# Algorithms for Sparse Convolution and Sublinear Edit Distance

**Nick Fischer**

A dissertation submitted towards the degree *Doctor of Natural Sciences* of the Faculty of Mathematics and Computer Science of Saarland University

Saarbrücken, 2023

# Colloquium

| | |
|---|---|
| **Date** | 29/08/2023 |
| **Dean of the Faculty** | Univ.-Prof. Dr. Jürgen Steimle |
| **Chair of the Committee** | Prof. Danupon Nanongkai, Ph.D. |
| **Reporters** | Prof. Dr. Karl Bringmann |
| | Prof. Dr. Robert Krauthgamer |
| | Prof. Dr. Timothy Chan |
| **Academic Assistant** | Dr. Karol Węgrzycki |

# Abstract

In this PhD thesis on fine-grained algorithm design and complexity, we investigate output-sensitive and sublinear-time algorithms for two important problems.

1. *Sparse Convolution:* Computing the convolution of two vectors is a basic algorithmic primitive with applications across all of Computer Science and Engineering. In the *sparse* convolution problem we assume that the input and output vectors have at most $t$ nonzero entries, and the goal is to design algorithms with running times dependent on $t$. For the special case where all entries are nonnegative, which is particularly important for algorithm design, it is known since twenty years that sparse convolutions can be computed in near-linear randomized time $O(t \log^2 n)$.

   In this thesis we develop a randomized algorithm with running time $O(t \log t)$ which is *optimal* (under some mild assumptions), and the first near-linear *deterministic* algorithm for sparse nonnegative convolution. We also present an application of these results, leading to seemingly unrelated fine-grained lower bounds against distance oracles in graphs.

2. *Sublinear Edit Distance:* The edit distance of two strings is a well-studied similarity measure with numerous applications in computational biology. While computing the edit distance exactly provably requires quadratic time, a long line of research has lead to a constant-factor approximation algorithm in almost-linear time. Perhaps surprisingly, it is also possible to approximate the edit distance $k$ within a large factor $O(k)$ in *sublinear time* $\widetilde{O}(\frac{n}{k} + \mathrm{poly}(k))$.

   We drastically improve the approximation factor of the known sublinear algorithms from $O(k)$ to $k^{o(1)}$ while preserving the $O(\frac{n}{k} + \mathrm{poly}(k))$ running time.

# Zusammenfassung

In dieser Doktorarbeit über feinkörnige Algorithmen und Komplexität untersuchen wir ausgabesensitive Algorithmen und Algorithmen mit sublinearer Laufzeit für zwei wichtige Probleme.

1. *Dünne Faltungen:* Die Berechnung der Faltung zweier Vektoren ist ein grundlegendes algorithmisches Primitiv, das in allen Bereichen der Informatik und des Ingenieurwesens Anwendung findet. Für das *dünne* Faltungsproblem nehmen wir an, dass die Eingabe- und Ausgabevektoren höchstens $t$ Einträge ungleich Null haben, und das Ziel ist, Algorithmen mit Laufzeiten in Abhängigkeit von $t$ zu entwickeln. Für den speziellen Fall, dass alle Einträge nicht-negativ sind, was insbesondere für den Entwurf von Algorithmen relevant ist, ist seit zwanzig Jahren bekannt, dass dünn besetzte Faltungen in nahezu linearer randomisierter Zeit $O(t \log^2 n)$ berechnet werden können.

   In dieser Arbeit entwickeln wir einen randomisierten Algorithmus mit Laufzeit $O(t \log t)$, der (unter milden Annahmen) optimal ist, und den ersten nahezu linearen deterministischen Algorithmus für dünne nichtnegative Faltungen. Wir stellen auch eine Anwendung dieser Ergebnisse vor, die zu scheinbar unverwandten feinkörnigen unteren Schranken gegen Distanzorakel in Graphen führt.

2. *Sublineare Editierdistanz:* Die Editierdistanz zweier Zeichenketten ist ein gut untersuchtes Ähnlichkeitsmaß mit zahlreichen Anwendungen in der Computerbiologie. Während die exakte Berechnung der Editierdistanz nachweislich quadratische Zeit erfordert, hat eine lange Reihe von Forschungsarbeiten zu einem Approximationsalgorithmus mit konstantem Faktor in fast-linearer Zeit geführt. Überraschenderweise ist es auch möglich, die Editierdistanz $k$ innerhalb eines großen Faktors $O(k)$ in *sublinearer* Zeit $\widetilde{O}(\frac{n}{k} + \text{poly}(k))$ zu approximieren.

   Wir verbessern drastisch den Approximationsfaktor der bekannten sublinearen Algorithmen von $O(k)$ auf $k^{o(1)}$ unter Beibehaltung der $O(\frac{n}{k}+\text{poly}(k))$-Laufzeit.

# Acknowledgements

My deepest thanks go to my advisor Karl Bringmann. He is an excellent teacher, and keeping up with his speed has spurred me on and made me grow a lot. He never pressured me, but offered me all the opportunities and help to explore whatever I wanted. All in all, I cannot think of a better advisor than Karl!

I would also like to thank my other coauthors, especially Alejandro Cassis, Marvin Künnemann and Vasileios Nakos. I had a lot of fun, both academically and non-academically, with Alejandro as my office mate and fellow PhD student. Marvin and Vasileios have taught me much, for which I am very grateful.

Studies aside, I would like to thank my amazing family and friends for making these last years a fantastic time, and for reminding me from time to time that life is not (at all) just about computer science. I especially thank my brother Dan and my parents Joma and Martin, on whom I can always rely. I would also like to thank Lennart, Frederik and many more friends for all the fun evenings with and without discussions on computer science. A very big thank you goes to my wonderful girlfriend Leo; in general, and in particular for designing together this style sheet. Finally, special credit goes to my grandfather Alois who already passed away—he taught 6-year-old me about basic math, which I believe is a main reason for my passion today.

# Preface

In this thesis I present five publications related to sparse convolutions [58, 61, 60, 2] and sublinear edit distance [56]. During my PhD I have worked on some other projects besides these five. In [98] we have proved that it is #P-hard to compute certain mathematical constants called *plethysm coefficients*; this problem is of importance in algebraic complexity theory. In [59] we have established a fine-grained classification of a natural class of polynomial-time decision problems, inspired by Schaefer's seminal classification theorem. Then, in two follow-up papers [55, 54], we have generalized this class to optimization problems and have proved several completeness and classification results. As an extension to our sublinear-time algorithm for edit distance [56], in [57] we have designed an algorithm that first *preprocesses* one of the strings (or both strings) in almost-linear time and then approximates the edit distance in time $(n/k+k)^{1+o(1)}$ (or time $k^{1+o(1)}$, respectively).

This is a complete list of my publications at the time of submitting this thesis (including my undergraduate thesis [97]):


2 Amir Abboud, Karl Bringmann, and Nick Fischer. "Stronger 3-SUM lower bounds for approximate distance oracles via additive combinatorics". In: *55th annual ACM symposium on theory of computing (STOC 2023)*. To appear. ACM, 2023. 10.48550/arXiv.2211.07058.

54 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. "A structural investigation of the approximability of polynomial-time problems". In: *49th international colloquium on automata, languages, and programming (ICALP 2022)*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pages 30:1–30:20. 10.4230/LIPIcs.ICALP.2022.30.

55 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. "Fine-grained completeness for optimization in P". In: *24th international conference on approximation, randomization, and combinatorial optimization (APPROX/RANDOM 2021)*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pages 9:1–9:22. 10.4230/LIPIcs.APPROX/RANDOM.2021.9.

56 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Almost-optimal sublinear-time edit distance in the low distance regime". In: *54th annual ACM symposium on theory of computing (STOC 2022)*. ACM, 2022, pages 1102–1115. 10.1145/3519935.3519990.

57 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Improved sublinear-time edit distance for preprocessed strings". In: *49th international colloquium on automata, languages, and programming (ICALP 2022)*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pages 32:1–32:20. 10.4230/LIPIcs.ICALP.2022.32.

58 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. "Faster minimization of tardy processing time on a single machine". In: *47th international colloquium on automata, languages, and programming (ICALP 2020)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pages 19:1–19:12. 10.4230/LIPIcs.ICALP.2020.19.

59 Karl Bringmann, Nick Fischer, and Marvin Künnemann. "A fine-grained analogue of schaefer's theorem in P: dichotomy of $\exists^k\forall$-quantified first-order graph properties". In: *34th computational complexity conference (CCC 2019)*. Vol. 137. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pages 31:1–31:27. 10.4230/LIPIcs.CCC.2019.31.

60 Karl Bringmann, Nick Fischer, and Vasileios Nakos. "Deterministic and Las Vegas algorithms for sparse nonnegative convolution". In: *33rd annual ACM-SIAM symposium on discrete algorithms (SODA 2022)*. SIAM, 2022, pages 3069–3090. 10.1137/1.9781611977073.119.

61 Karl Bringmann, Nick Fischer, and Vasileios Nakos. “Sparse nonnegative convolution is equivalent to dense nonnegative convolution”. In: *53rd annual ACM symposium on theory of computing (STOC 2021)*. ACM, 2021, pages 1711–1724. 10.1145/3406325.3451090.

97 Nick Fischer and Rob van Glabbeek. “Axiomatising infinitary probabilistic weak bisimilarity of finite-state behaviours”. In: *J. log. algebraic methods program.* 102 (2019), pages 64–102. 10.1016/j.jlamp.2018.09.006.

98 Nick Fischer and Christian Ikenmeyer. “The computational complexity of plethysm coefficients”. In: *Comput. complex.* 29.2 (2020), pages 8. 10.1007/s00037-020-00198-4.

# Content

# 1 Introduction

The overarching goal of the theory of computation is to classify each problem as either *easy* or *hard*—in terms of running time, space complexity or other resource requirements. Our strategies to prove easiness and hardness differ: On the one hand, *algorithm design* sets out to prove upper bounds by finding better and better algorithms. On the other hand, the goal of *complexity theory* is to prove hardness of problems either unconditionally or, more commonly, by means of reductions. Algorithm design and complexity theory often work hand-in-hand—for instance, it is not uncommon that a failed attempt for a hardness reduction turns into a surprising algorithm.

Historically, the most successful notion of easy and hard is *polynomial-time solvability* and *NP-hardness*: A problem is easy if it has a polynomial-time algorithm and hard if a polynomial-time algorithm for the problem would contradict the important P $\neq$ NP conjecture by means of a polynomial-time reduction. This classical view on algorithms and complexity has lead to a rich landscape of easy and hard problems. A major drawback of this approach is the inherent insensitivity with respect to different polynomial running times. For instance, we cannot distinguish "easy" problems with best-possible linear running time $\Theta(n)$ from "easy" problems with best-possible running time $\Theta(n^{100})$. However, these running times differ significantly, both from a theoretical and a practical perspective.

**From Coarse-Grained to Fine-Grained.** This is the starting point and motivation of the relatively new field *fine-grained complexity theory*, with the goal to pinpoint the exponents of polynomial-time problems. This typically means that a problem can be solved in time $O(n^{\alpha})$ for some exponent $\alpha$ (often by a naive algorithm), but that any algorithm with running time $O(n^{\alpha-\epsilon})$ with $\epsilon > 0$ would contradict a fine-grained hypothesis.[1]

[1] *Note that these hardness results leave room for sub-polynomial $n^{o(1)}$ algorithmic improvements. And indeed, for many central fine-grained problems it is known how to achieve substantial savings up to $2^{\sqrt{\log n}} = n^{o(1)}$ [201, 72].*

Beginning ten years ago, in a couple of landmark results this field has successfully determined the best-possible running time of many important problems. The list includes string problems such as Edit Distance [28] and Longest Common Subsequence [1, 63], geometric problems such as Collinearity testing [101] and Fréchet Distance [53], graph problems including computing the graph radius [5] and approximating the graph diameter [183], and many more. See [202] for a detailed survey. In even more recent times the theory is applied also to dynamic algorithms, quantum algorithms and cryptography.

Similar to the theory of NP-hardness, these aforementioned hardness results are conditioned on a small set of believable hypotheses, mostly concerning the fundamental Satisfiability, 3-SUM and APSP problems. The fine-grained analogue of polynomial-time reductions are *fine-grained reductions* that pay particular attention to the running time overhead due to the reduction.

**From Fine-Grained to Finer-Grained.** This leaves us with an acceptable state of affairs for most problems. But again, there are "easy" problems that are "easier" than others. For instance, the established fine-grained view on algorithms does not distinguish problems solvable in linear time $O(n)$, in near-linear time $\widetilde{O}(n) = n(\log n)^{O(1)}$ or in almost-linear time $n^{1+o(1)}$. Can we distinguish these running times, or does the story end here?

The perhaps most obvious way to obtain further speed-ups is to improve the lower-order factors. In most cases, this means reducing the number of log-factors which is often called the *art of log-shaving* [69]. On the complexity side, it is possible to define analogous *finer-grained* reductions that preserve the number of log-factors. These reductions are much less common though—unfortunately, while there are several well-established hypotheses about the polynomial running times of problems, we are lacking hypotheses about the exact number of log-factors. One

rare exception is the hypothesis that computing the Discrete Fourier Transform requires time $\Omega(n \log n)$, and as one of the main results in this thesis, we present a finer-grained reduction under this hypothesis (see Theorem 1.2). Another example of finer-grained reductions in the literature, with a different flavor, is that shaving a certain number of log-factors from the $O(n^2)$ running time of computing the edit distance between two strings leads to new circuit lower bounds [6].

Putting lower-order factors aside, can we obtain more substantial improvements? It seems that the story indeed ends here, as most problems of size $n$ require time $\Omega(n)$ (simply to read the input or to write the output). Therefore, in most cases an almost-linear time algorithm is optimal (up to lower-order factors). In this thesis, we focus on two natural situations where designing faster algorithms is nevertheless possible.

## 1.1 Output-Sensitive Algorithms

One way to achieve algorithms faster than linear time is to consider cases where input and output are sufficiently *sparse*. When the input is sparsely encoded with size *in* and the output is sparsely encoded with size *out*, the former lower bound becomes $\Omega(in + out)$, which, in many cases, is much faster than $\Omega(n)$ for worst-case input and output-size. Input-sensitive algorithms are omnipresent, and it is similarly natural to consider output-sensitive algorithms.

**Input-Sensitive Algorithms.** Representing inputs sparsely is so natural that we rarely ever explicitly say "input-sensitive" algorithms. For graph problems, for example, the dense representation by *adjacency matrices* inherently leads to algorithms with running time $\Omega(n^2)$, while many graphs in theory and practice have significantly fewer edges, $m \ll n^2$. In these numerous cases, the more reasonable approach is to represent graphs sparsely by *adjacency lists* which allows many problems to be solved in almost-linear time $m^{1+o(1)}$ (such as Single-Source Shortest Paths using Dijkstra's algorithm, or Maximum Flow [78], to name some fundamental examples).

Other examples of inputs that are commonly represented sparsely are matrices, polynomials, databases and even strings (such as the *sparse* pattern matching problem [84]).

**Output-Sensitive Algorithms.** It is equally natural, but slightly less common, to consider problems where the output is encoded sparsely. Obviously, for problems with negligibly small output size (in particular, for decision problems) it does not make sense to think about output-sensitive algorithms. Instead, think about problems where the output is at least as large as the input. Typical examples include listing or enumeration problems, for which it is desirable, especially from a practical perspective, to achieve running times of the form $O(f(in) + out)$.

One example is to list, in a given graph, all subgraphs of a certain pattern. For instance, we can list all 4-cycles in time $\widetilde{O}(m^{4/3} + out)$ [130, 7]. (As a side result of this thesis we prove a matching conditional lower bound in Theorem 1.12.)

Another natural example is (a variant of) the well-studied *Subset Sum* problem: Given a set of $n$ positive numbers $x_0, \ldots, x_{n-1}$ and a target $\tau$, list all numbers $x \in [0\,..\,\tau]$ that can be expressed as subset sum $x = \sum_{i \in I} x_i$ for some $I \subseteq [n]$. This problem requires time $\Omega(\tau)$ just to produce the worst-case output. However, for instances where the output list is significantly smaller than $\tau$, the problem can be solved in time $O(n + out^{4/3})$ [66].

The first major part of this thesis is concerned with the fundamental problem of multiplying two integer polynomials $A, B$ of degree $n$. Using the Fast Fourier Transform, we can compute the product in near-linear time $\widetilde{O}(n)$, and this problem clearly requires time $\Omega(n)$ both for reading the input and writing the output. However, when the input and output polynomials are represented sparsely (as a list of nonzero coefficients), it is in fact possible to multiply polynomials in

time $\widetilde{O}(in+out)$ [84, 164]. We also call this the *sparse (integer) convolution* problem. In the upcoming sections, we will thoroughly treat sparse convolutions in several nuanced settings.

## 1.2 Sublinear-Time Algorithms

In the previous section we have surveyed several examples of algorithms running in better-than-worst-case time for sparse inputs (and outputs). However, even sparsity-sensitive algorithms can never beat the unconditional barrier $\Omega(in+out)$. Recall that for *exact* problems changing a single bit in the input may generally lead to drastically different outputs, and therefore any correct algorithm necessarily has to probe all input bits.

For *approximations* on the other hand, it is a priori not clear whether the same lower bound applies: Changing a single bit in the input may lead to a *different*, but *approximately equal* output. In some cases, it is therefore possible to read the input only partially and obtain *truly sublinear* algorithms. The field investigating what approximations can be achieved in sublinear time is called *property testing* [113, 112].[2]

[2] *The typical view on property testing is slightly different: A property testing problem is to decide whether an input satisfies a property or is "far" from satisfying the property, in the sense that a significant fraction of input bits must change to satisfy the property.*

A toy example is to count the number of 1's in a bit-vector. Any exact algorithm requires $n$ probes, where $n$ is the length of the vector. However, using only a constant number of queries into the vector, we can compute the number of 1's up to an additive approximation error $\epsilon n$, for any constant $\epsilon > 0$.

Perhaps surprisingly, property testing can be successfully applied to much more difficult tasks, say in the context of string problems. Consider for instance the *Longest Increasing Subsequence* (LIS) problem: Given a string over an ordered alphabet, determine the length of the longest increasing subsequence, that is, the longest (not necessarily consecutive) subsequence with increasing characters. It is well-known that this problem can be solved in time $O(n \log n)$ where $n$ is the length of the given string [156], and any algorithm requires time $\Omega(n)$ to read the input. But is there hope for faster *approximation* algorithms? The answer depends on the length $k$ of the LIS. Note that $k = 1$ if and only if the given string is monotonically decreasing. Therefore, any better-than-2 approximation must test whether the input is monotonically decreasing which requires $\Omega(n)$ probes. More generally, computing a constant-factor (or subpolynomial $n^{o(1)}$-factor) approximation requires time $\Omega(n/k)$ (up to subpolynomial factors, respectively). We can therefore only hope to get sublinear-time algorithms for super-constant $k$. And indeed, as the final result in a series of papers [187, 185, 160, 166, 23], Andoni, Nosatzki, Sihna and Stein [23][3] recently proved that an $n^{o(1)}$-approximation of the LIS can be computed in almost-optimal time $O(n/k)$.

[3] *To obtain this statement, guess $k$ up to a factor 2 and plug in $\lambda = kn^{o(1)-1}$ into their main theorem [23]. For a sufficiently large $o(1)$ term, the running time is $O(n/k)$ and the approximation factor is $\lambda^{-o(1)} \leq n^{o(1)}$. Their theorem requires that $\lambda = o(1)$, so if instead $\lambda = \Omega(1)$ the algorithm can simply report "n" which is an $n^{o(1)}$-approximation of the length of the LIS.*

As the second direction of this thesis, we study the related problem of approximating the *edit distance* of two given strings in sublinear time. We similarly achieve that an $n^{o(1)}$-approximation can be computed in time $O(n/k + \text{poly}(k))$, where $k$ is the edit distance of the given strings (see Theorem 1.15). This is sublinear for sufficiently small $k$ and almost-optimal in this regime by a similar argument as for LIS.

**Organization.** In the following sections we describe our results in more detail. Section 1.3 is concerned with our output-sensitive algorithms for sparse convolution. In Sections 1.4 and 1.5 we present two applications of fast (sparse) convolution algorithms in fine-grained complexity and algorithm design. Section 1.6 states our result for sublinear-time edit distance.

## 1.3 Sparse Convolution

The first major contribution of this thesis is a collection of algorithms for the sparse convolution problem. We begin with some context on how computing (dense) convolutions finds applications in algorithm design and beyond.

**Dense Convolution.** The *convolution* of two integer vectors $A, B$ is the integer vector $A \star B$ which is defined coordinate-wise by $(A \star B)[k] = \sum_{i+j=k} A[i] \cdot B[j]$. Computing convolutions of integer vectors $A, B$ is a fundamental computational primitive, which arises in several disciplines of science and engineering. It has been a vital component in fields like signal processing, deep learning (convolutional neural networks) and computer vision. As computing convolutions of integer vectors is in fact the same problem as multiplying two integer polynomials, this is also one of the most central problems in computer algebra.

In a breakthrough result from 1965, Cooley and Tukey [85] discovered the fundamental *Fast Fourier Transform (FFT)* which solves the convolution problem in near-linear time $O(n \log n)$, where $n$ is the length of $A$ and $B$.[4] This algorithm had revolutionary consequences for all the aforementioned areas, and was praised as one of the "top-10 algorithms of the 20th century" [88]. Inside algorithm design the FFT became the algorithmic thrust for various state-of-the-art algorithms. The list includes many "additive problems" such as $k$-SUM [71], Subset Sum [52, 142, 129, 66, 64, 27], but also string problems [96, 10, 124, 84] and many others.

For these applications it often suffices to deal with *nonnegative convolution*, where the vectors $A, B$ have nonnegative entries. In fact, for many applications it suffices to solve the even simpler *Boolean convolution* problem—here, the vectors $A, B$ have 0–1 entries and the task is to compute the vector $A \circledast B$ with entries $(A \circledast B)[k] = \bigvee_{i+j=k} A[i] \wedge B[j]$. To convey a feeling on how computing Boolean and nonnegative convolutions can be handy in algorithms design, we present details on three typical applications.

**Application 1: Sumsets.** The Boolean convolution problem of length-$n$ vectors is equivalent to the computation of the *sumset* $X + Y = \{x + y : x \in X, y \in Y\}$ of two given sets $X, Y \subseteq [n]$.[5] This interpretation shows up often in $k$-SUM [71, 2, 130] and Subset Sum [52, 142, 66, 64] algorithms (for instance, the central 3-SUM problem can be seen as the problem of testing $0 \in X + Y + Z$ for three given sets $X, Y, Z$).

We later present a scenario where an algorithm for structured 3-SUM constitutes the backbone of a series of conditional hardness results against seemingly unrelated graph problems; see Section 1.4 (with details in Chapter 5).

**Application 2: String Algorithms.** Another successful application of nonnegative convolution is in the area of string problems, specifically pattern matching problems. This connection was first discovered by Fischer and Paterson [96]. The possibly simplest example is the *Sliding-Window Hamming Distance* problem over small (say, binary) alphabets [10]. In this problem the task is to compute the *Hamming distance* (i.e., the number of mismatching characters) from a short pattern $P \in \{0,1\}^m$ to all possible length-$m$ windows in a longer text $T \in \{0,1\}^n$. By appropriately encoding $P$ and $T$ into bit-vectors of length $O(n)$, we can read off the Hamming distances to all windows from the convolution vector.

**Application 3: The Partition-and-Convolve Design Paradigm.** As another systematic application, Boolean and nonnegative convolution form the essential ingredients to the *partition-and-convolve* design paradigm. The typical task for a problem approachable by a partition-and-convolve algorithm is to check for solutions of *all prescribed sizes* $k$. The standard approach for these problems is to come up with an appropriate dynamic program. The improved idea behind the partition-and-convolve paradigm is to partition the search space into (usually) two parts, each of which is solved recursively. In this way, a size-$k$ solution to the original problem is split into two parts of sizes $i, j$ such that $i + j = k$. Therefore, to

[4] *In fact, the FFT evaluates the discrete Fourier transform (DFT) in near-linear time. However, it is known that computing convolutions of vectors where the entries are nonnegative integers, integers and complex numbers, and the computation of DFTs are computationally equivalent, as each one can be reduced to the other. The nonnegativity assumption can be removed by appropriately increasing all entries. For the equivalence of computing convolutions and DFTs we remark that it is standard to express convolutions using DFT and inverse DFT, and the reverse direction is known as well [49] (assuming complex exponentials can be evaluated in constant time). See also [109, pp. 213–215].*

[5] *Simply take the vectors $A, B$ to be the indicator vectors of the sets $X, Y$.*

check whether there exists a size-$k$ solution to the original problem we recombine the recursive computations by a Boolean convolution. This approach is very flexible and can also be applied to other convolution-type problems; for instance, using nonnegative convolutions in place of Boolean convolutions corresponds to *counting* solutions of all prescribed sizes.

With an even more flexible adaption of this approach, we may even attempt to speed-up the computation of more general dynamic programs (DPs). The procedure is simple: Figure out how we can combine, say, two independent halves of the DP table in one shot. Hopefully the combination rule looks like a convolution problem—possibly with operations other than the usual integer addition and multiplication. If that is the case, it remains to design a nontrivial algorithm for the new convolution-type problem. In Section 1.5 (with details in Chapter 6) we give an example where this approach succeeded to design faster algorithms for a scheduling problem.

**Faster Than Fast Fourier Transform?** Given these various applications, it is natural and important to ask whether the $O(n \log n)$ running time is necessary or whether this running time can be improved? The trivial lower bound for dense vectors is $\Omega(n)$, so a logarithmic gap remains. It is widely conjectured that this log-factor is necessary but the evidence is scarce. For instance, there are lower bounds in restricted models [12] and there is a connection to the network coding conjecture [11].

Since progress in terms of lower-order factors seems out of reach, can we nevertheless compute convolutions faster?

**Sparse Convolution.** Suppose that the input and output vectors $A$, $B$ and $A \star B$ are *sparse*.[6] Can we design *output-sensitive* algorithms where we analyze the running time in terms of $t$, the combined number of nonzero entries in $A$, $B$ and $A \star B$? Note that in the Boolean and nonnegative cases, $t$ is dominated by the number of nonzero entries in $A \star B$.

The need for such a primitive appears in many situations, specifically in algorithms for variants of $k$-SUM and Subset Sum [71, 66, 65], several string problems including sparse wildcard matching, geometric pattern matching [67, 84] and block-mass pattern matching [17] and the fine-grained complexity of some graph problems [2, 130]. These types of problems have been investigated by different communities, including not only fine-grained complexity and string algorithms, but also computer algebra [180] and compressed sensing [118, 100], and they are very closely related to the famous sparse recovery problem, see e.g. [104, 100].

We start with a review about the most relevant literature on sparse convolutions prior to our work, separated into randomized and deterministic algorithms.

**Randomized Algorithms for Sparse Convolution.** A large body of work addresses this problem [163, 84, 179, 161, 121, 25, 71, 180, 164, 108, 61]. As a first breakthrough result, Cole and Hariharan showed that sparse nonnegative convolutions can indeed be computed in near-linear time in $t$ [84]. Their result is a randomized Las Vegas algorithm in time $O(t \log^2 n)$. Subsequent work improved upon this result by removing the nonnegativity assumption: With a Monte Carlo algorithm in the same running time $O(t \log^2 n)$ [164], or with bit-complexity $\widetilde{O}(t \log n)$ [108][7]. While Cole and Hariharan's algorithm uses a mix of complicated machinery, these algorithms rely on a relatively simple hashing-based approach.

Another more algebraic approach to sparse convolution algorithms is via polynomial evaluation and interpolation. At the heart of this approach lies an old algorithm called *Prony's method* [177] which allows to efficiently interpolate a sparse polynomial.[8] This algorithm involves heavy algebraic computations that can be implemented in randomized time $\widetilde{O}(t \log^2 n)$ [180].

[6] *In this setting we have to be careful how to represent the vectors: For dense convolutions we could simply write down the vectors as arrays of length n. For sparse convolutions we will represent the vectors as a list of (index, entry)-pairs for all nonzero entries.*

[7] *It seems that in the standard word RAM model with word size $\Theta(\log n)$ their algorithm would run in randomized time $O(t \log^5 t \operatorname{polyloglog} n)$.*

[8] *Formally, Prony's method allows to recover a t-sparse polynomial from 2t given evaluations at carefully chosen evaluation points. We give a quick overview of Prony's method in Section 2.3.*

This research is closely related to the extensively studied sparse Fourier transform problem, see e.g. [103, 106, 118, 125, 126]. Indeed, one can obtain a sparse convolution algorithm with running time $O(t \log^2 n)$, albeit with a more complicated algorithm and under the assumption that complex exponentials can be evaluated in constant time, by combining the state-of-the-art sparse Fourier transform with the semi-equispaced Fourier transform, see Section 2.4.

In summary: On the one hand the known upper bounds are $O(t \log^2 n)$ [84, 164] or $O(t \log^5 t \operatorname{polyloglog} n)$ [108]. On the other hand, the most plausible lower bound is $\Omega(t \log t)$ which transfers from FFT. It is natural to ask whether this gap can be closed:

*Question 1: Can sparse nonnegative convolutions be computed in (randomized) time $O(t \log t)$?*

**Deterministic Algorithms for Sparse Convolution.** In terms of deterministic algorithms, the state of affairs is worse. The first nontrivial deterministic result for sparse nonnegative convolution is a data structure that, after preprocessing one of the vectors in time $\Theta(t^2)$, computes the convolution with any given query vector in near-linear time $O(t \log^3 t)$ [18]. Later, Chan and Lewenstein [71] devised a deterministic algorithm running in time $t \cdot 2^{O(\sqrt{\log t \log\log n})}$, without preprocessing. Their algorithm is limited in the sense that it expects as an additional input the support (i.e., the set of nonzero coordinates) of $A \star B$. This assumption can be removed as shown by Bringmann and Nakos [65]; in Section 2.6 we provide more details. In summary, the state-of-the-art deterministic algorithms for computing sparse nonnegative convolutions either require heavy precomputations or fail to achieve near-linear time $O(t \operatorname{polylog} n)$. We therefore ask:

*Question 2: Can sparse nonnegative convolutions be computed in deterministic time $O(t \operatorname{polylog} n)$?*

### 1.3.1 Machine model

In the context of log-factor improvements we have to carefully consider the machine model of our algorithms. Throughout this thesis we employ the standard word RAM model where the word size $w$ is logarithmic in the input size of the respective problem. In particular, for the sparse convolution problem we assume that the words have size $w = \Theta(\log n + \log \Delta)$, where $\Delta$ is the largest entry in the given vectors. In this way we can perform basic arithmetic and operations on entries and indices in constant time.

### 1.3.2 Time-Optimal Sparse Nonnegative Convolution

In two papers [61, 60] we have answered both questions positively. In [61] we design the following novel Monte Carlo algorithm for nonnegative sparse convolution.

**Theorem 1.1 (Time-Optimal Sparse Nonnegative Convolution).** *There is a randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \log t + \operatorname{polylog}(n\Delta))$ and with error probability $2^{-\sqrt{\log t}}$, where $t = \|A \star B\|_0$ and $\Delta = \|A \star B\|_\infty$.*

This result affirmatively answers Question 1 for $t \gg \operatorname{polylog}(n\Delta)$, by a randomized algorithm. In fact, our algorithm can be phrased as a reduction from the

sparse case to the dense case of nonnegative convolution. In the following, we denote by $D_\delta(n)$ the running time of a randomized algorithm for dense nonnegative convolution with failure probability $\delta$ (for any $\delta \geq 0$). For convenience this notation hides the dependence on $\Delta$. In the sparse setting, where we denote the output size by $t$, we will denote by $S_\delta(t)$ the running time of a randomized algorithm for sparse nonnegative convolution with failure probability $\delta$; this hides the dependence on $n$ and $\Delta$. In this language, our main result is the following theorem.[9]

**Theorem 1.2 (Sparse and Dense Nonnegative Convolution Are Equivalent).** *Any randomized algorithm for dense nonnegative convolution with running time $D_\delta(n)$ and error probability $\delta > 0$ can be turned into a randomized algorithm for sparse nonnegative convolution with error probability $\delta$ running in time*

$$S_\delta(t) = O\big(D_\delta(t) + t\log^2(\log(t)/\delta) + \mathrm{polylog}(n\Delta)\big).$$

Since $D(t) = O(t\log t)$ (by the Fast Fourier Transform), setting $\delta = 2^{-\sqrt{\log t}}$ yields time $O(t\log t + \mathrm{polylog}(n\Delta))$, which proves Theorem 1.1. Furthermore, any future algorithmic improvement for the dense case automatically yields an improved algorithm for the sparse case by our reduction. In fact, under the mild conditions that $t \gg \mathrm{polylog}(n\Delta)$ and that the optimal running time $D_{1/3}(t)$ is $\Omega(t(\log\log t)^2)$, we obtain an *asymptotic equivalence* with respect to constant-error randomized algorithms:

- $S_{1/3}(t) = O(D_{1/3}(t))$ holds by Theorem 1.2 and the mild conditions, and
- $D_{1/3}(t) = O(S_{1/3}(t))$ holds since the sparse case trivially is a special case of the dense one.

We prove Theorems 1.1 and 1.2 in Chapter 4.

