Worst-Case to Average-Case Reductions via Additive Combinatorics*

Vahid R. Asadi
vrasadi@uwaterloo.ca
Univesity of Waterloo
Canada

Alexander Golovnev
alexgolovnev@gmail.com
Georgetown University
USA

Tom Gur
tom.gur@warwick.ac.uk
Univesity of Warwick
UK

Igor Shinkar
ishinkar@sfu.ca
Simon Fraser University
Canada

ABSTRACT

We present a new framework for designing worst-case to average-case reductions. For a large class of problems, it provides an explicit transformation of algorithms running in time T that are only correct on a small (subconstant) fraction of their inputs into algorithms running in time $\widetilde{O}(T)$ that are correct *on all* inputs.

Using our framework, we obtain such efficient worst-case to average-case reductions for fundamental problems in a variety of computational models; namely, algorithms for matrix multiplication, streaming algorithms for the online matrix-vector multiplication problem, and static data structures for all linear problems as well as for the multivariate polynomial evaluation problem.

Our techniques crucially rely on additive combinatorics. In particular, we show a local correction lemma that relies on a new probabilistic version of the quasi-polynomial Bogolyubov-Ruzsa lemma.

CCS CONCEPTS

• **Theory of computation → Problems, reductions and completeness**; **Cell probe models and lower bounds**.

KEYWORDS

average-case complexity, matrix multiplication, data structures

ACM Reference Format:
Vahid R. Asadi, Alexander Golovnev, Tom Gur, and Igor Shinkar. 2022. Worst-Case to Average-Case Reductions via Additive Combinatorics. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3519935.3520041

*A full version of the paper [4] is available at ECCC

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3520041

1 INTRODUCTION

Worst-case to average-case reductions provide a method for transforming algorithms that can only solve a problem for a fraction of the inputs into algorithms that can solve the problem *for all* inputs.

For instance, consider one of the most fundamental algorithmic problems: matrix multiplication. Suppose we have an average-case algorithm ALG that can correctly compute the product $A \cdot B$ on an α-fraction of matrices $A, B \in \mathbb{F}^{n \times n}$; that is, $\Pr[\mathrm{ALG}(A, B) = A \cdot B] \geq \alpha$. Is it possible to use ALG to obtain an algorithm that computes $A \cdot B$ *for all* input matrices? A worst-case to average-case reduction will give a positive answer to this question, boosting the *success rate* α to 1, without incurring significant overhead. Of course, the same question can be asked with respect to any other computational problem.

In this paper, we study such reductions for average-case algorithms where the success rate α could be very small, such as in the %1 regime, and even when α tends to zero rapidly (i.e., for algorithms that are only correct on a vanishing fraction of their inputs). There are two natural perspectives in which we can view such reductions. On the one hand, they can provide a proof that a problem retains its hardness even in the average case. On the other hand, they provide a paradigm for designing worst-case algorithms, by first constructing algorithms that are only required to succeed on a small fraction of their inputs, and then using the reduction to obtain algorithms that are correct on all inputs.

Background and context. The study of the average-case complexity originates in the work of Levin [31], and followup works such as [8]. A long line of works established various barriers to designing worst-case to average-case reductions for **NP**-complete problems (see, e.g., [25, 26] and references therein). We refer the reader to the classical surveys by Impagliazzo [24], Bogdanov and Trevisan [11], and Goldreich [20] on this topic.

On the positive side, Lipton [32] proved that the matrix permanent problem admits a polynomial-time worst-case to average-case reduction. Ajtai [1] designed worst-case to average-case reductions for certain lattice problems, which led to constructions of efficient cryptographic primitives from worst-case assumptions [2, 35]. Other number-theoretic problems in cryptography have been long known to admit such reductions due to random self-reducibility: the discrete logarithm problem, the RSA problem, and the quadratic residuosity problem (see, e.g., [37]). For the matrix multiplication

problem, there is a weak reduction that requires the average-case algorithm to succeed with very high probability 3/4 (see Section 2.1). There are also known worst-case to average-case reductions for many problems that are not thought to be in **NP** [5, 18, 38].

Recently, the study of fine-grained complexity [39] of algorithmic problems sparked interest in designing *efficient* worst-case to average-case reductions for such problems as orthogonal vectors, 3SUM, online matrix-vector multiplication, k-clique, and others. Such reductions are motivated by fine-grained cryptographic applications. A large body of work is devoted to establishing fine-grained worst-case to average-case reductions for the k-clique problem, orthogonal vectors, 3SUM, and various algebraic problems, as well as building certain cryptographic primitives from them [6, 7, 12, 16, 21, 30]. Since there are no known constructions of one-way functions and public-key cryptography from well-established fine-grained assumptions, the question of constructing efficient worst-case to average-case reductions for other fine-grained problems still attracts much attention.

1.1 Our Contribution

We design a framework for showing explicit worst-case to average-case reductions, and we use it to obtain reductions for fundamental problems in a variety of computational models. Informally, we show that if a problem has an algorithm that runs in time T and succeeds on α-fraction of its inputs (even for sub-constant success rate α), then there exists a worst-case algorithm for this problem, which runs in time $\widetilde{O}(T)$. We design such reductions for the matrix multiplication problem in the setting of algorithms, for the online matrix-vector multiplication problem in the streaming setting, for *all* linear problems in the setting of static data structures, and for the problem of multivariate polynomial evaluation. We describe these results in detail below.

1.1.1 Algorithms for Matrix Multiplication. Recall that in the matrix multiplication problem, the goal is simply to compute the product of two given matrices $A, B \in \mathbb{F}^{n\times n}$. A long line of research, culminating in the work of Alman and Vassilevska Williams [3], led to matrix multiplication algorithms performing $O(n^{2.37286})$ operations. We present a worst-case to average-case reduction for the matrix multiplication problem over prime fields. Namely, we show that if there exists a (randomized) algorithm that, given two matrices $A, B \in \mathbb{F}^{n\times n}$, runs in time $T(n)$ and correctly computes their product for a small fraction of all possible inputs, then there exists a (randomized) algorithm that runs in $\widetilde{O}(T(n))$ time and outputs the correct answer for all inputs. Formally, we have the following theorem.

Theorem 1. *Let $\mathbb{F} = \mathbb{F}_p$ be a prime field, $n \in \mathbb{N}$, and $\alpha := \alpha(n) \in (0, 1]$. Suppose that there exists an algorithm* ALG *that, on input two matrices $A, B \in \mathbb{F}^{n\times n}$ runs in time $T(n)$ and satisfies*

$$\Pr[\mathsf{ALG}(A, B) = A \cdot B] \geq \alpha,$$

where the probability is taken over the random inputs $A, B \in \mathbb{F}^{n\times n}$ and the randomness of ALG.

