets $\mathcal{X}_i$ only depend on n and do *not* depend on a HamPath instance.)

Next, we describe a RAM algorithm solving HamPath for a given n-vertex graph using the advice string σ and the query algorithm $\mathcal{A}$ for 3-SUM with preprocessing. First, we use the dynamic programming algorithm of Proposition 5.1.9 to find all simple paths of length exactly $n/3$ in time $\binom{n}{\leq n/3} \cdot \mathrm{poly}(n) \leq 2^{nH(1/3)} \cdot \mathrm{poly}(n)$. For a

set $S \in \mathcal{S}_{n/3}$, and vertices $u, v \in [n]$, we define the following variable,

$$D[S,u,v] = \begin{cases} 1 & \text{if } u \in S \text{ and } \exists \text{ a simple path from } u \text{ to a neighbor of } v \text{ using all vertices in } S, \\ 0 & \text{otherwise.} \end{cases}$$

Using the fact that each Hamiltonian path can be partitioned into 3 disjoint simple paths of length $n/3$, we have the following. G has a Hamiltonian path if and only if there exists a set of 4 vertices $v_1, \ldots, v_4 \in [n]$ and a collection of 3 disjoint sets: $S_1, S_2, S_3 \in \mathcal{S}_{n/3}$ such that

$$\bigwedge_{i \in [3]} D[S_i, v_i, v_{i+1}] = 1 \ .$$

Next, we design n^4 3-SUM queries that will determine whether G has a Hamiltonian path. For every $q = (v_1, \ldots, v_4)$, we compute the following subsets:

$$\mathcal{X}_i^q = \{\mathsf{Int}(S) \colon S \in \mathcal{S}_{n/3},\ v_i \in S, \text{ and } D[S, v_i, v_{i+1}] = 1\} \text{ for } i \in [1,2] \ ,$$
$$\mathcal{X}_3^q = \{2^n - 1 - \mathsf{Int}(S) \colon S \in \mathcal{S}_{n/3},\ v_k \in S, \text{ and } D[S, v_3, v_4] = 1\} \ .$$

Since for every $i \in [3]$, $\mathcal{X}_3^q \subseteq \mathcal{X}_3$, we can use the query algorithm $\mathcal{A}$ for the 3-SUM with preprocessing to check if there exist $x_1 \in \mathcal{X}_1^q, x_2 \in \mathcal{X}_2^q, x_3 \in \mathcal{X}_3^q$ such that $x_1 + x_2 = x_3$. Or, equivalently, if there exist $v_1, \ldots, v_4 \in [n]$ and $S_1, S_2, S_3 \in \mathcal{S}_{n/3}$ such that

$$\mathsf{Int}(S_1) + \mathsf{Int}(S_2) + \mathsf{Int}(S_3) = 2^n - 1 \quad \text{and} \quad D[S_i, v_i, v_{i+1}] = 1 \text{ for all } i \in [3]. \tag{5.1}$$

The algorithm reports that there exists a Hamiltonian Path in G if and only if at least one of the 3-SUM queries returns 1,

$$\bigvee_{q=(v_1,\ldots,v_4) \in [n]^4} \mathcal{A}^\sigma(\mathcal{X}_1^q, \mathcal{X}_2^q, \mathcal{X}_3^q) \ .$$

This completes the reduction. We first argue the correctness of the reduction. Namely, we show that Equation (5.1) holds for one of the queries if and only if G

has a Hamiltonian Path. Let $(a_1, \ldots, a_n)$ be a Hamiltonian path in G. For $i \in [3]$, we set $v_i = a_{1+(i-1)n/3}$, and we set v_4 to be an arbitrary neighbor of a_n. Similarly, $S_i = \{a_{1+(i-1)n/3}, \ldots, a_{in/3}\}$ for every $i \in [3]$. Note that $D[S_i, v_i, v_{i+1}] = 1$ for all $i \in [3]$. Finally, since all S_i are disjoint, we have that $\mathsf{Int}(S_1)+\mathsf{Int}(S_2)+\mathsf{Int}(S_3) = 2^n-1$, which implies that Equation (5.1) holds for the query $q = (v_1, \ldots, v_4)$.

For the other direction, let us assume that Equation (5.1) holds for some $q = (v_1, \ldots, v_4)$ and S_1, S_2, S_3. We use the fact that the sum of 3 numbers, each having exactly $n/3$ ones in their binary representation, has n ones in the binary representation if and only if the supports of the ones are disjoint. Thus, Equation (5.1) implies that all S_i are disjoint. This, together with $D[S_i, v_i, v_{i+1}] = 1$ for all $i \in [3]$, gives us a Hamiltonian path in G.