[9] *To be precise, we should take the dependence on $\Delta$ (and $n$) into account. Expressing the running time for the dense case as $D_\delta(n,\Delta)$ and for the sparse case as $S_\delta(t,n,\Delta)$, our reduction actually shows that*

$$\begin{aligned} &S_\delta(t,n,\Delta) \\ &\quad = O(D_\delta(t,\mathrm{poly}(n\Delta))) \\ &\quad + O(t\log^2(\log(t)/\delta)) \\ &\quad + \mathrm{polylog}(n\Delta). \end{aligned}$$

### 1.3.3 Deterministic Sparse Nonnegative Convolution

In [60] we have again studied sparse nonnegative convolution; this time from the perspective of deterministic algorithms. We answer Question 2 by a near-linear-time algorithm that improves upon the previously best time $t \cdot 2^{O(\sqrt{\log t \log\log n})}$, obtained by [71, 65].

**Theorem 1.3 (Deterministic Sparse Nonnegative Convolution).** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t\,\mathrm{polylog}(n\Delta))$, where $t = \|A \star B\|_0$ and $\Delta = \|A \star B\|_\infty$.*

As a corollary we can efficiently derandomize known algorithms for several problems which use sparse nonnegative convolution as a subroutine. For all these applications, we can simply replace the former randomized algorithms with our deterministic one in a black-box manner. Of course, for the same derandomization we could alternatively use the $t \cdot 2^{O(\sqrt{\log t \log\log n})} = t \cdot n^{o(1)}$-time algorithm from [71, 65] and therefore our contribution can alternatively be seen as improving the best deterministic time from $T^{1+o(1)}$ to $\widetilde{O}(T)$. Specifically, we obtain improvements for the following problems:

- *Output-Sensitive Subset Sum:*
  Given a set $X$ of integers and a threshold $\tau$, compute the set $S$ of all numbers less than $\tau$ which can be expressed as a subset sum of $X$. The best-known randomized algorithm runs in time $\widetilde{O}(|S|^{4/3})$ [66], and it can be derandomized in same running time.

- *N-fold Boolean Convolution:*
  Given $N$ Boolean vectors $A_1, \ldots, A_N \in \{0,1\}^n$, compute their Boolean convolution $A_1 \circledast \cdots \circledast A_N$ (with or without wrap-around) in input- plus output-sensitive time. It was recently shown that this problem can be solved in randomized near-linear time $O(t\,\mathrm{polylog}\,n)$ [65]. Our derandomization achieves the same

running time. This yields a new deterministic near-linear-time algorithm for Modular Subset Sum which is rather different from the known ones [27], as discussed in [65].

- *Block-Mass Pattern Matching:*
  Given a length-$n$ text $T$ and a length-$m$ pattern $P$ over the alphabet $\mathbf{N}$, the task is to output all possible indices $0 \leq k_0 \leq \cdots \leq k_m \leq n$ such that $P[i] = \sum_{k_i \leq j < k_{i+1}} T[j]$ for all positions $i \in [m]$. Building on the data structure from [18], this problem is known to be solvable in deterministic time $\widetilde{O}(n+m)$ after preprocessing the text in time $O(n^2)$ [17]. The preprocessing time was later reduced to $O(n^{1+\epsilon})$, for any $\epsilon > 0$ [71]. We entirely remove the necessity to precompute and thereby reduce the total running time to $\widetilde{O}(n+m)$.

- *3-SUM in Special Cases:*
  In a breakthrough paper, Chan and Lewenstein [71] used sophisticated techniques to obtain randomized and deterministic subquadratic algorithms for a variety of problems related to 3-SUM, such as bounded monotone two-dimensional 3-SUM, bounded monotone $(\min,+)$-convolution, clustered integer 3-SUM, etc. The precise running time of their deterministic algorithm for these problems is $O(n^{1.864})$. Here we remove an $o(1)$ overhead in the exponent which is invisible due to rounding the constant in the exponent.

We give a detailed proof of Theorem 1.3 in Chapter 3.

### 1.3.4 Las Vegas Algorithms for Sparse Nonnegative Convolution

In addition to our new deterministic algorithm, in [60] we also improve the state-of-the-art *Las Vegas* algorithms for sparse nonnegative convolution in two regards: *simplicity* and *efficiency*. In fact, to the best of our knowledge the only known Las Vegas algorithm is due to Cole and Hariharan [84]; all randomized algorithms published later have only Monte Carlo guarantees [179, 25, 180, 164, 108, 61]. The expected running time of [84] is $O(t \log^2 n)$ and moreover, they prove the additional guarantee that their algorithm terminates in time $O(t \log^2 n)$ with high probability $1 - \frac{1}{n}$. However, the algorithm is very complicated, involves various string problems as subtasks and the precomputation of a large prime number. We provide an accessible alternative with the same theoretical guarantees; the simplest version can be summarized in 13 lines of pseudocode (Algorithm 3.1 on Page 59).

**Theorem 1.4 (Simple Las Vegas for Sparse Nonnegative Convolution).** *There is a Las Vegas randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in expected time $O(t \log^2 t)$, where $t = \|A \star B\|_0$. Moreover, with probability $1 - \delta$ the running time is bounded by $O(t \log^2(t/\delta))$.*

In comparison to Cole and Hariharan's algorithm, our algorithm runs slightly faster in expectation (at least if $t \ll n$) and achieves the same high-probability guarantee (indeed, by setting $\delta = \frac{1}{n}$ the running time is bounded by $O(t \log^2 n)$ with probability at least $1 - \frac{1}{n}$). We further show how to reduce the expected running time, achieving optimality up to a log log factor.

**Theorem 1.5 (Fast Las Vegas for Sparse Nonnegative Convolution).** *There is a Las Vegas randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in expected time $O(t \log t \log\log t)$, where $t = \|A \star B\|_0$.*

Assuming that FFT-time $O(n \log n)$ is best-possible for computing dense convolutions, the best-possible algorithm for computing sparse convolutions requires time $\Omega(t \log t)$. Hence, our algorithm is likely optimal up to the $\log\log t$ factor. The proofs of these theorems are deferred to Chapter 3.

### 1.3.5 Open Problems

Our work on sparse convolutions raises a plethora of open problems that we discuss in the following.

**1** *Are there better randomized algorithms?*
There are at least three interesting directions:

**1a** *Can the error probability of the $O(t \log t)$ algorithm be reduced?*
Specifically, can Theorem 1.1 be improved from $\delta = 2^{-\sqrt{\log t}}$ to $1/\mathrm{poly}(t)$ or even $1/\mathrm{poly}(n)$?

**1b** *Is there a Las Vegas algorithm in expected time $O(t \log t)$?*
That is, can we shave the $\log\log t$ factor from Theorem 1.5? While the tricks from Theorem 1.4 can be worked into parts of our $O(t \log t)$ Monte Carlo algorithm, the algorithm involves some other steps that seem hard to verify.

**1c** *Can the* $\mathrm{polylog}(n\Delta)$ *term in Theorem 1.2 be removed?*
This would make the algorithm efficient also for very small $t$. We believe that a quite different approach would be necessary, since already for finding a prime field large enough to store $n$ and $\Delta$, or for computing a single multiplicative inverse in such a prime field, the fastest algorithms that we are aware of run in time $O(\mathrm{polylog}(n\Delta))$, even for the word RAM model.

**2** *Are there better deterministic algorithms?*
The best deterministic time we achieve is $O(t \log^5(n\Delta)\,\mathrm{polyloglog}(n\Delta))$; is it possible to reduce the number of log factors, and can we remove the dependence on $n$ and $\Delta$?

**3** *Can any of our algorithms be extended to sparse* integer *convolution?*
The critical difference between integer and nonnegative convolution is that many terms in the convolution can *cancel*. As an extreme case, there are dense integer vectors $A, B$ such that $A \star B$ is 2-*sparse*.[10] This renders many approaches for nonnegative convolution hopeless for the general integer case, specifically non-algebraic techniques (such as almost-additive hashing and the scaling trick described in Chapter 2).

[10] *As an easy example, where at least one of the vectors $A, B$ is dense, take*

$$A = (1, \ldots, 1),$$
$$B = (1, -1, 0, \ldots, 0).$$

*Their convolution $A \star B$ has only two nonzero entries, in the first and last position respectively.*

**4** *Are there output-sensitive algorithms for generalized sumsets?*
Recall that for integer sets $A, B$ we define $A + B = \{a + b : a \in A, b \in B\}$ as the *sumset* of $A$ and $B$. It is natural to generalize sumsets to other additive groups $G$—but can we similarly hope for output-sensitive algorithms? An easy case are finite-field vectors $G = \mathbf{F}_p^d$ for which we design efficient algorithms in Chapter 2.

After a long line of research (which we discuss in more depth in Section 2.1), Umans [198] established an algebraic algorithm to compute sumsets over *all* finite groups $G$ in time $O(|G|^{\omega/2})$, where $\omega < 2.372$ is the exponent of matrix multiplication [14, 91]. Assuming that $\omega = 2$, this is almost-linear in $|G|$ and it remains an interesting question whether we can achieve almost-linear running time in $t$.

**5** *Can we obtain further improvements by bit tricks, say for Boolean convolution?*

**6** *Can we reduce sparse Fourier transform to dense Fourier transform in a finer-grained way?*
Computing convolutions is intimately connected to the Discrete Fourier Transform (DFT). The sparse case of Fourier transform, where one has oracle access to some vector $A$ and wants to compute its DFT $\widehat{A}$ under a $t$-sparsity assumption, is also extensively studied [103, 106, 118, 119, 125, 126, 175, 134, 133, 135, 165]. We ask whether a reduction like Theorem 1.2 is similarly possible. The algorithm in [118] runs time $O(t \log(n\Delta))$, but the running time is not dominated by the calls to FFT.

## 1.4 Application: 3-SUM Lower Bounds for Approximate Distance Oracles

As a complementary application of sparse convolution, we present the following result from fine-grained complexity [2]: We prove conditional lower bounds for the well-studied approximate shortest paths problem. This problem does a priori not look like a convolution-style problem at all. Nevertheless, our hardness result relies on an efficient *algorithm* for 3-SUM, which is powered by efficiently computing sparse nonnegative convolutions besides several other tools from additive combinatorics.

**Distance Oracles.** An approximate distance oracle is an algorithm that preprocesses a graph[11] efficiently and can then quickly return the distance between any given pair of nodes, up to a small error. After being implicitly studied for some time [158, 26, 82, 89, 83], Thorup and Zwick [197] formally introduced the distance oracle problem in 2001 suggesting that it is perhaps the most natural formulation of the classical all-pairs shortest paths problem. Distance oracles quickly rose to prominence and the techniques developed for them found deep connections to other popular topics such as sublinear algorithms, spanners, labelling schemes, routing schemes, and metric embeddings.

[11] *Throughout we assume that graphs are undirected, unweighted and have $n$ nodes and $m$ edges.*

Distance oracles have been thoroughly investigated with the primary goal of understanding the best possible trade-off between the four main parameters: the multiplicative error factor (aka the *stretch*), the query time, the space usage, and the preprocessing time; see e.g. [36, 159, 33, 34, 35, 170, 193, 171, 206, 207, 74, 75, 192, 138, 13, 182, 77, 90] and the list is still growing. They have also been studied from other perspectives, for example more efficient distance oracles for restricted classes of graphs were sought after (e.g. [73, 155, 150] for planar graphs), and also their complexity in *dynamic* graphs is of great interest (e.g. [76, 117, 99, 90]). Despite all this, perhaps the first question one might ask remains poorly understood:

*Question 3: What is the best stretch $f(k)$ we can achieve with preprocessing time $O(mn^{1/k})$ and almost-constant query time $n^{o(1)}$?*

The seminal Thorup-Zwick oracle [197] achieves stretch $2k-1$ after preprocessing a graph in $O(kmn^{1/k})$ time (it also achieves $O(k)$ query time and uses $O(n^{1+1/k})$ space). Better trade-offs exist in the small-$k$ regime of stretch below 3 [170, 171, 34, 192, 138, 13, 77]. In dense enough graphs, the results are even better [36, 33, 35, 206]; in particular if $m = \Omega(n^{1+c/\sqrt{k}})$ Wulff-Nilsen [206] obtained *linear* $O(m)$ preprocessing time. However, in the setting of *sparse* graphs and large $k$ (where the running time is close to linear), the Thorup-Zwick bound remains the state of the art.

Most of the existing lower bound techniques are incapable of answering the above question. Incompressibility arguments [50, 158, 197], typically based on Erdős' girth conjecture, can show the optimality of the $n^{1+1/k}$ space bound of Thorup and Zwick, but cannot be used to prove any lower bound higher than $m$. In the cell probe model, Sommer, Verbin, and Yu [193] show that $m^{1+1/k}$ space (and therefore time) is required for stretch $f(k) = O(k/t)$ if the query time is $t$; this lower bound is meaningless when the query time is super-constant and is far from the Thorup-Zwick upper bound even when $t$ is a small. Finally, under a conjecture about the space complexity of Set Intersection, Pătraşcu, Roditty, and Thorup [170, 171] show $\Omega(mn^{\epsilon})$ lower bounds on the space complexity, but their techniques only address stretch $3-\delta$; alas, they cannot prove that the error must grow above 3 in the close-to-linear time regime.

Recently, Abboud, Bringmann, Khoury, and Zamir [4] introduced the *short cycle removal* technique for hardness of approximation in fine-grained complexity

and applied it to prove that the stretch must be $f(k) > k/6.3776 \pm O(1)$, assuming the 3-SUM or APSP conjectures. Thus, $f(k)$ must grow with $k$ and it is a linear function. However, there is still a huge gap in our understanding of this basic question; e.g. the optimal stretch for $O(mn^{0.1})$ preprocessing time could be anything between 21 and 3. Whether the short cycle removal technique could achieve tight bounds was left as the main open question [4].

### 1.4.1 Optimal Short Cycle Removal

Our contribution in [2] is that we take the short cycle removal technique to its limit and prove much higher and, in some cases, tight lower bounds. Let us begin by introducing this technique.

Triangle finding problems are a common starting point for fine-grained hardness results. The following *all-edge* version is particularly interesting, since it is known to require $n^{2-o(1)}$ time in $\Theta(n^{1/2})$-regular graphs assuming either the 3-SUM conjecture [172, 143] or the APSP conjecture [204].

**Definition 1.6 (All-Edges Triangle).** *Given a tripartite graph $G = (V, E), V = X \cup Y \cup Z$ determine which edges in $E \cap (Y \times Z)$ are in at least one triangle.*

Let us recall the popular 3-SUM conjecture that implies the hardness of All-Edges Triangle.

**Conjecture 1.7 (3-SUM).** *For any $\epsilon > 0$, no $O(n^{2-\epsilon})$-time algorithm can determine whether a given set $A$ of $n$ integers contains $a, b, c \in A$ such that $a + b + c = 0$.*

**From All-Edges Triangle to Approximate Distance Oracles.** It is easy to reduce from All-Edges Triangle to distance oracles. Construct a distance oracle for a new graph $G'$ that is obtained from $G$ by deleting all edges in $Y \times Z$. To determine if an edge $(y, z) \in E(G) \cap (Y \times Z)$ is in a triangle, we query the oracle for the distance between $y$ and $z$ in $G'$: It must be exactly 2 if $(y, z)$ is in a triangle in $G$, and it is at least 3 otherwise.

To prove hardness for *approximate* distance oracles we would want the distance in $G'$ to be much larger than 3 if $(y, z)$ was not in a triangle in $G$. Now, the key observation is that a path of length $k-1$ in $G'$ implies that $(y, z)$ was in a $k$-cycle in $G$. In other words, if the edge $(y, z)$ is *not* in a $k$-cycle in $G$ then a $\frac{k}{2}$-approximation to the distance suffices for determining if $(y, z)$ participates in a triangle.

**Short Cycle Removal.** The basic idea of the *short cycle removal* technique is to reduce *the number of short cycles* in a graph without eliminating its triangles. The goal is that the number of pairs $(y, z)$ that are in short cycles but not in triangles will be small, since such pairs incur a false positive in the above reduction. The main tool towards this is to show that in subquadratic time the number of $k$-cycles can be reduced from the worst case $O(n^{k/2+1/2})$ to only $O(n^{k/2+\gamma})$ for $\gamma < 1/2$ which is closer to the random case (where $\gamma = 0$). The quality of the lower bounds obtained by this technique depends directly on the value of $\gamma$ for which such a statement can be proved.

In [4], the authors use the following structure versus randomness argument: If the graph has many $k$-cycles (more than the random case) then it must have a structure in the form of a *dense piece* (a subgraph with disproportionately many edges). They use fast matrix multiplication to check for triangles that use the dense pieces and then remove them from the graph, reducing its number of $k$-cycles significantly and making it more random.

**Theorem (Short Cycle Removal [4]).** *For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$-time algorithm for All-Edge Triangles in a $\Theta(n^{1/2})$-regular $n$-vertex graphs which contains at most $O(n^{k/2+\gamma})$ $k$-cycles for all $k \geq 3$ and for $\gamma = 0.345 + o(1)$, unless the 3-SUM and APSP conjectures fail.*

The value of $\gamma$ that [4] achieve depends on the fast matrix multiplication exponent $\omega < 2.37188$ [14, 91], and even if $\omega = 2$ they only get $\gamma = 1/4$; going beyond this seems difficult. The authors suggest an approach for getting $\gamma \to 0$ but there are three major barriers. First, one needs to prove an unproven combinatorial conjecture about the relationship between the number of cycles and the existence of dense subgraphs. Second, one has to turn the proof into an efficient algorithm for finding the dense pieces. And third, it is conceptually impossible to remove the dense pieces without using fast matrix multiplication, which means that one must first prove that $\omega \to 2$ before getting $\gamma \to 0$.

**Optimal Short Cycle Removal.** In our paper [2] we take a different approach: We look at the reduction from 3-SUM to All-Edges Triangle and ask: *What structure in the 3-SUM instance causes the resulting graph to have too many k-cycles?*[12] The answer turns out to be related to the *additive energy* of the 3-SUM instance, namely to the number of quadruples $a_1, a_2, a_3, a_4 \in A$ such that $a_1 + a_2 = a_3 + a_4$. Thus, our goal changes from "short cycle removal" in graphs to "*energy reduction*" on a set of numbers. The latter can be done much more effectively using machinery from additive combinatorics (overviewed in depth in Section 5.1) such as the celebrated Balog-Szemerédi-Gowers theorem [30, 114], and the aforementioned use of sparse convolution.

[12] *In fact, we design a more transparent such reduction that could be of independent interest.*

Our main technical result is an optimal short cycle removal for All-Edges Triangle that is obtained via an optimal energy reduction for 3-SUM. This can be seen as an *algorithm* for 3-SUM: If the given instance is very structured, then we can solve (parts of) it faster using additive combinatorics and sparse convolution,[13] until the remaining instance looks random-like. Then we run the reduction to All-Edges Triangle which produces a random-like graph without many cycles. Notably, we achieve $\gamma = 0$ even without assuming that $\omega = 2$.

[13] *We believe that this part is of independent interest. See Theorem 5.4 in Chapter 5.*

**Theorem 1.8 (Optimal Short Cycle Removal).** *For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$-time algorithm for All-Edges Triangle in a $\Theta(n^{1/2})$-regular $n$-vertex graph which contains at most $O(n^{k/2})$ $k$-cycles for all $k \geq 3$, unless the 3-SUM conjecture fails.*

### 1.4.2 New Lower Bounds for Distance Oracles

Our main corollary is an improvement of the lower bound for distance oracles with $n^{o(1)}$ query time and close-to-linear $O(mn^{1/k})$ preprocessing time, from stretch $\geq k/6.3772$ to stretch $\geq k \pm O(1)$. This is only a factor 2 away from the Thorup-Zwick upper bound. We find it astonishing that the strongest known lower bound to our basic question about distance oracles involves tools from additive combinatorics.

**Theorem 1.9 (Hardness of Distance Oracles with Stretch $k$).** *For any integer constant $k \geq 2$, there is no approximate distance oracle for sparse graphs with stretch $k$, preprocessing time $\widetilde{O}(m^{1+p})$ and query time $\widetilde{O}(m^q)$ with $kp + (k+1)q < 1$, unless the 3-SUM conjecture fails.*

Our lower bound is proved for sparse graphs where $m = O(n)$. Consequently, it cannot be bypassed even by $(\alpha, \beta)$-distance oracles that have an additive error of $\beta = n^{o(1)}$ in addition to a multiplicative stretch of $\alpha$.[14] As in [4], our lower bound also holds for the *offline* problem where we are given the queries before preprocessing. We also obtain a trade-off between the query and preprocessing time; e.g. if the query time is $O(n^{1/k})$ rather than $n^{o(1)}$ then the stretch is $k/2 \pm O(1)$ rather than $k \pm O(1)$.

[14] *This is because the additive error is insignificant in the sparse regime where we can subdivide edges.*

**Tight Bounds?** While being pleased by the dramatic improvement in the lower bound, we are disappointed that we did not get a tight lower bound despite optimizing the short cycle removal technique to its limit. *Is the short cycle removal technique inherently insufficient for proving a tight lower bound?*

The following three theorems indicate that our technique may well be the "right" one. This calls for revisiting the 20-year-old *upper bounds* in the hope of closing the gap by improving the stretch from $2k-1$ to $k \pm O(1)$. There is ample evidence that this may be around the corner. Better algorithms already exist in the regimes of dense graphs or when the stretch is small (some are very recent [13, 77]). For large $k$, Roditty and Tov [182] recently improved the $2k-1$ factor slightly to $2k-4$ while keeping the same space and query time as Thorup-Zwick (but not preprocessing time). Moreover, in the closely related setting of *graph spanners* where there is a similar trade-off saying that $2k-1$ stretch can be achieved with a subgraph on $n^{1+1/k}$ edges, it was shown by Parter [169] that the stretch can be improved to $k$ for all pairs of nodes at distance $> 1$ (see also [93, 43]). Alas, beating the Thorup-Zwick bound for general $k$ has been elusive; perhaps knowing that the gap from the lower bound is only 2 (following this thesis) will motivate the community to find better algorithms. Such a result would not only be pleasing, but it could also be useful in practice (see e.g. [178]).

**The Small Stretch Regime.** Recall that the smallest stretch attainable by the Thorup-Zwick oracle is 3 (i.e., $k = 2$), in which case their preprocessing time is $O(m\sqrt{n})$. Let us focus on the case of sparse graphs where $m = O(n)$ and this time bound becomes $O(n^{3/2})$. Subsequent work [34, 13, 77, 90] showed that interesting results can also be achieved for smaller stretch factors (if we allow constant additive error). The best result in this regime is a recent result by Dory, Forster, Nazari and de Vos [90] designing a distance oracle with stretch exactly 2, constant query time and preprocessing time $O(mn^{2/3})$.

Using our optimal cycle removal, we prove that the Thorup-Zwick oracle is optimal in the following sense: If we want to improve the stretch to $3-\epsilon$ then the running time must grow polynomially to $m^{3/2+\Omega(\epsilon)}$. In addition, we prove the optimality of Dory et al.'s algorithm [90] in the sense that $m^{5/3-o(1)}$ time is required for stretch 2.

**Theorem 1.10 (Hardness of Distance Oracles with Stretch $2 \le \alpha < 3$).** *For any $2 \le \alpha < 3$ and $\epsilon > 0$, in sparse graphs there is no distance oracle with stretch $\alpha$, query time $n^{o(1)}$ and preprocessing time $O(m^{1+\frac{2}{1+\alpha}-\epsilon})$, unless the 3-SUM conjecture fails.*

Recall that there are techniques besides short cycle removal that can prove lower bounds for stretch up to 3. Indeed, the lower bounds of Pătraşcu, Roditty, and Thorup [170, 171] are similar to ours for stretches $2+\epsilon$ and $3-\epsilon$, except that they are concerned with *space* and not just preprocessing time. On the one hand, this makes their lower bounds stronger. On the other hand, they need to rely on a strong conjecture about the *space* versus query time trade-off of Set Intersection, rather than the 3-SUM conjecture that is simply about the *time* complexity. While Set Intersection is a common starting point for data structure lower bounds, the particular variant they use is not standard and was not used in any other paper to our knowledge. Basing the same results also on one of the most popular conjectures in fine-grained complexity is desirable. In any case, the more important message of Theorem 1.10 is to show that our techniques *can* prove tight bounds.

**Dynamic Graphs.** Extending our basic question to the dynamic setting we seek the optimal stretch for a distance oracle that achieves $n^{o(1)}$ query time and $O(n^{1/k})$ time for *updates* that add or remove an edge. In this case, we give a more efficient reduction from All-Edges Triangle (inspired by the reduction of Abboud and Vassilevska Williams [9] to dynamic matching) and prove that the stretch must be $2k \pm O(1)$.

**Theorem 1.11 (Hardness of Dynamic Distance Oracles).** *For any integer constant $k \ge 2$, there is no dynamic approximate distance oracle with stretch $2k-1$, update time $O(m^u)$ and query time $O(m^q)$ with $ku + (k+1)q < 1$, unless the 3-SUM conjecture fails.*

This lower bound would be tight if the Thorup-Zwick bound extends to the dynamic setting. This was indeed accomplished by Chechik [76] (see also [117, 90]) in the *decremental* setting where only edge deletions are allowed, but not yet in the fully dynamic case (the best bounds appear in [99]).

### 1.4.3 A Tight Lower Bound for 4-Cycle Listing

Finding 4-cycles in a graph is one of the simplest non-trivial cases of the classical *Subgraph Isomorphism* problem. The longstanding upper bound for testing 4-cycle freeness is $O(\min(n^2, m^{4/3}))$ [16, 209]. It is conjectured that no $O(n^{2-\epsilon})$ algorithm exists; proving this under one of the more popular conjectures of fine-grained complexity has been a well-known open question. In fact, the 4-cycle problem is infamous for eluding even any super-linear lower bound via the standard reduction techniques; [4] highlight this problem as encapsulating the challenge in proving hardness of approximation results for distance oracles.

In the *listing* version we are asked to output all 4-cycles in the graph. Such problems are well-studied and are closely related to the enumeration of query answers in databases. It is known that *all* cycles in a graph can be listed in linear time $O(m+t)$ where $t$ is the output size [47]. But for a fixed length $k$, the listing $k$-cycles problem is not as easy. In a landmark result in fine-grained complexity, that implied the aforementioned 3-SUM-hardness for All-Edge Triangle, Pătraşcu [172] proved an essentially tight lower bound for *triangle* listing (see [143, 48]). The first and only super-linear lower bound for 4-cycle listing, however, came only a decade later via the short cycle removal technique [4]. We improve their lower bound from $(m^{1.1927}+t)^{1-o(1)}$ to a *completely tight* lower bound matching the $O(\min(n^2, m^{4/3})+t)$ upper bound.[15]

**Theorem 1.12 (Hardness of Listing 4-Cycles).** *For any $\epsilon > 0$, there is no algorithm listing all 4-cycles in time $\widetilde{O}(n^{2-\epsilon}+t)$ or in time $\widetilde{O}(m^{4/3-\epsilon}+t)$ (where $t$ is the number of 4-cycles), unless the 3-SUM conjecture fails.*

We prove this theorem as well as the previous hardness results for distance oracles in Chapter 5.

[15] *The $O(n^2+t)$ upper bound is simple: Create an array of size $n^2$. For each node $x$ and all pairs of neighbors $u, v \in N(x)$ store $x$ in the $(u,v)$ entry of the array. If we access an entry that already contains nodes $y_1, \ldots, y_k$ we output the 4-cycles $(x, u, y_i, v)$ for all $i$. The time is $n^2$ plus the number of 4-cycles because each time we access an entry (except for the first time) we output at least one 4-cycle. The $O(m^{4/3}+t)$ algorithm is more involved [130, 7].*

### 1.4.4 Open Problems

The most important question which remains open is whether the gap between our lower bound for stretch $k$ and the Thorup-Zwick upper bound for stretch $2k-1$ can be closed:

1. *Is there a distance oracle with stretch $k$, preprocessing time $O(mn^{1/k})$ and almost-constant query time $n^{o(1)}$, or can we rule out distance oracles with stretch better than $2k-1$?*

Besides that, our work inspires more questions on the nature of 3-SUM and related problems:

2. *Does our machinery for structured 3-SUM find more applications?*
   In spirit, our result proves that the 3-SUM problem is not quadratic-time hard for additively structured instances. Instances not covered by this case are Sidon sets and in particular *random instances.* Can this insight be leveraged?

3. *Are there 3-SUM lower bounds for other subgraph isomorphism problems?*
   In particular, can we show matching lower bounds for listing (or even detecting) $k$-cycles for $k > 4$? The currently fastest algorithms for detecting $k$-cycles in undirected graphs depend on whether $k$ is odd or even [209, 16]: For odd $k$ the problem behaves "triangle-like" and the best algorithm runs in time $O(n^\omega)$. For even $k$ the problem behaves "4-cycle-like" and can be solved in time $O(n^2)$ (without using fast matrix multiplication). It seems plausible that our lower bounds could extend to lower bounds against listing $k$-cycles for even $k$.

## 1.5 Fast Scheduling via Partition-and-Convolve

We include yet another application of convolutions in algorithm design in this thesis, based on [58]. As a proof of concept of the aforementioned *partition-and-convolve* design paradigm, we design an algorithm for a seemingly unrelated *scheduling* problem in this framework. (In this section we momentarily digress from the design of output-sensitive and sublinear-time algorithms.)

**The Scheduling Problem.** Consider the problem of minimizing the total processing times of tardy jobs on a single machine. In this problem we are given $n$ jobs, where each job $j$ has a *processing time* $p_j \in \mathbf{N}$ and a *due date* $d_j \in \mathbf{N}$. The goal is to find a single-machine schedule (i.e., a permutation of the jobs) that minimizes the processing time of all *tardy* jobs—that is, of all jobs not completely processed before their due date. In the standard three field notation for scheduling problems of Graham [115], this problem is denoted as the $1||\sum p_jU_j$ problem.[16]

The $1||\sum p_jU_j$ problem is arguable one of the simplest natural scheduling problems, modeling a basic scheduling scenario. As it includes Subset Sum as a special case (see below), the $1||\sum p_jU_j$ problem is NP-hard. However, it is only hard in the weak sense, meaning it admits pseudo-polynomial time algorithms. Our focus is on developing fast pseudo-polynomial time algorithms for $1||\sum p_jU_j$, improving in several settings on the best previously known solution from the late 60s. Before we describe our results, we discuss the previously known state of the art of the problem, and describe how our results fit into this line of research.

[16] *The 1 in the first field indicates a single machine model, the empty second field indicates there are no additional constraints, and the third field indicates that the goal is to minimize $\sum_j p_jU_j$ where $U_j \in \{0,1\}$ indicates whether a job is tardy.*

**State of the Art: The Lawler-Moore Algorithm.** $1||\sum p_jU_j$ is a special case of the famous $1||\sum w_jU_j$ problem. Here, each job $j$ also has a weight $w_j$ in addition to its processing time $p_j$ and due date $d_j$, and the goal is to minimize the total weight (as opposed to total processing times) of tardy jobs. This problem has already been studied in the 60s, and even appeared in Karp's fundamental paper from 1972 [136]. The classical dynamic programming algorithm by Lawler and Moore [149] solves the $1||\sum w_jU_j$ problem (and hence also $1||\sum p_jU_j$) in time $O(P \cdot n)$, where $P = \sum_{j \in J} p_j$ denotes the total processing time of all jobs. This is one of the earliest and most prominent examples of pseudo-polynomial algorithms, and remains the fastest known algorithm even for the special case of $1||\sum p_jU_j$ (prior to our work).

As we assume that all processing times are nonnegative integers and as we can ignore jobs with processing time 0, we have $n \leq P$. The Lawler-Moore algorithm therefore runs in time $O(P^2)$. In fact, it makes perfect sense to analyze the time complexity of a pseudo-polynomial time algorithm for either problems only in terms of $P$, as $P$ directly corresponds to the total input length when integers are encoded in unary. Observe that while the case of $n = P$ (all jobs have unit processing times) essentially reduces to sorting, there are several non-trivial cases where $n$ is smaller than $P$ yet still quite significant in the $O(P \cdot n)$ time. In our paper [58], we have investigated how the $O(P^2)$-time algorithm can be improved.

**Improvements for $1||\sum w_jU_j$?** For $1||\sum w_jU_j$ there is some evidence that the answer to the analogous question should be negative. Karp [136] observed that the special case of the $1||\sum w_jU_j$ problem where all jobs have the same due date $d$ (the $1|d_j = d|\sum w_jU_j$ problem) is essentially equivalent to the classical 0/1-Knapsack problem. In two independent papers, Cygan, Mucha, Węgrzycki and Włodarczyk [86] and Künnemann, Paturi and Schneider [146] studied the $(\min,+)$-Convolution problem[17], and conjectured that the $(\min,+)$-convolution between two vectors of length $n$ cannot be computed in $O(n^{2-\epsilon})$ time, for any $\epsilon > 0$. Under this $(\min,+)$-convolution conjecture, they obtained lower bounds for several Knapsack related problems. In our terms, their result is that the $1|d_j = d|\sum w_jU_j$ problem cannot be solved in subquadratic time $O(P^{2-\epsilon})$ for any $\epsilon > 0$, unless

[17] *In analogy to standard integer convolution, the $(\min,+)$-convolution problem is to compute, given two integer vectors $A, B$, the vector $C$ defined by*

$$C[k] = \min_{i+j=k} A[i] + B[j].$$

*See also Section 2.1.*

the $(\min,+)$-convolution conjecture is false. In particular, $1||\sum w_jU_j$ has no subquadratic algorithm under this conjecture.

**Improvements for $1||\sum p_jU_j$?** Analogous to the situation with $1||\sum w_jU_j$, the special case of $1||\sum p_jU_j$ where all jobs have the same due date $d$ (the $1|d_j = d|\sum p_jU_j$ problem) is equivalent to the classical Subset Sum problem. Recently, there has been significant improvements for pseudo-polynomial-time algorithms for Subset Sum resulting in algorithms with $\widetilde{O}(t+n)$ running times [52, 129], where $n$ is the number of integers in the instance and $t$ is the target. In our language, this implies that the $1|d_j = d|\sum p_jU_j$ problem can be solved in near-linear time $\widetilde{O}(P)$.