- *If $|\mathbb{F}| \leq 2/\alpha$, then there exists a randomized algorithm* ALG′ *that for* every *input $A, B \in \mathbb{F}^{n\times n}$ and $\delta > 0$, runs in time $\frac{\exp(O(\log^5(1/\alpha)))}{\delta} \cdot T(n)$ and outputs AB with probability at least $1 - \delta$.*
- *If $|\mathbb{F}| \geq 2/\alpha$, then there exists a randomized algorithm* ALG′ *that for* every *input $A, B \in \mathbb{F}^{n\times n}$ and $\delta > 0$, runs in time $O(\frac{1}{\delta\cdot\alpha^4} \cdot T(n))$ and outputs AB with probability at least $1 - \delta$.*

For example, if we have an algorithm that succeeds on α fraction of the inputs for $\alpha > \exp(-\sqrt[6]{\log(n)})$ in time $T(n) = n^c$, then we get an algorithm that works for all inputs and runs in time $n^{c+o(1)}$. In particular, if we have an $n^{2+o(1)}$ algorithm that succeeds on $\alpha > \exp(-\sqrt[6]{\log(n)})$ fraction of the inputs, then there is a worst case algorithm with running time $n^{2+o(1)}$.

1.1.2 Data Structures for All Linear Problems. The class of linear problems plays a central role throughout computer science and mathematics, yielding a myriad of applications both in theory and practice. Our next contribution gives worst-case to average-case reductions for static data structures for *all linear problems*. Recall that a linear problem L_A over a field $\mathbb{F}$ is defined by a matrix $A \in \mathbb{F}^{m\times n}$.[1] An input to the problem is a vector $v \in \mathbb{F}^n$, which is preprocessed into s memory cells. Then, given a query $i \in [m]$, the goal is to output $\langle A_i, v\rangle$, where A_i is the i'th row of A, by probing at most t of the memory cells, where t is called the query time.

Note that the trivial solutions for data structure problems are to either: (i) store only $s = n$ memory cells containing the input v, and for each query $i \in [m]$, read v entirely and compute the answer in query time $t = n$; or (ii) use $s = m$ memory cells, where the i'th cell contains the answer to the query $i \in [m]$, thus allowing for query time $t = 1$. In a typical application, the number of queries $m = \mathrm{poly}(n) \gg n$, and a data structure is efficient if it uses space $s = \widetilde{O}(n)$ (or $s \ll m$) and has query time $t = \mathrm{poly}(\log(n))$ (or $t = n^\varepsilon$ for a small constant $\varepsilon > 0$). Note that the two trivial solutions do not lead to such efficient data structures for $m \gg n$.

We consider randomized data structures, where both the preprocessing stage and the query stage use randomness, and are expected to output the correct answer with high probability (over the randomness of both stages). In average-case randomized data structures, the success rate of the algorithm is taken over both the inner randomness and the random input, whereas in worst-case randomized data structure, the success rate is taken only over the inner randomness of the algorithm (i.e., the algorithm succeeds with high probability *on all* inputs).

We present a worst-case to average-case reduction showing that if there exists a data structure DS that uses s memory cells, has query time t, and success rate such that for a small fraction of inputs the data structure answers all queries correctly, then there exists another data structure DS′ that uses $4s$ memory cells, has query time $4t$, and success rate such that *for all inputs* the data structure answers all queries correctly with high probability.

Theorem 2. *Let $\mathbb{F} = \mathbb{F}_p$ be a prime field, $\alpha := \alpha(n) \in (0, 1]$, $n, m \in \mathbb{N}$, and a matrix $A \in \mathbb{F}^{m\times n}$. Denote by L_A the linear problem of outputting $\langle A_i, x\rangle$ on input $x \in \mathbb{F}^n$ and query $i \in [m]$. Suppose*

[1] Formally, L_A is defined by an infinite sequence of matrices $(A_n)_{n\geq 1}$, where $A_n \in \mathbb{F}^{m\times n}$ for $m = m(n)$.

that [2]

$$L_A \in \mathrm{DS}\left[\begin{array}{ll} pt: & p \\ mu: & s \\ qt: & t \\ sr: & \Pr_{x\in\mathbb{F}^n}[\mathrm{DS}_x(i) = \langle A_i, x\rangle \ \forall i \in [m]] \geq \alpha \end{array}\right].$$

Then for every $\delta > 0$,

$$L_A \in \mathrm{DS}\left[\begin{array}{ll} pt: & p + \exp(\log^4(1/\alpha)) \cdot \mathrm{poly}\log(1/\delta) \cdot \mathrm{poly}(n) \\ mu: & 4s + O(\log^4(1/\alpha)\log(n)) \\ qt: & 4t + O(\log^4(1/\alpha)\log(n)) \\ sr: & \forall x \in \mathbb{F}^n \ \Pr[\mathrm{DS}'_x(i) = \langle A_i, x\rangle \ \forall i \in [m]] \geq 1-\delta \end{array}\right].$$

We stress that in the average-case data structure we start with, the probability is taken over a random input (as well as the inner randomness of the algorithm), whereas in the worst-case data structure that we obtain, with high probably the algorithm is successful *on all* inputs.

The reduction above shows that for any linear problem L_A, if a data structure succeeds on an arbitrary small constant $\alpha > 0$ fraction of the inputs, then we can obtain a data structure that succeeds on all inputs with parameters that essentially differ only by a constant multiplicative factor, and the query complexity t translates into query complexity $4t + O(\log(n))$.

We note that the $O(\log^4(1/\alpha)\log(n))$ overhead in the space complexity of the constructed data structure is caused by storing $O(\log^4(1/\alpha))$ numbers from $[n]$. In particular, if the word size of the data structure is $w \geq \log(n)$, then the space complexity of the resulting data structure is $4s + O(\log^4(1/\alpha))$. Similarly, in this case the query complexity of the resulting data structure is $4t + O(\log^4(1/\alpha))$.

Note that for any non-trivial data structure problem, a data structure must use at least $\Omega(n)$ memory cells (only to store a representation of the input). Therefore, even for α as small as $\alpha = 2^{-n^\eta}$ for a small constant $\eta > 0$, the overhead in the space complexity is negligible. For typical query times of data structures, such as $t = \mathrm{poly}(\log(n))$ and $t = n^\varepsilon$, the overhead in the query time is negligible even for $\alpha = 1/\mathrm{poly}(n)$ and $\alpha = 2^{-n^\eta}$, respectively.

1.1.3 Online Matrix-Vector Multiplication. Next we turn to the core data structure problem in fine-grained complexity, the online matrix-vector multiplication problem (OMV). In the data structure variant of this streaming problem, one needs to preprocess a matrix $M \in \mathbb{F}^{n\times n}$, such that given a query vector $v \in \mathbb{F}^n$, one can quickly compute Mv. The study of OMV (over the Boolean semiring) and its applications to fine-grained complexity originates from [22], and [13, 29] give surprising upper bounds for the problem. Over finite fields, [15, 19] give lower bounds for OMV, and [14] proves lower bounds for a related vector-matrix-vector multiplication problem. We prove an efficient worst-case to average-case reduction for OMV over prime fields. A concurrent and independent work [23] studies worst-case to average-case reductions for OMV over the Boolean semiring and their applications.

Note that OMV is, in fact, *not* a linear problem, because for a query v the output is not a single field element, but rather a vector $Mv \in \mathbb{F}^n$. Moreover, the average case condition only guarantees success with probability taken over *both* the matrix M as well as the vector v. Nevertheless, we can exploit the fact that each coordinate of the correct output is a linear function in the entries of M, and extend our techniques to the more involved setting of OMV.