We now analyze the complexity of the presented algorithm for HamPath. The non-uniform advice has length $N^\delta = 2^{\delta n H(1/3)} \cdot \mathrm{poly}(n)$. The running time of the algorithm is bounded by the running of the dynamic programming algorithm of Proposition 5.1.9 and n^4 queries to $\mathcal{A}$,

$$2^{nH(1/3)} \cdot \mathrm{poly}(n) + n^4 \cdot N^\delta = (2^{nH(1/3)} + 2^{\delta n H(1/3)}) \cdot \mathrm{poly}(n) = (2^{\delta n H(1/3)}) \cdot \mathrm{poly}(n) \; .$$

Plugging in $\delta = 1.088$, we have that $2^{\delta H(1/3) n} = 2^{0.9991n} \ll 2^n$. Thus, we get a faster than 2^n non-uniform algorithm for HamPath, and refute the Non-uniform Hamiltonian Path Conjecture. $\square$

Bibliography

[1] Scott Aaronson. Oracles Are Subtle But Not Malicious. In *CCC*, 2006. 37

[2] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA*, 2014. 64, 68

[3] Amir Abboud, Karl Bringmann, Nick Fischer, and Marvin Künnemann. The Time Complexity of Fully Sparse Matrix Multiplication. In *SODA*, 2024. 56

[4] Divesh Aggarwal, Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. Fine-grained hardness of CVP(P)—Everything that we can prove (and nothing else). In *SODA*, 2021. 12, 69

[5] Thomas D. Ahle and Jakob B. T. Knudsen. Subsets and supermajorities: Optimal hashing-based set similarity search. In *FOCS*, 2020. 64, 65

[6] Joah Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *SODA*, 2021. 48, 65

[7] Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In *FOCS*, 2019. 61, 62

[8] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *SODA*, 2025. 8, 48

[9] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *ICDT*, 2009. 56

[10] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, 2017. 65

[11] Alexandr Andoni, Shunhua Jiang, and Omri Weinstein. A framework for building data structures from communication protocols. In *STOC*, 2025. 10, 64, 65

[12] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 2006. 30

[13] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. 14, 15, 16, 25

[14] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *FOCS*, 2009. 13

[15] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962. 70

[16] Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Matrix Multiplication Verification Using Coding Theory. *RANDOM*, 2024. 47, 57

[17] Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Output-Sparse Matrix Multiplication Using Compressed Sensing. *arXiv preprint arXiv:2508.10250*, 2025. 47, 50

[18] Radu Berinde, Anna C. Gilbert, Piotr Indyk, Howard Karloff, and Martin J. Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *Allerton*, 2008. 50

[19] Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs. In *FOCS*, 2020. 61, 62

[20] Andreas Björklund. Determinant Sums for Undirected Hamiltonicity. In *FOCS*, 2010. 70

[21] Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *STOC*, 1999. 65

[22] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and Queries That Are Sufficient for Exact Learning. *Journal of Computer and System Sciences*, 1996. 37

[23] Jin-Yi Cai. $S_2P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 2007. 28, 35, 37, 44

[24] Jin-Yi Cai and Osamu Watanabe. On Proving Circuit Lower Bounds against the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 2004. 38

[25] Ran Canetti. More on BPP and the Polynomial-Time Hierarchy. *Information Processing Letters*, 1996. 27, 36

[26] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS*, 2016. 4, 12, 18, 69

[27] Venkatesan T. Chakaravarthy and Sambuddha Roy. Oblivious Symmetric Alternation. In *STACS*, 2006. 28, 38, 44

[28] Venkatesan T. Chakaravarthy and Sambuddha Roy. Arthur and Merlin as oracles. *Computational Complexity*, 2011. 37

[29] Timothy M. Chan. Speeding up the four Russians algorithm by about one more logarithmic factor. In *SODA*, 2014. 64

[30] Timothy M. Chan. Orthogonal range searching in moderate dimensions: KD trees and range trees strike back. In *SoCG*, 2017. 10, 64, 65

[31] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *STOC*, 2015. 13

[32] Timothy M. Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and more: Quickly derandomizing Razborov-Smolensky. In *SODA*, 2016. 64

[33] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman's trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM Counting, and more. In *STOC*, 2023. 13

[34] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *ICALP*, 2002. 11, 64, 65, 68

[35] Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *FOCS*, 2019. 4

[36] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: eliminating randomness at almost no cost. In *STOC*, 2021. 1

[37] Lijie Chen and Roei Tell. Hardness vs. randomness, revised: uniform, non-black-box, and instance-wise. *SIAM Journal on Computing*, 2024. 1

[38] Lijie Chen and Ryan Williams. An equivalence class for Orthogonal Vectors. In *SODA*, 2019. 63, 64, 68