In contrast, due to the equivalence of $1|d_j = d|\sum p_jU_j$ and Subset Sum, we also know that this algorithm cannot be significantly improved unless the Strong Exponential Time Hypothesis (SETH) fails. Specifically, combining a recent reduction from $k$-SAT to Subset Sum [3] with the equivalence of $1|d_j = d|\sum p_jU_j$ and Subset Sum, yields that there is no $\widetilde{O}(P^{1-\epsilon})$-time algorithm for the $1|d_j = d|\sum p_jU_j$ problem for any $\epsilon > 0$, unless SETH fails.

### 1.5.1 Our Contribution

These known results leave quite a big gap for the true time complexity of $1||\sum p_jU_j$, as it can potentially be anywhere between the quadratic-time $O(P^2)$ upper bound and the linear-time $P^{1-o(1)}$ lower bound. In particular, the $1||\sum p_jU_j$ and $1||\sum w_jU_j$ problems have not been distinguished from an algorithmic perspective so far. Our contribution is a new pseudo-polynomial time algorithms for the $1||\sum p_jU_j$ problem improving on Lawler and Moore's algorithm.

**Theorem 1.13 (Faster Minimization of Tardy Processing Time).** *The $1||\sum p_jU_j$ problem can be solved in time $\widetilde{O}(P^{7/4})$.*

Our approach is a prototypical application of the partition-and-convolve design paradigm: Starting from the dynamic programming algorithm by Lawler and Moore, we observe that the DP table can be built more efficiently. Specifically, after *partitioning* the DP table in two halves and recursively filling the respective entries, we can obtain the DP entries for the full problem by solving the following *convolution*-style problem.

**Definition 1.14 (Skewed Convolution).** *Let $A, B, S \in \mathbf{Z}^n$. The $(\max,\min)$-skewed convolution of $A, B, S$ is defined as the vector $C \in \mathbf{Z}^{2n-1}$ with entries*

$$C[k] = \max_{\substack{i,j\in[n]\\ i+j=k}} \min\{A[i], B[j] + S[k]\}.$$

One of our main technical contributions is a faster-than-brute-force algorithm for computing $(\max,\min)$-skewed convolutions.

We give the proof of Theorem 1.13 in Chapter 6. In the paper version [58] we also give pseudo-polynomial algorithms in terms of different parameters such as the sum of due dates $D$ and the number of distinct due dates $D_\#$.

### 1.5.2 Open Problems

Our work on the $1||\sum p_jU_j$ problem leaves open some interesting questions. We remark that some of these questions have already been tackled by follow-up work [137, 120, 188].

1 *Can the $1||\sum w_jU_j$ be solved even faster?*
In the meantime, Klein, Polak and Rohwedder [137] and Schieber and Sitaraman [188] have improved the running time to $\widetilde{O}(P^{7/5})$. The first improvement is a faster algorithm for $(\max,\min)$-skewed convolution [137] and the second

improvement is a more efficient reduction to that problem [188]. It is interesting to see whether the time complexity can be improved further, possibly even to near-linear time. Improving the time complexity of (max, min)-skewed convolution beyond $\widetilde{O}(P^{3/2})$ seems difficult as this would directly imply an improvement to the well-studied (max, min)-convolution problem [144].

Conversely, one could try to obtain fine-grained lower bounds for the problem, possibly in the same vein as [3].

2 *Can the algorithm be extended to multiple machines?*
That is, can we obtain similar improvements for the $P_m||\sum p_jU_j$ problem which allows to schedule the jobs on $m$ parallel machines? The baseline running time for this problem is $O(P^m \cdot n) = O(P^{m+1})$ (by an easy extension of Lawler and Moore's algorithm), but it is entirely possible that this problem can be solved in time $\widetilde{O}(P^m)$, or even faster.

3 *Can the techniques in this paper be applied to any other interesting scheduling problems?*
A good place to start might be to look at other problems which directly generalize Subset Sum. Hermelin, Molter and Shabtay [120], apply similar ideas to the $1||\sum w_jU_j$ problem for a variety of parameters (other than $P$, as this problem is quadratic-time hard for the parameter $P$).

## 1.6 Sublinear Algorithms for Approximate Edit Distance

We finally turn to our second pillar: Property Testing. Specifically, we include our result on approximating the edit distance of two strings in truly sublinear time [60].

**Edit Distance.** The *edit distance* (also called *Levenshtein distance* [151]) between two strings $X$ and $Y$ is the minimum number of character insertions, deletions and substitutions required to transform $X$ into $Y$. It constitutes a fundamental string similarity measure with applications across several disciplines, including computational biology, text processing and information retrieval.

Computational problems involving the edit distance have been studied extensively. A textbook dynamic programming algorithm computes the edit distance of two strings of length $n$ in time $O(n^2)$ [199, 200]. It is known that beating this quadratic time by a polynomial improvement would violate the Strong Exponential Time Hypothesis [28, 1, 63, 6], one of the cornerstones of fine-grained complexity theory. For faster algorithms, we therefore have to resort to *approximating* the edit distance. A long line of research (starting even before the hardness result emerged) lead to successively improved approximation algorithms: The first result established that in linear time the edit distance can be $O(\sqrt{n})$-approximated [147]. The approximation ratio was improved to $O(n^{3/7})$ in [31] and to $n^{1/3+o(1)}$ in [38]. Making use of the Ostrovsky-Rabani technique for embedding edit distance into the $\ell_1$-metric [167], Andoni and Onak [24] gave a $2^{O(\sqrt{\log n})}$-approximation algorithm which runs in time $n^{1+o(1)}$. Later, Andoni, Krauthgamer and Onak achieved an algorithm in time $O(n^{1+\epsilon})$ computing a $(\log n)^{O(1/\epsilon)}$-approximation [20]. A breakthrough result by Chakraborty, Das, Goldenberg, Koucký and Saks showed that it is even possible to compute a constant-factor approximation in strongly subquadratic time [68]. Subsequent work [51, 145] improved the running time to close-to-linear for the regime of near-linear edit distance. Very recently, Andoni and Nosatzki [22] extended this to the general case, showing that in time $O(n^{1+\epsilon})$ one can compute a $f(1/\epsilon)$-approximation for some function $f$ depending solely on $\epsilon$.

**Sublinear Algorithms.** As outlined before, while for exact algorithms linear-time algorithms are the gold standard, for approximation algorithms it is not clear

whether a $O(n^{1+\epsilon})$ running time is desired, as in fact one might hope for *sublinear-time* algorithms. Indeed, another line of research explored the edit distance in the sublinear setting, where one has random access to the strings $X$ and $Y$, and the goal is to compute the edit distance between $X$ and $Y$ without even reading the whole input. Formally, in the $(k, K)$-gap edit distance problem the task is to distinguish whether the edit distance is at most $k$ or at least $K$. The running time is analyzed in terms of the string length $n$ and the gap parameters $k$ and $K$. Initiating the study of this problem, Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami [37] showed how to solve the $(k, \Omega(n))$-gap problem in time $O(k^2/n + \sqrt{k})$, assuming $k \leq n^{1-\Omega(1)}$. The aforementioned algorithm by Andoni and Onak [24] can be viewed in the sublinear setting and solves the $(k, K)$-gap problem in time $n^{2+o(1)} \cdot k/K^2$, assuming $K/k = n^{\Omega(1)}$. In a major contribution, Goldenberg, Krauthgamer and Saha [111] showed how to solve the $(k, K)$-gap problem in time $\widetilde{O}(nk/K + k^3)$, assuming $K/k = 1 + \Omega(1)$. Subsequently, Kociumaka and Saha [141] improved the running time to $\widetilde{O}(nk/K + k^2 + \sqrt{nk^5}/K)$. They focus their presentation on the $(k, k^2)$-gap problem, where they achieve time $\widetilde{O}(n/k + k^2)$.

How far from optimal are these algorithms? It is well-known that $(k, K)$-gap edit distance requires time $\Omega(n/K)$ (this follows from the same bound for Hamming distance). Batu et al. [37] additionally proved that $(k, \Omega(n))$-gap edit distance requires time $\Omega(\sqrt{k})$. Together, $(k, K)$-gap edit distance requires time $\Omega(n/K + \sqrt{k})$, but this leaves a big gap to the known algorithms. In particular, in the low distance regime (where, say, $K \leq n^{0.01}$) the lower bound is $\Omega(n/K)$ and the upper bound is $\widetilde{O}(nk/K)$. Closing this gap has been raised as an open problem by the authors of the previous sublinear-time algorithms.[18] In particular, a natural question is to determine the smallest possible gap which can be distinguished within the time budget of the previous algorithms:

*Question 4: What is the best approximation factor of edit distance in sublinear time $O(n/k + \text{poly}(k))$?*

In particular, is the currently-best $(k, k^2)$-gap barrier penetrable or can one prove a lower bound? This quadratic gap appears to be the limit of the techniques of previous work [111, 141].

[18] *For the open problems raised in the conference talk on [111] see* https://youtu.be/WFxk3JAOC84?t=1104. *Similarly, for the open problems raised in the conference talk on [141] see* https://youtu.be/3jfHHEFNRU4?t=1159.

### 1.6.1 Our Contribution

We make significant progress on Question 4 by proving that the gap can be reduced to *subpolynomial* $k^{o(1)}$. Specifically, we show that $(k, k^{1+o(1)})$-gap edit distance can be solved in time $O(n/k + k^{4+o(1)})$. In the low distance regime ($k \leq n^{0.19}$) this runs in the same time as the previous algorithms for the $(k, k^2)$-gap problem. Formally, we obtain the following results.

**Theorem 1.15 (Subpolynomial Gap Edit Distance).** *The $(k, k \cdot 2^{\widetilde{O}(\sqrt{\log k})})$-gap edit distance problem can be solved in time $O(n/k + k^{4+o(1)})$.*

**Theorem 1.16 (Polylogarithmic Gap Edit Distance).** *The $(k, k \cdot (\log k)^{O(1/\epsilon)})$-gap edit distance problem can be solved in time $O(n/k^{1-\epsilon} + k^{4+o(1)})$, for any $\epsilon \in (0, 1)$.*

We provide the proofs of Theorems 1.15 and 1.16 in Chapter 7.

Note that one can solve the $(k, k^2)$-gap edit distance problem by running our algorithm from Theorem 1.15 for $\bar{k} := k^{2-o(1)}$. This runs in time $O(n/k^{2-o(1)} + \text{poly}(k))$, which improves the previously best running time of $\widetilde{O}(n/k + \text{poly}(k))$ for the $(k, k^2)$-gap edit distance problem [111, 141] by a factor $k^{1-o(1)}$ in the low distance regime.

**Edit Distance versus Hamming Distance.** It is interesting to compare our result against the best-possible sublinear-time algorithms for approximating the *Hamming distance*. For Hamming distance, it is well known that the $(k, K)$-gap problem has complexity $\Theta(n/K)$, with matching upper and (unconditional) lower bounds, assuming that $K/k = 1 + \Omega(1)$. In the large distance regime, Hamming distance and edit distance have been *separated* by the $\Omega(\sqrt{k})$ lower bound for $(k, \Omega(n))$-gap edit distance [37], because $(k, \Omega(n))$-gap Hamming distance can be solved in time $O(1)$.

Our results show a surprising *similarity* of Hamming distance and edit distance: In the low distance regime ($k \leq n^{0.19}$) the complexity of the $(k, k^{1+o(1)})$-gap problem is $n/k^{1\pm o(1)}$ for both Hamming distance and edit distance. Thus, up to $k^{o(1)}$-factors in the gap and running time, their complexity is the same in the low distance regime.[19]

[19] *We remark that this similarity does not yet follow from previous sublinear algorithms, since they solve the $(k, k^2)$-gap edit distance in time $O(n/k + \text{poly}(k))$ [111, 141], while $(k, k^2)$-gap Hamming distance can be solved much faster, namely in time $O(n/k^2)$.*

**Our Techniques.** To achieve our result, we depart from the framework of subsampling the Landau-Vishkin algorithm [148, 147], which has been developed by the state-of-the-art algorithms for sublinear edit distance [111, 141]. Instead, we pick up the thread from the *almost-linear-time* algorithm by Andoni, Krauthgamer and Onak [20]: First, we give (what we believe to be) a more accessible view on that algorithm. Subsequently, we design a sublinear version of it by *pruning* certain branches in its recursion tree and thereby avoid spending time on "cheap" subproblems. To this end, we use a variety of structural insights on the (local and global) patterns that can emerge during the algorithm and design property testers to effectively detect these patterns.

**Comparison to Goldenberg, Kociumaka, Krauthgamer and Saha [110].** Independently from our result, Goldenberg et al. [110] studied the complexity of the $(k, k^2)$-gap edit distance problem in terms of *non-adaptive* algorithms. Their main result is an $O(n/k^{3/2})$-time algorithm, and a matching query complexity lower bound. We remark that our results are incomparable: For the $(k, k^2)$-gap problem they obtain a non-adaptive algorithm (which is faster for large $k$), whereas we present an adaptive algorithm (which is faster for small $k$). Moreover, we can improve the gap to $(k, k^{1+o(1)})$, while still running in sublinear time $O(n/k)$ when $k$ is small. Our techniques also differ substantially: Goldenberg et al. build on the work of Andoni and Onak [24] and Batu et al. [37], whereas our algorithm borrows from [20].

### 1.6.2 Open Problems

Our work is the first step towards proving that decent approximations of the edit distance can be computed in sublinear time. We leave open many questions.

1. *Can we obtain almost-optimal algorithms in the high distance regime?*
   Specifically, can we reduce the $\text{poly}(k)$ term in the running time to match the $\Omega(n/k + \sqrt{n})$ lower bound?

2. *Can we reduce the gap to logarithmic or even constant?*
   In the almost-linear-time setting, Andoni and Nosatzki [22] have established a constant-factor approximation. It seems very hard to speed up their algorithm to sublinear time. Our take is that this complicated algorithm first has to undergo serious simplifications before being amenable to improvements.

3. *Can we at least obtain improved upper bounds in terms of the query complexity?*
   While finding algorithms with sublinear running time and constant gap seem out of reach, it might be easier to find a constant-factor approximation with unbounded running time, but with a truly sublinear number of *queries*.

## 1.7 Preliminaries

**Machine Model.** Throughout this thesis, we work in the word RAM model. This model is inspired by modern real-world computers and can be informally described as follows: The memory consists of *words* storing bit-strings of fixed length $w$. Given the address of a word (stored itself in a word), we can access the addressed memory cell in constant time. In addition, basic logical operations (such as bit-wise "and", "or" and "not") and arithmetic operations (such as addition, subtraction, multiplication and division with remainders) on single words run in constant time. The word size is typically $w = \Theta(\log n)$, where $n$ is the input size of the respective problem; in this way the words are large enough to address the whole input.

For problems like sparse convolution, where the input consists of numbers up to $\Delta$, we typically assume that the word size is $\Theta(\log n + \log \Delta)$ so that each input number can be stored in a single word.

**Randomized Algorithms.** Following standard terminology, we distinguish between two types of randomized algorithms: A *Monte Carlo* algorithm errs with some probability (returning no output or a wrong output) but has a worst-case time bound. A *Las Vegas* algorithm is always correct (with probability 1), but only has an *expected* running time bound. Whenever unspecified, by *algorithm* we mean a Monte Carlo algorithm with constant failure probability.

**Basic Notation.** We denote the nonnegative integers by $\mathbf{N}$, the integers by $\mathbf{Z}$, the rationals by $\mathbf{Q}$, the reals by $\mathbf{R}$, and the complex numbers by $\mathbf{C}$, respectively. For a prime power $q$, we let $\mathbf{F}_q$ denote the finite field of order $q$.

We write $[n] = \{0, \dots, n-1\}$ and define the integer intervals $[a\,..\,b] = \{a, \dots, b\}$ and similarly for the (half-)open intervals $[a\,..\,b)$, $(a\,..\,b]$, $(a\,..\,b)$. For intervals of real numbers, we set $[a, b] = \{x \in \mathbf{R} : a \le x \le b\}$ and similarly define the (half-)open intervals $[a, b)$, $(a, b]$, $(a, b)$.

In the running times, we often write $\mathrm{poly}(n) = n^{O(1)}$, $\mathrm{polylog}(n) = (\log n)^{O(1)}$ and $\mathrm{polyloglog}(n) = (\log \log n)^{O(1)}$. Moreover, we write $\widetilde{O}(T) = T\,\mathrm{polylog}(T)$.

Finally, we sometimes use the Iverson notation $[E] \in \{0, 1\}$ to denote the truth value of the expression $E$.

**Sumsets.** Let $G$ be an additive group, and consider sets $A, B \subseteq G$. We define the *sumset* notation $A + B = \{a + b : a \in A, b \in B\}$ and $A - B = \{a - b : a \in A, b \in B\}$. We will occasionally apply more general functions $f : G \to G$ to sets in the natural way, $f(A) = \{f(a) : a \in A\}$.

**Vectors.** We typically denote vectors by capital letters $A, B, C$. Throughout this thesis, we will index vectors starting at zero (and similarly for all related objects such as matrices and strings). For a length-$n$ vector $A$ and an index $i \in [n]$ we write $A[i]$ to access the $i$-th entry of $A$. The *support* $\mathrm{supp}(A)$ of $A$ is the set of nonzero coordinates, $\mathrm{supp}(A) = \{x \in [n] : A[x] \neq 0\}$. We write $\|A\|_0 = |\mathrm{supp}(A)|$ and say that $A$ is *$s$-sparse* if $\|A\|_0 \le s$. For a vector $A$ with real (or complex) entries, we additionally define the $\ell_p$-norms $\|A\|_p = (\sum_{i=0}^{n-1} |A[i]|^p)^{1/p}$ for all $p \in [1, \infty)$ and the $\ell_\infty$-norm $\|A\|_\infty = \max_{i=0}^{n-1} |A[i]|$. Finally, for a function $f : [n] \to [m]$ and a length-$n$ vector $A$, we define the vector $f(A)$ via

$$f(A)[x] = \sum_{\substack{i \in [n] \\ f(i) = x}} A[i].$$

We formally define the convolution $A \star B$ of two vectors at the beginning of Chapter 2.

# 2 Sparse Convolution Toolkit

In this chapter we introduce the known techniques for computing sparse convolutions. This chapter is not based on a publication, but rather summarizes the diverse approaches and the rich toolkit that I have learned (and helped to develop) while working on sparse convolutions. As a bonus we design simple algorithms for sparse integer convolution and for sparse nonnegative convolution over finite-field vector spaces, that have not appeared in the literature before. This chapter also serves as preliminaries to the upcoming Chapters 3 and 4 both of which specialize on improved sparse convolution algorithms.

**Organization.** We start with a recap about the dense convolution problem in Section 2.1. In the following three Sections 2.2 to 2.4 we describe three approaches to computing sparse convolutions and highlight their respective benefits and drawbacks. In Sections 2.5 to 2.7 we equip ourselves with some more tools that will be handy later on.

## 2.1 Dense Convolution

The *convolution* of two integer vectors $A, B$ (of length $n$) is defined as the integer vector $A \star B$ (of length $2n-1$) with entries

$$(A \star B)[k] = \sum_{\substack{i,j \in [n] \\ i+j=k}} A[i] \cdot B[j].$$

Many researchers prefer to equivalently view the convolution of two vectors of length $n$ as the multiplication of two (univariate) polynomials of degree $n-1$. In this thesis we will stick to vectors, mainly for two reasons: First, while the polynomial view point often plays nicer with algebraic operations, it is at times awkward to use combinatorial tricks (such as hashing). Second, we will soon introduce generalizations of the standard integer convolution which no longer admit equivalent views in terms of polynomials.

We also define the *cyclic convolution*

$$(A \star_n B)[k] = \sum_{\substack{i,j \in [n] \\ i+j=k \pmod n}} A[i] \cdot B[j].$$

It is easy to see that computing cyclic and computing non-cyclic convolutions have the same asymptotic running time,[20] but in many situations it is more convenient to compute cyclic convolutions. Both can be computed by the Fast Fourier Transform (FFT) in near-linear time:

[20] *Interpret the vectors $A, B$ as vectors of length $2n$. Then the cyclic convolution $A \star_n B$ equals the non-cyclic convolution $A \star B$ (up to trailing zeros).*

**Theorem 2.1 (Fast Fourier Transform, [85]).** *Given integer vectors $A, B$ of length $n$, we can compute $A \star B$ in time $O(n \log n)$.*

**Boolean and Nonnegative Convolution.** Our study particularly focuses on two special cases of integer convolution. The *Boolean convolution* of two Boolean vectors $A, B \in \{0,1\}^n$ is the Boolean vector $A \circledast B \in \{0,1\}^{2n-1}$ defined by

$$(A \circledast B)[k] = \bigvee_{\substack{i,j \in [n] \\ i+j=k}} A[i] \wedge B[j].$$

Equivalently, note that $A \circledast B$ is the vector obtained from $A \star B$ by replacing positive entries by 1. In particular, note that computing Boolean convolutions is a special

case of computing integer convolutions. Recall that the Boolean convolution problem is equivalent to the computation of sumsets $X + Y = \{x \in X,\ y \in Y\}$ for two given sets $X, Y \subseteq [n]$. An interesting result due to Indyk [124] is how to leverage bit tricks to obtain a different kind of algorithm: In linear time we can compute an approximation of $A \star B$ that is correct on a constant fraction of entries. Specifically:

**Theorem 2.2 (Approximate Sumsets, [124]).** *There exists a randomized algorithm which, given two sets $X, Y \subseteq [n]$ computes in time $O(n)$ a set $\widetilde{Z} \subseteq X + Y$, such that for all $z \in X + Y$ we have $\mathbf{P}(x \in \widetilde{Z}) \geq \frac{2}{3}$.*

In the slightly more general *nonnegative convolution* problem we require the vectors $A, B$ to have with nonnegative entries. We will simply say that an integer vector $A$ is *nonnegative* whenever all of its entries are nonnegative. In the sumset analogy, computing the nonnegative convolution of the indicator vectors results not only in computing $X + Y$ but also the *multiplicities* of the sums $x + y$. In this thesis we focus algorithms for *sparse nonnegative* convolution, due to the numerous algorithmic applications.

**General Convolution.** We will occasionally consider convolutions in a more general context. Let $(R, \oplus, \otimes)$ be a ring and let $(G, +)$ be a group. Consider vectors $A, B \in R^G$ (that is, we view $A$ and $B$ as functions $G \to R$). Then we define the convolution $A \star B \in R^G$ via[21]

$$(A \star B)[k] = \bigoplus_{\substack{i,j \in G \\ i+j=k}} A[i] \otimes B[j].$$

[21] *In more algebraic terms, the set $R^G$ equipped with coordinate-wise* addition *and convolution as* multiplication *is often referred to as the* group algebra of $G$ over $R$ *and often denoted by $R[G]$.*

This is indeed a generalization, as by taking $G = C_n$ to be the cyclic group of size $n$ and by taking $R = \mathbf{Z}$ to be the integer ring we exactly recover cyclic convolutions. It is however not clear that an FFT algorithm exists for this more general problem. A long line of research [85, 46, 39, 81, 40, 41, 181, 198] starting with Cooley and Tukeys seminal result [85] has lead to successively more and more general (in terms of $G$) convolution algorithms culminating in the following two state-of-the-art results. Both results are stated in terms of *arithmetic circuits*[22] which is a natural model of computation in this context. Here, $2 \leq \omega < 2.372$ is the exponent of matrix multiplication [14, 91].

[22] *An* arithmetic circuit *is a directed acyclic graph where each source node is labeled with an input, each sink node is labeled with an output and each internal node is labeled with an operation $+$ or $\times$. The outputs are computed by propagating the inputs through the graph while applying the respective operations. See [102] for a formal definition.*

**Theorem 2.3 (Fast Fourier Transform for Abelian Groups, [41]).** *Let $G$ be a finite Abelian group. There is an arithmetic circuit to compute the convolution of given vectors $A, B \in \mathbf{C}^G$ in $O(|G| \log |G|)$ arithmetic operations.*

**Theorem 2.4 (Fast Fourier Transform for Arbitrary Groups, [198]).** *Let $G$ be an arbitrary finite group. There is an arithmetic circuit to compute the convolution of given vectors $A, B \in \mathbf{C}^G$ in $|G|^{\omega/2+o(1)}$ arithmetic operations.*

To be applicable in our context, we need to turn these results into word RAM algorithms. It is possible to compute general convolutions of integer vectors in the word RAM model by simulating the arithmetic circuits from Theorems 2.3 and 2.4 (either using complex arithmetic with bounded precision or by substituting $\mathbf{C}$ with an appropriate finite field). Here, however, the representation of the group $G$ matters critically. For instance, each finite Abelian group can be represented as the product of cyclic groups of prime power order, and given this representation the arithmetic circuit from Theorem 2.3 leads to a word RAM algorithm running in time $O(|G| \log |G|)$. For more complicated groups it is a priori not clear how to turn Theorem 2.4 into an efficient algorithm.

**Even More General Convolution.** While throughout this thesis we mostly focus on the case $R = \mathbf{Z}$, it is possible to define convolutions when $R$ is only a semiring[23]. For nice rings $R$ it is often possible to compute convolutions in near-linear time in the same vein as the previous two theorems. In contrast, computing convolutions

[23] *Recall that a* semiring *is a ring that does not necessarily have additive inverses.*

over semirings leads to significant changes in the complexity. Note that we can always compute $A \star B$ naively in time $O(n^2)$, and it is an interesting question from fine-grained complexity for which semirings there are better algorithms.

For example, a popular hypothesis in fine-grained complexity states that computing the $(\min,+)$-convolution[24] indeed requires quadratic time (up to lower-order factors) [86, 146]:

**Conjecture 2.5 ($(\min,+)$-Convolution).** *For any $\epsilon > 0$, there is no algorithm to compute the $(\min,+)$-convolution of two given length-$n$ vectors in time $O(n^{2-\epsilon})$.*

In fact, in the course of the last decade the fine-grained study of generalized convolution problems has lead to a whole landscape of complexities from near-linear to almost-quadratic time [86, 146, 153, 58, 137]. One interesting intermediate problem with conjectured running time $n^{3/2\pm o(1)}$ is $(\min,\max)$-convolution[25].

We remark that the landscape of semiring-convolution problems is mirrored by *matrix products* over semirings. For example, the matrix analogue of $(\min,+)$-convolution is the $(\min,+)$-product problem[26], which is fine-grained equivalent to the famous All-Pairs Shortest Paths problem in graphs [95, 203]. Corresponding convolution and matrix problems often behave very similarly in terms of their algorithmic and lower bound techniques, and sometimes there are even fine-grained reductions from semiring-convolution problems to their respective matrix problems.

[24] *The convolution over the semiring $(\mathbf{Z} \cup \{\infty\}, \min, +)$. At times this is called the* tropical *semiring.*

[25] *The convolution over the semiring $(\mathbf{Z} \cup \{-\infty, \infty\}, \min, \max)$. At times this is called the* subtropical *semiring.*

[26] *Given two $n \times n$ integer matrices $A, B$, compute the integer matrix $C$ defined by $C[i,j] = \min_k A[i,k]+B[k,j]$.*

**Sparse Convolution.** From now on we focus on the *sparse* convolution problem, where the goal is to design algorithms sensitive to the input plus output size

$$t = \|A\|_0 + \|B\|_0 + \|A \star B\|_0.$$

In the following three Sections 2.2 to 2.4 we describe three approaches to computing sparse convolutions and highlight their respective benefits and drawbacks. In the following Sections 2.5 to 2.7 we establish more tools which we need in the later Chapters 3 and 4.

## 2.2 Sparse Convolution via Additive Hashing

The simplest approach to sparse convolutions is via hashing. Specifically, via hashing with hash functions $h : [n] \to [m]$ satisfying the *additiveness* (or *linearity*) property that $h(x) + h(y) = h(x+y) \pmod m$. We start with a brief exposition in Section 2.2.1 about known additive hash functions all which will later play important roles. Later, in Section 2.2.2 we design a simple algorithm for sparse integer convolution based on one of these hash functions.

### 2.2.1 Collection of Additive Hash Families

In this section, we introduce four families of hash functions each satisfying an additiveness(-like) property.

**Family 1: Hashing Modulo a Random Prime.** The most basic additive hash family is the family of functions $h(x) = x \bmod p$ where $p$ is a random prime. (In more mathy terms, this is a projection to a random subgroup of $\mathbf{Z}$.)

**Lemma 2.6 (Random Prime Hashing).** *Let $n \geq m$ be arbitrary. The family of hash functions $h(x) = x \bmod p$ where $p \in [m\,..\,2m]$ is a random prime satisfies the following three properties.*

1. Efficiency: *Sampling $h$ takes time* polylog $m$ *and evaluating takes constant time.*
2. $O(\log n)$-Universality: *For distinct keys $x, y \in [n]$: $\mathbf{P}(h(x) = h(y)) \leq \frac{2\log n}{m}$.*
3. Additiveness: *For all keys $x, y \in [n]$: $h(x) + h(y) \equiv h(x+y) \pmod p$.*

**Proof.** ▶ The additiveness property is obvious, so focus on universality and fix two distinct keys $x, y \in [U]$. It holds that $h(x) = h(y)$ if and only if $p$ divides the difference $x - y$. Since $|x - y| \leq n$, $x - y$ has at most $\log_m(n)$ distinct prime factors in the range $[m \,..\, 2m]$. On the other hand, by a quantitative version of the Prime Number Theorem [184, Corollary 3], there are at least $\frac{3m}{5 \ln m}$ primes in the range $[m \,..\, 2m]$ (for sufficiently large $m$). Hence, the probability that $p$ divides $x-y$ is at most $5 \log_m(n) \ln(m)/(3m) \leq 2 \log(n)/m$. ◀

The main selling point of this hash function is that it is truly additive and very simple. Unfortunately, it is not perfectly uniform. The next families 2 and 3 will perform better in this regard, but come with a weaker additiveness-like property.

**Family 2: Linear Hashing.** *Linear hashing* is one of the classic textbook hash functions $h : [n] \to [m]$ defined as follows:

$$h(x) = ((\sigma x + \tau) \bmod p) \bmod m.$$

Here, $p \geq n$ is some fixed prime, $m \leq p$ is the fixed number of buckets and $\sigma$ and $\tau$ are chosen uniformly and independently at random from $[p]$. We say that $h$ is a *linear hash function with parameters $p$ and $m$*. Linear hashing is particularly powerful as this family is $O(1)$-universal (even $(1+o(1))$-universal, to be precise).

**Lemma 2.7 (Linear Hashing).** *Let $h$ be a linear hash function with parameters $p$ and $m$ drawn uniformly at random. Then the following properties hold:*

1. Efficiency: *Sampling and evaluating $h$ takes constant time (assuming that $p$ is fixed).*
2. Almost-Additiveness: *For all keys $x, y \in [n]$ there is an offset $o \in \{-p, 0, p\}$ such that $h(x) + h(y) = h(0) + h(x + y) + o \pmod m$.*
3. $O(1)$-Universality: *For distinct keys $x, y \in [n]$: $\mathbf{P}(h(x) = h(y)) \leq O(\frac{1}{m})$.*
4. Pairwise Independence: *For distinct keys $x, y \in [n]$ and buckets $a, b \in [m]$:* $|\, \mathbf{P}(h(x) = a \wedge h(y) = b) - \frac{1}{m^2} \,| \leq \frac{3}{mp} \leq \frac{3}{m^2}$.

We include a proof for completeness.

**Proof.** ▶ Universality follows directly from pairwise independence, so we start proving the pairwise independence property. We rewrite $h$ as $h(x) = g(x) \bmod m$, where $g(x) = (\sigma x + \tau) \bmod p$ for uniformly random $\sigma, \tau \in [p]$. The first step is to prove that $\mathbf{P}(g(x) = a' \wedge g(y) = b') = 1/p^2$ for distinct keys $x, y$ and arbitrary buckets $a', b' \in [p]$. Note that the event $g(x) = a'$ and $g(y) = b'$ can be rewritten as $g(x) = a'$ and $g(y) - g(x) = b' - a' \pmod p$ and the claim follows immediately by observing that the random variables $g(x)$ and $g(y) - g(x) = g(y - x) \pmod p$ are independent.

We get back to $h$. Clearly,

$$\mathbf{P}(h(x) = a \wedge h(y) = b) = \sum_{\substack{a', b' \\ a = a' \bmod m \\ b = b' \bmod m}} \mathbf{P}(g(x) = a' \wedge g(y) = b') = \sum_{\substack{a', b' \\ a = a' \bmod m \\ b = b' \bmod m}} \frac{1}{p^2},$$

There are at least $\lfloor p/m \rfloor$ and at most $\lceil p/m \rceil$ such values $a'$ and $b'$, respectively, and we conclude that the desired probability is at least $\lfloor p/m \rfloor^2/p^2 \geq (p/m - 1)^2/p^2 \geq \frac{1}{m^2} - \frac{2}{mp}$ and at most $\lceil p/m \rceil^2/p^2 \leq (p/m + 1)^2/p^2 \leq \frac{1}{m^2} + \frac{3}{mp}$.