Theorem 3. *Let $\mathbb{F} = \mathbb{F}_p$ be a prime field, $n \in \mathbb{N}$, and $\alpha := \alpha(n) \in (0, 1]$. Consider the matrix-vector multiplication problem $OMV_\mathbb{F}$ for dimension n, and suppose that for some $\alpha > 0$ it holds that*

$$OMV_\mathbb{F} \in \mathrm{DS}\left[\begin{array}{ll} pt: & p \\ mu: & s \\ qt: & t \\ sr: & \Pr_{M,v}[\mathrm{DS}_M(v) = Mv] \geq \alpha \end{array}\right].$$

Then for every $\delta > 0$,

$$OMV_\mathbb{F} \in \mathrm{DS}\left[\begin{array}{ll} pt: & 4p + \exp(\log^4(1/\alpha)) \cdot \mathrm{poly}\log(1/\delta) \cdot \mathrm{poly}(n) \\ mu: & 4s + O(\log^4(1/\alpha)n) + O(n^2) \\ qt: & (4t+n) \cdot \mathrm{poly}(1/\alpha) \cdot \mathrm{poly}\log(1/\delta) \\ sr: & \forall M, v : \Pr[\mathrm{DS}_M(v) = Mv] \geq 1-\delta \end{array}\right].$$

We stress that in the assumed data structure, the success rate asserts that for *a random input M and query v*, the data structure produces the correct answer with (an arbitrary small) probability $\alpha > 0$, where the probability is over (i) the random input M (ii) random query v (iii) and the randomness of the preprocessing and the query phases of the data structure. On the other hand, the conclusion holds for worst case inputs and queries. That is, *for every* input M and query v, the obtained data structure produces the correct answer with high probability, where the probability is only over the randomness used in the preprocessing stage and the query phase of the data structure (i.e., with high probability we can compute *all* of the inputs).

To understand the parameters of the reduction, note that in the the OMV problem with $n \times n$ matrices, the preprocessing must be at least n^2, as this is the size of the input matrix, and the query time must be at least n, as information-theoretically we need to output n field elements. Our worst-case to average-case reduction is essentially optimal in these parameters for a constant α, as a weak data structure that uses s memory cells and query time t is translated into a data structure that works for all inputs and all queries using space $4s + O(n^2)$ and query time $4t + O(n) = O(t)$. In fact, even for α as small as $\alpha = 1/n^{o(1)}$, the space complexity is increased by at most $O(n^2)$, and the query time is multiplied by at most $n^{o(1)}$.

1.1.4 Worst-Case to Weak-Average-Case Reductions. In the following, we discuss how to obtain worst-case algorithms starting from a very weak, but natural, notion of average-case reductions that we discuss next.

Recall that in the standard definition of average-case data structures, the algorithm preprocesses its input and is then required to correctly answer all queries for an α-fraction of all possible inputs. However, in many cases (such as in the online matrix-vector multiplication problem), we only have an average-case guarantee on both inputs and queries. In this setting, we should first ask what is a natural notion of an average-case condition.

A strong requirement for an average-case algorithm in this case is to correctly answer *all queries* for at least α-fraction of the inputs. However, it is desirable to only require the algorithm to correctly answer on an *average input and query*. That is, a *weak average-case* data structure for computing a function $f : \mathbb{F}^n \times [m] \to \mathbb{F}$ with

[2] In our notation for data structures, *pt* stands for pre-processing time, *mu* stands for memory used, *qt* stands for query time, and *sr* stands for success rate.

success rate $\alpha > 0$ receives an input $x \in \mathbb{F}^n$, which is preprocessed into s memory cells. Then, given a query $i \in [m]$, the data structure $\mathsf{DS}_x(i)$ outputs $y \in \mathbb{F}^{n'}$ such that $\Pr_{x\in\mathbb{F}^n, i\in[m]}[\mathsf{DS}_x(i) = f(x,i)] \geq \alpha$.

The challenge in this setting is that the errors may be distributed between both the inputs and the queries. On one extreme, the error is concentrated on selected inputs, and then the data structure computes *all queries* correctly for α-fraction of the inputs. On the other extreme, the error is spread over all inputs, and then the data structure may only answer α-fraction of the queries on any inputs. Of course, the error could be distributed anywhere in between these extremes.

While we showed that every linear problem has an efficient worst-case to average-case reduction, not all linear (and non-linear) problems admit a worst-case to *weak*-average-case reductions (details of this part can be found in the full version of the paper [4, Section 6.3]). Nevertheless, we overcome this limitation for certain problems of interest.

One of the most-studied problems in static data structures is the polynomial evaluation problem [17, 27, 28]. Here, one needs to preprocess a degree-d polynomial $q\colon \mathbb{F}^m \to \mathbb{F}$ into s memory cells, and then for a query $x \in \mathbb{F}^m$, quickly compute $q(x)$. We study the problem of evaluating a low degree polynomial in the regime where the average-case data structure might only succeed on a small α fraction of the queries (outside of the unique decoding regime, see discussion below). We show that we can use such an average-case data structure to obtain a worst-case data structure that can compute q on any $x \in \mathbb{F}^m$.

Theorem 4. *Let $\mathbb{F} = \mathbb{F}_p$ be a prime field, $\alpha := \alpha(n) \in (0,1]$, and let $m, d \in \mathbb{N}$ be parameters. Consider the problem $\mathsf{RM}_{\mathbb{F},m,d}$ of evaluating polynomials of the form $q\colon \mathbb{F}^m \to \mathbb{F}$ of total degree d (i.e., the problem of evaluating the Reed-Muller encoding of block length $n = \binom{m+d}{d}$).*

Suppose that

$$\mathsf{RM}_{\mathbb{F},m,d} \in \mathsf{DS}\left[\begin{array}{ll} pt: & p \\ mu: & s \\ qt: & t \\ sr: & \Pr_{q,x}[\mathsf{DS}_q(x)] \geq \alpha \end{array}\right].$$

Then

$$\mathsf{RM}_{\mathbb{F},m,d} \in \mathsf{DS}\left[\begin{array}{ll} pt: & p + \exp(\log^4(1/\alpha)) \cdot \mathrm{poly}(n) \\ mu: & 4s + O(\log^4(1/\alpha)\log(n)) \\ qt: & O(|\mathbb{F}|^2 \cdot t + |\mathbb{F}|\log^4(1/\alpha) + |\mathbb{F}|\log(n)) \\ sr: & \forall q, x : \Pr[\mathsf{DS}_q(x) = q(x)] > 1 - O\left(\sqrt{\frac{d}{|\mathbb{F}|}}\right) \end{array}\right].$$

Here, similarly to Theorem 3, the assumed data structure succeeds only for a small fraction of inputs and queries, while in the conclusion the data structure succeeds with high probability on *every input* and *every query*.