[39] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *FOCS*, 2020. 4

[40] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric Exponential Time Requires Near-Maximum Circuit Size. In *STOC*, 2024. 4, 34, 36, 38, 42

[41] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *STOC*, 2023. 4, 29

[42] Yijia Chen, Kord Eickmeyer, and Jörg Flum. The exponential time hypothesis and the parameterized clique problem. In *IPEC*, 2012. 12, 69

[43] Yilei Chen and Jiatu Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. In *STOC*, 2024. 24

[44] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *STOC*, 2004. 64

[45] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *SIGMOD*, 2020. 56

[46] Peter Dixon, Aduri Pavan, Jason Vander Woude, and NV Vinodchandran. Pseudodeterminism: promises and lowerbounds. In *STOC*, 2022. 3

[47] Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *FOCS*, 2023. 48

[48] Zeev Dvir, Alexander Golovnev, and Omri Weinstein. Static data structure lower bounds imply rigidity. In *STOC*, 2019. 11

[49] Rūsiņš Freivalds. Fast probabilistic algorithms. In *MFCS*, 1979. 45

[50] Joel Friedman. A note on matrix rigidity. *Combinatorica*, 1993. 61, 62

[51] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range Avoidance for Constant Depth Circuits: Hardness and Algorithms. *RANDOM*, 2023. 24, 31

[52] Karthik Gajulapalli, Zeyong Li, and Ilya Volkovich. Oblivious Complexity Classes Revisited: Lower Bounds and Hierarchies. In *FSTTCS*, 2024. 24, 41, 42, 43

[53] Karthik Gajulapalli, Surendra Ghentiyala, Zeyong Li, and Sidhant Saraogi. Online Orthogonal Vectors Revisited. *SODA*, 2026. 66, 74, 76

[54] Karthik Gajulapalli, Surendra Ghentiyala, Zeyong Li, and Sidhant Saraogi. Downward self-reducibility in the total function polynomial hierarchy. *SODA*, 2026. 24, 33, 74

[55] Robert Ganian, Petr Hlineny, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower Bounds on the Complexity of MSO_1 Model-Checking. In *STACS*, 2012. 12, 69

[56] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *SODA*, 2017. 63

[57] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. 15

[58] Oded Goldreich and Avishay Tal. Matrix rigidity of random Toeplitz matrices. In *STOC*, 2016. 61, 62

[59] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *WADS*, 2017. 65

[60] Isaac Goldstein, Moshe Lewenstein, and Ely Porat. Orthogonal vectors indexing. In *ISAAC*, 2017. 10, 65

[61] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM*, 56(4):1–34, 2009. 50

[62] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. *ACM Transactions on Computation Theory*, 2022. 23, 29, 30

[63] Fred G. Gustavson. Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition. *ACM Transactions on Mathematical Software*, 1978. 56

[64] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann Publishers, 2011. 64

[65] Prahladh Harsha, Daniel Mitropolsky, and Alon Rosen. Downward Self-Reducibility in TFNP. *ITCS*, 2023. 33

[66] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210, 1962. 70

[67] Rahul Ilango, Jiatu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *STOC*, 2023. 24

[68] Russell Impagliazzo and Ramamohan Paturi. The Complexity of k-SAT. In *CCC*, 1999. 18

[69] Russell Impagliazzo and Avi Wigderson. P=BPP if E requires exponential circuits: derandomizing the XOR lemma. In *STOC*, 1997. 1, 2

[70] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? In *FOCS*, 1998. 18

[71] Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *arXiv:1401.5512*, 2014. 64

[72] Piotr Indyk. On approximate nearest neighbors in non-euclidean spaces. In *FOCS*, 1998. 64, 65, 68

[73] Piotr Indyk. *High-dimensional computational geometry*. PhD thesis, Stanford University, 2001. 65

[74] Mark A. Iwen and Craig V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Information Processing Letters*, 2009. 56

[75] Riko Jacob and Morten Stöckel. Fast Output-Sensitive Matrix Multiplication. In *ESA*, 2015. 56

[76] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *ICALP*, 2015. 4, 18

[77] Yonggang Jiang, Merav Parter, and Asaf Petruschka. New oracles and labeling schemes for vertex cut queries. *arXiv:2501.13596*, 2025. 65

[78] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, 2003. 1, 3

[79] Ravindran Kannan. Circuit-Size Lower Bounds and Non-Reducibility to Sparse Sets. *Information and Control*, 1982. 37

[80] Richard M. Karp and Richard J. Lipton. Some Connections between Nonuniform and Uniform Complexity Classes. In *STOC*, 1980. 37

[81] Shashwat Kasliwal, Adam Polak, and Pratyush Sharma. 3SUM in preprocessed universes: Faster and simpler. In *SOSA*, 2025. 13