Finally, we prove that $h$ is almost-additive. For the inner function $g$ it is clear that $g(x) + g(y) = g(0) + g(x + y) \pmod p$. As each side of the equation is a nonnegative integer less than $2p$, it follows that $g(x) + g(y) = g(0) + g(x + y) + o$, where $o \in \{-p, 0, p\}$. By taking residues modulo $m$, the claim follows. ◀

We remark that linear hashing is closely related to another family of hash functions, which satisfy the same properties as in Lemma 2.7:

$$h(x) = \left\lfloor \frac{m \cdot ((\sigma x + \tau) \bmod p)}{p} \right\rfloor.$$

(Here, as before $p$ and $m$ are fixed and $\sigma, \tau \in [p]$ are uniformly random.) While most of our algorithms relying on linear hashing would also work with this hash function, for simplicity we will stick to linear hashing throughout.

**Family 3: Linear Hashing without Primes.** Linear hashing comes with the drawback that we have to precompute a (possibly large) prime number $p$ in polylogarithmic time. For most applications this overhead is negligible—but in some cases it critically incurs an additive polylog $n$ term to the running time which is otherwise independent of $n$. One can remove this overhead and perform linear hashing *without* prime numbers as proven e.g. in [87]. We provide a different self-contained proof.

**Lemma 2.8 (Linear Hashing without Primes).** *Let $n \geq m$ be arbitrary. There is a family of hash functions $h : [n] \to [m]$ with the following three properties.*

1. Efficiency: *Sampling and evaluating $h$ takes constant time.*
2. Uniform Differences: *For any distinct keys $x, y \in [n]$ and for any $q \in [m]$, the probability that $h(x) - h(y) \equiv q \pmod m$ is at most $O(\frac{1}{m})$.*
3. Almost-Additiveness: *There exists a constant-size set $\Phi \subseteq [m]$ such that for all keys $x, y \in [n]$ it holds that $h(x)+h(y) \equiv h(x+y)+\phi \pmod m$ for some $\phi \in \Phi$.*

For the proof we need the following lemma. A set $A = \{r + ia : i \in [|A|]\} \subseteq \mathbf{Z}$ is called an *arithmetic progression* with *step-width* $a$. The following lemma proves that two arithmetic progressions with coprime step-widths are as uncorrelated as possible.

**Lemma 2.9 (Coprime Step-Widths).** *Let $A$ and $B$ be arithmetic progressions with coprime step-widths $a$ and $b$, respectively. Then $|A \cap B| \leq \min\{\frac{|A|-1}{b}, \frac{|B|-1}{a}\} + 1$.*

**Proof.** ▸ We may assume that $A = \{0, a, \dots, (|A|-1)a\}$ and $B = \{0, b, \dots, (|B|-1)b\}$ (remove all points before the first common element of $A$ and $B$ and shift such that the first common element becomes zero). Since $a$ and $b$ are coprime, the intersection $A \cap B$ consists only of multiples of $ab$ and thus $|A \cap B| \leq \lfloor \frac{(|A|-1)a}{ab} \rfloor + 1$. The same bound holds symmetrically for $B$. ◂

**Proof of Lemma 2.8.** ▸ First, assume that $m$ is odd. Let $N$ be the smallest power of two larger than $n \cdot m$. We then define the family of hash functions as

$$h(x) = (\sigma x \bmod N) \bmod m,$$

where $\sigma \in [N]$ is a random *odd* number. We will now prove the three claimed properties for this family.

1. *Efficiency:* Sampling $h$ only involves constructing $N$ and sampling a random odd number. Both operations take constant time in the word RAM model. Evaluating $h$ is also in constant time.
3. *Almost-Additiveness:* Fix any keys $x, y \in [n]$. Then for one of the two choices $\phi \in \{0, N\}$ it holds that $(\sigma x \bmod N) + (\sigma y \bmod N) = (\sigma(x + y) \bmod N) + \phi$. By reducing this equation modulo $m$, it follows that $\Phi = \{0, N \bmod m\}$ is a suitable choice.
2. *Uniform Differences:* To prove that $h$ satisfies the uniform difference property, it suffices to prove that $h$ is $O(1)$-uniform, i.e., to prove that $\mathbf{P}(h(z) = a) \leq O(\frac{1}{m})$ for all keys $z \in [n]$ and all buckets $a \in [m]$. Indeed, from the previous paragraph we learn that $h(x) - h(y) \equiv q \pmod m$ only if $h(x - y) \equiv q - \phi \pmod m$ for some $\phi \in \Phi$. Taking a union bound over the constant number of elements $\phi$ then yields the claim.

   To check that $h$ is $O(1)$-uniform, we write $z = 2^k \cdot w$ where $w$ is odd. Then $\sigma z \bmod N$ is uniformly distributed in $A = \{2^k \cdot i : i \in [2^{-k} \cdot N]\}$.

Indeed, by identifying $[N]$ with the finite ring $\mathbf{Z}/N\mathbf{Z}$, $A$ is the smallest additive subgroup of $\mathbf{Z}/N\mathbf{Z}$ which contains $z$, and thus multiplying with a random unit $\sigma \in (\mathbf{Z}/N\mathbf{Z})^\times$ randomly permutes $z$ within that subgroup. It follows that

$$\mathbf{P}(h(z) = a) = \frac{2^k \cdot |A \cap B|}{N},$$

where $B \subseteq [N]$ consists of all numbers equal to $a$ modulo $m$. Observe that $A$ and $B$ are both arithmetic progressions with step-widths $2^k$ and $m$, respectively. Recall that $m$ is odd, therefore $2^k$ and $m$ are coprime and Lemma 2.9 applies and yields $|A \cap B| \leq \frac{N}{2^k m} + 1$. We finally obtain $\mathbf{P}(h(z) = a) \leq \frac{1}{m} + \frac{n}{N} \leq O(\frac{1}{m})$.

Finally, we remove the assumption that $m$ is odd. If $m$ is even, then we simply apply the previous construction to obtain a linear hash function $h : [n] \to [m-1]$ and reinterpret this as a function $h : [n] \to [m]$. It is easy to see that this preserves efficiency and uniform differences, and we claim that is also preserves almost-additiveness. Indeed, fix arbitrary keys $x, y$. From the construction we know that $h(x) + h(y) \equiv h(x+y) + \phi \pmod{(m-1)}$ for some $\phi \in \Phi$. Both sides of the equation are integers less than $2m-2$ and hence their images modulo $m-1$ and $m$, respectively, differ by at most 1. Therefore, we have $h(x)+h(y) \equiv h(x+y)+\phi' \pmod{m}$ for some $\phi' \in \Phi' = \{\phi + \psi : \phi \in \Phi, \psi \in \{-1, 0, 1\}\}$. ◀

**Family 4: Projection to Random Subspaces.** As the last example of additive hashing, we consider rings other than the integers, namely finite-field vector spaces $\mathbf{F}_p^d$. There is a simple family of hash functions—projections to random subspaces:

**Lemma 2.10 (Hashing via Random Linear Maps).** *Let $h : \mathbf{F}_p^d \to \mathbf{F}_p^{d'}$ be a random linear map (i.e., let $H \in \mathbf{F}_p^{d' \times d}$ be a random matrix, and let $h(x) = Hx$). Then the following properties are satisfied:*

1. Additiveness: $h(x+y) = h(x) + h(y)$ *for all* $x, y \in \mathbf{F}_p^d$.
2. Independence: *For any linearly independent vectors $x_1, \ldots, x_k \in \mathbf{F}_p^d$ (in particular, the $x_i$'s must be nonzero), the random variables $h(x_1), \ldots, h(x_k)$ are independent, and for any $a_1, \ldots, a_k \in \mathbf{F}_p^{d'}$ we have*

$$\mathbf{P}(h(x_1) = a_1 \text{ and } \ldots \text{ and } h(x_k) = a_k) = (p^{d'})^{-k},$$

*More generally, for any $x_1, \ldots, x_k \in \mathbf{F}_p^d$ (not necessarily linearly independent) and any $a_1, \ldots, a_k \in \mathbf{F}_p^{d'}$ we have that*

$$\mathbf{P}(h(x_1) = a_1 \text{ and } \ldots \text{ and } h(x_k) = a_k) \leq (p^{d'})^{-s},$$

*where $s = \dim\langle x_1, \ldots, x_k \rangle$.*

**Proof.** ▶ The additiveness property is obvious. To prove the independence statement, first recall that any set of linearly independent vectors $x_1, \ldots, x_k$ can be written as $x_i = Me_i$, where $M$ is a full-rank matrix and $e_i$ is the all-zeros vector with a single 1 in position $i$. Next, observe that the matrix $HM$ is uniformly random (indeed for any fixed matrix $N$ we have that $\mathbf{P}(HM = N) = \mathbf{P}(H = NM^{-1})$ and $H$ is uniformly random). It follows that the hash values $h(x_i) = HMe_i$ are the columns of a uniformly random matrix and therefore independent. Hence:

$$\mathbf{P}(h(x_1) = a_1 \text{ and } \ldots \text{ and } h(x_k) = a_k) = (p^{d'})^{-k},$$

for any $a_1, \ldots, a_k \in \mathbf{F}_p^{d'}$.

To obtain the more general statement for vectors which are not necessarily linearly independent, select a subset from $\{x_1, \ldots, x_k\}$ of $\dim\langle x_1, \ldots, x_k \rangle$ linearly independent vectors. For this subset, the hash values behave independently. ◀

### 2.2.2 Sparse Integer Convolution via Additive Hashing

In this section we give a simple algorithm for sparse integer convolution (that is, for sparse polynomial multiplication). The algorithm is Monte Carlo randomized. The idea is to find the convolution $A \star B$ via *iterative recovery*.

While this algorithm has never appeared in the literature, it borrows heavily from common ideas used in several papers [164, 123, 127, 105].[27] It follows three key ideas:

**Key Idea 1: Additive Hashing.** Let $h : [n] \to [m = \widetilde{\Theta}(t)]$ be an additive hash function.[28] We lift $h$ to vectors, and define $h(A)$ and $h(B)$ as length-$m$ vectors via

$$h(A)[x] = \sum_{\substack{i \in [n] \\ h(i)=x}} A[i], \qquad h(B)[y] = \sum_{\substack{j \in [n] \\ h(j)=y}} B[i].$$

That is, $h(A)$ is the vector obtained from $A$ by hashing each entry $A[i]$ to the bucket $h(i)$ and summing all entries falling into the same bucket. We can compute $h(A)$ and $h(B)$ in linear time $O(t)$, and we can compute their cyclic convolution $h(A) \star_m h(B)$ via FFT in time $\widetilde{O}(t)$. The crucial insight is the following lemma:

**Lemma 2.11 (Convolutions and Additive Hashing).** *Let $h : [n] \to [m]$ be an additive hash function (that is, for all $x, y$ we have $h(x) + h(y) = h(x + y) \pmod m$). Then $h(A) \star_m h(B) = h(A \star B)$.*

**Proof.** ▸ Let $z$ be arbitrary. The proof is a simple calculation; for simplicity we denote equality modulo $m$ by $\equiv$.

$$h(A \star B)[z] = \sum_{\substack{k \in [n] \\ h(k)=z}} (A \star B)[k] = \sum_{\substack{i,j \in [n] \\ h(i+j)=z}} A[i] \cdot B[j] = \sum_{\substack{i,j \in [n] \\ h(i)+h(j) \equiv z}} A[i] \cdot B[j]$$
$$= \sum_{\substack{x,y \in [m] \\ x+y \equiv z}} h(A)[x] \cdot h(B)[y] = (h(A) \star_m h(B))[z]. \quad \blacktriangleleft$$

With this lemma in mind, we have access to the vector $h(A \star B)$. With good probability, many nonzero entries $(A \star B)[k]$ are *isolated* under the hashing (that is, there is no other nonzero entry $(A \star B)[k']$ such that $h(k) = h(k')$), and our goal is to recover all of these entries. For each isolated entry $k$, we clearly have that $h(A \star B)[h(k)] = (A \star B)[k]$. This seems helpful but not sufficient, as the algorithm cannot infer $k$ by only seeing the entry $h(A \star B)[h(k)]$.

**Key Idea 2: Sparse Recovery and the Derivative Trick.** The solution is the standard idea from sparse recovery: We need to add some sort of *identifier* to hashed vector. In our scenario, an elegant solution is the derivative trick used by Huang [123]: Let $\partial A$ denote the *derivative* vector with entries $(\partial A)[i] = i \cdot A[i]$.[29] The following familiar lemma applies:

**Lemma 2.12 (Product Rule).** *Let $A, B$ be vectors. Then $\partial(A \star B) = (\partial A) \star B + A \star (\partial B)$.*

**Proof.** ▸ Again, the proof is a simple calculation. Let $k$ be arbitrary, then:

$$\partial(A \star B)[k] = k \cdot (A \star B)[k] = \sum_{\substack{i,j \in [n] \\ i+j=k}} (i + j) \cdot A[i] \cdot B[j]$$
$$= \sum_{\substack{i,j \in [n] \\ i+j=k}} i \cdot A[i] \cdot B[j] + \sum_{\substack{i,j \in [n] \\ i+j=k}} A[i] \cdot j \cdot B[j] = (\partial A \star B)[k] + (A \star \partial B)[k]. \quad \blacktriangleleft$$

Using this idea, we can compute $h(\partial A \star B)$ and $h(A \star \partial B)$ (as described in Key Idea 1) and thereby obtain their sum $h(\partial(A \star B))$. Coming back to our previous

[27] *More precisely, we essentially follow Nakos' approach to integer convolution [164], but replace the way the sparse recovery identifiers are chosen by Huang's derivative trick [123]. This removes the need for complex arithmetic and a tedious precision analysis.*

[28] *Here, we assume for simplicity that $t = \|A \star B\|_0$ is known, but this assumption is not necessary in the algorithm.*

[29] *This naming is in analogy to derivatives of polynomials which in addition to scaling also* shift *the coefficient vectors. While our definition seems odd at first, this version leads to cleaner algorithms.*

**Algorithm 2.1.** A simple algorithm for sparse integer convolution (aka polynomial multiplication). Given two vectors $A, B \in \mathbf{Z}^n$, this algorithm computes their convolution $A \star B$.

```
1  t_0 ← ‖A‖_0 + ‖B‖_0
2  for t ← t_0, 2t_0, 4t_0, ..., ∞ do
3      C ← (0, ..., 0)
4      for ℓ ← 0, 1, ..., ⌈log t⌉ do
5          Sample a random prime m ∈ [128/2^ℓ · t log n, 256/2^ℓ · t log n]
6          Let h(x) = x mod m
7          Compute V ← h(A) ⋆_m h(B) − h(C) via FFT
8          Compute W ← h(∂A) ⋆_m h(B) + h(A) ⋆_m h(∂B) − h(∂C) via FFT
9          for each z ∈ [m] do
10             k ← W[z] / V[z]
11             if k ∈ [2n − 1] then
12                 C[k] ← C[k] + V[z]
13     if VERIFY(A, B, C) then return C
```

problem, assume that $v = (A \star B)[k]$ is an isolated entry. Then it is easy to check that $h(A \star B)[h(k)] = v$ and $h(\partial(A \star B))[h(k)] = k \cdot v$. Taking the quotient of these two values, we have correctly identified $k$.

Consider the following recovery algorithm which uses this idea: We maintain a vector $C$ which is initially all-zeros. For each bucket $z \in [m]$, we compute the quotient $k \leftarrow h(\partial(A \star B))[z] \,/\, h(A \star B)[z]$. We test whether $k$ is an integer in the feasible range $[2n-1]$, and only in this case update $C[k] \leftarrow (A \star B)[z]$. At this point we exploit a second property about the hash function $h$: Assuming that the hash family is $\widetilde{O}(1)$-universal, at least a constant fraction, say $\frac{3}{4}$, of entries becomes isolated under $h$. Therefore, this recovery loop correctly recovers at least a $\frac{3}{4}$ fraction of the entries in $A \star B$ correctly. Unfortunately, the non-isolated entries might introduce some errors to the recovery; note that we cannot guarantee that each index $k$ as computed by the algorithm actually corresponds to an isolated index—non-isolated buckets can coincidentally lead to integers quotients $k$.

A standard idea is to repeat this algorithm $\log t$ times so that with high probability each index was recovered correctly in the majority of cases. Unfortunately, this is not as is easy as it looks: The errors in the recovery could, in principle, corrupt the same index $k$ in each repetition. We take another approach to fix this remaining issue.

**Key Idea 3: Iterative Recovery.** Instead, we will run the algorithm for several iterations and fix these errors one by one until we have computed $C = A \star B$. That is, we start from the all-zeros vector $C$ and apply the previous two steps to arrive at an approximation $C \approx A \star B$. In the next iteration, our goal is to recover the *residual vector* $A \star B - C$. Again we recover a constant fraction of coordinates correctly and thereby correct some errors introduced in the first iteration (while introducing some new errors). After $O(\log t)$ iterations we have removed all errors with high probability.

This idea can be implemented very efficiently by taking into account that the sparsity of the residual vector decreases geometrically across the iterations. We can therefore also use geometrically smaller and smaller bucket sizes and ultimately save a factor $\log t$ in the running time.

We summarize the complete algorithm in Algorithm 2.1 and include a short formal analysis. The algorithm does not assume that the sparsity $t$ is known, but instead guesses $t$ by exponential search. We also assume access to an algorithm $\textsc{Verify}(A, B, C)$ that tests whether $A \star B = C$ (with high probability); we will design this algorithm in Section 2.5.

**Theorem 2.13 (Sparse Integer Convolution).** *With high probability, Algorithm 2.1 returns the convolution $A \star B$ of the given vectors $A, B \in \mathbf{Z}^n$, and it runs in expected time $O(t \log^2 n)$ where $t = \|A\|_0 + \|B\|_0 + \|A \star B\|_0$.*

**Proof.** ▸ We will ignore the iterations of the outer loop until we have reached the critical value $t \geq \|A \star B\|_0$. As the verifier is correct with high probability, the algorithm will not prematurely stop and return a wrong answer with high probability. Focus on one iteration after the critical threshold. We claim that with constant probability the inner loop correctly computes $C = A \star B$ and the algorithm terminates.

We refer to the iterations of the inner loop as *levels* $\ell$. Let $C_\ell$ denote the vector $C$ after executing the $\ell$-th iteration of the inner loop; i.e., $C_0$ is the all-zeros vector and $C_L$ for $L = \lceil \log t \rceil$ hopefully equals $C$. We say that the $\ell$-th level is *successful* if the residual vector $A \star B - C$ has sparsity $\|A \star B - C_\ell\| \leq t/4^\ell$. We prove by induction that, conditioned on the events that the levels $0, 1, \ldots, \ell - 1$ are successful, also the $\ell$-th level is successful with probability at least $1 - 2^{-\ell-1}$.

For the base case, note that $\|A \star B - C_0\| = \|A \star B\| \leq t$. So let $\ell > 0$ and assume that all previous levels $0, 1, \ldots, \ell - 1$ were successful. Let $m$ be the prime sampled in the $\ell$-th level and let $h(x) = x \bmod m$ as Algorithm 2.1. We say that an index $k \in \operatorname{supp}(A \star B - C_{\ell-1})$ is *isolated* if there is no other index $k' \in \operatorname{supp}(A \star B - C_{\ell-1})$ with $h(k) = h(k')$. By the universality of our hash family (see Lemma 2.6), the event $h(k) = h(k')$ happens with probability at most

$$\frac{2 \log n}{\frac{128t}{2^\ell} \cdot \log n} = \frac{2^\ell}{64}$$

for any fixed keys $k, k'$. Taking a union bound over the at most $t/4^{\ell-1}$ options for $k'$ (here we use the induction hypothesis), the probability that $k$ is non-isolated is at most $\frac{2^\ell}{64} \cdot \frac{t}{4^{\ell-1}} \leq \frac{1}{16 \cdot 2^\ell}$. The expected number of non-isolated indices is $\frac{t}{4^{\ell-1}} \cdot \frac{1}{16 \cdot 2^\ell}$, and by Markov's inequality the number of non-isolated indices exceeds $\frac{t}{2 \cdot 4^\ell}$ with probability at most $2^{-\ell-1}$.

We come back to the algorithm and consider the vector $C_\ell$ as computed in the $\ell$-th level. By the previous two Lemmas 2.11 and 2.12 we have correctly computed the vectors $V = h(A \star B - C_{\ell-1})$ and $W = h(\partial(A \star B - C_{\ell-1}))$ in Lines 7 and 8. Therefore, for each isolated index $k$ we have that $V[z] = (A \star B - C_{\ell-1})[k]$ and $W[z] = k \cdot V[z]$ for $z = h(k)$. It follows that the algorithm eventually identifies $k$ in Line 10 and correctly updates $C_\ell[k] \leftarrow C_{\ell-1}[k] + V[z] = (A \star B)[k]$. This proves that the algorithm correctly recovers all isolated indices, but we also have to account for the non-isolated indices. There is no hope to recover the non-isolated indices, and moreover each non-isolated index might lead to one false recovery in Lines 10 and 12. It follows that $\|A \star B - C_\ell\|_0$ is at most twice the number of non-isolated indices. In combination with the previous paragraph we conclude that $\|A \star B - C_\ell\|_0 \leq \frac{t}{4^\ell}$, and thus the $\ell$-th level is successful.

By a union bound over all levels $\ell = 0, 1, \ldots, L = \lceil \log t \rceil$, we obtain that the $L$-th level is successful with probability at least $1 - \sum_\ell 2^{-\ell-1} \geq \frac{1}{2}$. In this case we have that $\|A \star B - C_L\| \leq \frac{t}{4^L} < 1$ and therefore $A \star B = C_L$ as claimed.

Finally, it remains to analyze the running time. We prove that each iteration of the outer loop (with value $t$) runs in time $O(t \log^2 n)$. Then, by taking into account that each iteration after the critical threshold $t \geq \|A \star B\|_0$ terminates with probability at least $\frac{1}{2}$, the expected running time is also bounded by $O(t \log^2 n)$.

We analyze the running time per level $\ell$. Sampling the prime $p$ takes time $\operatorname{polylog}(t \log n)$ (which is negligible). It takes time $O(t)$ to prepare the derivatives and the hashed vectors of $A$, $B$ and $C$. The dominant step is to compute the three convolutions in FFT time $O(m \log m) = O(\frac{t \log n}{2^\ell} \log(t \log n)) = O(\frac{t}{2^\ell} \log^2 n)$.

Finally, the recovery loop in Line 9 runs in time $O(m)$. Summing over all levels $\ell = 0, 1, \ldots, \lceil \log t \rceil$, the running time per outer iteration becomes

$$\sum_{\ell=0}^{\lceil \log t \rceil} O(t + \frac{t}{2^\ell} \log^2 n) = O(t \log t + t \log^2 n) = O(t \log^2 n)$$

as claimed. ◀

**Advantages and Disadvantages.** The hashing approach powers most recent algorithms [164, 108, 61, 60] and has the clear advantages of being simple and flexible. For instance, while the algorithm we just described is Monte Carlo randomized (due to the Monte Carlo verifier) and it is possible to work in more tricks to get a Las Vegas algorithms for *nonnegative* convolution [60] (see Chapter 3).

In terms of the running time, this algorithm runs in time $O(t \log^2 n)$ which is one $\log n$ factor away from the optimal time bound $O(t \log t)$. One might guess that a log-factor can be saved by using more efficient hash functions in terms of universality. For instance, we could use linear hashing (Lemma 2.7). Unfortunately, since linear hashing is only almost-additive, the iterative recovery loop does no longer work,[30] and the resulting algorithm still runs in time $O(t \log^2 n)$.

The most severe disadvantage is that this approach is inherently randomized. Derandomizations are known but only at the cost of super-polylogarithmic $t^{o(1)}$-factors.

[30] *The problem is that each entry $A \star B$ is split into a constant number of buckets under the linear hashing. As a consequence, we can no longer cancel out the contribution of the residual vector and the sparsity of $A \star B - C$ stays large throughout all levels.*

## 2.3 Sparse Convolution via Algebraic Methods

A radically different approach to computing sparse convolutions by viewing the vectors $A$ as polynomials $A(X) = \sum_i A[i] \cdot X^i$ and by exploiting their algebraic structure. Recall that in this view, the role of convolution is taken by polynomial multiplication.

One prime example is by a classical algorithm due to Gaspard de Prony from 1796, called *Prony's method* [177]. This algorithm was rediscovered several times since then, for decoding BCH codes [205] and in the context of polynomial interpolation [44]; see also [180]. The statement is that we can recover a $t$-sparse polynomial $P(X)$ from $2t$ evaluations $P(x_0), \ldots, P(x_{2t-1})$ at strategically chosen points $x_0, \ldots, x_{2t-1}$.

For our purposes, we can apply Prony's method in three steps to compute the convolution $C = A \star B$ of two sparse vectors $A, B$:

1. *Evaluate* $A(x_0), \ldots, A(x_{2t-1})$ and $B(x_0), \ldots, B(x_{2t-1})$.
2. *Multiply* $C(x_0) \leftarrow A(x_0) \cdot B(x_0), \ldots, C(x_{2t-1}) \leftarrow A(x_{2t-1}) \cdot B(x_{2t-1})$
3. *Interpolate* the $t$-sparse polynomial $C$ from its evaluations $C(x_0), \ldots, C(x_{2t-1})$ using Prony's method.

This is a promising approach, but steps 3 and 1 (note that even the multipoint evaluation of a sparse polynomial is not obvious) require heavy algebraic machinery. In the remainder of this section we provide some more details on Prony's method, and even prove some of its algebraic ingredients.

### 2.3.1 Prony's Method in Detail

In this section we attempt to give a detailed overview of Prony's method. Our aim is to be more intuitive than formally precise. Recall that our goal is to interpolate a $t$-sparse polynomial $P(X)$ (with degree $n \gg t$) by evaluating the polynomial at $2t$ strategically chosen evaluation points. For a more involved discussion see [132, 44] and specifically the survey [180].

**Which Evaluation Points?** We choose as evaluation points the geometric sequence $\omega^0, \omega^1, \ldots, \omega^{2t-1}$. This choice turns out to be convenient for several reasons which will become clear soon. Assume for now that $\omega > 1$ is an integer and that all computations are over the integers. (Later we will work over a finite field and pick $\omega$ more carefully.)

Throughout, we write $P = \sum_{j=0}^{t-1} P[x_j] \cdot X^{x_j}$ and refer to $x_0, \ldots, x_{t-1}$ as the *support* of $P$. Initially, neither the support nor the coefficients of $P$ are known; we only have access to the evaluations $P(\omega^0), \ldots, P(\omega^{2t-1})$. The two major phases of Prony's method are to first recover the support, and then the coefficients. We start with the second phase which is conceptually easier.

**Phase 2: Recovering the Coefficients.** Observe that we can express the evaluation of $P$ by means of the following matrix-vector product:

$$\begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ \vdots \\ P(\omega^{t-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{x_0} & \omega^{x_1} & \cdots & \omega^{x_{t-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(t-1)x_0} & \omega^{(t-1)x_1} & \cdots & \omega^{(t-1)x_{t-1}} \end{bmatrix}}_{W} \begin{bmatrix} P[x_0] \\ P[x_1] \\ \vdots \\ P[x_{t-1}] \end{bmatrix}. \tag{1}$$

Here, the matrix $W$ is a *transposed Vandermonde matrix.* This matrix has full rank (as can easily be checked via its determinant $\det(W) = \prod_{i<j}(\omega^{x_i} - \omega^{x_j})$) and therefore any polynomial with support $x_0, \ldots, x_{t-1}$ is *uniquely* determined by its evaluations $P(\omega^0), \ldots, P(\omega^{t-1})$. In particular, knowing the support $x_0, \ldots, x_{t-1}$ (which we can assume was recovered in step 1) we can interpolate $P$ by solving this equation system for the coefficients $P[x_0], \ldots, P[x_{t-1}]$.

Note that for the recovery of the coefficients, it was sufficient to have access to the $t$ evaluations $P(\omega^0), \ldots, P(\omega^{t-1})$. We need the full sequence of $2t$ evaluations only for recovering the support.

**Phase 1: Recovering the Support.** The more involved task is to recover the support set $x_0, \ldots, x_{t-1}$ from the sequence of evaluations $P(\omega^0), \ldots, P(\omega^{2t-1})$. We start with some background on linear recurrences:

**Definition 2.14 (Linear Recurrence).** *A sequence $S_0, S_1, \ldots$ is* linearly recurrent with degree $r$ *if there is a degree-$r$ polynomial $\Lambda(X) = \sum_{\ell=0}^{r} \lambda_\ell X^\ell$ such that each term in the sequence is determined by a linear combination of its $r$ preceding terms, weighted with $\lambda_0, \ldots, \lambda_r$:*

$$\sum_{\ell=0}^{r} \lambda_\ell S_{i+\ell} = 0 \quad \textit{for all } i \geq 0.$$

*If $\Lambda$ is the monic polynomial (i.e., with leading coefficient 1) with smallest-possible degree satisfying this condition then we call $\Lambda$ the* minimal polynomial *of the recurrence.*

(At first, it seems odd why we view the $\lambda_i$'s as a polynomial, but this viewpoint hopefully soon starts to make sense.)

It turns out that the sequence of evaluations $P(\omega^0), \ldots, P(\omega^{2t-1})$ is linearly recurrent with degree exactly $t$. And, more importantly, we can read of the support from the minimal polynomial of this recurrence:

**Lemma 2.15.** *For any $t$-sparse polynomial $P$, the sequence $P(\omega^0), \ldots, P(\omega^{2t-1})$ is linearly recurrent with degree $t$. Moreover, the minimal polynomial of this recurrence is*

$$\Lambda(X) = \prod_{x \in \mathrm{supp}(P)} (X - \omega^x).$$

**Proof.** ▶ Let $x_0, \ldots, x_{t-1}$ denote the support of $P$ and write $P(X) = \sum_{j=0}^{t-1} P[x_j] \cdot X^{x_j}$. Assume that the sequence $P(\omega^0), \ldots, P(\omega^{2t-1})$ is linearly recurrent with degree $r$ for some polynomial $\Lambda(X) = \sum_{\ell=0}^{r} \lambda_\ell X^\ell$. We can therefore express the recurrence condition as follows, for any $i \geq 0$:

$$\begin{aligned} 0 &= \sum_{\ell=0}^{r} \lambda_\ell P(\omega^{i+\ell}) \\ &= \sum_{\ell=0}^{r} \lambda_\ell \sum_{j=0}^{t-1} P[x_j] \cdot \omega^{(i+\ell)x_j} \\ &= \sum_{j=0}^{t-1} \omega^{ix_j} \cdot P[x_j] \cdot \sum_{\ell=0}^{r} \lambda_\ell \omega^{\ell x_j} \\ &= \sum_{j=0}^{t-1} \omega^{ix_j} \cdot P[x_j] \cdot \Lambda(\omega^{x_j}). \end{aligned}$$

We see two consequences: First, picking the polynomial $\Lambda(X) = \prod_{j=0}^{t-1}(X - \omega^{x_j})$ with roots at all powers $\omega^{x_0}, \ldots, \omega^{x_{t-1}}$ satisfies this recurrence condition for all $i$. In particular, the recurrence has degree at most $t$.

Second, suppose that the minimal polynomial $\Lambda$ has degree less than $t$. Then for at least one power $\omega^{x_0}, \ldots, \omega^{x_{t-1}}$, $\Lambda$ does not have a root. Consider the polynomial $Q$ defined by $Q(X) = \sum_{j=0}^{t-1} P[x_j] \cdot \Lambda(\omega^{x_j}) \cdot X^{x_j}$; note that $Q$ is not identically zero. We can rewrite the recurrence from before as $Q(\omega^i) = 0$ for all $i \in [t]$. This yields a contradiction: On the one hand $Q$ is not identically zero, but on the other hand, $Q$ is $t$-sparse and therefore uniquely determined by its evaluations $Q(\omega^0), \ldots, Q(\omega^{t-1})$ all of which are zero. ◀

The previous lemma suggests the following algorithm: Compute the minimal polynomial $\Lambda$ of the sequence $P(\omega^0), \ldots, P(\omega^{2t-1})$. Knowing that the roots of $\Lambda$ are exactly the powers $\omega^{x_0}, \ldots, \omega^{x_{t-1}}$, we can extract the support set by computing the roots of $\Lambda$ and taking their logarithms (with base $\omega$).

**Finite-Field Arithmetic.** Before stating the complete algorithm, we point out a serious issue in this approach: The integers involved in the computations are guaranteed to become huge—to represent a single power $\omega^{x_j}$ we need at least $n$ bits.

We will therefore perform all computations modulo some prime $p$. This leads to another complication: So far we have implicitly assumed that the powers $\omega^{x_0}, \ldots, \omega^{x_{t-1}}$ are distinct (and this is in fact the only property about $\omega$ that we need). To preserve this property, we will pick $\omega \in \mathbf{F}_p$ in such a way that its multiplicative order is greater than the degree of $P$.

**The Complete Algorithm.** We finally summarize Prony's method as an algorithm with six steps. (Phase 1 consists of steps 3–5 and phase 2 consists of step 6.)

1. *Find an element $\omega$ with multiplicative order at least $n$:*
   In most cases, it suffices to pick a random element, but there are also deterministic methods for this task [80, 79].
2. *Evaluate $P$:* Compute $P(\omega^0), \ldots, P(\omega^{2t-1})$.
3. *Compute the minimal polynomial $\Lambda$ of the linear recurrence $P(\omega^0), \ldots, P(\omega^{2t-1})$:*
   This task can be classically solved in quadratic time using the Berlekamp-Massey algorithm [45, 157], but it is also known how to solve this task in near-linear time using a Toeplitz solver [132] (which in this case is essentially an application of the Extended Euclidian Algorithm).
4. *Compute the roots $\omega^{x_0}, \ldots, \omega^{x_{t-1}}$ of $\Lambda$:*
   (This is indeed the set of roots by Lemma 2.15.) It is known how to implement

this step in *randomized* near-linear time [131], but in terms of deterministic methods nothing comparable is known.