As for the effect of the reduction on the parameters, we see that for any $\alpha > 1/\mathrm{poly}(n)$ the preprocessing time changes only by an additive $\mathrm{poly}(n)$, the space complexity changes from s to $4s + \mathrm{poly}(\log(n))$, and the query time changes from t to $O(|\mathbb{F}|^2 \cdot t + |\mathbb{F}| \cdot \mathrm{poly}\log(n))$. In the data structure setting, the number of queries is usually polynomial in input length. Thus, in a typical setting of parameters for $\mathsf{RM}_{\mathbb{F},m,d}$, the field size is $|\mathbb{F}| = \mathrm{poly}(\log n)$, and, therefore, the blow-up of $|\mathbb{F}|^2$ is not critical.

A coding-theoretic perspective. For small values of average-case rate $\alpha > 0$, the polynomial evaluation problem can be cast as *list decoding with preprocessing*, by viewing the outputs of the query phase of the data structure as a function $h\colon \mathbb{F}^m \to \mathbb{F}$ that agrees with the input polynomial $q\colon \mathbb{F}^m \to \mathbb{F}$ on some small fraction of the queries, and the goal is to recover q from h.

Indeed, note that for a small $\alpha > 0$, if a function $h\colon \mathbb{F}^m \to \mathbb{F}$ agrees with some unknown low-degree polynomial q on α fraction of the inputs, then there are potentially $O(1/\alpha)$ possible low-degree polynomials that are equally close to h. Hence, without preprocessing it is impossible to recover the original polynomial q. However, in the data structure settings, we can use the preprocessing to obtain an auxiliary structural information that would later allow us to transition from the list decoding regime to the unique decoding regime, and in turn, compute the values of the correct polynomial q with high probability (see more details in Section 2.4).

1.2 Open Problems

Our work leaves many natural open problems, such as obtaining reductions for various natural problems in other computational models (e.g., communication complexity, property testing, PAC learning, and beyond). However, for brevity, we would like to focus on and highlight one direction that we find particularly promising.

In Theorem 2, we design worst-case to average-case reductions for linear problems in the setting of static data structures. An immediate and alluring question is whether our local correction via additive combinatorics framework can also be used to show worst-case to average-case reductions for all linear problems for both circuits and uniform algorithms. We observe that using similar techniques as in Theorem 2, our framework can be used to show that given an efficient average-case circuit or uniform algorithm and an efficient *verifier* for the problem, one can indeed design an explicit efficient worst-case circuit or uniform algorithm. A natural open problem here is to eliminate the assumption about the verifier and answer the aforementioned question to the affirmative.

2 TECHNICAL OVERVIEW

We provide an overview of the main ideas and techniques that we use to obtain our results. For concreteness, we illustrate our techniques by first focusing on the matrix multiplication problem.

We start in Section 2.1, where we explain the challenge and discuss why the naive approach fails. In Section 2.2 we present the technical components that lie at the heart of this work: *local correction lemmas via additive combinatorics*. Equipped with these technical tools, in Section 2.3 we present the main ideas in our worst-case to average-case reduction for matrix multiplication. Finally, in Section 2.4 we briefly discuss how to obtain the rest of our main results.

2.1 The Challenge: Low-Agreement Regime

Recall that in the matrix multiplication problem we are given two matrices $A, B \in \mathbb{F}^n$, and the goal is to compute their matrix product $A \cdot B$. For simplicity of the exposition, unless specified otherwise, in this overview we restrict our attention to the field $\mathbb{F}_2$, and to constant values of the success rate parameter $\alpha > 0$ of average-case algorithms.

We would like to show that if there is an *average-case* algorithm ALG that can compute matrix multiplication for an α-fraction of all pairs of matrices $A, B \in \mathbb{F}^n$ in time $T(n)$, then there is a *worst-case* randomized algorithm ALG′ that runs in time $O(T(n))$ and computes $A \cdot B$ with high probability for *every* pair of matrices A and B.

We start with the elementary case where the average-case guarantee is in the *high-agreement regime*, i.e., where the algorithm succeeds on, say, 99% of the inputs; that is,

$$\Pr_{A,B\in\mathbb{F}^{n\times n}}[\mathrm{ALG}(A,B) = A\cdot B] \geq \alpha\,, \tag{1}$$

for $\alpha = 0.99$. In this case, a folklore local correction procedure (see, e.g., [10]) will yield a worst-case algorithm that succeeds with high probability on all inputs. We next describe this procedure.

Given an *average-case* algorithm ALG satisfying Eq. (1) with $\alpha = 0.99$, consider the worst-case algorithm ALG′ that receives any two matrices $A, B \in \mathbb{F}^{n\times n}$ and first samples uniformly at random two matrices $R, S \in \mathbb{F}^{n\times n}$. Next, writing $A = R + (A - R)$ and $B = S + (B - S)$, the algorithm ALG′ computes

$$M = \mathrm{ALG}(R,S)+\mathrm{ALG}(A-R,S)+\mathrm{ALG}(R,B-S)+\mathrm{ALG}(A-R,B-S)\,. \tag{2}$$

Denote by X the set of matrix pairs (A, B) for which $\mathrm{ALG}(A, B) = A \cdot B$, and recall that by Eq. (1) the density of X is 0.99. Note that: (a) the matrices R, $A-R$, S, and $B-S$ are uniformly distributed, and (b) if the pairs (R, S), $(A-R, S)$, $(R, B-S)$, and $(A-R, B-S)$ are in the set X, then by Eq. (2) we have $M = A \cdot B$, and the algorithm ALG′ computes the multiplication correctly. Hence, by a union bound we have $\Pr[M = AB] \geq 1 - 4 \cdot 0.01 > 0.9$ *for all* matrices $A, B \in \mathbb{F}^{n\times n}$. Of course, the error probability can be further reduced by repeating the procedure and ruling by majority.

Unfortunately, this argument breaks when the average-case guarantee is weaker; namely, in the *low-agreement regime*, where the algorithm succeeds on, say, only 1% of the inputs. Here, when trying to self-correct as above, the vast majority of random choices would lead to a wrong output, and so at a first glace, the self-correction approach may seem completely hopeless.[3]

Nevertheless, using more involved tools from additive combinatorics such as a probabilistic version of the quasi-polynomial Bogolyubov-Ruzsa lemma that we show, as well as tools such as small-biased sample spaces and the Goldreich-Levin algorithm, we can construct different local correction procedures that work in the *low-agreement regime*. We proceed to describe our framework for local correction using the aforementioned tools.

2.2 Local Correction via Additive Combinatorics

Additive combinatorics studies approximate notions of algebraic structures via the perspective of combinatorics, number theory, harmonic analysis and ergodic theory. Most importantly for us, it provides tools for transitioning between algebraic and combinatorial notions of approximate subgroups with only a small loss in the underlying parameters (see surveys [33, 34]).

The starting point of our approach for local correction is a fundamental result in additive combinatorics, known as *Bogolyubov's lemma*, which shows that the 4-ary sumset of any dense set in $\mathbb{F}_2^n$ contains a large linear subspace. More accurately, recall that the sumset of a set X is defined as $X + X = \{x_1 + x_2 : x_1, x_2 \in X\}$, and similarly $4X = \{x_1 + x_2 + x_3 + x_4 : x_1, x_2, x_3, x_4 \in X\}$. These quantities can be thought of as quantifying a combinatorial analogue of an approximate subgroup. Bogolyubov's lemma states that for any subset $X \subseteq \mathbb{F}_2^n$ of density $|X|/2^n \geq \alpha$, there exists a subspace $V \subseteq 4X$ of dimension at least $n - \alpha^{-2}$.