[82] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, 2021. 4, 22, 25

[83] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *STOC*, 1999. 36

[84] Donald E. Knuth. *The art of computer programming. Vol. 3: Sorting and searching*. Addison-Wesley, 1973. 64

[85] Johannes Köbler and Osamu Watanabe. New Collapse Consequences of NP Having Small Circuits. *SIAM Journal on Computing*, 1998. 37

[86] Oliver Korten. The hardest explicit construction. In *FOCS*, 2022. 1, 4, 6, 23, 24

[87] Oliver Korten. Range Avoidance and the Complexity of Explicit Constructions. *Bulletin of EATCS*, 2025. 22

[88] Oliver Korten and Toniann Pitassi. Strong vs. Weak Range Avoidance and the Linear Ordering Principle. In *FOCS*, 2024. 33, 34, 35, 36

[89] Marvin Künnemann. On Nondeterministic Derandomization of Freivalds' Algorithm: Consequences, Avenues and Algorithmic Progress. In *ESA*, 2018. 8, 46, 47, 54, 55, 56, 57

[90] Konstantin Kutzkov. Deterministic algorithms for skewed matrix products. In *STACS*, 2013. 8, 56, 57

[91] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 1983. vii, 35

[92] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *FOCS*, 2012. 48

[93] François Le Gall. Faster rectangular matrix multiplication by combination loss analysis. In *SODA*, 2024. 48, 54

[94] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *SODA*, 2018. 48

[95] Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *STOC*, 2022. 2, 24

[96] Zeyong Li. Symmetric Exponential Time Requires Near-Maximum Circuit Size: Simplified, Truly Uniform. In *STOC*, 2024. 4, 34, 36, 38, 42

[97] Satyanarayana V. Lokam. Complexity lower bounds using linear algebra. *Foundations and Trends in Theoretical Computer Science*, 2009. 56

[98] Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *FOCS*, 2022. 65

[99] Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 1983. 48, 49

[100] Dieter van Melkebeek and Konstantin Pervyshev. A generic time hierarchy with one bit of advice. *Computational Complexity*, 2007. 44

[101] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017. 8

[102] Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for NP and NQP. In *STOC*, 2018. 4

[103] Sivaramakrishnan Natarajan Ramamoorthy and Cyrus Rashtchian. Equivalence of systematic linear data structures and matrix rigidity. In *ITCS*, 2020. 11

[104] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 1994. 1, 22

[105] Rasmus Pagh. Compressed matrix multiplication. In *ITCS*, 2012. 56

[106] Mihai Pătraşcu. *Lower bound techniques for data structures*. PhD thesis, Massachusetts Institute of Technology, 2008. 65

[107] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. 65

[108] Pavel Pudlák and Vojtech Rödl. Some combinatorial-algebraic problems from complexity theory. *Discrete Mathematics*, 1994. 61, 62

[109] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *FOCS*, 2022. 4, 29

[110] Ronald L. Rivest. *Analysis of associative retrieval algorithms*. PhD thesis, Stanford University, 1974. 64

[111] Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976. 64, 65

[112] Daniel S. Roche. What Can (and Can't) we Do with Sparse Polynomials? In *ISSAC*, 2018. 56

[113] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 1998. 27, 36

[114] Rahul Santhanam. Circuit Lower Bounds for Merlin-Arthur Classes. *SIAM Journal on Computing*, 2009. 37, 38

[115] Rahul Santhanam. The complexity of explicit constructions. *Theory of Computing Systems*, 2012. 22

[116] Claude E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 1949. 21

[117] Mohammad A. Shokrollahi, Daniel A. Spielman, and Volker Stemann. A remark on matrix rigidity. *Information Processing Letters*, 1997. 61, 62

[118] Michael Sipser. A complexity theoretic approach to randomness. In *STOC*, 1983. vii, 35

[119] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 2022. 8, 45

[120] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, 1977. 16, 17

[121] Dirk Van Gucht, Ryan Williams, David P. Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *PODS*, 2015. 56

[122] Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In *IPEC*, 2015. 18, 19, 63

[123] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *ICM*, 2018. 18, 63

[124] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, 2024. 48, 54, 55, 56

[125] N. V. Vinodchandran. A note on the circuit complexity of PP. *Theoretical Computer Science*, 2005. 37

[126] Emanuele Viola. Lower bounds for data structures with space close to maximum imply circuit lower bounds. *Theory of Computing*, 15(1):1–9, 2019. 11

[127] Nikhil Vyas and Ryan Williams. On oracles and algorithmic methods for proving lower bounds. In *ITCS*, 2023. 4

[128] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. 10, 63, 65

[129] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *STOC*, 2010. 4

[130] Ryan Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM (JACM)*, 2014. 4

[131] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions On Algorithms*, 2005. 54, 56