**5** *Compute the discrete logarithms $x_0, \dots, x_{t-1}$ of the roots to the base $\omega$:*
Here we use again that $\omega$ has multiplicative order at least $n$ (as otherwise we could not distinguish two exponents $x, x'$ that are equal modulo $n$). The complexity of this step depends on the multiplicative structure of $\mathbf{F}_p$, but is typically expensive.

However, if $P$ is an integer polynomial, we can use the following trick [180]. We work over $\mathbf{F}_p$ for some carefully chosen prime $p$ where $p-1$ is divisible by a large power of two. Then we can pick an element $\omega$ whose order is a large power of two. In this case it is possible to compute discrete logarithms in time $\operatorname{polylog}(n)$.

**6** *Compute the coefficients $P[x_0], \dots, P[x_{t-1}]$ by solving a transposed Vandermonde system in Equation* (1)*:*
The magic is that these structured linear systems can be solved in near-linear time $O(t \log^2 t)$; see the upcoming Theorem 2.18.

**Advantages and Disadvantages.** Unfortunately, it seems difficult to use this algorithm to answer either of our driving questions: For a deterministic algorithm, the main obstacle is step 4 which is currently not known to run in deterministic near-linear time. For optimal algorithms (in terms of the running time), many steps are prohibitively expensive such as computing discrete logarithms computation in step 5, but also solving the transposed Vandermonde system in step 6. There are other techniques for sparse convolution using polynomial interpolation, see [180], but they do not seem sufficient in going beyond a $O(t \operatorname{polylog} n)$-time algorithm in any variation of the problem, owing to the usage of a variety of tools from structured linear algebra which come with additional log-factors.

Our answer to both driving questions nevertheless relies on ideas from Prony's method, specifically from the final step 6. In both Chapters 3 and 4 we will compute the support $\operatorname{supp}(P) = \{x_0, \dots, x_{t-1}\}$ by other means, and then use the efficient transposed Vandermonde solver to find the coefficients $P[x_0], \dots, P[x_{t-1}]$. For this reason we include a formal proof of the transposed Vandermonde solver in the next Section 2.3.2.

**Prony's Method and Derivatives.** We remark that the recipe behind Prony's method as outlined before can be generalized: For instance, we can recover a $t$-sparse given the $2t$ evaluations of $P$'s derivatives, e.g. given $\frac{d^0}{dx^0}P(1), \dots, \frac{d^{2t-1}}{dx^{2t-1}}(1)$; see [122] and see [194] for a generalization to even more general operators. This derivative version saves some algebraic machinery (in particular, we can skip step 2 and the computationally expensive step 5).

### 2.3.2 Algebraic Tools

On more than one occasion we need to efficiently perform simple algebraic computations such as computing powers or inverses. The next two lemmas describe how to easily obtain improved algorithms for the exponentiation and division of many numbers.

**Lemma 2.16 (Bulk Exponentiation).** *Let $R$ be a ring. Given an element $x \in R$, and a set of nonnegative exponents $k_1, \dots, k_n \le k$, we can compute $x^{k_1}, \dots, x^{k_n}$ in time $O(n \log_n k)$ using $O(n \log_n k)$ multiplications.*

The naive way to implement exponentiations is via repeated squaring in time $O(n \log k)$. There are methods [208, 173] improving the dependence on $k$, but for our purposes this simple algorithm suffices.

**Proof.** ▶ First, compute the base-$n$ representations of all exponents $e_i = \sum_j k_{i,j} n^j$; then $k_{i,j} \in [n]$ where $j = 0, \ldots, \lceil \log_n k \rceil$. For all $i = 1, \ldots, n$ and $j = 0, \ldots, \lceil \log_n k \rceil$ we precompute the powers $x^{in^j}$ using the rules $x^{n^{j+1}} = (x^{n^j})^n$ and $x^{(i+1)n^j} = x^{in^j} x^{n^j}$. Finally, each output $x^{k_i}$ is the product of $\lceil \log_n k \rceil$ numbers $\prod_j x^{k_{i,j} n^j}$. The correctness is immediate, and it is easy to check that all steps takes time $O(n \log_n k)$. ◀

**Lemma 2.17 (Bulk Division).** *Let $F$ be a field. Given $n$ field elements $a_1, \ldots, a_n \in F$, we can compute their inverses $a_1^{-1}, \ldots, a_n^{-1} \in F$ in time $O(n)$ using $O(n)$ multiplications and a single inversion.*

**Proof.** ▶ First, we compute the $n$ prefix products $b_j = a_1 \cdots a_i$. It takes a single inversion to compute $b_n^{-1}$. Then, for $i = n, n-1, \ldots, 2$, we compute $a_i^{-1} = b_i^{-1} b_{i-1}$ and $b_{i-1}^{-1} = b_i^{-1} a_i$. Finally, $a_1^{-1} = b_1^{-1}$. As claimed, this algorithm takes time $O(n)$ and it uses $O(n)$ multiplications and a single inversion. ◀

Finally, a crucial ingredient to our core algorithm is the following theorem about solving transposed Vandermonde systems.

**Theorem 2.18 (Transposed Vandermonde Systems).** *Let $F$ be a field. For distinct field elements $a_1, \ldots, a_n \in F$, let*

$$V = V(a) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ a_1^2 & a_2^2 & \cdots & a_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{bmatrix}.$$

*Given $x \in F^n$ we can compute $Vx$ in time $O(n \log^2 n)$ using at most $O(n \log^2 n)$ ring operations, and we can compute $V^{-1}x$ in time $O(n \log^2 n)$ using at most $O(n \log^2 n)$ ring operations and one division.*

Note that the distinctness condition ensures that the matrix $V$ has full rank (as the Vandermonde determinant $\det(V) = \prod_{i<j}(a_j - a_i)$ is nonzero). This algorithm has been discovered several times [132, 152, 168]. Although none of these sources pays attention to the number of divisions, one can check that applying the bulk division strategy from Lemma 2.17 suffices to obtain the claimed bound. For the sake of completeness, we include a full proof.

Throughout, let $W = W(a) = V(a)^T$ denote the transpose of the matrix $V$, i.e., $W$ is a Vandermonde matrix. The proof of Theorem 2.18 is by the so-called *transposition principle*: First, classic algorithms show that $Wx$ and $W^{-1}x$ can be computed by efficient arithmetic circuits (Lemma 2.19). Second, whenever $Ax$ can be computed efficiently by an *arithmetic circuit*, then also $A^T x$ can be computed similarly efficiently (Lemma 2.20). See [102] for a definition of arithmetic circuits. For our purposes it suffices to only consider addition, subtraction and multiplication gates.

**Lemma 2.19 (Polynomial Evaluation and Interpolation).** *There are algorithms $\mathcal{A}$ and $\mathcal{A}'$ which, given $a \in F^n$ with pairwise distinct entries $a_i$, respectively compute arithmetic circuits $C$ and $C'$ such that:*

- *$\mathcal{A}$ and $\mathcal{A}'$ run in time $O(n \log^2 n)$,*
- *$\mathcal{A}$ and $\mathcal{A}'$ use at most $O(n \log^2 n)$ ring operations and $\mathcal{A}'$ additionally uses at most 1 division,*
- *On input $x \in F^n$, the circuits compute $C'(x) = W(a)x$ and $C'(x) = W(a)^{-1}x$.*

**Proof sketch.** ▶ For $C$, note that evaluating $C(x) = W(a)x$ is exactly the problem of evaluating the polynomial $\sum_i x_i X^i \in F[X]$ at the points $a_1, \ldots, a_n$. Hence, we can use the classical $O(n \log^2 n)$-time algorithm for *polynomial multi-point evaluation*, see e.g. [102, Algorithm 10.7]. This algorithm can be interpreted to compute the arithmetic circuit $C$ rather than computing the multi-point evaluation directly. As this algorithm works over rings, it does not use any divisions.

For $C'$, observe that computing $C'(x) = W(a)^{-1}x$ corresponds to *polynomial interpolation*, which again has a classical $O(n \log^2 n)$-time algorithm [102, Algorithm 10.11]. This time however, we have to pay attention to the number of divisions performed in the process. Note that [102, Algorithm 10.11] only computes divisions in the second step, all of which can be bulked together by Lemma 2.17, and moreover all inputs to these divisions only depend on $a$ and can thus be performed by the algorithm $\mathcal{A}'$, rather than by the arithmetic circuit $C'$. This proves the claim. ◂

Next, we need the following lemma, see [168, Theorem 3.4.1]. It can be proven in several ways, for instance via the Baur-Strassen Theorem [42, 162].

**Lemma 2.20 (Transposition Principle).** *Let $C$ be an arithmetic circuit of size $s$ computing some linear function $x \mapsto Ax$ for some $A \in F^{n \times n}$. Then there is an arithmetic circuit $C^T$ of size $O(s+n)$ computing the function $x \mapsto A^T x$. Moreover, one can compute $C^T$ from $C$ in time $O(s+n)$.*

**Proof of Theorem 2.18.** ▸ For the computation of $Vx$, we first run the algorithm $\mathcal{A}$ from Lemma 2.19 to compute an arithmetic circuits $C$ of size $s = O(n \log^2 n)$. Recall that $\mathcal{A}$ uses $O(n \log^2 n)$ ring operations. The circuit computes $C(x) = W(a)x$. Second, use the transposition principle (Lemma 2.20) to compute the circuits $C^T$ computing $C^T(x) = W(a)^T x = V(a)x$. By Lemma 2.20, $C^T$ has size $O(s+n) = O(n \log^2 n)$, and it can be computed in the same running time. Finally, we evaluate $C^T$ at $x$, which again takes time $O(n \log^2 n)$ and uses only ring operations.

The computation of $V^{-1}x$ follows exactly the same lines, using the algorithm $\mathcal{A}'$ in place of $A$. Recall that $\mathcal{A}'$ additionally computes one division. ◂

## 2.4 Sparse Convolution via Sparse Fourier Transform

For completeness, we mention a third approach via *sparse Fourier transforms* that also results in a $O(t \log^2(n\Delta))$-time algorithm. This idea is not explicitly written down as far as we know. It has been established in the celebrated work of Hassanieh, Indyk, Katabi and Price [118] that one can recover a $t$-sparse vector $A \in \mathbf{C}^n$ in time $O(t \log(n\Delta))$ by only accessing a small subset of its Fourier transform $\widehat{A}$. This alone might not seem sufficient, but spelling out the details of [118] reveals that the pattern of accesses to $\widehat{A}$ is a random arithmetic progression of length $O(t \log(n\Delta))$. In light of this, one can additionally leverage known techniques from so-called *semi-equispaced Fourier transforms* [92], [126, Section 12] to obtain a $O(t \log^2(n\Delta))$-time algorithm. The semi-equispaced Fourier transform is a well-studied subfield of computational Fourier transforms, and results from that area show that $s$ equally spaced Fourier coefficients of a length-$n$ and $s$-sparse vector can be computed in time $O(s \log(n\Delta))$ [126, Section 12]. Combining this with the algorithm of [118] yields an $O(t \log^2(n\Delta))$-time algorithm for sparse integer convolution. The inherent reason for this logarithmic blow-up is that going back and forth in Fourier and time domain is more costly in the sparse case than in the dense case.

## 2.5 Verifier using Polynomial Identity Testing

In this section we prove the following useful tool by again exploiting the algebraic nature of taking convolutions.

**Lemma 2.21 (Sparse Verification).** *Given three integer vectors $A, B, C \in \mathbf{Z}^n$ of sparsity at most $t$, there is a randomized algorithm testing whether $A \star B = C$. It runs in time $O(t \log_t n + \mathrm{polylog}(\|A\|_\infty \|B\|_\infty))$ and fails with probability at most $1/\mathrm{poly}(t)$.*

The typical application of this verifier is to boost the success probability of sparse convolution algorithms: If there is a sparse convolution algorithm with

constant success probability, say, then we can construct an algorithm succeeding with high probability by rerunning the algorithm whenever this verifier fails.

The idea behind the lemma is standard: We view $A$, $B$ and $C$ as polynomials via $A(X) = \sum_i A[i] \cdot X^i$ and similarly for $B$ and $C$. Recall that taking convolutions is the same as multiplying polynomials in that viewpoint, i.e., it suffices to check whether $AB = C$. This is a polynomial identity testing problem which can be classically solved by the Schwartz-Zippel lemma.

**Proof.** ▸ First check whether $\|C\|_\infty > t\|A\|_\infty\|B\|_\infty$ and reject in this case. Otherwise compute a prime $p > \text{poly}(t) \cdot n + t\|A\|_\infty\|B\|_\infty$. We view $A$, $B$ and $C$ as polynomials over $\mathbf{Z}_p$, by interpreting $A(X) = \sum_i A[i] \cdot X^i$ and similarly for $B$ and $C$. Next, sample a random point $x \in \mathbf{Z}_p$. We use the bulk exponentiation algorithm (Lemma 2.16) to precompute all relevant powers $x^i$ and then evaluate $A(x)$, $B(x)$ and $C(x)$ at $x$. If $A(x)B(x) = C(x)$, then we accept (confirming that $A \star B = C$), otherwise we reject.

If $AB = C$, then this algorithm is always correct. So suppose that $AB \neq C$, and let $D = AB - C$. Over the integers it is clear that $D$ is not the zero polynomial, and since $p$ is large enough it also holds that $D$ is nonzero over $\mathbf{Z}_p$. The algorithm essentially evaluates $D$ at a random point $x \in \mathbf{Z}_p$ and accepts if and only if $D(x) = 0$. The error event is that $D(x) = 0$ despite $D$ being nonzero as a polynomial. Recall that $D$ has degree at most $n$, so it has at most $n$ zeros. Therefore, the probability of hitting a zero is at most $n/p < 1/\text{poly}(t)$.

Finally, we analyze the running time. Precomputing the prime $p$ takes time $\text{polylog}(t\|A\|_\infty\|B\|_\infty)$. Using the bulk exponentiation trick (Lemma 2.16), computing the powers of $x$ takes time $O(t \log_t n)$. Finally, evaluating $A$, $B$ and $C$ at $x$ takes time $O(t)$. Note that all arithmetic operations carried out in these steps are over $\mathbf{Z}_p$ and since a single element of $\mathbf{Z}_p$ can be written down using a constant number of machine words, each ring operation takes constant time. The total time is $O(t + \text{polylog}(\|A\|_\infty\|B\|_\infty))$. ◂

## 2.6 The Scaling Trick

Another handy ingredient to many of our algorithms is the so-called *scaling trick*; see the followimg Lemma 2.22. The scaling trick was first applied to the context of convolutions in [65]; see also [66, 58]. It states that, whenever we want to compute the sumset $X + Y$ of two given sets $X + Y$, we can assume that we have access to a small superset $\widetilde{Z} \supseteq X + Y$ at the cost of worsening the running time by a factor $O(\log n)$. Formally:

**Lemma 2.22 (Scaling Trick).** *Assume that there exists an algorithm that, given sets $X, Y, \widetilde{Z} \subseteq [n]$ such that $X + Y \subseteq \widetilde{Z}$, computes $X + Y$ in time $T(|\widetilde{Z}|)$. Then there is an algorithm that, given sets $X, Y \subseteq [n]$, computes $X + Y$ in time $O(T(|X + Y|) \log n)$.*

**Proof.** ▸ Let $\mathcal{A}$ be the algorithm to compute the sumset $X + Y$ given a small superset $\widetilde{Z} \supseteq X + Y$. We design a simple recursive algorithm that computes the sumset $X + Y$ without any further assumptions:

1. Let $X' = \{\lfloor \frac{x}{2} \rfloor : x \in X\}, Y' = \{\lfloor \frac{y}{2} \rfloor : y \in Y\}$ and recursively compute $Z' = X' + Y'$.
2. Let $\widetilde{Z} = \{2z, 2z + 1, 2z + 2 : z \in Z'\}$. We will prove that this set $\widetilde{Z}$ satisfies the desired condition $X + Y \subseteq \widetilde{Z}$.
3. Compute $X + Y$ by calling $\mathcal{A}(X, Y, \widetilde{Z})$.

For the correctness of this algorithm it suffices to argue that $X + Y \supseteq \widetilde{Z}$. Take any pair of elements $x \in X$, $y \in Y$. We clearly have that $z' = \lfloor \frac{x}{2} \rfloor + \lfloor \frac{y}{2} \rfloor \in Z'$ and using

this we prove that $x + y \in \widetilde{Z}$. By the elementary bounds $a - 1 \leq \lfloor a \rfloor \leq a$, for any rational $a \in \mathbf{Q}$, we have that

$$x + y = \frac{2x}{2} + \frac{2y}{2} \leq 2\left\lfloor \frac{x}{2} \right\rfloor + 1 + 2\left\lfloor \frac{y}{2} \right\rfloor + 1 = 2z' + 2,$$
$$x + y = \frac{2x}{2} + \frac{2y}{2} \geq 2\left\lfloor \frac{x}{2} \right\rfloor + 2\left\lfloor \frac{y}{2} \right\rfloor = 2z'.$$

In summary, $x+y$ is an integer in the range $[2z' \,..\, 2z'+2]$. By construction, $x+y \in \widetilde{Z}$. From the same argument it follows that $\widetilde{Z} \leq 3|X+Y|$ as any element in $X+Y$ causes at most three elements in $\widetilde{Z}$.

It remains to bound the running time. As $|\widetilde{Z}| \leq 3|X+Y|$, calling the algorithm $\mathcal{A}$ takes time $T(3|X+Y|) = O(T(|X+Y|))$. The recursive call continues to compute a sumset $X' + Y'$ of size $|X' + Y'| \leq |X + Y|$, and since the recursion tree reaches depth $\log n$, the total time is $O(T(|X+Y|)\log n)$. ◀

We remark that the scaling trick only works for sumsets (or more generally, nonnegative convolutions), but it cannot be applied to approximate the support of integer convolutions. Recall that for integer convolutions, there can be cancellations leading to unexpectedly small support sets. For instance, consider the length-$n$ vectors $A = (1, \ldots, 1)$ and $B = (1, -1, 0, \ldots, 0)$. Their convolution is the vector $A \star B = (1, 0, \ldots, 0, -1, 0, \ldots, 0)$ with sparsity 2. Attempting to apply the scaling trick to recursively approximate the support $\mathrm{supp}(A \star B)$ leads to failure, as in this case the vector $B'$ would collapse $B$ to the zero-vector. It remains an interesting question to find an appropriate substitute of the scaling trick for integer convolutions, which would possibly lead to new (in particular, deterministic) algorithms for computing integer convolutions.

**Scaling Trick over $\mathbf{F}_p^d$.** The scaling trick can however be applied to approximate the support of nonnegative convolutions over groups other than the integers. We specifically need the following variant for finite-field vector spaces:

**Lemma 2.23 (Scaling Trick over $\mathbf{F}_p^d$).** *Assume that there exists an algorithm that, given sets $X, Y, \widetilde{Z} \subseteq \mathbf{F}_p^d$ such that $X+Y \subseteq \widetilde{Z}$, computes $X+Y$ in time $T(|\widetilde{Z}|)$. Then there is an algorithm that, given sets $X, Y \subseteq [n]$, computes $X+Y$ in time $O(T(p|X+Y|)d)$.*

**Proof.** ▶ This proof is very similar to the proof of Lemma 2.22. We let $\mathcal{A}$ be the algorithm computing the sumset $X+Y$ provided a small superset $\widetilde{Z} \supseteq X+Y$ is given, and design a recursive algorithm computing the sumset $X+Y$ unconditionally.

**1** Let

$$X' = \{(x_1, \ldots, x_{d-1}) : (x_1, \ldots, x_d) \in X\},$$
$$Y' = \{(y_1, \ldots, y_{d-1}) : (y_1, \ldots, y_d) \in Y\}$$

and recursively compute $Z' = X' + Y'$. Notice that $Z'$ can be obtained from $X+Y$ by chopping of the last coordinates from all elements.

**2** Let

$$\widetilde{Z} = \{(z_1, \ldots, z_d) : (z_1, \ldots, z_{d-1}) \in Z', z_d \in \mathbf{F}_p\}.$$

It is fairly obvious that this set $\widetilde{Z}$ satisfies the desired condition $X + Y \subseteq \widetilde{Z}$.

**3** Compute $X+Y$ by calling $\mathcal{A}(X, Y, \widetilde{Z})$.

As the correctness should be clear from the previous explanations, it merely remains to bound the running time. Note that $|\widetilde{Z}| \leq p|X+Y|$ as $\widetilde{Z}$ is obtained from $X+Y$ by replacing in each element $z \in X+Y$ the last coordinate by an arbitrary field element from $\mathbf{F}_p$. It follows that calling $\mathcal{A}$ takes time $T(p|X+Y|)$.

Moreover, as before, the recursion tree reaches depth $d$ and the size of the recursively computed sumsets $X' + Y'$ does not increase, i.e., $|X' + Y'| \le |X + Y|$. In total, the algorithm therefore runs in time $O(T(p|X + Y|)d)$ as claimed. ◀

Exploiting this scaling trick, it is easy to obtain an efficient algorithm to compute nonnegative convolutions over $\mathbf{F}_p^d$ (whenever both $p$ and $d$ are small). We use the hashing approach outlined in Section 2.2.

**Theorem 2.24 (Sparse Nonnegative Convolution over $\mathbf{F}_p^d$).** *Let $G = \mathbf{F}_p^d$. Given nonnegative vectors $A, B \in \mathbf{N}^G$, we can compute $A \star B$ in time $\widetilde{O}(t \cdot \mathrm{poly}(pd))$ by a Monte Carlo randomized algorithm, where $t = \|A \star B\|_0$.*

**Proof.** ▶ Let $X = \mathrm{supp}(A), Y = \mathrm{supp}(B)$. By the previous Lemma 2.23 we can assume that we have access to a superset $\widetilde{Z} \supseteq X + Y = \mathrm{supp}(A \star B)$, and it remains to design an algorithm running in near-linear time with respect to $t = |\widetilde{Z}|$.

We apply the hashing approach: Let $d' = \lceil \log_p(100t) \rceil$ and let $G' = \mathbf{F}_p^{d'}$; i.e., the subgroup $G'$ has size $100t \le |G'| \le 100p \cdot t$. Let $h : G \to G'$ be a random linear map (see Lemma 2.10). We claim that for any element $z \in \widetilde{Z}$, the probability that $z$ is *isolated* under the hashing (that is, that there is no other $z' \in Z$ with $h(z) = h(z')$) is at least $\frac{9}{10}$. Indeed, the collision probability is $\mathbf{P}(h(z) = h(z')) \le |G|^{-1} \le \frac{1}{100t}$, therefore is suffices to take a union bound over all possible $t = |\widetilde{Z}|$ elements.

Our goal is to compute $A \star B$ on the subset of entries which are isolated under $h$. To this end, we compute $h(A)$ and $h(B)$, compute their convolution using Theorem 2.3, and recover $(A \star B)[z] = (h(A) \star h(B))[h(z)]$ for all isolated elements $z$. By the isolation property, it is easy to check that this is correct.

As we have argued before, each element is isolated with probability at least $\frac{9}{10}$. Hence, by repeating the process for $O(\log t)$ iterations, each element in $Z$ was isolated at least once with high probability, and at this point we have recovered the complete vector $A \star B$.

The total running time (ignoring the scaling trick) can be bounded as follows: Constructing $h(A)$ and $h(B)$ takes time $O(\|A\|_0 + \|B\|_0) = O(t)$. Computing their convolution $h(A) \star h(B)$ takes time $O(|G'| \log |G'|)$ using Theorem 2.3, and by our choice of $G'$ this becomes $\widetilde{O}(t \cdot p)$. In the same time budget we can also test for each element in $\widetilde{Z}$ whether it is isolated under the hashing. In total the running time is $\widetilde{O}(t \cdot p)$ and the repetitions only add a logarithmic overhead of $O(\log t)$. The scaling trick incurs a factor $pd$, thereby worsening the running time to $\widetilde{O}(t \cdot p^2 d)$. ◀

## 2.7 Witness Finding

Specifically in the context of Boolean convolution (aka computing sumsets $X + Y$), it is sometimes desirable to find *witnesses*. Here, we say that a witness of $z \in X + Y$ is a pair $(x, y) \in X \times Y$ such that $x + y = z$; we denote the set of witness of $z$ by $W_{X,Y}(z)$.

**Theorem 2.25 (Witness Finding).** *Assume that, given $X, Y \subseteq G$ we can compute $X+Y$ in time $T(|X+Y|)$. Then there is a Monte Carlo randomized algorithm that uniformly samples, for each $z \in X + Y$, a* witness *from $W_{X,Y}(z) = \{(x, y) \in X \times Y : x + y = z\}$. The algorithm runs in time $\widetilde{O}(T(|X + Y|))$.*

We remark that in order to list $w$ witnesses from $W_{X,Y}(z)$ for each $z \in X + Y$ (or all witnesses, if $|W_{X,Y}(z)| \le w$), it suffices to run this algorithm $\widetilde{O}(w)$ times. If $z$ has more than $w$ witnesses, then with high probability we have sampled at least $w$ different witnesses in this way. And if $z$ has at most $w$ many witnesses, then with high probability we have sampled all witnesses.

**Proof.** ▶ The algorithm follows a standard recipe. We first apply the isolation technique to isolate a single witness (for many elements $z$). Then, we identify this unique witness bit-by-bit.

**Step 1: Isolating a Single Witness.** As the first step we repeatedly subsample the set $X$. The hope is that in this way, for any fixed $z$, we isolate a *single witness*. Formally, we define a sequence of sets $X_0, X_1, \dots \subseteq X$ by letting $X_0 = X$ and letting $X_i \subseteq X_{i-1}$ be a random subset that including each element with probability $\frac{1}{2}$. With high probability, after at most $L = O(\log |X|)$ steps this process has reached the empty set $X_L = \emptyset$.

Fix any $z \in X+Y$, and consider the sequence $W_0 = W_{X_0,Y}(z), W_1 = W_{X_1,Y}(z), \dots$. Note that this sequence is equivalently obtained by $W_0 = W_{X,Y}(z)$ and by subsampling a random subseteq $W_i \subseteq W_{i-1}$ with rate $\frac{1}{2}$ for all $i > 0$. We show that with constant probability, there is a set $W_i$ in the sequence with size exactly $|W_i| = 1$. Indeed, suppose that $|W_i| \geq 2$. Then the only error event is the number of witnesses drops from at least 2 to 0. In case that the number of witnesses stays at least 2, we simply move on to the next set $W_{i+1}$. This error event happens with probability at most

$$\mathbf{P}(|W_{i+1}| = 0 \mid |W_{i+1}| \leq 1) = \frac{\mathbf{P}(|W_{i+1}| = 0)}{\mathbf{P}(|W_{i+1}| \leq 1)} = \frac{2^{-|W_i|}}{(1+|W_i|) \cdot 2^{-|W_i|}} = \frac{1}{1+|W_i|} \leq \frac{1}{3}.$$

Moreover, note that by symmetry the only surviving witness is uniformly distributed.

**Step 2: Identifying the Unique Witness.** Focus on any set $X_i$ as constructed in the previous step, and assume that for the fixed $z$ we have isolated a single witness, i.e., $|W_i| = 1$. Our goal is to identify that unique witness. Let $X_{i,0}, \dots, X_{i,R-1} \subseteq X_i$ be random subsets of $X_i$ including each element independently with probability $\frac{1}{2}$. Using the fast sumset algorithm, we compute $Z_{i,0} = X_{i,0} + Y, \dots, Z_{i,R-1} = X_{i,R-1} + Y$. For each element $z \in X + Y$, we define its *identifier* as the length-$R$ bit-vector indicating which sets $Z_{i,0}, \dots, Z_{i,R-1}$ contain $z$. We similarly define the identifier of an element $x \in X$ by the length-$R$ bit-vector indicating which sets $X_{i,0}, \dots, X_{0,R-1}$ contain $x$. Note that for each isolated element $z$, the unique witness $x$ of $z$ shares the same identifier. Moreover, by choosing $R = \Omega(\log |X|)$ with high probability the identifiers of all elements in $X$ are pairwise distinct. Therefore, by computing the identifiers of all elements in $X$ and $Z$, and comparing these to each other, we can identify the unique witnesses of all isolated elements $z$.

The running time of this approach is dominated by the $O(\log^2 |X|)$ calls to the sumset algorithm. Each call is of the form $X_{i,j}+Y$ where $X_{i,j} \subseteq Y$, hence the running time can be bounded by $O(T(|X+Y|) \log^2 |X|)$ (assuming that $T(\cdot)$ is nondecreasing). Note that this algorithm succeeds for each $z$ with constant probability, hence we have to repeat the whole procedure $O(\log |X|)$ times. ◂

We remark that finding a single witness can be derandomized at polylogarithmic cost using $\epsilon$-biased sets similar to [15].

# 3 Deterministic and Las Vegas Algorithms for Sparse Nonnegative Convolution

In this chapter we provide technical details on the deterministic and Las Vegas algorithms for sparse nonnegative convolution in Theorems 1.3 to 1.5. The results of this chapter are originally based on the same-named paper [60].

60 Karl Bringmann, Nick Fischer, and Vasileios Nakos. "Deterministic and Las Vegas algorithms for sparse nonnegative convolution". In: *33rd annual ACM-SIAM symposium on discrete algorithms (SODA 2022)*. SIAM, 2022, pages 3069–3090. 10.1137/1.9781611977073.119.

**Organization.** We organize this chapter as follows: In Section 3.1 we give a quick high-level overview of our techniques. Then, in Sections 3.3 and 3.4 we respectively give details. For this chapter we presuppose several tools from the previous Chapter 2.

## 3.1 Overview

We start with a technical overview of the ideas behind Theorems 1.3 to 1.5. The deterministic algorithm uses a radically different approach than the Las Vegas algorithms.

### 3.1.1 Deterministic Algorithm

**Theorem 1.3 (Deterministic Sparse Nonnegative Convolution).** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \operatorname{polylog}(n\Delta))$, where $t = \|A \star B\|_0$ and $\Delta = \|A \star B\|_\infty$.*

The algorithm is in fact easy to describe, given the tool set we established in the last section. The main parts of the algorithm are algebraic and we will view the vectors $A$ and $B$ as univariate polynomials via the natural correspondence $A(X) = \sum_{i=0}^{n-1} A[i] \cdot X^i$. We follow the approach outlined in Section 2.3 to compute the product polynomial $C = AB$:

1. *Evaluate* $A(x_0), \dots, A(x_{2t-1})$ and $B(x_0), \dots, B(x_{2t-1})$ at some carefully chosen points $x_0, \dots, x_{2t-1}$.
2. *Multiply* $C(x_0) \leftarrow A(x_0) \cdot B(x_0), \dots, C(x_{2t-1}) \leftarrow A(x_{2t-1}) \cdot B(x_{2t-1})$
3. *Interpolate* the $t$-sparse polynomial $C$ from its evaluations $C(x_0), \dots, C(x_{2t-1})$ using a variant of Prony's method.

Unfortunately, it is *not* known how to implement Prony's method in deterministic near-linear time, so we have to improvise. In the following we reiterate our explanation of Prony's method from Section 2.3.1 and describe how we implement each step for our convolution algorithm. Recall that in Prony's method we evaluate the polynomial at the points of a geometric series $\omega^0, \dots, \omega^{2t-1}$, for a particularly chosen $\omega$. The remaining algorithm runs in two phases: In the first phase we recover the support of polynomial and in the second phase we recover its coefficients. We again start with the second phase which is conceptually simpler.

**Phase 2: Recovering the Coefficients.** Suppose that we have already solved phase 1 and have successfully recovered the support $\operatorname{supp}(C) = \{z_0, \dots, z_{t-1}\}$. Then, given the evaluations $C(\omega^0), \dots, C(\omega^{t-1})$, how can we recover the coeffi-

cients $C[z_0], \ldots, C[z_{t-1}]$? The trick is to observe that the evaluation of a sparse polynomial can be expressed as a nicely structured matrix-vector product:

$$\begin{bmatrix} C(\omega^0) \\ C(\omega^1) \\ \vdots \\ C(\omega^{t-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{z_0} & \omega^{z_1} & \cdots & \omega^{z_{t-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(t-1)z_0} & \omega^{(t-1)z_1} & \cdots & \omega^{(t-1)z_{t-1}} \end{bmatrix} \begin{bmatrix} C[z_0] \\ C[z_1] \\ \vdots \\ C[z_{t-1}] \end{bmatrix}.$$

This matrix is the transpose of a Vandermonde matrix. It is known that performing linear algebra operations (such as computing matrix-vector products, or solving linear systems) with transposed Vandermonde matrices can be implemented in $O(t \log^2 t)$ field operations (see Theorem 2.18 [132, 152, 168]). Therefore, to interpolate the coefficients $C[z_0], \ldots, C[z_{t-1}]$, we solve this system of linear equations with indeterminates $C[z_0], \ldots, C[z_{t-1}]$. Note that for this step is necessary to know the support $z_0, \ldots, z_{t-1}$ in advance, as otherwise cannot even write down the equation system.

Another critical condition is that the equation system must be nonsingular in order to have a unique solution. This condition is equivalent to $\omega^{z_0}, \ldots, \omega^{z_{t-1}}$ being pairwise distinct. Thus, a reasonable way to achieve the condition is to let $\omega$ be a field element with multiplicative order at least $n \geq \deg(C)$. We soon describe how to obtain such an element deterministically.

**Phase 1: Recovering the Support.** In Prony's method we run involved algebraic machinery such as solving a linear recurrence, polynomial root finding and computing discrete logarithms to get access to the support $z_0, \ldots, z_{t-1}$. Unfortunately, it is not known how to efficiently implement the root-finding step by a deterministic algorithm.