We will show that statements of the above form can be used towards obtaining a far stronger local correction paradigm than the one outlined in Section 2.1. To see the initial intuition, consider an average-case algorithm that is guaranteed to correctly compute α-fraction of the inputs, and denote by X the set of these correctly computed inputs. Then $|X|/2^n \geq \alpha$, and Bogolyubov's lemma shows that there exists a large subspace V such that every $v \in V$ can be expressed as a sum of four elements in X, each of which can be computed correctly by the average-case algorithm.

The approach above suggests a paradigm for local correction, however, there are several non-trivial problems in implementing this idea. For starters, how could we handle inputs that lay *outside of the subspace* V? To name a few others: how can we amplify the success probability in the low-agreement regime? How do we algorithmically obtain the decomposition? Can we handle finite fields beyond $\mathbb{F}_2^n$? How do we handle average-case where the success rate α is sub-constant?

Indeed, for our worst-case to average-case reductions, we will need local correction lemmas with stronger structural properties than those admitted by Bogolyubov's lemma, as well as new ideas for each one of the settings. In the following, we discuss the main hurdles for the foregoing approach and the tools that are needed to overcome them, leading to our main technical tool, which is a probabilistic version of the quasi-polynomial Bogolyubov-Ruzsa lemma that we obtain. Then, we present our framework for local correction using these techniques. Finally, in Sections 2.3 and 2.4 we show the additional ideas that are necessary for applying the local correction lemmas in the settings of matrix multiplication, online matrix-vector multiplication, and data structures.

A probabilistic Bogolyubov lemma. An immediate problem with the aforementioned local correction scheme is that while Bogolyubov's lemma asserts that *there exists* a decomposition of each input into a sum of four elements in X, it does not tell us how to obtain this decomposition.

Toward this end, we further show that each vector $v \in V$ has many "representations" as a sum of four elements from X. This way, for any $v \in V$ we can efficiently sample a representation $v = x_1+x_2+x_3+x_4$, where each $x_i \in X$. More accurately, let $X \subseteq \mathbb{F}_2^n$ be a set of density α, let $R = \{r \in \mathbb{F}^n \setminus \{0\} : \left|\hat{1}_X(r)\right| \geq \alpha^{3/2}\}$, and let $V = \{v \in \mathbb{F}^n : \langle v, r\rangle = 0\ \forall r \in R\}$ be a linear subspace defined by R. Then $|R| \leq 1/\alpha^2$ and for all $v \in V$ it holds that

$$\Pr_{x_1,x_2,x_3}[x_1, x_2, x_3, v - x_1 - x_2 - x_3 \in X] \geq \alpha^5\,.$$

[3] Indeed, consider the counterexample where the average-case algorithm $\mathrm{ALG}(A, B)$ outputs $A \cdot B$ in case the first element of A is 0 and returns the zero matrix in case the first element of A is 1. Note that in this case $\Pr_{A,B\in\mathbb{F}^{n\times n}}[\mathrm{ALG}(A, B) = A \cdot B] \geq 1/2$, yet no decomposition of $A = \sum_i A_i$ and $B = \sum_i B_i$ as described above could self-correct matrix multiplication where the first element of A is 1. Indeed, any such composition would have an A_i with the first element 1, where $\mathrm{ALG}(A_i, B_j)$ fails.

Sparse-shift subspace decomposition. The probabilistic Bogolyubov lemma allows us to locally correct inputs inside the subspace $V \subseteq 4X$. However, we need to be able to handle any vector in the field. Towards that end, we show an algebraic lemma that allows us to decompose each element of the field into a sum of an element v in the subspace V and a *sparse* shift-vector s. More accurately, let $R \subseteq \mathbb{F}^n \setminus \{\vec{0}\}$ and $V = \{v \in \mathbb{F}^n : \langle v, r\rangle = 0 \ \forall r \in R\}$. We show that there exists a collection of $t \leq |R|$ vectors $B = \{b_1, \ldots, b_t\}$, $b_i \in \mathbb{F}^n$ and indices $k_1, \ldots, k_t \in [n]$ such that $\mathrm{span}(B) = \mathrm{span}(R)$ and every vector $y \in \mathbb{F}^n$ can be written as $y = v + s$, where $v \in V$ and $s = \sum_{j=1}^{t} c_j \cdot \vec{e}_{k_j}$ for $c_j = \langle y, b_j \rangle$ and $\vec{e}_{k_j}$ is a unit vector.

We stress that the sparsity of the decomposition is essential to our applications, as we cannot locally correct the shift part of the decomposition, and instead we need to compute it explicitly. We remark that for matrix multiplication we can obtain a stronger guarantee by dealing with matrices outside of the subspace V via a low-rank random matrix shifts (see Section 2.3).

Subspace computation via the Goldreich-Levin lemma. In order to perform local correction using additive combinatorics machinery as above while maintaining computational efficiency, we need to be able to compute the aforementioned basis $b_1, \ldots, b_t \in \mathbb{F}^n$ and indices $k_1, \ldots, k_t \in [n]$ efficiently. We note that, in essence, this problem reduces to learning the heavy Fourier coefficients of the set X. Thus, using ideas from [9] and an extension of the Goldreich-Levin algorithm to arbitrary finite fields, we can perform the latter in a computationally efficient way.

Probabilistic quasi-polynomial Bogolyubov-Ruzsa lemma. The main weakness of Bogolyubov's lemma is that the co-dimension of the subspace that it admits is polynomial in $1/\alpha$, where α is the success rate of the average-case algorithm. While this dependency on α allows us to locally correct in the 1% agreement regime, it becomes degenerate when α tends to 0 rapidly.

A natural first step towards overcoming this barrier is to use a seminal result due to Sanders [36], known as the *quasi-polynomial Bogolyubov-Ruzsa lemma*, which shows the existence of a subspace whose co-dimension's dependency on $1/\alpha$ is *exponentially* better. That is, the lemma shows that for a set $X \subseteq \mathbb{F}_2^n$ of size $\alpha \cdot |\mathbb{F}_2|^n$, where $\alpha \in (0, 1]$, there exists a subspace $V \subseteq \mathbb{F}_2^n$ of dimension $\dim(V) \geq n - O(\log^4(1/\alpha))$ such that $V \subseteq 4X$. However, as in the case of Bogolyubov's lemma, we have the problem that the statement is only *existential*.