We will therefore compute the support differently, exploiting that we only deal with *nonnegative* convolution. In fact, it suffices to compute a small superset $\widetilde{Z}$ of the support. To this end, we use the *scaling trick* described in Section 2.6. In a nutshell, the scaling trick is to construct smaller vectors $A', B'$ of length $\frac{n}{2}$ defined by $A'[i] = A[2i] + A[2i+1]$ and $B'[j] = B[2j] + B[2j+1]$, to compute their convolution $C' = A' \star B'$ *recursively*, and to obtain the superset as

$$\widetilde{Z} = \{2k, 2k+1, 2k+2 : k \in \operatorname{supp}(C')\}.$$

This choice is correct: Clearly $|\widetilde{Z}| \leq 3t$, and it is easy to verify that $\widetilde{Z}$ is indeed a superset of $\operatorname{supp}(A \star B)$. The recursion only reaches depth $\log n$, and thus incurs a logarithmic overhead in the running time.

**Which Evaluation Points?** Finally, let us revisit how to pick the evaluation points. We will work over some appropriately large finite field $\mathbf{F}_p$ so that we do not suffer from losses in the running due to large integers. (We remark that this algorithm is numerically unstable, so using complex arithmetic is not an option.) As outlined before, it is necessary to pick the evaluation points $\omega^0, \ldots, \omega^{x_{t-1}}$ for some $\omega \in \mathbf{F}_p$, but how to pick $\omega$? The only relevant property is that $\omega$ has multiplicative order at least $n$. (In particular, this requires $p \geq n$.)

There is a simple randomized algorithm: Pick a random element. Unfortunately, the best-known *deterministic* algorithms for finding a large-order element in a given prime field $\mathbf{F}_p$ require time polynomial in $p$ [80]. Thus, it seems intractable to work over a finite field $\mathbf{F}_p$ with $p \geq n$ as originally intended.

Fortunately, in a finite field $\mathbf{F}_q = \mathbf{F}_{p^m}$ with prime power order, it is possible to find large-order elements in time $\operatorname{poly}(p, m)$ [79, 190, 191]. Specifically, setting $p, m = \operatorname{polylog}(n)$ we can find an element $\omega$ with order at least $n$ in time polylogarithmic time. Working over a finite field with small characteristic $p \leq \operatorname{polylog}(n)$ has another drawback though: We cannot recover the entries of the vector $A \star B$ (which can have size up to $n$, even if $A$ and $B$ are bit-vectors

**Algorithm 3.1.** Given two nonnegative vectors $A, B \in \mathbf{N}^n$, this Las Vegas algorithm correctly computes their convolution $A \star B$.

```
1  for m ← 1, 2, 4, ..., ∞ do
2      repeat 2 log m times
3          Sample a linear hash function h : [n] → [m]
4          Compute X ← h(A) ⋆_m h(B)
5          Compute Y ← h(∂A) ⋆_m h(B) + h(A) ⋆_m h(∂B)
6          Compute Z ← h(∂²A) ⋆_m h(B) + 2h(∂A) ⋆_m h(∂B) + h(A) ⋆_m h(∂²B)
7          Initialize R ← (0, ..., 0)
8          for each k ∈ [m] do
9              if X[k] ≠ 0 and Y[k]² = X[k] · Z[k] then
10                 z ← Y[k] / X[k]
11                 R[z] ← R[z] + X[k]
12     Let C be the coordinate-wise maximum of all vectors R
13     if ‖C‖₁ = ‖A‖₁ · ‖B‖₁ then return C
```

to begin with). We remedy this problem by computing the convolution $A \star B$ over *several* finite fields $\mathbf{F}_{q_1}, \mathbf{F}_{q_2}, \ldots$, and use the Chinese Remainder Theorem to identify the correct integer solution afterwards.

This completes the description of the deterministic algorithm. We provide the details in Section 3.3.

### 3.1.2 Simple Las Vegas Algorithm

Next, we state our simple Las Vegas algorithm and outline the idea behind the proof.

**Theorem 1.4 (Simple Las Vegas for Sparse Nonnegative Convolution).** *There is a Las Vegas randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in expected time $O(t \log^2 t)$, where $t = \|A \star B\|_0$. Moreover, with probability $1 - \delta$ the running time is bounded by $O(t \log^2(t/\delta))$.*

The proof of Theorem 1.4 is essentially given by Algorithm 3.1—this is a simple Las Vegas algorithm with expected running time $O(t \log^2 t)$ as claimed in Theorem 1.4; however, to obtain the tail bound on the running time one has to slightly refine Algorithm 3.1. We provide this refinement along with a detailed analysis in Section 3.4; for the rest of the overview we will analyze the simple version in Algorithm 3.1.

To understand the pseudocode, we first recall some notation: For a vector $A$, we denote by $\partial A$ its *derivative* defined coordinate-wise as $(\partial A)[i] = i \cdot A[i]$. More generally, we denote by $\partial^d A$ its *d-th derivative* with $(\partial^d A)[i] = i^d \cdot A[i]$. This definition is in slight dissonance with the analogous definition for polynomials (which would require the derivative vector to be scaled and *shifted*), but we prefer this version as it leads to a slightly simpler algorithm.

We also define hashing for vectors: For a hash function $h : [n] \to [m]$ and a length-$n$ vector $A$, define the length-$m$ vector $h(A)$ via $h(A)[j] = \sum_{i:h(i)=j} A[i]$. The operator $\star_m$ denotes convolution with wrap-around (see Section 2.1 for details).

Let us outline the high-level idea of Algorithm 3.1. The outer loop (Line 1) guesses the correct sparsity, i.e., as soon as the outer loop reaches a value $m \geq \Omega(t)$ we expect the algorithm to terminate. Each iteration of the repeat-loop (Line 2) is supposed to produce a vector $R$ which closely approximates $A \star B$. More specifically, we prove that $R$ satisfies the following two properties:

1. It always holds that $R \leq A \star B$ (coordinate-wise).
2. Equality is achieved at any coordinate with constant probability (provided that the outer loop has reached a sufficiently large value $m \geq \Omega(t)$).

It follows that $C$, the coordinate-wise maximum of several vectors $R$, also always satisfies $C \leq A \star B$. Hence, the algorithm never outputs an incorrect solution. Indeed, since $C$ and $A \star B$ are nonnegative vectors, the vector $C = A \star B$ is the only one simultaneously satisfying $C \leq A \star B$ and $\|C\|_1 = \|A \star B\|_1 = \|A\|_1 \cdot \|B\|_1$. To see that Algorithm 3.1 terminates fast, note that the repeat-loop runs for $\Omega(\log m)$ iterations and thus, using the second claim we correctly assign *all* coordinates with high probability.

The crucial part is to prove that $R$ satisfies the claims 1 and 2. Intuitively, $R$ consists of all nonzero entries from $A \star B$ which did not suffer from a collision with another nonzero entry. For a more formal argument, we analyze the inner-most loop (Line 8). For starters, focus on an iteration $k \in [m]$ and suppose that there is only a single nonzero entry in $A \star B$, say at $z$, which is hashed to the bucket $k$.[31] In this case we have the three identities $X[k] = (A \star B)[z]$, $Y[k] = z \cdot (A \star B)[z]$ and $Z[k] = z^2 \cdot (A \star B)[z]$. As a consequence, the conditions "$X[k] \neq 0$" and "$Y[k]^2 = X[k] \cdot Z[k]$" in Line 9 are satisfied. The algorithm then correctly identifies $z$ in Line 10 and updates "$R[z] \leftarrow R[z] + (A \star B)[z]$" as intended.

[31] *Strictly speaking, that condition is not sufficient because linear hashing is only "almost" additive. We ignore this technical issue in the overview and give the full analysis in Section 3.4.*

However, to prove claim 1 (which is ultimately responsible for the Las Vegas guarantee), we have to be certain that Lines 10 and 11 are only executed if there is a single entry hashed to the $k$-th bucket (otherwise, the index $z$ computed in Line 10 is likely to be nonsense). The key insight is that the simple test "$Y[k]^2 = X[k] \cdot Z[k]$" in Line 9 suffices, as can be proven by the following lemma (see Section 3.4 for a proof).

**Lemma 3.1 (Testing 1-Sparsity).** *Let $V$ be a nonnegative vector. Then $\|\partial V\|_1^2 \leq \|V\|_1 \cdot \|\partial^2 V\|_1$. This inequality is tight if and only if $\|V\|_0 \leq 1$.*

This new tester is one of the reasons why we can achieve the claimed Las Vegas running time simplifying (and slightly improving) upon Cole and Hariharan's algorithm. This concludes the overview of our simple Las Vegas algorithm (Theorem 1.4).

## 3.2 Accelerated Las Vegas Algorithm

We continue with an exposition of our accelerated Las Vegas algorithm:

**Theorem 1.5 (Fast Las Vegas for Sparse Nonnegative Convolution).** *There is a Las Vegas randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in expected time $O(t \log t \log\log t)$, where $t = \|A \star B\|_0$.*

The insight behind this result is that Algorithm 3.1 already reaches a very good approximation after much less than $O(\log m)$ iterations of the inner loop. Indeed, after only $O(\log\log n)$ iterations we expect that algorithm has already recovered $A \star B$ correctly up to a $(\log n)^{-\Omega(1)}$ fraction of the entries. At this point it becomes more efficient to switch to another recovery approach which exploits that $A \star B - C$ is already quite sparse, as in [61]. In particular, since $A \star B - C$ is a *nonnegative* vector and its sparsity is at most $t/\log n$, say, we can use the hash function $h(x) = x \bmod p$ for $p$ being a random prime in $[t, 2t]$. This family of hash functions (1) satisfies that $h(A) \star_m h(B) - h(C) = h(A \star B - C)$ (and thus preserves all cancellations) and (2) isolates a constant fraction of elements in $A \star B - C$ with constant probability to clear up the rest of the elements. Note that it is important that $A \star B - C$ is $t/\log n$ sparse instead of $t$ sparse for (2) to hold, because $h$ is only $O(\log n)$-universal. Choosing $O(\log t)$ different random primes and using the 1-sparsity testing we arrive at our desired algorithm. For the sparsity test we require that the vector $A \star B - C$ is nonnegative.

One catch is that this approach only gives a $O(t \log t \cdot \log\log n)$-time algorithm (instead of the desired time with $\log\log t$ in place of $\log\log n$) due to the fact that $h(x)$ is $O(\log n)$-universal and hence the random prime must be chosen in an interval that is also dependent on $n$ rather than solely on $t$. To address this issue we apply the following precomputation: We hash to a $\mathrm{poly}(t)$-size universe and

verify that this hashing was successful in Las Vegas randomized time, again using our 1-sparsity tester. The details of this step appear in Section 3.4.4.

### 3.2.1 Beyond 1-Sparsity?

An interesting technical open question is whether our sparsity-testing technique can be extended? Specifically, can the technique be adapted to obtain the optimal running time $O(t \log t)$? Recall that the key step in the analysis is the application of Lemma 3.1. This lemma can be generalized as follows: A nonnegative vector $V$ is at most $s$-sparse if and only if the following positive-semidefinite matrix is non-singular:

$$\begin{bmatrix} \|\partial^0 V\|_1 & \|\partial^1 V\|_1 & \cdots & \|\partial^s V\|_1 \\ \|\partial^1 V\|_1 & \|\partial^2 V\|_1 & \cdots & \|\partial^{s+1} V\|_1 \\ \vdots & \vdots & \ddots & \vdots \\ \|\partial^s V\|_1 & \|\partial^{s+1} V\|_1 & \cdots & \|\partial^{2s} V\|_1 \end{bmatrix};$$

see for instance [154, Theorem 3A] for a proof. One approach for an improved Las Vegas algorithm would be to hash to $t/\log t$ buckets using a linear hash function, recover each bucket as in [61] in $O(t \log t)$ time and, using the generalized sparsity-testing technique, verify that most buckets indeed have sparsity $O(\log t)$, which in turn means that all but a $1/\log t$-fraction of $A \star B$ has been successfully recovered; then one can continue and recover the rest with $h(x) = x \bmod p$. Although promising, this approach suffers from precision issues (when implementing the $O(\log t)$-tester the numbers get too large) and hence does not lead to the desired $O(t \log t)$ time. It would be very interesting to find a way to circumvent this obstacle and obtain the ideal $O(t \log t)$ Las Vegas running time.

## 3.3 Deterministic Algorithm

In this section we prove Theorem 1.3. We proceed in three steps, as outlined before. We start with some preliminaries on finite field arithmetic.

**Finite Field Arithmetic.** Let $q = p^m$ be a prime power. Recall that the prime field $\mathbf{F}_p$ can be represented as $\mathbf{Z}/p\mathbf{Z}$, the integers modulo $p$. The field $\mathbf{F}_q$ can be represented as $\mathbf{F}_p[X]/\langle f \rangle$ where $f \in \mathbf{F}_p[X]$ is an arbitrary irreducible degree-$m$ polynomial. There is a deterministic algorithm to precompute such an irreducible polynomial $f \in \mathbf{F}_p$ in time $\mathrm{poly}(p, m)$ [189]; we will point out this step in our algorithms. Having precomputed $f$, we can perform the basic field operations in $\mathbf{F}_q$ using polynomial arithmetic in time $\widetilde{O}(\log q)$ [102].

Let us quickly recall some definitions from field theory. The *multiplicative order* of an element $x$ is the smallest positive integer $i$ such that $x^i = 1$; we also call $x$ an *$i$-th root of unity*. The *minimal polynomial* of a field element $x \in \mathbf{F}$ is defined as the smallest-degree monic polynomial (i.e., with leading coefficient 1) over $\mathbf{F}$ which vanishes at $x$. We say that two field elements $x, y$ are *conjugate* if their minimal polynomials coincide.

### 3.3.1 Sparse Polynomial Evaluation and Interpolation

The key ingredient to our deterministic algorithm is the efficient evaluation and interpolation of sparse polynomials based on fast linear algebra with transposed Vandermonde matrices.

**Lemma 3.2 (Sparse Evaluation and Interpolation).** *Let $\mathbf{F}$ be a field and let $\omega \in \mathbf{F}$ have multiplicative order at least $n$. The following two computational problems can be solved in deterministic time $O(t \log^2 t + t \log n)$:*

1. Evaluation: *Given a $t$-sparse degree-$n$ polynomial $A$, output $A(\omega^0), \dots, A(\omega^{t-1})$.*
2. Interpolation: *Given $a_0, \dots, a_{t-1} \in \mathbf{F}$ and a size-$t$ set $\widetilde{X} \subseteq [n]$, interpolate the unique polynomial $A$ with support $\operatorname{supp}(A) \subseteq \widetilde{X}$ and with evaluations $A(\omega^i) = a_i$ for all $i \in [t]$.*

**Proof.** ▸ We build the two algorithms separately.

1. *Evaluation:* Assume that $A$ has the form $A(X) = \sum_{i=0}^{t-1} A[x_i] \cdot X^{x_i}$. We precompute the powers $\omega^{x_0}, \dots, \omega^{x_{t-1}}$ by repeated squaring in time $O(t \log n)$. We can then compute the evaluations $A(\omega^0), \dots, A(\omega^{t-1})$ by computing the following transposed Vandermonde matrix-vector product:

$$\begin{bmatrix} A(\omega^0) \\ A(\omega^1) \\ \vdots \\ A(\omega^{t-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{x_0} & \omega^{x_1} & \cdots & \omega^{x_{t-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(t-1)x_0} & \omega^{(t-1)x_1} & \cdots & \omega^{(t-1)x_{t-1}} \end{bmatrix} \begin{bmatrix} A[x_0] \\ A[x_1] \\ \vdots \\ A[x_{t-1}] \end{bmatrix}.$$

   Since $\omega$ has order at least $n$, the elements $\omega^{x_1}, \dots, \omega^{x_t}$ are pairwise distinct. Therefore, this matrix is nonsingular and we may apply Theorem 2.18 to efficiently evaluate the product in time $O(t \log^2 t)$.

2. *Interpolation:* Let $x_0, \dots, x_{t-1}$ denote the elements in $\widetilde{X}$. We similarly prepare the powers $\omega^{x_1}, \dots, \omega^{x_t}$ via repeated squaring. To interpolate $A$, we use Theorem 2.18 to solve the following transposed Vandermonde equation system with indeterminates $A[x_1], \dots, A[x_t]$:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega^{x_0} & \omega^{x_1} & \cdots & \omega^{x_{t-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(t-1)x_0} & \omega^{(t-1)x_1} & \cdots & \omega^{(t-1)x_{t-1}} \end{bmatrix} \begin{bmatrix} A[x_0] \\ A[x_1] \\ \vdots \\ A[x_{t-1}] \end{bmatrix}.$$

   Again, this matrix is nonsingular and thus Theorem 2.18 applies to compute a solution in time $O(t \log^2 t)$. Setting $A(X) = \sum_{i=0}^{t-1} A[x_i] \cdot X^{x_i}$, we clearly reconstructed a polynomial with the correct evaluations $A(\omega^i) = a_i$ and support $\operatorname{supp}(A) \subseteq \widetilde{X}$. Moreover, $A$ is the only polynomial satisfying these conditions, since the equation system is nonsingular and therefore $A$ is uniquely determined. ◂

Using Lemma 3.2 we the deterministic algorithm is immediate, knowing a superset of the support and an appropriate element $\omega$.

**Lemma 3.3 (Sparse Convolution over a Large Field).** *Let $\mathbf{F}$ be a field. Given vectors $A, B \in \mathbf{F}^n$, a set $\widetilde{Z} \supseteq \operatorname{supp}(A \star B)$ and an element $\omega \in \mathbf{F}$ with multiplicative order at least $n$, we can compute $A \star B$ in deterministic time $O(t \log^2 t + t \log n)$ using $O(t \log^2 t + t \log n)$ field operations. Here, $t = \|A\|_0 + \|B\|_0 + |\widetilde{Z}|$.*

**Proof.** ▸ We follow an evaluation–interpolation approach. Let us identify vectors with polynomials via $A(X) = \sum_{i=0}^{n-1} A[i] \cdot X^i$. In this correspondence, taking convolutions $A \star B$ corresponds to multiplying polynomials $A(X) \cdot B(X)$.

We first evaluate $A(\omega^0), \dots, A(\omega^{t-1})$ and $B(\omega^0), \dots, B(\omega^{t-1})$ using Lemma 3.2. We then apply Lemma 3.2 again to interpolate a polynomial $C(X)$ with support $\operatorname{supp}(C) \subseteq \widetilde{Z}$ and evaluations $C(\omega^i) = A(\omega^i) \cdot B(\omega^i)$ for all $i \in [t]$. One solution is the correct polynomial $C(X) = A(X) \cdot B(X)$, and Lemma 3.2 guarantees that this is the unique solution. The running time is $O(t \log^2 t + t \log n)$ as claimed. ◂

It remains to construct $\omega$ (see Section 3.3.2) and to find a superset of the support (see the complete algorithm in Section 3.3.3).

### 3.3.2 Finding Large-Order Elements

We next solve the sparse convolution problem for integer vectors $A, B$ assuming that we know the support of $A \star B$, using what we have established in the last section. We start with the following two lemmas due to Cheng [79]; for completeness we include short proofs.

**Lemma 3.4 ([79]).** *Let $\beta \in \mathbf{F}_p$ be primitive. Then $X^{p-1} - \beta \in \mathbf{F}_p[X]$ is irreducible.*

**Proof.** ▸ Let $f = X^{p-1} - \beta$ and let $f = f_1 \dots f_m$ denote its factorization into monic irreducibles. We first prove that all factors have the same degree. Let $\alpha$ be a root of $f$ (in a field extension). Then $\{x\alpha : x \in \mathbf{F}_p^\times\}$ must be the full set of roots of $f$. Indeed, $(x\alpha)^{p-1} - \beta = x^{p-1}\beta - \beta = 0$ by Fermat's Little Theorem, and there cannot be other roots since $f$ has degree $p-1$. Pick arbitrary distinct indices $1 \le i, j \le m$; we prove that $\deg(f_i) \le \deg(f_j)$. Let $x, y \in \mathbf{F}_p^\times$ be such that $x\alpha$ is a root of $f_i$ and $y\alpha$ is a root of $f_j$. We can construct a polynomial $f_j'(X) = f_j(yx^{-1}X)$, which by construction has degree $\deg(f_j)$ and has $x\alpha$ as a root. But recall that $f_i$ is irreducible (and monic) and therefore the minimal polynomial of $x\alpha$. It follows that $\deg(f_i) \le \deg(f_j') = \deg(f_j)$. Since $i, j$ were arbitrary we conclude that all polynomials $f_1, \dots, f_m$ must have common degree $d = \frac{p-1}{m}$.

Next, we prove that $m = 1$. Let $\alpha_1, \dots, \alpha_d$ denote the roots of $f_1$ (in a field extension). As observed before, we have that $\alpha_i \alpha_j^{-1} \in \mathbf{F}_p$ for all $i, j$. Moreover, $\prod_i \alpha_i$ is the constant coefficient of $f_1$ and thus $\prod_i \alpha_i \in \mathbf{F}_p$. It follows that $\alpha_1^d = \prod_{i=1}^d \alpha_1 \alpha_i^{-1} \alpha_i$ is an element of $\mathbf{F}_p$. Recall that $\alpha_1$ is a root of $f$ and hence $\alpha_1^{p-1} = (\alpha_1^d)^m = \beta$. Finally, any value $m > 1$ would contradict the primitivity of $\beta$. ◂

**Lemma 3.5 ([79]).** *Let $f = X^{p-1} - \beta \in \mathbf{F}_p[X]$ be an irreducible polynomial. Then $X + 1$ has multiplicative order at least $2^p$ in $\mathbf{F}_p[X]/\langle f \rangle$ provided that $p \ge 7$.*

**Proof.** ▸ Let $\mathbf{F}_{p^{p-1}}$ denote the field $\mathbf{F}_p[X]/\langle f \rangle$. Let $s$ denote the order of $X+1 \in \mathbf{F}_{p^{p-1}}$ and let $S$ denote the set of $s$-th roots of unity in $\mathbf{F}_{p^{p-1}}$ (that is, $S$ is the set of all polynomials $g \in \mathbf{F}_p[X]/\langle f \rangle$ such that $g^s = 1 \pmod f$). We show that $S$ must be large. We clearly have $X + 1 \in S$. More generally, for any $i \in \mathbf{F}_p^\times$ we also have $iX + 1 \in S$ since $X + 1$ and $iX + 1$ are conjugate over $\mathbf{F}_p$. Furthermore, $S$ is closed under multiplication.

Let $E \subseteq \mathbf{N}^{p-1}$ be the set of all sequences $e = (e_1, \dots, e_{p-1})$ with entry sum $\sum_i e_i = p-2$. For any such sequence $e \in E$, we define $\phi(e) = \prod_{i=1}^{p-1} (iX+1)^{e_i} \in \mathbf{F}_{p^{p-1}}$. By the previous paragraph, $\phi$ is a map $\phi : E \to S$. We claim that $\phi$ is injective. If $\phi(e) = \phi(e')$ for distinct $e, e' \in E$, then by definition

$$\prod_{i=1}^{p-1} (iX+1)^{e_i} = \prod_{i=1}^{p-1} (iX+1)^{e_i'} \pmod f.$$

Recall that $f$ has degree $p-1$, but $\sum_i e_i = \sum_i e_i' < p-1$. It follows that the equation remains true even without computing modulo $f$:

$$\prod_{i=1}^{p-1} (iX+1)^{e_i} = \prod_{i=1}^{p-1} (iX+1)^{e_i'}.$$

However, this identity contradicts unique factorization in $\mathbf{F}_p[X]$. It follows that $\phi$ is injective and therefore $s \ge |S| \ge |E|$. Finally, by a simple counting argument one can show that $|E| = \binom{2p-4}{p-2} \ge 2^p$, for all $p \ge 7$. ◂

For the rest of this section, we will analyze Algorithm 3.2.

**Lemma 3.6 (Correctness of Algorithm 3.2).** *Given integer vectors $A, B$ and an arbitrary set $\widetilde{Z} \supseteq \operatorname{supp}(A \star B)$, Algorithm 3.2 correctly returns $C = A \star B$.*

**Proof.** ▸ First, focus on an arbitrary iteration $i$ of the loop in Lines 3 to 8. We prove that the algorithm correctly computes the vector $C_i \in \mathbf{F}_{p_i}$ which is obtained from

**Algorithm 3.2.** Given vectors $A, B \in \mathbf{Z}^n$ and a set $\widetilde{Z} \supseteq \operatorname{supp}(A \star B)$, this deterministic algorithm computes the convolution $C = A \star B$.

1 Let $k = \lceil \log(n\|A\|_\infty\|B\|_\infty) \rceil$
2 Compute the smallest $k$ primes $p_1, \ldots, p_k$ larger than $\lceil \log n \rceil$
3 **for** $i \leftarrow 1, \ldots, k$ **do**
4  Find a primitive element $\beta \in \mathbf{F}_{p_i}$ by brute-force
5  Let $q_i = p_i^{p_i - 1}$ and represent $\mathbf{F}_{q_i}$ as $\mathbf{F}_{p_i}[X]/\langle X^{p_i-1} - \beta \rangle$
6  Let $\omega = X + 1 \in \mathbf{F}_{q_i}$
7  Reduce the coefficients of $A, B$ modulo $p_i$ to obtain $A_i, B_i \in \mathbf{F}_{p_i}^n \subseteq \mathbf{F}_{q_i}^n$
8  Compute $C_i \leftarrow A_i \star B_i$ over $\mathbf{F}_{q_i}$ using Lemma 3.3 with $\widetilde{Z}$ and $\omega$
9 **for each** $x \in \widetilde{Z}$ **do**
10  Recover $C[x] \in \mathbf{Z}$ from $C_1[x] \in \mathbf{F}_{p_1}, \ldots, C_k[x] \in \mathbf{F}_{p_k}$ using Chinese Remaindering
11 **return** $C$ with entries $C[x]$ for $x \in \widetilde{Z}$ and zeros elsewhere

$C = A \star B$ by reducing all coefficients modulo $p_i$. The polynomial $X^{p_i-1} - \beta$ computed in Lines 4 and 5 is indeed irreducible by Lemma 3.4, so we can represent $\mathbf{F}_{q_i}$ as $\mathbf{F}_{p_i}/\langle X^{p_i-1} - \beta \rangle$ as claimed. Moreover, the element $\omega \in \mathbf{F}_{q_i}$ constructed in Line 6 has multiplicative order at least $2^{p_i} \geq n$ by Lemma 3.5. The preconditions of Lemma 3.3 are satisfied ($\widetilde{Z} \supseteq \operatorname{supp}(A \star B) \supseteq \operatorname{supp}(A_i \star B_i)$ and $\omega$ has order at least $n$), hence we correctly compute $C_i = A_i \star B_i$ in Line 8. Note that although we carry out the computations over the extension field $\mathbf{F}_{q_i}$, the vector $C_i$ is guaranteed to have coefficients in $\mathbf{F}_{p_i}$.

We finally use the Chinese Remainder Theorem to recover $C$ from its images modulo $p_1, \ldots, p_k$. As $\prod_{i=1}^k p_i \geq 2^k \geq n\|A\|_\infty\|B\|_\infty$ exceeds the maximum coefficient in $C$, this recovery step correctly identifies $C = A \star B$. ◀

**Lemma 3.7 (Running Time of Algorithm 3.2).** *The running time of Algorithm 3.2 is bounded by $O(t \log^4 n \operatorname{polyloglog} n)$ where $t = \|A\|_0 + \|B\|_0 + |\widetilde{Z}|$, assuming that $\|A\|_\infty, \|B\|_\infty \leq \operatorname{poly}(n)$.*

**Proof.** ▶ Assuming that $\|A\|_\infty, \|B\|_\infty \leq \operatorname{poly}(n)$, we have $k = \lceil \log(n\|A\|_\infty\|B\|_\infty) \rceil \leq O(\log n)$. Note that $p_1, \ldots, p_k \leq \widetilde{O}(\log n)$ by the Prime Number Theorem, and therefore computing these primes in Line 2 takes time $\widetilde{O}(\log n)$, using for instance Eratosthenes' sieve. Finding a primitive element $\beta \in \mathbf{F}_{p_i}$ in Line 4 takes time $\widetilde{O}(\log n)$ as well and Lines 5 to 7 have negligible costs. Per iteration, running the convolution algorithm in Line 8 takes time $O(t \log^2 t + t \log n) = O(t \log^2 n)$ and requires the computation of at most $O(t \log^2 n)$ field operations in $\mathbf{F}_{q_i}$, thus amounting for time $O(t \log^3 n \operatorname{polyloglog} n)$. In total the loop in Line 3 takes time $O(t \log^4 n \operatorname{polyloglog} n)$. Finally, each call to the algorithmic Chinese Remainder Theorem in Line 10 takes time $O(\log^2(\prod_{i=1}^k p_i)) = O(\log^2 n \operatorname{polyloglog} n)$ [102]. ◀

We remark that our algorithm can be somewhat simplified by exploiting the following result: For any finite field $\mathbf{F}_{p^m}$, one can construct in time $\operatorname{poly}(p, m)$ a (simple-structured) set which is guaranteed to contain a primitive element [190, 191]. The drawback is that the running time worsens by a couple of log factors.

### 3.3.3 Complete Algorithm

We finally remove the assumption that the support of $A \star B$ is given as part of the input by applying Lemma 2.22. The following lemma is immediate:

**Lemma 3.8 (Deterministic Sparse Nonnegative Convolution).** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \log^5 n \operatorname{polyloglog} n)$ where $t = \|A \star B\|_0$, assuming that $\|A\|_\infty, \|B\|_\infty \leq \operatorname{poly}(n)$.*

**Algorithm 3.3.** This algorithm takes two nonnegative vectors $A, B \in \mathbf{N}^n$ and a parameter $m$ and computes a vector $R \leq A \star B$. For details see Lemma 3.10. The algorithm works as well when $h$ is sampled from Lemma 2.8, and in this case the algorithm avoids computing a prime number.

1 Sample a linear hash function $h : [n] \to [m]$
2 Compute $X \leftarrow h(A) \star_m h(B)$
3 Compute $Y \leftarrow h(\partial A) \star_m h(B) + h(A) \star_m h(\partial B)$
4 Compute $Z \leftarrow h(\partial^2 A) \star_m h(B) + 2h(\partial A) \star_m h(\partial B) + h(A) \star_m h(\partial^2 B)$
5 Initialize $R \leftarrow (0, \ldots, 0)$
6 **for each** $k \in [m]$ **do**
7  **if** $X[k] \neq 0$ **and** $Y[k]^2 = X[k] \cdot Z[k]$ **then**
8  $z \leftarrow Y[k]/X[k]$
9  $R[z] \leftarrow R[z] + X[k]$
10 **return** $R$

This lemma yields the proof of Theorem 1.3; to analyze the algorithm for vectors with entries of size $\Delta$, simply view the vectors as having length $n' = \max\{n, \Delta\}$.

## 3.4 Las Vegas Algorithms

The goal of this section is to prove Theorems 1.4 and 1.5. We first prove the sparsity testing lemma (in Section 3.4.1). Then we prove Theorem 1.4 (in Section 3.4.2) and Theorem 1.5 (in Sections 3.4.3 and 3.4.4).

### 3.4.1 Sparsity Testing

Recall that we define the *derivative* $\partial A$ coordinate-wise by $(\partial A)[i] = i \cdot A[i]$, and we define the *d-th derivative* $\partial^d A$ by $(\partial^d A)[i] = i^d \cdot A[i]$. The crucial ingredient for the Las Vegas guarantee is the following lemma about testing 1-sparsity of a vector, having access to its first and second derivatives.

**Lemma 3.1 (Testing 1-Sparsity).** *Let $V$ be a nonnegative vector. Then $\|\partial V\|_1^2 \leq \|V\|_1 \cdot \|\partial^2 V\|_1$. This inequality is tight if and only if $\|V\|_0 \leq 1$.*

**Proof.** ▸ Note that since $V$ is nonnegative, we can rewrite $V[i] = \sqrt{V[i]} \cdot \sqrt{V[i]}$. The proof is a straightforward application of the Cauchy-Schwartz inequality:

$$\|\partial V\|_1^2 = \left(\sum_i iV[i]\right)^2 = \left(\sum_i \sqrt{V[i]} \cdot i\sqrt{V[i]}\right)^2$$
$$\leq \left(\sum_i V[i]\right)\left(\sum_i i^2 V[i]\right) = \|V\|_1 \cdot \|\partial^2 V\|_1.$$

Recall that the Cauchy-Schwartz inequality is tight if and only if the involved vectors $V$ and $\partial^2 V$ are scalar multiples of each other. In our case this is possible if and only if $\|V\|_0 \leq 1$. ◂

### 3.4.2 Simple Algorithm

We are finally ready to analyze Algorithm 3.1. For the ease of presentation, we have extracted the core part of Algorithm 3.1 (Lines 3 to 11) as Algorithm 3.3, and our first goal is a detailed analysis of that core part.

To increase clarity we shall adopt the following naming convention for the rest of this section: The indices $x, y, z \in [n]$ exclusively denote coordinates of large vectors, whereas $i, j, k \in [m]$ denote coordinates of the hashed vectors, or

equivalently, buckets of a hash function $h$. The first lemma analyzes the vectors $X, Y, Z$ computed by the algorithm.

**Lemma 3.9.** *Let $h, X, Y, Z$ be as in Algorithm 3.3. Moreover, for a bucket $k \in [m]$ define the nonnegative vector $V_k \in \mathbf{N}^n$ by*

$$V_k[z] = \sum_{\substack{x+y=z \\ h(x)+h(y)\equiv k \pmod m}} A[x] \cdot B[y].$$

*Then $X[k] = \|V_k\|_1$, $Y[k] = \|\partial V_k\|_1$ and $Z[k] = \|\partial^2 V_k\|_1$.*

Note that $A \star B = \sum_k V_k$. Intuitively, the vector $V_k$ is that part of $A \star B$ which is hashed into the $k$-th bucket.