We thus prove a probabilistic version of the quasi-polynomial Bogolyubov-Ruzsa lemma (see the full version [4]) over any field $\mathbb{F} = \mathbb{F}_p$, which asserts that for an α-dense set $X \subseteq \mathbb{F}^n$, there exists a subspace $V \subseteq \mathbb{F}^n$ of dimension $\dim(V) \geq n - O(\log^4(1/\alpha))$ such that for all $v \in V$ it holds that

$$\Pr_{x_1, x_2, x_3 \in \mathbb{F}^n}[x_1, x_2 \in A, x_3, x_4 \in -A] \geq \Omega(\alpha^5) \ ,$$

where $x_4 = v - x_1 - x_2 - x_3$. Furthermore, by combining the techniques above, we show that given a query access to the set X, there is an algorithm that runs in time $\exp(\log^4(1/\alpha)) \cdot \mathrm{poly}\log(1/\delta) \cdot \mathrm{poly}(n)$ and with probability $1-\delta$ computes a set of vectors $R \subseteq \mathbb{F}^n$ such that $V = \{v \in \mathbb{F}^n : \langle v, r\rangle = 0 \ \forall r \in R\}$.

We are grateful to Tom Sanders for providing us with the argument for showing this lemma, and we provide the proof in [4, Appendix A].

Our local correction lemma. We are now ready to provide an informal statement of our local correction lemma, which builds on the machinery above, and in particular, on the probabilistic quasi-polynomial Bogolyubov-Ruzsa lemma.

Loosely speaking, our local correction allows us to decompose any vector $y \in \mathbb{F}^n$ as a linear combination of the form

$$y = x_1 + x_2 - (x_3 + x_4) + s \ ,$$

where $x_1, x_2, x_3, x_4 \in X$ and $s \in \mathbb{F}^n$ is a *sparse* vector.

Lemma 2.1 (informally stated, see [4, Lemma 3.4])**.** *For a field $\mathbb{F} = \mathbb{F}_p$ and α-dense set $X \subseteq \mathbb{F}^n$, there exists $t \leq 1/\alpha^2$ vectors $b_1, \ldots, b_t \in \mathbb{F}_2^n$ and indices $k_1, \ldots, k_t \in [n]$ satisfying the following. Given a vector $y \in \mathbb{F}_2^n$, let $s = \sum_{j=1}^{t} \langle y, b_j \rangle \cdot \vec{e}_{k_j}$ we have*

$$\Pr_{x_1, x_2, x_3 \in \mathbb{F}^n}[x_1, x_2 \in X, x_3, x_4 \in -X] \geq \Omega(\alpha^5) \ ,$$

where $x_4 = y - s - x_1 - x_2 - x_3$.

Furthermore, given an oracle that computes $1_X(x)$ with probability at least $2/3$, there exists an algorithm that makes $\exp(\log^4(1/\alpha)) \cdot \mathrm{poly}\log(1/\delta) \cdot \mathrm{poly}(n)$ oracle calls and field operations, and with probability at least $1-\delta$ outputs $b_1, \ldots, b_t$ and $k_1, \ldots, k_t$.

The aforementioned local correction lemmas lie at the heart of our average-case to worst-case reductions, which we discuss next.

2.3 Illustrating Example: Matrix Multiplication

We present a high-level overview of our reductions for matrix multiplication, which illustrates the key ideas that go into the proof. Let ALG be an average-case algorithm that can compute matrix multiplication for an α-fraction of all pairs of matrices $A, B \in \mathbb{F}^n$ in time $T(n)$. We use the *average-case* algorithm ALG to construct a *worst-case* randomized algorithm ALG′ that runs in time $O(T(n))$ and computes $A \cdot B$ with high probability for *every* pair of matrices A and B. For simplicity of the exposition, in this overview we make the following assumptions: (1) the algorithm ALG is *deterministic*, (2) the input is a pair (A, B) such that A is a matrix satisfying $\Pr_{B'}[\mathsf{ALG}(A, B') = A \cdot B'] \geq \alpha$, (3) the success rate α is a constant, and (4) the field $\mathbb{F}$ is $\mathbb{F}_2$.

We start by noting two simple facts. First, given the algorithm's (potentially wrong) output $\mathsf{ALG}(A, B)$, we can efficiently check whether the computation is correct using Freivalds' algorithm. Second, denoting by $X = \{B' \in \mathbb{F}_2^{n \times n} : \mathsf{ALG}(A, B') = A \cdot B'\}$ the set of "good" matrices, we have that if $B \in X$, then the average-case algorithm correctly outputs $\mathsf{ALG}(A, B) = A \cdot B$. Hence, the main challenge is in dealing with the case that $B \notin X$, in which we need to locally correct the value of the multiplication.

Local correction via Bogolyubov's lemma. The first idea is to reduce the problem to the case where the set of good matrices contains a large subspace, and hence admits local correction, as discussed in Section 2.2. Specifically, by the probabilistic Bogolyubov lemma, given X we can choose a subspace $V \subseteq \mathbb{F}_2^{n \times n}$ of matrices, where $\dim(V) \geq n^2 - 1/\alpha^2$, such that for any $B' \in V$, if we sample

M_1, M_2, M_3 uniformly at random, then

$$\Pr[M_1, M_2, M_3, M_4 \in X] \geq \alpha^5 \ , \text{ where } M_4 = B' - M_1 - M_2 - M_3 \ .$$

Note that if the matrices M_1, M_2, M_3, M_4 produced by our sampling are all in the set of good matrices X, then we can self-correct the value of $\mathsf{ALG}(A, B')$ by evaluating $\{\mathsf{ALG}(A, M_i)\}_{i\in[4]}$ and computing the linear combination

$$\sum_{i=1}^{4} \mathsf{ALG}(A, M_i) = \sum_{i=1}^{4} A \cdot M_i = A \cdot (\sum_{i=1}^{4} M_i) = A \cdot B' \ .$$

Note that this event is only guaranteed to occur with probability α^5, which is far smaller than $1/2$. Nevertheless, since we can verify the computation using Freivalds' algorithm, we can boost this probability to be arbitrarily close to 1 by repeating the random sampling step $O(1/\alpha^5)$ times, each time computing $\sum_{i=1}^{4} \mathsf{ALG}(A, M_i)$ and verifying if the obtained result is indeed correct using Freivalds' algorithm. Therefore, if B belongs to the (unknown) subspace V, then the algorithm described above indeed computes $A \cdot B$ with high probability in time $O(T(n)/\mathrm{poly}(\alpha)) = O(T(n))$.

However, the approach above does not work for matrices B that do not lie in the subspace V described above. To deal with this case, our next goal is to "shift" the matrix into the subspace V using low-rank random shifts, which can then be computed efficiently and used for local correction. We describe this procedure next.

Low-rank random matrix shifts. We start by making the following key observation: if we have an arbitrary matrix A, and a matrix $B \in \mathbb{F}^{n\times n}$ of rank k, then their product AB can be computed in time $O(kn^2)$, given a rank-k decomposition of B. Details follow.

To see this, suppose that the first k columns of B denoted by $(B_i)_{i=1}^{k}$, are linearly independent, and for each of the remaining $n-k$ columns $(B_j)_{j=k+1}^{n}$, we know the linear combination $B_j = \sum_{i=1}^{k} d_{i,j} \cdot B_i$ for some coefficients $d_{i,j} \in \mathbb{F}$. We can first multiply A by each of the k linearly independent columns of B. Then, to compute the remaining columns, for each $i = 1, \ldots, k$ let $C_i = A \cdot B_i$ be the i'th column of the matrix $C = AB$, and observe that if $B_j = \sum_{i=1}^{k} d_{i,j} B_i$, then $C_j = A \cdot B_j = A \cdot (\sum_{i=1}^{k} d_{i,j} B_i) = \sum_{i=1}^{k} d_{i,j} \cdot C_i$, which can be computed in $O(kn)$ time for each j. Therefore the total running time of multiplying A by B is $O(kn^2)$.

We are now ready to describe our method for shifting the matrices into the subspace V using low-rank matrices, capitalizing on the observation above. Given the matrix B (that is, possibly, not in V), we sample a random matrix $R_B \in \mathbb{F}^{n\times n}$ of rank $2k$ by randomly choosing $2k$ columns and filling them with uniformly random field elements. Note that with high probability these $2k$ columns are linearly independent. Then, we let the rest of the columns be random linear combinations of the first $2k$ columns we chose. We observe that if $\dim(V) = n - k$, then

$$\Pr[B + R_B \in V] \geq \frac{1}{2|\mathbb{F}|^k} \ .$$

If indeed $B + R_B \in V$, then we can compute $A \cdot (B + R_B)$ using the procedure discussed above, by writing $B + R_B$ as a sum of 4 random matrices $B + R_B = M_1 + M_2 + M_3 + M_4$, applying $\mathsf{ALG}(A, M_i)$ for each $i = 1 \ldots 4$, and using Freivalds' algorithm to efficiently check if the produced output is correct or not.

Note that since we have a lower bound on the probability that $B + R_B$ belongs to the desired subspace, we have an upper bound on the expected number of attempts required until this event occurs. When we obtain such low-rank matrix shifts, which we verify using Freivalds' algorithm, we proceed by computing $A \cdot R_B$. Since R_B is a matrix of rank at most $2k$, the total running time of this will be $O(kn^2)$. Finally, we return

$$\mathsf{ALG}(A, B + R_B) - A \cdot R_B \ ,$$

which indeed produces the correct answer assuming that $\mathsf{ALG}(A, B + R_B)$ is correct.

Remark 2.2. *The discussion above made the simplifying assumption that the inputs we are getting are pairs (A, B) such that A is a matrix satisfying $\Pr_{B'}[\mathsf{ALG}(A, B') = A \cdot B'] \geq \alpha$. The actual proof require also handling the inputs for which the matrix A does not satisfy this requirement, which is done using similar ideas by applying the local correction procedure first to A and then to B.*

2.4 Beyond Matrix Multiplication

We conclude the technical overview by briefly sketching some of the key ideas in the rest of our worst-case to average-case reductions, building on the local correction lemmas outlined in Section 2.2. Below we assume that all data structures are deterministic, but by standard techniques this assumption is without loss of generality. We start with the simplest setting, and then proceed to the more involved ones.

Worst-case to average-case reductions for all linear data structure problems. The setting here is the closest to that of matrix multiplication. Let DS_A be an average-case data structure for a linear problem defined by A, where we preprocess an input vector x and the answer to query i is $\langle A_i, x\rangle$, and A_i is the i'th row of A.

Given a vector $y \in \mathbb{F}^n$, we use our local correction lemma to obtain a decomposition of the form $y = x_1 + x_2 - (x_3 + x_4) + v$, where $x_1, x_2, x_3, x_4 \in X$ (i.e., on which $DS_{x_j}(i) = \langle A_i, x_j\rangle$ for all i) and a sparse shift vector $v = \sum_{j=1}^{t} \langle y, b_j\rangle \cdot \vec{e}_{k_j}$. We then preprocess each of the x_j's by applying DS_A to it, and we also compute $\langle A_i, v\rangle$ efficiently by using its sparse representation. The idea is that by the linearity of the problem, we can locally correct according to $\sum_{j=1}^{4} DS_{x_j}(i) + \langle A_i, v\rangle$.

It is important to note that, unlike in the setting of matrix multiplication, we cannot use the random low-rank matrix shifts, nor Freivald's algorithm for verification. However, this is where we rely on the sparse subspace decomposition to shift the input into the subspace V implied by the quasi-polynomial Bogolyubov-Ruzsa lemma. In addition, instead of relying on Freivalds' algorithm for verification, here we use the guarantee about the correctness of computation in the subspace V together with the sparsity of the shift vector, which allows us to correct its corresponding contribution via explicit computation. See details in [4, Section 6.1].

Online matrix-vector multiplication (OMV).. The online setting of the OMV problem poses several additional challenges. Recall that in the average-case reductions above, the input is a vector $x \in \mathbb{F}_2^n$, each query is a coordinate $i \in [n]$, and the matrix $M \in \mathbb{F}_2^{n\times n}$ is a hard-coded parameter. In the OMV problem, the matrix M is the *input*, the vector x is the *query*, and answer to a query is not a scalar but rather

a *vector*. Hence we need to use a two-step local correction where we first decompose the matrix and then decompose the vector. Observe that we can use our additive combinatorics mechanism to preprocess the matrix M and get a description of the subspace V that it asserts, as well as the formula that is required to compute the shift vector s given x, but the problem is that here we cannot preprocess x, as it arrives online. Thus, in the query phase, when the algorithm receives x, we want to find the decomposition $x = x_1+x_2+x_3+x_4+s$. We then compute the shift vector s, and then sample x_i's whose sum is $x - s$. However this leaves us with the task of checking that all of the x_i's are computed correctly. To this end, we rely on a generalization of *small-bias sample spaces* to finite fields in order to obtain an efficient verification procedure. See details in [4, Section 5].

Weak-average-case reductions. As discussed in the introduction, in the setting of weak-average-case we cannot expect a reduction for all linear problems. In turn, this leads to substantially different techniques. We concentrate on the multivariate polynomial evaluation problem. Here, we are given a polynomial $p\colon \mathbb{F}^m \to \mathbb{F}$ of degree d, where for simplicity, in this overview we fix the parameters $d = \log(n)$, $|\mathbb{F}| = \mathrm{poly}(\log(n))$, and $m = \log(n)/\log\log(n)$, so that we encode n field elements using a codeword of length $\mathrm{poly}(n)$, and the distance is $1 - dm/|\mathbb{F}| > 0.99$. The polynomial is given as input by its $n = d^m$ coefficients, the queries are of the form $x \in \mathbb{F}^m$, and the goal is to output $p(x)$. The key difficulty here, is that for small values of the average-case rate $\alpha > 0$, we need to be able to deal with the *list decoding regime* (see discussion in Section 1.1.4).

The first step is to rely on our additive combinatorics local correction tools similarly as in the OMV reduction. Here the idea is to preprocess the polynomial p and obtain a decomposition of the form $p = p_1 + p_2 + p_3 + p_4 + s$, where again s is a sparse shift-vector. We then construct a data structure for each p_i. However, since we cannot process the queries $x \in \mathbb{F}^m$, we are left with the task of locally correcting the noisy polynomials $\{p_i\}$. If a polynomial p_i is only slightly corrupted (i.e., within the unique decoding regime), we can easily locally correct it without using any preprocessing. However, we also need to deal with noisy polynomials p_i in the *list decoding regime* in which only α-fraction of the points are evaluated correctly, for an arbitrarily small α.