**Proof.** ▸ We merely showcase that $Y[k] = \|\partial V_k\|_1$; the other proofs are very similar. For convenience, let us denote equality modulo $m$ by $\equiv$. It holds that:

$$\begin{aligned}
Y[k] &= (h(\partial A) \star_m h(B) + h(A) \star_m h(\partial B))[k] \\
&= \sum_{\substack{i,j\in[m] \\ i+j\equiv k}} h(\partial A)[i] \cdot h(B)[j] + h(A)[i] \cdot h(\partial B)[j] \\
&= \sum_{\substack{x,y\in[n] \\ h(x)+h(y)\equiv k}} (\partial A)[x] \cdot B[y] + A[x] \cdot (\partial B)[y] \\
&= \sum_{\substack{x,y\in[n] \\ h(x)+h(y)\equiv k}} (x+y) \cdot A[x] \cdot B[y] \\
&= \sum_{z\in[n]} z \cdot \sum_{\substack{x,y\in[n] \\ x+y=z \\ h(x)+h(y)\equiv k}} A[x] \cdot B[y] \\
&= \sum_{z\in[n]} z \cdot V_k[z] \\
&= \|\partial V_k\|_1.
\end{aligned}$$

◂

Next, we will prove that in every iteration the algorithm computes a feasible approximation $R$ to the target vector $A \star B$.

**Lemma 3.10 (Correctness and Running Time of Algorithm 3.3).** *Given nonnegative vectors $A, B$ and any parameter $m$, Algorithm 3.3 runs in time $O(m \log m)$ and computes a vector $R$ such that for every $z \in [n]$:*

- *$R[z] \le (A \star B)[z]$ (always), and*
- *$R[z] < (A \star B)[z]$ with probability at most $c \cdot \|A \star B\|_0 / m$ for some constant $c$.*

**Proof.** ▸ Fix an iteration $k \in [m]$ of the loop (Line 6) and suppose that the condition in Line 7 is satisfied. Defining $V_k$ as in the previous lemma, we claim that $V_k$ is exactly 1-sparse. Indeed, on the one hand, $V_k$ is not the all-zeros vector as $\|V_k\|_1 = X[k] > 0$. On the other hand, since $\|V_k\|_1 \cdot \|\partial^2 V_k\|_1 = X[k] \cdot Z[k] = Y[k]^2 = \|\partial V_k\|_1^2$ we have that $\|V_k\|_0 \le 1$ by Lemma 3.1. Given that $V_k$ is 1-sparse, it is easy to check that the value $z := Y[k]/X[k]$ as computed in Line 8 is the unique nonzero coordinate in $V_k$, i.e., $\mathrm{supp}(V_k) = \{z\}$. It follows that the update in Line 9 is in fact an update of the form "$R \leftarrow R + V_k$". Recall that $\sum_k V_k = A \star B$, and thus the first item follows directly.

Next, we focus on the second item. We can assume that $z \in \mathrm{supp}(A \star B)$ as otherwise the statement is trivial given the previous paragraph. Let $\Phi \subseteq [m]$ be the set from Lemma 2.8. We say that $z$ *collides* with another index $z'$ if there are $\phi, \phi' \in \Phi$ such that $h(z) + \phi \equiv h(z') + \phi' \pmod m$. If $z$ does not collide with any other $z' \in \mathrm{supp}(A \star B)$ then we say that $z$ is *isolated*. The remaining proof splits into the following two statements:

- *Each index $z \in \operatorname{supp}(A \star B)$ is isolated with probability $1 - O(\|A \star B\|_0/m)$:*
  If $z$ collides with another index $z'$ then we have $h(z) - h(z') \equiv q \pmod m$ for some $q = \phi - \phi'$, $\phi, \phi' \in \Phi$. For any fixed $q$ this event occurs with probability at most $O(\frac{1}{m})$ by the uniform difference property of linear hashing (Lemma 2.8). Taking a union bound over the constant number of elements $q$, we conclude that $z$ collides with $z'$ with probability at most $O(\frac{1}{m})$. Hence the expected number of collisions is $O(\|A \star B\|/m)$. Using Markov's inequality we finally obtain that a collision occurs with probability at most $O(\|A \star B\|/m)$ and only in that case $z$ fails to be isolated.

- *Whenever $z$ is isolated we have $R[z] = (A \star B)[z]$:*
  To see this, it suffices to argue that for all $k$ of the form $k \equiv h(z)+\phi \pmod m$, for some $\phi \in \Phi$, the vectors $V_k$ are at most 1-sparse. In that case the corresponding iterations $k$ each perform the update "$R \leftarrow R + V_k$" in Line 9 and the claim follows since $R[z] = \sum_k V_k[z] = (A \star B)[z]$. So suppose that some vector $V_k$ is at least 2-sparse. In this case there exist $x, x' \in \operatorname{supp}(A)$ and $y, y' \in \operatorname{supp}(B)$ such that $h(x) + h(y) \equiv h(x') + h(y') \equiv k \pmod m$ and $x + y \neq x' + y'$. Then either $z' := x + y$ or $z' := x' + y'$ differs from $z$, and we have witnessed a collision between $z$ and $z'$. This contradicts the assumption that $z$ is isolated.

Finally, note that the running time is dominated by the six calls to FFT in Lines 2 to 4 taking time $O(m \log m)$. The loop (Line 6) only takes linear time. ◀

Recall that Algorithm 3.1 simply calls Algorithm 3.3 repeatedly and returns the coordinate-wise maximum $C$ of all vectors $R$ as soon as $\|C\|_1 = \|A\|_1 \cdot \|B\|_1$. The bucket size $m$ increases from iteration to iteration. Given the analysis of Algorithm 3.3, it remains to prove that Algorithm 3.1 is correct and fast, thereby proving Theorem 1.4.

**Lemma 3.11 (Correctness of Algorithm 3.1).** *Whenever Algorithm 3.1 outputs a vector $C$, then $C = A \star B$ (with error probability 0).*

**Proof.** ▶ In Line 11, $C$ is computed as the coordinate-wise maximum of several vectors $R$ computed by Algorithm 3.3. The previous lemma asserts that $R \leq A \star B$ (coordinate-wise) and therefore also $C \leq A \star B$. Moreover, since $C$ was returned by the algorithm we must have $\|C\|_1 = \|A\|_1 \cdot \|B\|_1$ (Line 13). In conjunction, these facts imply that $C = A \star B$, since both $C$ and $A \star B$ are nonnegative vectors. ◀

**Lemma 3.12 (Running Time of Algorithm 3.1).** *The expected running time of Algorithm 3.1 is $O(t \log^2 t)$, where $t = \|A \star B\|_0$.*

**Proof.** ▶ We first prove that the algorithm terminates with high probability as soon as the outer loop (Line 1) reaches a sufficiently large value. More precisely, let $c$ be the constant from Lemma 3.10 and fix any iteration of the outer loop with value $m \geq 2ct$. We claim that the algorithm terminates within this iteration with probability at least $1 - t^{-1}$. To this end we analyze the probability of the event $C[z] = (A \star B)[z]$, for any fixed index $z$. Recall that $C$ is the coordinate-wise maximum of $2 \log m \geq 2 \log t$ vectors $R$ computed by Algorithm 3.3. For any such vector $R$, Lemma 3.10 guarantees that $R[z] = (A \star B)[z]$ with probability at least $1 - ct/m \geq \frac{1}{2}$. Hence, the probability that $C[z] = (A \star B)[z]$ is at least $1 - 2^{-2 \log t} = 1 - t^{-2}$. By a union bound over the $t$ nonzero entries $z$, the probability that algorithm correctly computes $C = A \star B$ in this iteration is at least $1 - t^{-1}$.

The running time of a single iteration with value $m$ is dominated by the $2 \log m$ calls to Algorithm 3.3 taking time $O(m \log m)$. Sampling the hash functions $h$ has negligible cost (by Lemma 2.8) and so does running the inner-most loop (Line 8). The previous paragraph in particular shows that the algorithm terminates before the $\eta$-th iteration after crossing the critical threshold $m \geq 2ct$, with probability at least $1 - t^{-\eta} \geq 1 - 4^{-\eta}$. Hence, we can bound the expected running time by the total

**Algorithm 3.4.** A more careful implementation of Algorithm 3.1. The running time of this algorithm can be bounded sharper than only in expectation, see Lemma 3.13. In the pseudocode, $\epsilon > 0$ is a parameter.

```
1  C ← (0, ..., 0)
2  for μ ← 0, 1, 2, ..., ∞ do
3      for ν ← 0, 1, 2, ..., μ do
4          repeat μ · 2^{ν/(1+ε)} times
5              Compute R by Algorithm 3.3 with parameter m = 2^{μ−ν}
6              Update C ← max{C, R} (coordinate-wise)
7              if ‖C‖_1 = ‖A‖_1 · ‖B‖_1 then return C
```

time before this threshold ($m < 2ct$) plus the expected time after ($m = 2^\eta \cdot 2ct$) which can be bounded by a geometric series:

$$\sum_{\mu=0}^{\log(2ct)} O(2^\mu \log^2(2^\mu)) + \sum_{\eta=0}^{\infty} 4^{-\eta} \cdot O((2^\eta \cdot t) \cdot \log^2(2^\eta \cdot t)) = O(t \log^2 t). \qquad \blacktriangleleft$$

This finishes the analysis of Algorithm 3.1, but not yet the proof of Theorem 1.4 which additionally claims a tail bound on the running time. To get this additional guarantee, we can modify Algorithm 3.1 to increase $m$ more carefully; see the pseudocode in Algorithm 3.4.

**Lemma 3.13 (Correctness and Running Time of Algorithm 3.4).** *Given nonnegative vectors $A, B \in \mathbf{N}^n$ and any parameter $\epsilon > 0$, Algorithm 3.4 correctly computes their convolution $A \star B$ in expected time $O(t \log^2 t)$, where $t = \|A \star B\|_0$. Moreover, with probability $1 - \delta$ it terminates in time*

$$O\left(t \log^2(t) \cdot \left(\frac{\log(t/\delta)}{\log t}\right)^{1+\epsilon+o(1)}\right).$$

**Proof.** ▸ The correctness proof is exactly as in Lemma 3.11 and can therefore be omitted. We prove the improved running time bound. Let $c$ be the constant from Lemma 3.10 and focus on the iterations of the outer loops (Lines 2 and 3) with values $\mu = M$ and $\nu = N$, where

$$N = \left\lceil (1+\epsilon) \log\left(\frac{\log(t/\delta)}{\log t}\right)\right\rceil \quad \text{and} \quad M = \lceil \log(2ct) \rceil + N.$$

In this case we have $m = 2^{\mu-\nu} \geq 2ct$. We claim that the algorithm terminates in this iteration with probability at least $1 - \delta$. To prove this, we again analyze the probability of the event $C[z] = (A \star B)[z]$ for any fixed index $z$. The vector $C$ is the coordinate-wise maximum of all vectors $R$ computed in the inner loop (Line 2) and Lemma 3.10 proves that the event $R[z] = (A \star B)[z]$ happens with probability at least $1 - ct/m \geq \frac{1}{2}$. Since the inner loop is repeated $\mu \cdot 2^{\nu/(1+\epsilon)}$ times, the event $C[z] = (A \star B)[z]$ happens with probability at least

$$1 - 2^{-\mu \cdot 2^{\nu/(1+\epsilon)}} \geq 1 - 2^{-\log(t) \cdot \log(t/\delta)/\log(t)} = 1 - \frac{\delta}{t}.$$

By a union bound over the $t$ nonzero coordinates $z$, the algorithm computes $C = A \star B$ (and consequently terminates) with probability at least $1 - \delta$.

Now, to analyze the running time, we have to bound the running time until the algorithm reaches the required values $\mu = M$ and $\nu = N$. The running time of a single execution of the inner-most loop is dominated by the call to Algorithm 3.3

which takes time $O(m \log m)$ by Lemma 3.10. Thus, with probability $1 - \delta$ the total running time is bounded by

$$\sum_{\mu=0}^{M} \sum_{\nu=0}^{\mu} \mu \cdot 2^{\nu/(1+\epsilon)} \cdot O(2^{\mu-\nu} \log(2^{\mu-\nu})) \leq O\left( M^2 \sum_{\mu=0}^{M} 2^{\mu} \sum_{\nu=0}^{\mu} 2^{-\epsilon\nu/(1+\epsilon)} \right) \leq O(2^M \cdot M^2).$$

Plugging in the definition of $M$ this becomes

$$O(2^M \cdot M^2) = O\left( t \cdot \left( \frac{\log(t/\delta)}{\log t} \right)^{1+\epsilon} \cdot M^2 \right) = O\left( t \log^2(t) \cdot \left( \frac{\log(t/\delta)}{\log t} \right)^{1+\epsilon+o(1)} \right).$$

Finally, we derive from the previous paragraph that expected running time is bounded by $O(t \log^2 t)$. Indeed, the total running time exceeds $\ell \cdot t \log^2 t$ with probability $\ll O(\ell^{-3})$, and thus the expected running time is $t \log^2 t \cdot \sum_{\ell=1}^{\infty} \ell \cdot O(\ell^{-3}) \leq O(t \log^2 t)$. ◀

This completes the proof of Theorem 1.4: Plugging in any constant $0 < \epsilon < 1$ into Lemma 3.13 yields running time $O(t \log^2(t/\delta))$.

### 3.4.3 Accelerated Algorithm

We now speed up Algorithm 3.4 in expectation. The crucial subroutine in that algorithm is Algorithm 3.3 which computes a good approximation $R$ of $A \star B$. For the improvement we design a similar subroutine which instead computes a good approximation of $A \star B - C$; see Algorithm 3.5.

**Lemma 3.14 (Correctness and Running Time of Algorithm 3.5).** *Given vectors $A, B, C \in \mathbf{Z}^n$ such that $A \star B - C$ is nonnegative, and any parameter $m$, Algorithm 3.5 runs in time $O(m \log m)$ and computes a vector $R$ such that for every $z \in [n]$:*

- $R[z] \leq (A \star B - C)[z]$ *(always), and*
- $R[z] < (A \star B - C)[z]$ *with probability at most $c \log n \cdot \|A \star B - C\|_0 / m$ for some constant $c$.*

**Proof.** ▶ Recall that by Lemma 2.6 the family of hash functions $h(x) = x \bmod p$ is truly additive, i.e., satisfies $h(x) + h(y) \equiv h(x + y) \pmod p$ for all keys $x, y$. As a consequence, it holds that $X = h(A) \star_p h(B) - h(C) = h(A \star B - C)$ and similarly $Y = h(\partial(A \star B - C))$ and $Z = h(\partial^2(A \star B - C))$; the proofs of these statements are straightforward calculations.

The rest of the proof is very similar to Lemma 3.10 and we merely sketch the differences. We analogously define vectors $V_k$ by $V_k[z] = (A \star B - C)[z]$ if $z \equiv k \bmod p$ and $V_k[z] = 0$ otherwise. Then, by the previous paragraph we have $X[k] = \|V_k\|_1$, $Y[k] = \|\partial V_k\|_1$ and $Z[k] = \|\partial^2 V_k\|_1$. It follows by the same argument, using the sparsity tester (Lemma 3.1), that the recovered vector $R$ is exactly $R = \sum_k V_k$, where the sum is over all vectors $V_k$ which are at most 1-sparse. The first item is immediate since $\sum_{k \in [p]} V_k = A \star B - C$.

To prove the second item, it suffices to argue that with good probability each nonzero entry $z$ does not collide with any other nonzero entry $z'$ under $h$. In that case, the vector $V_k$ for $k = h(z)$ is 1-sparse and the algorithm correctly computes $R[z] = (A \star B - C)[z]$. To see that each index $z$ is likely isolated, we apply the $O(\log n)$-universality of $h$ (Lemma 2.6): The probability that $z$ collides with some fixed index $z'$ is at most $O(\log(n)/p) \leq O(\log(n)/m)$. Taking a union bound over the $\|A \star B - C\|_0$ nonzero entries $z'$ yields the claimed bound.

Finally, observe that the running time is again dominated by the six calls to FFT in Lines 2 to 4, which take time $O(m \log m)$. Sampling $h$ takes time $\text{polylog}(m)$ and the loop in Line 6 takes linear time. ◀

**Algorithm 3.5.** Given integer vectors $A, B, C \in \mathbf{Z}^n$ such that $A \star B - C$ is nonnegative and a parameter $m$, this algorithm computes a nonnegative vector $R \le A \star B - C$. For details see Lemma 3.14.

1 Sample a random prime $p \in [m, 2m]$ and let $h(x) = x \bmod p$
2 Compute $X \leftarrow h(A) \star_p h(B) - h(C)$
3 Compute $Y \leftarrow h(\partial A) \star_p h(B) + h(A) \star_p h(\partial B) - h(\partial C)$
4 Compute $Z \leftarrow h(\partial^2 A) \star_p h(B) + 2h(\partial A) \star_p h(\partial B) + h(A) \star_p h(\partial^2 B) - h(\partial^2 C)$
5 Initialize $R \leftarrow (0, \ldots, 0)$
6 **for each** $k \in [p]$ **do**
7  **if** $X[k] \neq 0$ **and** $Y[k]^2 = X[k] \cdot Z[k]$ **then**
8  $z \leftarrow Y[k]/X[k]$
9  $R[z] \leftarrow R[z] + X[k]$
10 **return** $R$

**Algorithm 3.6.** Given two nonnegative vectors $A, B \in \mathbf{N}^n$, this Las Vegas algorithm correctly computes their convolution $A \star B$. This is the accelerated version of Algorithm 3.1; see Lemmas 3.15 and 3.16.

1 $C \leftarrow (0, \ldots, 0)$
2 **for** $m \leftarrow 1, 2, 4, \ldots, \infty$ **do**
3  **repeat** $3 \log \log n$ **times**
4  Compute $R$ by Algorithm 3.3 with inputs $A, B$ and parameter $m$
5  Update $C \leftarrow \max\{C, R\}$ (coordinate-wise)
6  **repeat** $2 \log m$ **times**
7  Compute $R$ by Algorithm 3.5 with inputs $A, B, C, m' = \lceil \frac{m}{\log n} \rceil$
8  Update $C \leftarrow C + R$
9  **if** $\|C\|_1 = \|A\|_1 \cdot \|B\|_1$ **then return** $C$

Next, to obtain the speed-up over Algorithm 3.4, we combine Algorithms 3.3 and 3.5. The idea is that Algorithm 3.4 reaches a good (but imperfect) approximation $C$ of $A \star B$ after only $\log \log n$ iterations of the inner-most loop; after that point $A \star B - C$ is sufficiently sparse so that a few iterations with Algorithm 3.5 can correct the remaining errors. The resulting algorithm is summarized in Algorithm 3.6.

**Lemma 3.15 (Correctness of Algorithm 3.6).** *Whenever Algorithm 3.6 outputs a vector $C$, then $C = A \star B$ (with error probability 0).*

**Proof.** ▸ We first prove that the algorithm maintains the invariant $0 \le C \le A \star B$. There are two types of updates. First, for a vector $R$ computed by Algorithm 3.3, the algorithm updates "$C \leftarrow \max\{C, R\}$". Since $R$ satisfies $0 \le R \le A \star B$ by Lemma 3.10, this update maintains the invariant. Second, for a vector $R$ computed by Algorithm 3.5, the algorithm update "$C \leftarrow C + R$". Since $R$ satisfies $0 \le R \le A \star B - C$ by Lemma 3.14, this update also upholds the invariant.

It is easy to conclude that the algorithm outputs the correct solution $C = A \star B$, as this is the only vector $0 \le C \le A \star B$ which also satisfies $\|C\|_1 = \|A\|_1 \cdot \|B\|_1$. ◂

**Lemma 3.16 (Running Time of Algorithm 3.6).** *The expected running time of Algorithm 3.6 is $O(t \log t \log \log n)$, where $t = \|A \star B\|_0$.*

**Proof.** ▸ For the analysis, we split the execution of the algorithm into two *phases*: The first and initial phase ends as soon as $\|A \star B - C\|_0 \le t/\log^2 n$, and the second phase ends when the algorithm terminates. To analyze the expected running times of both phases, we assume that the outer loop (Line 2) has reached a value $m \ge 2ct$, where $c$ is the maximum of the constants in Lemmas 3.10 and 3.14. In this case we claim that a single execution of the loop body terminates both phases with probability at least $\frac{3}{4}$.

**1** For the first phase we analyze the pseudocode in Lines 3 to 5. Fix an arbitrary index $z \in \operatorname{supp}(A \star B)$. For a vector $R$ computed by Algorithm 3.3 we have $R[z] = (A \star B)[z]$ with probability at least $1 - ct/m \geq \frac{1}{2}$, by Lemma 3.10. If any of the vectors $R$ computed in Lines 3 to 5 satisfies $R[z] = (A \star B)[z]$, then we correctly assign "$C[z] \leftarrow \max\{C[z], R[z]\}$" in Line 5 (and we never change that entry for the remaining execution of the algorithm). Since the loop runs for $3 \log\log n$ iterations, the probability that $C[z]$ remains incorrect is at most $2^{-3\log\log n} = (\log n)^{-3}$. Therefore, the expected number of incorrectly assigned coordinates is at most $t/\log^3 n$ and by Markov's inequality that number exceeds $t/\log^2 n$ with probability at most $1/\log n$. This is less than $\frac{1}{8}$ for sufficiently large $n$.

**2** For the second phase we analyze the pseudocode in Lines 6 to 8. Assuming that the first phase is finished, we have $\|A \star B - C\|_0 \leq t/\log^2 n$. The argument is similar to the first phase: A vector $R$ computed by Algorithm 3.5 satisfies with probability at least $1 - c \log n \cdot \|A \star B - C\|_0/m' \geq 1 - ct/m \geq \frac{1}{2}$ that $R[z] = (A \star B - C)[z]$, for any fixed $z$. Moreover, if any of the vectors $R$ computed in Lines 6 to 8 satisfies $R[z] = (A \star B - C)[z]$ then we correctly update "$C[z] \leftarrow C[z] + R[z]$" (and this entry is unchanged for the remaining execution). The probability that $C[z]$ is still incorrect after $2 \log m \geq 2 \log t$ iterations is $2^{-2\log t} = t^{-2}$. By a union bound over the $t$ nonzero entries $z$, we have correctly computed $C = A \star B$ after finishing the loop with probability at least $1 - t^{-1}$. For sufficiently large $t$, this is at least $\frac{7}{8}$.

In combination, with probability $\frac{3}{4}$ both phases finish and therefore the algorithm terminates within a single iteration of the outer loop. Each iteration takes time $O(m \log m \cdot \log\log n)$ (Lemma 3.10) plus $O(m' \log m' \cdot \log m) = O(m \log^2 m/\log n)$ (Lemma 3.14). To bound the total running time, we use that only with probability $4^{-\eta}$ the algorithm continues for another $\eta$ iterations of the outer loop after crossing the critical threshold $m \geq 2ct$. Hence, the expected running time is bounded by $O(t \log t \log\log n)$ before that threshold and by

$$\sum_{\eta=1}^{\infty} 4^{-\eta} \cdot O\left((2^\eta \cdot t) \log(2^\eta \cdot t) \log\log n + (2^\eta \cdot t) \frac{\log^2(2^\eta \cdot t)}{\log n}\right) = O(t \log t \log\log n)$$

after. In total, the expected time is $O(t \log t \log\log n)$ as claimed. ◀

### 3.4.4 Las Vegas Length Reduction

As the final step, we can reduce the running time of Lemma 3.16 by replacing the $\log\log n$ factor with $\log\log t$. To this end, we implement a length reduction which reduces the convolution of arbitrary-length vectors to a small number of convolutions of length-poly($t$) vectors. The pseudocode is given in Algorithm 3.7. The proof of the following Lemma 3.17 completes the proof of Theorem 1.5.

**Lemma 3.17 (Correctness and Running Time of Algorithm 3.7).** *Given nonnegative vectors $A, B$, Algorithm 3.7 correctly computes their convolution $A \star B$. The expected running time is $O(t \log t \log\log t)$, where $t = \|A \star B\|$.*

**Proof.** ▶ We skip the correctness part since the proof is exactly like the correctness argument of Algorithm 3.1; the only difference here is that $X, Y, Z$ are computed by Algorithm 3.6 instead of FFT, however, Algorithm 3.6 is a Las Vegas algorithm and therefore also always correct.

To analyze the running, we start by lower bounding the probability that any iteration terminates the algorithm. We say that a linear hash function $h$ as sampled in Line 3 is *good* if for all distinct $z, z' \in \operatorname{supp}(A \star B)$ and all $\phi, \phi' \in \Phi$ it holds that $h(z) + \phi \not\equiv h(z') + \phi' \pmod m$; here $\Phi$ is the set in Lemma 2.8. Following the same arguments as in Section 3.4.2 one can prove that Algorithm 3.7 terminates as soon as a good hash function is sampled. Therefore, we now lower bound the

**Algorithm 3.7.** Given two nonnegative vectors $A, B \in \mathbf{N}^n$, this Las Vegas algorithm correctly computes their convolution $A \star B$. This algorithm improves the running time of Algorithm 3.6 by replacing the dependence on $\log\log n$ by $\log\log t$.

```
1  repeat
2      Let m = ||A||_0^3 · ||B||_0^3
3      Sample a linear hash function h : [n] → [m]
4      Compute X ← h(A) ⋆_m h(B) by Algorithm 3.6
5      Compute Y ← h(∂A) ⋆_m h(B) + h(A) ⋆_m h(∂B) by Algorithm 3.6
6      Compute Z ← h(∂²A) ⋆_m h(B) + 2h(∂A) ⋆_m h(∂B) + h(A) ⋆_m h(∂²B)
       by Algorithm 3.6
7      Initialize C ← (0, ..., 0)
8      for each k ∈ [m] do
9          if X[k] ≠ 0 and Y[k]² = X[k] · Z[k] then
10             z ← Y[k]/X[k]
11             C[z] ← C[z] + X[k]
12     if ||C||_1 = ||A||_1 · ||B||_1 then return C
```

probability that a random linear hash function $h$ is good. For fixed $z, z', \phi, \phi'$, the probability that $h(z) + \phi \equiv h(z') + \phi' \pmod m$ is at most $O(\frac{1}{m})$. We take a union bound over the $O(t^2)$ choices of $z, z', \phi, \phi'$ and conclude that a random function $h$ is good with probability at least $1 - O(t^2/m)$. Observe that $\|A\|_0 + \|B\|_0 - 1 \le t \le \|A\|_0 \cdot \|B\|_0$, and thus $t^3 \le m \le t^6$. Therefore, for sufficiently large $t$ each iteration of the loop terminates the algorithm with probability at least $\frac{1}{2}$.

The running time of each iteration $i$ is dominated by the six convolutions computed by Algorithm 3.6. Let $T_{i,1}, \ldots, T_{i,6}$ denote the running times of these calls, respectively. Moreover, let $S_i$ denote the random variable which indicates whether the $i$-th iteration takes place (or whether the algorithm has terminated before). By the previous paragraph we have that $\mathbf{P}(S_i = 1) \le 2^{-i}$. The total running time is bounded by

$$\sum_{i=1}^{\infty} S_i \cdot \sum_{j=1}^{6} T_{i,j}.$$

Hence, by linearity of expectation and since the random variables $S_i$ and $T_{i,j}$ are independent, the expected running time is at most

$$\sum_{i=1}^{\infty} \mathbf{E}(S_i) \cdot \sum_{j=1}^{6} \mathbf{E}(T_{i,j}) \le \sum_{i=1}^{\infty} 2^{-i} \cdot O(t \log t \log\log m) = O(t \log t \log\log t).$$

Here, we used the expected time bound from Lemma 3.16 to bound $\mathbf{E}(T_{i,j})$. ◂

# 4 An Optimal Algorithm for Sparse Nonnegative Convolution

In this chapter we present details on our optimal algorithm for sparse nonnegative convolution. This is based on the paper [61].

61 Karl Bringmann, Nick Fischer, and Vasileios Nakos. “Sparse nonnegative convolution is equivalent to dense nonnegative convolution”. In: *53rd annual ACM symposium on theory of computing (STOC 2021)*. ACM, 2021, pages 1711–1724. 10.1145/3406325.3451090.

**Organization.** This chapter is organized as follows. In Section 4.1 we sketch our algorithm and describe some technical difficulties and highlights. The reduction is split across several sections starting from Sections 4.2 to 4.6; we give an outline for these sections in Section 4.1. Finally, in Section 4.8, we show an improved concentration bound for linear hashing, which we used as an essential ingredient in our reduction, as well as an almost tight lower bound against a theorem from [139].

## 4.1 Overview

Our goal is to prove the following theorems, where, as argued before, Theorem 1.1 follows immediately from Theorem 1.2 by setting $\delta = 2^{\sqrt{\log t}}$.

**Theorem 1.1 (Time-Optimal Sparse Nonnegative Convolution).** *There is a randomized algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \log t + \mathrm{polylog}(n\Delta))$ and with error probability $2^{-\sqrt{\log t}}$, where $t = \|A \star B\|_0$ and $\Delta = \|A \star B\|_\infty$.*

**Theorem 1.2 (Sparse and Dense Nonnegative Convolution Are Equivalent).** *Any randomized algorithm for dense nonnegative convolution with running time $D_\delta(n)$ and error probability $\delta > 0$ can be turned into a randomized algorithm for sparse nonnegative convolution with error probability $\delta$ running in time*

$$S_\delta(t) = O\big(D_\delta(t) + t \log^2(\log(t)/\delta) + \mathrm{polylog}(n\Delta)\big).$$

As we will deal with a sequence of increasingly more and more specialized problems, let us formally introduce the convolution problem we are trying to solve:

**Problem (SparseConv).**
- Input: *Nonnegative vectors $A, B \in \mathbf{N}^n$.*
- Task: *Compute $A \star B$.*

In what follows, we assume that we are given a number $t$ so that $\|A \star B\|_0 \le t$, and we want to recover $A \star B$ in time $O(t \log t)$. This assumption will be removed in Section 4.7 using standard techniques. For the sake of simplicity, we will focus on how to obtain a constant-error randomized algorithm for sparse convolution from a deterministic algorithm for dense convolution.

**Step 1: Universe Reduction from Large to Small.** The first step is to reduce our problem to vectors of length $n = \mathrm{poly}(t)$. We will occasionally refer to $n$ as the *universe size* and we refer to this regime of $n$ as a *small* universe. We say that the universe is *large* if there is no bound on $n$. Formally, we introduce the following problem.

**Problem (SmallUniv-SparseConv).**
- Input: *An integer $t$ and nonnegative vectors $A, B \in \mathbf{N}^{\mathrm{poly}(t)}$ so that $\|A \star B\|_0 \le t$.*
- Task: *Compute $A \star B$.*

We show in Section 4.6 how to reduce the general problem of computing $A \star B$ in a large universe to three instances in a small universe $n$. This step is very similar to the algorithm in Section 3.4.4.

The idea is that in this parameter regime the linear hash function $h$ is *perfect* with probability $1 - 1/\text{poly}(t)$. In combination with the derivative operator $\partial$, it suffices to compute three convolutions $h(A) \star_n h(B), h(\partial A) \star_n h(B), h(A) \star_n h(\partial B)$. Note that the cyclic convolution $\star_n$ can be reduced in the nonnegative case to the non-cyclic convolution at the cost of doubling the sparsity of the underlying vector, i.e., $\|h(A) \star h(B)\|_0 \leq 2\|h(A) \star_n h(B)\|_0 \leq 2\|A \star B\|_0 \leq 2t$. This yields the claimed reduction.

This universe reduction ensures that from now on that hashing via the hash family $h(x) = x \bmod p$ (for a random prime $p$) is $O(\log t)$-universal, i.e., we have removed its undesired dependence on $n$, which will be important for the next step. We stress as a subtle detail that this step crucially relies on the fact that we are dealing with *nonnegative* convolution.

**Step 2: Error Correction.** In the next step, we show that it suffices to compute the convolution $A \star B$ up to $t/\text{polylog}\, t$ errors, since we can correct these errors by iterative recovery with an additive hash function $h$. More precisely assume that we can can efficiently solve the following problem for an appropriate parameter $\epsilon = 1/\text{polylog}\, t$.

**Problem (SmallUniv-Approx-SparseConv).**

- Input: *An integer $t$ and nonnegative vectors $A, B \in \mathbf{N}^{\text{poly}(t)}$ so that $\|A \star B\|_0 \leq t$.*
- Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$.*

If we are able to solve this problem, then the remaining goal is to correct the error between $A \star B$ and $\widetilde{C}$. Let $h : [n = \text{poly}(t)] \to [m = O(t)]$ be a truly additive hash function. We can access the residual vector $A \star B - \widetilde{C}$ via $h(A) \star_m h(B) - h(\widetilde{C}) = h(A \star B - \widetilde{C})$. Thus, since the new universe size is a polylog factor larger than the sparsity of the residual vector, it is possible to continue in an iterative fashion using $h$ and still be within the $O(t \log t)$ time bound. Note that (i) it is crucial that we have recovered a $(1 - 1/\log t)$-fraction of the coordinates of $\widetilde{C}$ rather than only a constant fraction, and (ii) it can (and will) be the case that $\text{supp}(\widetilde{C}) \setminus \text{supp}(A \star B) \neq \emptyset$, i.e., there are spurious elements, but those spurious elements will be removed upon iterating.