We overcome the difficulty above by capitalizing the preprocessing power of the data structure. Namely, we will show how to boost the success probability from the list-decoding regime to the unique-decoding regime, in which case we can perfectly correct the polynomial via the local correction of the Reed–Muller code. The key idea is that by the generalized Johnson bound, there is only a list of $O(1)$ codewords that agree with the average-case data structure on at least $\alpha/2$-fraction of the points. We thus fix a reference point $w \in \mathbb{F}^m$ and explicitly compute the correct value of $p_i(w)$. Next, we sample a random point r and query the points of line $\ell_{x,w}$ incident to r and the reference point z. Then, we consider the list (of size $O(1)$) of all low-degree univariate polynomials that agree with the queried points on $\ell_{x,w}$, and trim the list by removing each polynomial that does not agree on the reference point. Using the sampling properties of lines in multivariate polynomials, we can show that answering accordingly to the remaining polynomials in the list would yield the right value with high probability.

ACKNOWLEDGMENTS

We are grateful to Tom Sanders for providing a sketch of the proof of the probabilistic version of the quasi-polynomial Bogolyubov-Ruzsa lemma. We would also like to thank Shachar Lovett and Tom Sanders for discussions regarding the quasi-polynomial Bogolyubov-Ruzsa lemma. Tom Gur is supported by the UKRI Future Leaders Fellowship MR/S031545/1.

REFERENCES

[1] Miklós Ajtai. 1996. Generating hard instances of lattice problems. In *STOC 1996*. 99–108.
[2] Miklós Ajtai and Cynthia Dwork. 1997. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC 1997*. 284–293.
[3] Josh Alman and Virginia Vassilevska Williams. 2021. A refined laser method and faster matrix multiplication. In *SODA 2021*. SIAM, 522–539.
[4] Vahid R. Asadi, Alexander Golovnev, Tom Gur, and Igor Shinkar. 2022. Worst-Case to Average-Case Reductions via Additive Combinatorics. *Electronic Colloquium on Computational Complexity* (2022). https://eccc.weizmann.ac.il/report/2022/020
[5] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. 1993. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity* 3, 4 (1993), 307–318.
[6] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2017. Average-case fine-grained hardness. In *STOC 2017*. 483–496.
[7] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2018. Proofs of work from worst-case assumptions. In *CRYPTO 2018*. Springer, 789–819.
[8] Shai Ben-David, Benny Chor, Oded Goldreich, and Michel Luby. 1992. On the theory of average case complexity. *J. Comput. System Sci.* 44, 2 (1992), 193–219.
[9] Eli Ben-Sasson, Noga Ron-Zewi, Madhur Tulsiani, and Julia Wolf. 2014. Sampling-Based Proofs of Almost-Periodicity Results and Algorithmic Applications. In *ICALP 2014*. Springer, 955–966.
[10] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. 1990. Self-Testing/Correcting with Applications to Numerical Problems. In *STOC 1990*. ACM, 73–83.
[11] Andrej Bogdanov and Luca Trevisan. 2006. Average-case complexity. *Foundations and Trends in Theoretical Computer Science* 2, 1 (2006), 1–106.
[12] Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. 2019. The Average-Case Complexity of Counting Cliques in Erdős–Rényi Hypergraphs. In *FOCS 2019*. IEEE, 1256–1280.
[13] Diptarka Chakraborty, Lior Kamma, and Kasper Green Larsen. 2018. Tight cell probe bounds for succinct boolean matrix-vector multiplication. In *STOC 2018*. 1297–1306.
[14] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. 2018. Simulation beats richness: New data-structure lower bounds. In *STOC 2018*. 1013–1020.
[15] Raphael Clifford, Allan Grønlund, and Kasper Green Larsen. 2015. New unconditional hardness results for dynamic and online problems. In *FOCS 2015*. IEEE, 1089–1107.
[16] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. 2020. New techniques for proving fine-grained average-case hardness. In *FOCS 2020*. IEEE, 774–785.
[17] Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová. 2021. Data Structures Lower Bounds and Popular Conjectures. In *ESA 2021*. LIPIcs.
[18] Joan Feigenbaum and Lance Fortnow. 1993. Random-self-reducibility of complete sets. *SIAM J. Comput.* 22, 5 (1993), 994–1005.
[19] Gudmund Skovbjerg Frandsen, Johan P. Hansen, and Peter Bro Miltersen. 2001. Lower bounds for dynamic algebraic problems. *Information and Computation* 171, 2 (2001), 333–349.
[20] Oded Goldreich. 2011. Notes on Levin's Theory of Average-Case Complexity. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. Springer, 233–247.
[21] Oded Goldreich and Guy Rothblum. 2018. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *FOCS 2018*. IEEE, 77–88.
[22] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC 2015*. 21–30.
[23] Monika Henzinger, Andrea Lincoln, and Barna Saha. 2021. The Complexity of Average-Case Dynamic Subgraph Counting. *Electronic Colloquium on Computational Complexity* (2021).
[24] Russell Impagliazzo. 1995. A personal view of average-case complexity. In *CCC 1995*. IEEE, 134–147.
[25] Russell Impagliazzo. 2011. Relativized separations of worst-case and average-case complexities for NP. In *CCC 2011*. IEEE, 104–114.
[26] Russell Impagliazzo and Leonid A. Levin. 1990. No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In *FOCS 1990*. IEEE.

[27] Kiran S. Kedlaya and Christopher Umans. 2008. Fast modular composition in any characteristic. In *FOCS 2008*. IEEE, 146–155.
[28] Kasper Green Larsen. 2012. Higher cell probe lower bounds for evaluating polynomials. In *FOCS 2012*. IEEE, 293–301.
[29] Kasper Green Larsen and Ryan Williams. 2017. Faster online matrix-vector multiplication. In *SODA 2017*. SIAM, 2182–2189.
[30] Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. 2019. Public-key cryptography in the fine-grained setting. In *CRYPTO 2019*. Springer, 605–635.
[31] Leonid A. Levin. 1986. Average case complete problems. *SIAM J. Comput.* 15, 1 (1986), 285–286.
[32] Richard Lipton. 1991. New directions in testing. *Distributed computing and cryptography* 2 (1991), 191–202.
[33] Shachar Lovett. 2015. An Exposition of Sanders' Quasi-Polynomial Freiman-Ruzsa Theorem. *Theory of Computing* (2015), 1–14.
[34] Shachar Lovett. 2017. Additive combinatorics and its applications in theoretical computer science. *Theory of Computing* (2017), 1–55.
[35] Oded Regev. 2004. New lattice-based cryptographic constructions. *J. ACM* 51, 6 (2004), 899–942.
[36] Tom Sanders. 2012. On the Bogolyubov–Ruzsa lemma. *IEEE Transactions on Information Theory* 5, 3 (2012), 627–655.
[37] Victor Shoup. 2009. *A computational introduction to number theory and algebra*. Cambridge.
[38] Madhu Sudan, Luca Trevisan, and Salil Vadhan. 2001. Pseudorandom generators without the XOR lemma. *J. Comput. System Sci.* 62, 2 (2001), 236–266.
[39] Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *ICM 2018*. World Scientific.