There is one catch: Iterative recovery creates a sequence of successive approximations $\widetilde{C}_1, \widetilde{C}_2, \ldots$ to $A \star B$, and the time to hash each such vector, i.e., to perform the subtraction $h_1(A) \star_m h_1(B) - h_1(\widetilde{C}_\ell)$, is $O(t)$. Since there are $O(\log t)$ such subtractions, the total cost spent on subtractions is $O(t \log t)$, which suffices for Theorem 1.1 but not for Theorem 1.2. The natural solution is to reduce the number of successive approximations (iterations), which is closely related to the column sparsity of linear sketches that allow iterative recovery. More sophisticated iterative loop invariants exist [127, 176, 107], but these all get $\Omega(\log t)$ column sparsity. What we observe is that, surprisingly, a small modification of the iterative loop in [105] finishes in $O(\log \log t)$ iterations, rather than $O(\log t)$. In the $\ell$-th iteration we hash to $O(t/\ell^2)$ buckets, and let $t_\ell = \|A \star B - \widetilde{C}_\ell\|_0$. An easy argument yields that with probability $1 - 1/\ell^2$ we have $t_{\ell+1} \leq 1/10 \cdot t_\ell^2/t \cdot \ell^4$, which yields $t_L < 1$ for $L = O(\log \log t)$. This means that the subtraction is performed $O(\log \log t)$ times, so the running time overhead is only $O(t \log \log t)$. A more involved implementation of this idea (due to the fact that we are interested in $o(1)$ failure probability) appears in Section 4.5.

**An Attempt Inspired by Prony's Method.** So far we have reduced to small universe and established that we can afford $t/\text{polylog}\, t$ errors. In the following we want to recover a $(1 - 1/\log t)$-fraction of the coordinates "in one shot". Consider the following line of attack. Fix a parameter $w \ll t$ and sample a linear hash function $h : [n = \text{poly}(t)] \to [m = t/w]$ (see Lemma 2.7). We aim to recover, for each

bucket $z \in [m]$, all entries of the convolution $A \star B$ that are hashed to bucket $z$.[32] This corresponds to hashing $C = A \star B$ to $m = t/w$ buckets; we expect to have $w$ elements per bucket and thus most buckets contain at most $2w$ elements, say. Note that we no longer expect isolated buckets, so we cannot simply use the first derivative $h(\partial C)$ to identify the contribution to the bucket as in Section 2.2. Instead, we compute the *first* $2w$ *derivatives* $h(\partial^1 C), \dots, h(\partial^{2w} C)$ in the following way. Here, as before, $\partial^d A$ denotes the vector with entries $(\partial^d A)[i] = i^d \cdot A[i]$. By the product rule we have that

$$\partial^c C = \partial^c (A \star B) = \sum_{a+b=c} \binom{c}{a} \cdot \partial^a A \star \partial^b B,$$

or equivalently,

$$\frac{1}{c!} \partial^c C = \sum_{a+b=c} \left( \frac{1}{a!} \partial^a A \right) \star \left( \frac{1}{b!} \partial^b B \right).$$

Using this identity we can encode the computation of $h(\partial^1 C), \dots, h(\partial^{2w} C)$ as a single (dense) convolution of size $O(wm) = O(t)$. Inspired by Prony's method we will recover $C$ from the precomputed vectors $h(\partial^1 C), \dots, h(\partial^{2w} C)$.[33]

Two problems remain: First, Prony's method requires heavy algebraic machinery which is too expensive for our needs. We will therefore only use the transposed Vandermonde solver (Theorem 2.18) that itself requires access to the *support* $\operatorname{supp}(C)$ (this is very similar to our deterministic algorithm in Section 3.3). We will deal with this obstacle later in Step 4.

The second problem is more severe. Since we want to successfully recover a $(1 - 1/\log t)$-fraction of elements in $A \star B$, for a $(1 - 1/\log t)$-fraction of support elements $k \in \operatorname{supp}(A \star B)$ it must be the case that $|h^{-1}(h(i))| \leq 2w$. This is a necessary condition in order to recover $(A \star B)[h^{-1}(h(i))]$ using the derivatives. If $h$ was three-wise independent, a standard argument using Chebyshev's inequality would show the desired concentration bound. However, since the linear hash function $h$ is only pairwise independent, we need to take a closer look at concentration of linear hashing.

**Intermezzo on Linear Hashing.** A beautiful paper by Knudsen [139] shows that the linear hash function $h$, despite being only pairwise independent, satisfies refined concentration bounds.

**Theorem 4.1 (Almost Three-Wise Independence [139, Theorem 5]).** *Let $X \subseteq [U]$ be a set of $t$ keys. Randomly pick a linear hash function $h$ with parameters $p > 4n^2$ and $m \leq n$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $y, z \in X$ be chosen independently and uniformly at random. Then:*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \leq \frac{1}{m^2} + \frac{2^{O(\sqrt{\log t \log\log t})}}{mt}. \tag{2}$$

Using the above theorem and Chebyshev's inequality, Knudsen arrives at a concentration bound on the number of elements falling in a fixed bucket, see [139, Theorem 2].[34] Up to the factor $2^{O(\sqrt{\log t \log\log t})} = t^{o(1)}$, this would indeed be the concentration bound satisfied by three-wise independent hash functions. However, this additional $t^{o(1)}$ factor is crucial for our application. Moreover, as we show in Section 4.8, the analysis in [139] is nearly tight. In particular, we show the existence of a set $X$ such that the $t^{o(1)}$ factor is necessary.

**Theorem 4.2 ([139, Theorem 5] is Almost Optimal).** *Let $t$ and $n$ be arbitrary parameters with $n \geq t^{1+\epsilon}$ for some constant $\epsilon > 0$, and let $h$ be a random linear hash*

[32] *Here and in the following for ease of exposition we ignore the issue that entries of $A \star B$ can be split up, due to $h$ being only almost-affine.*

[33] *Classically, Prony's method accesses specific* evaluations *of the respective vector, rather than its derivatives (see the overview in Section 2.3.1). That version comes with some complications though: For instance, it requires access to an element $\omega$ with large multiplicative order. The derivative version, while a little less intuitive, circumvents these computational difficulties. In the conference version of our paper [61] we stick to the classical version and thus obtain a slightly weaker result.*

[34] *We are referring to the FOCS 2016 proceedings version, which differs in an important way from the arXiv version.*

*function which hashes to m buckets. Then there exists a set $X \subseteq [n]$ of t keys, a fixed key $x \notin X$ and buckets $a, b \in [m]$ such that for uniformly random $y, z \in X$ we have*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \geq \frac{1}{mt} \exp\left(\Omega\left(\sqrt{\min\left(\frac{\log t}{\log\log t}, \frac{\log n}{\log^2 \log n}\right)}\right)\right).$$

This brings us to an unclear situation. The structured linear algebra machinery of Prony's method seems inadequate for our purposes and the state-of-the-art concentration bounds of linear hashing do not seem to be sufficiently strong. We show how to remedy this state of affairs, using the following new concentration bound for linear hashing:

**Theorem 4.3 (Closer to Three-Wise Independence in Tiny Universes).** *Let $X \subseteq [n]$ be a set of t keys. Randomly pick a linear hash function h with parameters $p > 4n^2$ and $m \leq n$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $y, z$ be chosen independently and uniformly at random. Then:*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \leq \frac{1}{m^2} + O\left(\frac{n \log n}{mt^2}\right).$$

Note that this theorem does not contradict our lower bound in Theorem 4.2. This result can be proved using the machinery established in [139] as well as some elementary number theory, and is actually simpler than the complete analysis of [139]. We state the relevant parts in Section 4.8.

Keeping this new concentration bound in mind, our next trick (Step 3) is to reduce to a *tiny* universe of size $t$ polylog $t$. In tiny universes, Theorem 4.3 shows that linear hashing is almost 3-wise independent up to polylogarithmic factors—a major improvement in comparison to the original $t^{o(1)}$-overhead. Another technical step is to approximate the support of $A \star B$ (Step 4), which can be done efficiently when the universe is tiny. This replaces the computationally expensive part of Prony's method. After that, we are ready to make the attempt work (Step 5). These steps are described in the following.

**Step 3: Universe Reduction from Small to Tiny.** We further reduce the universe size to $t$ polylog $t$; let us call this regime *tiny*. This is the smallest universe we can hash to while ensuring that with constant probability a $(1 - 1/\log t)$-fraction of coordinates is isolated under the hashing. Apart from this difference the reduction is very similar to Step 0. It remains to solve the following computational problem (again, think of $\epsilon = 1/\text{polylog}\, t$). This is done in Section 4.4.

**Problem (TinyUniv-Approx-SparseConv).**
- Input: *An integer t and nonnegative vectors $A, B \in \mathbf{N}^{\epsilon^{-2}t}$ such that $\|A \star B\|_0 \leq t$.*
- Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$.*

**Step 4: Approximating the Support.** Next we want to approximate the support set $\text{supp}(A \star B)$. Specifically, we want to recover a set $Z$ of size $|Z| = O(t)$ such that $|\text{supp}(A \star B) \setminus Z| \leq t/\text{polylog}\, t$. Since $\text{supp}(A \star B) = \text{supp}(A) + \text{supp}(B)$ (in the sumset notation), for $X = \text{supp}(A)$, $Y = \text{supp}(B)$ we formally want to solve the following problem.

**Problem (TinyUniv-ApproxSupp).**
- Input: *An integer t and sets $X, Y \subseteq [\epsilon^{-2}t]$, such that $|X + Y| \leq t$.*
- Task: *Compute a set Z of size $O(t)$ such that $|(X + Y) \setminus Z| \leq \epsilon t$.*

Note that this problem is merely the restriction to Boolean convolutions in the previous problem TinyUniv-Approx-SparseConv. To solve this problem, we create a sequence of successive approximations to $X + Y$. This approach is inspired by the scaling trick from Section 2.6. Specifically, consider the sets

$$X_\ell = \left\{\left\lfloor \frac{x}{2^\ell} \right\rfloor : x \in X\right\}, \quad Y_\ell = \left\{\left\lfloor \frac{y}{2^\ell} \right\rfloor : y \in Y\right\},$$

for $0 \le \ell \le \log(n/t)$ (where $n = \epsilon^2 t$ is the tiny universe size). For $\ell = \log(n/t)$, we have $X_\ell, Y_\ell \subseteq [t]$, and thus we can compute $Z_\ell := X_\ell + Y_\ell$ by one Boolean convolution in FFT time $O(t \log t)$. Since the universe size $n$ is tiny, the number of levels is just $\log(n/t) = O(\epsilon^{-1}) = O(\log\log t)$. It remains to argue how to go from level $\ell+1$ to $\ell$, to finally approximate $X_0 + Y_0 = X + Y$. We say that a set $Z_\ell$ *closely approximates* $X_\ell + Y_\ell$ if $|Z_\ell| = O(t)$, and $|(X_\ell + Y_\ell) \setminus Z_\ell| \le \epsilon t$. Given a set $Z_{\ell+1}$ which closely approximates $X_{\ell+1} + Y_{\ell+1}$, we want to find a set $Z_\ell$ which closely approximates $X_\ell + Y_\ell$. It is not hard to see that a candidate for $Z_\ell$ is $2Z_{\ell+1} + \{0,1,2\}$. Hence the main problem is keeping the size of $Z_\ell$ small by filtering out false positives. One way to do so would be to compute $h(X_\ell) + h(Y_\ell)$, for a random linear hash function $h : [n] \to [O(t)]$. We then throw away all coordinates $i \in 2Z_{\ell+1} + \{0,1,2\}$ for which the bucket $h(i)$ is empty. Naively computing the convolution would lead to time $\Omega(t \log t \log\log t)$. To improve this, we apply Indyk's algorithm for Boolean convolution:

**Theorem 2.2 (Approximate Sumsets, [124]).** *There exists a randomized algorithm which, given two sets $X, Y \subseteq [n]$ computes in time $O(n)$ a set $\widetilde{Z} \subseteq X + Y$, such that for all $z \in X + Y$ we have $\mathbf{P}(x \in \widetilde{Z}) \ge \frac{2}{3}$.*

Since Indyk's algorithm has a small probability of not reporting an element in the sumset, this leads to losing some elements in $\mathrm{supp}(A)+\mathrm{supp}(B)$, but we are fine with $t/\mathrm{polylog}\, t$ errors. On the positive side, compared to standard Boolean convolution this reduces the running time by a factor $\log t$. Putting everything together carefully, we show that $\mathrm{supp}(A \star B)$ can be approximated in time $O(t(\log\log t)^2)$. For the complete proof we refer to Section 4.3.

**Step 5: Approximate Set Query.** With all reductions and preparations discussed so far, it remains to solve the following problem to finish our algorithm, for details see Section 4.2.

**Problem (TinyUniv-Approx-SetQuery).**

- Input: *An integer $t$, nonnegative vectors $A, B \in \mathbf{N}^{\epsilon^{-2}t}$ and a set $Z$ with $|Z| = O(t)$ and $|\mathrm{supp}(A \star B) \setminus Z| \le \epsilon^3 t$.*
- Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \epsilon t$.*

This is the last step of the algorithm. As in the approach using Prony's method that we discussed above, we pick a parameter $w$, hash to $m = t/w$ buckets, and get access to $h(\partial^1 C), \ldots, h(\partial^{2w} C)$. Recall that in a tiny universe, the surprising observation is that the lower bound on the concentration of linear hashing does not apply, and we obtain the much stronger concentration bound from Corollary 4.27.

Furthermore, we can now circumvent the computationally expensive part of Prony's method, since we have knowledge of most of the support $\mathrm{supp}(A \star B)$. It turns out that we only need to solve $O(t/w)$ transposed Vandermonde systems of size $O(w) \times O(w)$ (over some finite field $\mathbf{Z}_q$). The part of the support we do not know might mess up some the estimates due to collisions, but it is such a small fraction that cannot make us fail on more than a $1/\mathrm{polylog}\, t$-fraction of the coordinates in $Z$ (and the errors that will be introduced due to misestimation will be cleaned up by the iterative recovery loop in Step 2). Using the improved concentration bound for linear hashing, a fast transposed Vandermonde solver (Theorem 2.18), and some additional tricks to compute all vectors $h(\partial^1 C), \ldots, h(\partial^{2w} C)$ simultaneously, we can pick $w = \mathrm{polylog}\, t$ and arrive at a $O(t \log t)$-time algorithm, that is also a reduction from sparse to dense convolution.

One last detail is that Vandermonde system solvers compute multiplicative inverses, which cost time $\Omega(\log q) = \Omega(\log(n\Delta))$ each, and thus account for time $\Omega(t \log(n\Delta))$ in total. We observe that, since we are solving several (in particular, $t/w$) Vandermonde systems, we can run all of them in parallel and batch the inversions across calls. We can then simulate $t/w$ inversions using $O(t/w)$ multiplications and just one division, see Lemma 2.17. This yields $O(t \log t)$ run-

ning time and, as claimed in Theorem 1.2, an additive polylog($n\Delta$) term (which is anyways already present, only for choosing the prime $q$).

## 4.2 Set Queries in a Tiny Universe

As the first step in our chain of reductions, the goal of this section is to give an efficient algorithm for the TinyUniv-Approx-SetQuery problem:

**Problem (TinyUniv-Approx-SetQuery).**

- Input: *An integer $t$, nonnegative vectors $A, B \in \mathbf{N}^{\epsilon^{-2}t}$ and a set $Z$ with $|Z| = O(t)$ and $|\operatorname{supp}(A \star B) \setminus Z| \le \epsilon^3 t$.*
- Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \epsilon t$.*

**Lemma 4.4 (TinyUniv-Approx-SetQuery).** *Let $\epsilon, \delta > 0$. The TinyUniv-Approx-SetQuery problem is in time $O(D_\delta(t) + t\log^2(\epsilon^{-1}) + t\log(\delta^{-1}) + \operatorname{polylog}(\|A\|_\infty, \|B\|_\infty))$ with error probability $\delta$, provided that $\log t \le \epsilon^{-1}, \delta^{-1} \le \operatorname{poly}(t)$.*

We show the lemma in two steps. In Section 4.2.1 we give two important preliminary lemmas, and in Section 4.2.2 we then present and analyze the algorithm which proves Lemma 4.4.

### 4.2.1 Derivative Representation

Let $m, w$ be parameters and let $f(x) = x \bmod m$. As in the previous chapter, we will represent $A$ by its derivatives. Specifically, in this section we will show that we can efficiently evaluate, and under certain restrictions also invert, the following map $A \to f(\partial^0 A), \dots, f(\partial^{w-1} A)$. We will refer to $f(\partial^0 A), \dots, f(\partial^{w-1} A)$ as the *derivative representation* of $A$.

For the inversion a crucial assumption is that we are given a close approximation $X$ of $\operatorname{supp}(A)$. The quality of the recovery is controlled by the following measure $F_m(X)$. We say that a bucket $i \in [m]$ is *overfull for $X$* if there are more than $\frac{2|X|}{m}$ elements $x \in X$ with $i = x \bmod m = f(x)$. We define $F_m(X)$ as the number of elements in $X$ falling into overfull buckets. In other words, we define

$$F_m(X) = \sum_{x \in X} \left[ \sum_{x' \in X} [x \equiv x' \pmod m] > \frac{2|X|}{m} \right],$$

where we use the Iverson notation $[P] \in \{0, 1\}$ to denote the truth value of $P$.

For the remainder of this subsection we assume as before that $A$ is an arbitrary length-$n$ vector with sparsity at most $t$. We further assume that $A$ is over some finite field $\mathbf{F}_q$ in order avoid precision issues in the underlying algebraic machinery.

**Lemma 4.5 (Transform to and from the Derivative Representation).** *Let $A \in \mathbf{F}_q^n$ be a $t$-sparse vector, let $m$ be a parameter, let $f(x) = x \bmod m$ and let $w = \lceil 2t/m \rceil$. Then:*

1. *Given $A$, we can compute $f(\partial^0 A), \dots, f(\partial^{w-1} A)$ in deterministic time $O(t\log^2 w)$.*
2. *Given $f(\partial^0 A), \dots, f(\partial^{w-1} A)$ and a size-$t$ set $X$, we can compute a vector $\widetilde{A}$ in deterministic time $O(t\log^2 w)$. The vector $\widetilde{A}$ satisfies*

$$\|A - \widetilde{A}\|_0 \le w \cdot |\operatorname{supp}(A) \setminus X| + F_m(X).$$

**Proof.** ▸ 1 We start with the first item. Let $X = \operatorname{supp}(A)$ (note that $|X| \le t$). We first partition $X$ into several *chunks* $X_{i,j}$. Start with $X_i = \{x \in X : f(x) = i\}$ and then subdivide each part $X_i$ into chunks $X_{i,1}, X_{i,2}, \dots$ of size $|X_{i,j}| = w$ (except for one chunk of size $\le w$). We claim that in this way we have constructed at most $O(m)$ chunks: On the one hand, there can be at most $m$ chunks of size

exactly $w$ since $A$ has sparsity $t \leq mw$. On the other hand, there can be at most $m$ chunks of size less than $w$ by the way the greedy algorithm works.

Now focus on an arbitrary chunk $X_{i,j}$; for simplicity assume that $|X_{i,j}| = w$. Let $x_1, \ldots, x_w$ denote the elements of $X_{i,j}$ in an arbitrary order. We compute the vector $S_{i,j} \in \mathbf{Z}_q^w$ as the following transposed Vandermonde matrix-vector product:

$$S_{i,j} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_w \\ x_1^2 & x_2^2 & \cdots & x_w^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{w-1} & x_2^{w-1} & \cdots & x_w^{w-1} \end{bmatrix} \begin{bmatrix} A[x_1] \\ A[x_2] \\ \vdots \\ A[x_w] \end{bmatrix}.$$

Since $n < q$ the elements $x_1, \ldots, x_w \in [n]$ are distinct modulo $q$ and therefore the matrix is nonsingular over $\mathbf{F}_q$. We can therefore apply Theorem 2.18 to compute $S_{i,j}$. It remains to return the vectors $f(\partial^\ell A)$ for all $\ell \in [w]$, which we compute as $f(\partial^\ell A)[i] = \sum_j S_{i,j}[\ell]$. It is easy to check that $S_{i,j}[\ell]$ equals $f(\partial^\ell A')[i]$ where $A'$ is the vector obtained from $A$ by restricting the support to $X_{i,j}$. The correctness of the whole algorithm follows immediately.

Finally, we analyze the running time. The construction of the chunks takes time $O(mw) = O(t)$, and also writing down all vectors $f(\partial^\ell A)$ takes time $O(t)$ given the vectors $S_{i,j}$. The dominant step is to compute transposed Vandermonde matrix-vector product for each chunk. Since there are $O(m)$ chunks in total and the running time for solving a single system is bounded by $O(w \log^2 w)$ (by Theorem 2.18), the total running time is $O(mw \log^2 w)$ plus $O(mw \log^2 w)$ ring operations running in constant time each. Therefore, the total running time is $O(t \log^2 w)$.

**2** Next, we prove the second item. Assume we have access to $f(\partial^0), \ldots, f(\partial^{w-1})$. Our goal is to recover a good approximation $\widetilde{A}$ of $A$, provided that an approximation $X$ of $\operatorname{supp}(A)$ is given. As before, the first step is to partition $X$ into buckets $X_i = \{x \in X : x \bmod m = i\}$. We say that the bucket $X_i$ is *overfull* if $|X_i| > w$. In contrast to before, we can afford to ignore all overfull buckets here, so focus on an arbitrary bucket $X_i$ with $|X_i| \leq w$. Letting $x_1, \ldots, x_w$ denote the elements in $X_i$ in an arbitrary order (and assuming for the sake of simplicity that there are exactly $w$ of these), it suffices to solve the following transposed Vandermonde system with indeterminates $\widetilde{A}[x_1], \ldots, \widetilde{A}[x_w]$:

$$\begin{bmatrix} f(\partial^0 A)[i] \\ f(\partial^1 A)[i] \\ \vdots \\ f(\partial^{w-1} A)[i] \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_w \\ x_1^2 & x_2^2 & \cdots & x_w^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{w-1} & x_2^{w-1} & \cdots & x_w^{w-1} \end{bmatrix} \begin{bmatrix} \widetilde{A}[x_1] \\ \widetilde{A}[x_2] \\ \vdots \\ \widetilde{A}[x_w] \end{bmatrix}.$$

We prove that $\|A - \widetilde{A}\|_0$ is small. A bucket $X_i$ is *successful* if (i) it is not overfull, and if (ii) there exists no support element $x \in \operatorname{supp}(A) \setminus X$ with $x \bmod m = i$. The claim is that whenever $X_i$ is successful, then $\widetilde{A}[x] = A[x]$ for all $x \in X_i$. Indeed, for any successful bucket one can verify that the equation system is valid for $A$ in place of $\widetilde{A}$, and as the Vandermonde matrix has full rank this is the unique solution.

Therefore, it suffices to bound the total size of all non-successful buckets: On the one hand, the number of elements in buckets for which condition (i) holds but (ii) fails is at most $w \cdot |\operatorname{supp}(A) \setminus X|$. On the other hand, the contribution of elements in buckets for which condition (i) fails is exactly $F_m(X)$ by definition. Together, these yield the claimed bound on $\|A - \widetilde{A}\|_0$.

The running time can be analyzed in the same way as before: Forming the buckets takes time $O(mw) = O(t)$ and solving the transposed Vandermonde

systems takes $O(mw \log^2 w) = O(t \log^2 w)$ arithmetic operations. However, we have to be careful: Theorem 2.18 states that we need $O(mw \log^2 w)$ ring operations (each running in constant time in the RAM model) *plus one division.* Divisions over $\mathbf{F}_q$ can be implemented in time $O(\log q)$ using the Euclidian algorithm, but it would be too costly to run the Euclidian algorithm $m$ times. Instead, we use the bulk division lemma (Lemma 2.17) which states that we can simulate $m$ parallel divisions by a single division and $O(m)$ multiplications. The running time overhead due to the Euclidian algorithm becomes $O(\log q)$ in total. ◀

The previous lemma shows that we can efficiently transition from $A$ to its derivative representation. The next lemma shows that in the derivative representation we can efficiently simulate the convolution of two vectors.

**Lemma 4.6 (Convolution in the Derivative Representation).** *Let $A, B \in \mathbf{F}_q^n$. Given their derivative representations $f(\partial^0 A), \ldots, f(\partial^{w-1} A)$ and $f(\partial^0 B), \ldots, f(\partial^{w-1} B)$, we can compute the derivative representation $f(\partial^0 C), \ldots, f(\partial^{w-1} C)$ of $C = A \star B$ in time $O(D_\delta(mw) + \log q)$ with error probability $\delta \geq 0$.*

**Proof.** ▶ Let $X$ be the integer vector of length $2mw$ with entries defined as follows: Initialize $X$ to be all-zeros and then set $X[i + 2am] = \frac{1}{a!} \cdot h(\partial^a A)[i]$ for all $i \in [m]$ and $j \in [w]$. Similarly construct $Y$ from $B$. We compute the convolution $Z := X \star Y$ using the dense convolution algorithm and extract the derivative representation of $C$ via

$$f(\partial^c C)[k] = c! \cdot (Z[k + 2cm] + Z[k + (2c + 1)m]).$$

We start with the running time analysis. The factorials $0!, 1!, \ldots, (w-1)!$ can be computed in time $O(w)$. Using the bulk division algorithm (Lemma 2.17) we can reduce the computation of $\frac{1}{0!}, \frac{1}{1!}, \ldots, \frac{1}{(w-1)!}$ to $O(w)$ multiplications running in constant time each plus one division running in time $O(\log q)$ using Euclid's algorithm. The rest of the algorithm runs in time $O(mw + D(mw)) = O(D(mw))$, using a possibly randomized dense convolution algorithm with error probability $\delta \geq 0$.

The correctness is by the following calculation. On the one hand, we have:

$$\begin{aligned}
&\frac{1}{c!} \cdot f(\partial^c C)[k] \\
&\quad = \frac{1}{c!} \cdot \sum_{\substack{z \in [n] \\ z \bmod m = k}} z^c \cdot C[z] \\
&\quad = \frac{1}{c!} \cdot \sum_{\substack{x,y \in [n] \\ (x+y) \bmod m = k}} (x + y)^c \cdot A[x] \cdot B[y]
\end{aligned}$$

We now apply the binomial theorem $(x+y)^c = \sum_{a+b=c} \binom{c}{a} x^a \cdot y^b$ and use that the binomial coefficient can be expressed as $\binom{a}{c} = \frac{c!}{a!(c-a)!}$.

$$
\begin{aligned}
&= \sum_{\substack{x,y\in[n] \\ (x+y) \bmod m = k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} \frac{\binom{c}{a}}{c!} \cdot x^a \cdot y^b \cdot A[x] \cdot B[y] \\
&= \sum_{\substack{x,y\in[n] \\ (x+y) \bmod m = k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} \frac{x^a \cdot A[x]}{a!} \cdot \frac{y^b \cdot B[y]}{b!} \\
&= \sum_{\substack{i,j\in[m] \\ (i+j) \pmod m = k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} \left( \sum_{\substack{x\in[n] \\ x \bmod m = i}} \frac{x^a \cdot A[x]}{a!} \right) \cdot \left( \sum_{\substack{y\in[n] \\ y \bmod m = j}} \frac{y^b \cdot B[y]}{b!} \right) \\
&= \sum_{\substack{i,j\in[m] \\ (i+j) \pmod m = k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} \frac{f(\partial^a A)[i]}{a!} \cdot \frac{f(\partial^b B)[j]}{b!} \\
&= \sum_{\substack{i,j\in[m] \\ (i+j) \pmod m = k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} X[i+2am] \cdot Y[j+2bm]
\end{aligned}
$$

On the other hand we have

$$
Z[k+2cm] = \sum_{\substack{i,j\in[m],\, a,b\in[w] \\ i+2am+j+2bm = k+2cm}} X[i+2am] \cdot Y[j+2bm]
$$

We claim that the condition $i+2am+j+2bm = k+2cm$ is equivalent to $i+j=k$ and $a+b+c$. Indeed, note that $a+b=c$ is implied since $|i+j-k| < 2m$. Therefore:

$$
Z[k+2cm] = \sum_{\substack{i,j\in[m] \\ i+j=k}} \sum_{\substack{a,b\in[w] \\ a+b=c}} X[i+2am] \cdot Y[j+2bm],
$$

and similarly

$$
Z[k+(2c+1)m] = \sum_{\substack{i,j\in[m] \\ i+j=k+m}} \sum_{\substack{a,b\in[w] \\ a+b=c}} X[i+2am] \cdot Y[j+2bm]
$$

In combination we have $h(A \star B)[k,c] = k! \cdot (Z[k+2cm] + Z[k+(2c+1)m])$ and the correctness of the whole algorithm follows. ◀

### 4.2.2 The Algorithm

We are ready to prove Lemma 4.4 by analyzing the pseudocode given in Algorithm 4.1. We start with the running time analysis.

**Lemma 4.7 (Running Time of Algorithm 4.1).** *With probability* $1-\frac{\delta}{2}$, *Algorithm 4.1 terminates in time* $O(D_\delta(t) + t\log^2(\epsilon^{-1}) + t\log(\delta^{-1}) + \text{polylog}(t, \|A\|_\infty, \|B\|_\infty))$, *provided that* $\log t \le \epsilon^{-1} \le \text{poly}(t)$.

**Proof.** ▶ We start analyzing the loop in Lines 4 and 8, and prove that a single iteration succeeds with constant probability. Having established that fact, it is clear that the loop is left after at most $O(\log(1/\delta))$ independent iterations with probability at least $1-\frac{\delta}{4}$. Recall that the loop ends as soon as $F_m(\widetilde{Z}) \le \frac{\epsilon t}{2}$, that is,

$$
F_m(X) = \sum_{z\in\widetilde{Z}} \left[ \sum_{z'\in\widetilde{Z}} [z \equiv z' \pmod m] > \frac{2|\widetilde{Z}|}{m} \right] \le \frac{\epsilon t}{2}. \tag{3}
$$

**Algorithm 4.1.** Solves the TinyUniv-Approx-SetQuery problem: Given integer vectors $A, B \in \mathbf{Z}^n$ where $n = \epsilon^{-2}t$ and a set $Z \approx \operatorname{supp}(A \star B)$, this algorithm computes an approximation $\widetilde{C}$ of $C = A \star B$ with $\|C - \widetilde{C}\|_0 \leq \epsilon t$.

**procedure** TinyUniv-Approx-SetQuery($A, B, Z, t, \epsilon$)
  Let $q > n \cdot \|A\|_\infty \cdot \|B\|_\infty$ be a prime and cast $A, B$ to vectors over $\mathbf{F}_q$
  Let $p > 4n^2$ be a prime, let $m = \Theta(\epsilon t)$ and let $w = \lceil 2t/m \rceil$
  **repeat**
    Pick $\sigma, \tau \in [p]$ uniformly at random
    Let $f(x) = x \bmod m$, $g(x) = (\sigma x + \tau) \bmod p$ and $h(x) = f(g(x))$
    $\widetilde{Z} \leftarrow g(Z) + \{0, p\}$
  **until** $F_m(\widetilde{Z}) \leq \frac{\epsilon t}{2}$
  Compute $A_0 \leftarrow f(\partial^0 g(A)), \ldots, A_{w-1} \leftarrow f(\partial^{w-1} g(A))$ using Lemma 4.5
  Compute $B_0 \leftarrow f(\partial^0 g(B)), \ldots, B_{w-1} \leftarrow f(\partial^{w-1} g(B))$ using Lemma 4.5
  Compute $R_0 \leftarrow f(\partial^0(g(A) \star g(B))), \ldots, R_{w-1} \leftarrow f(\partial^{w-1}(g(A) \star g(B)))$ using Lemma 4.6
  Recover $R$ from $R_0, \ldots, R_{w-1}$ and $\widetilde{Z}$ using Lemma 4.5
  **return** $\widetilde{C} = g^{-1}(R)$ (cast back to an integer vector)

Since by definition $\widetilde{Z} = g(X) + \{0, p\}$, we may fix offsets $o, o' \in \{0, p\}$ and instead bound

$$\sum_{z \in Z} \left[ \sum_{z' \in Z} [h(z) + o = h(z') + o' \pmod m] > \frac{2|Z|}{m} \right] \leq \frac{\epsilon t}{4}, \tag{4}$$

where $h(x) = g(x) \bmod m$ is a linear hash function with parameters $p$ and $m$. Indeed, if the latter event happens, then also the former event happens. Fix $o, o'$ and fix any $x \in X$. Then:

$$\begin{aligned}
&\mathbf{P}\left( \sum_{x' \in X} [h(x) + o = h(x') + o' \pmod m] > \frac{2|X|}{m} \right) \\
&= \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot \mathbf{P}\left( \sum_{x' \in X} [h(x') = (a + o - o') \bmod m] > \frac{2|X|}{m} \,\middle|\, h(x) = a \right)
\end{aligned}$$

This is where our concentration bounds come into play: This conditional probability can be bounded by Corollary 4.27 with buckets $a$ and $b = (a + o - o') \bmod m$. Let $F = \sum_{x' \in X} [h(x') = b]$, then $\mathbf{E}(F) = \frac{|X|}{m} + O(1)$. It follows that:

$$\begin{aligned}
&= \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot \mathbf{P}\left( F > \frac{2|X|}{m} \,\middle|\, h(x) = a \right), \\
&= \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot \mathbf{P}\left( F - \mathbf{E}(F) > \frac{|X|}{m} - O(1) \,\middle|\, h(x) = a \right), \\
&\leq \sum_{a \in [m]} \mathbf{P}(h(x) = a) \cdot O\left( \frac{mn \log n}{|X|^2} \right) \\
&= O\left( \frac{mn \log n}{|X|^2} \right)
\end{aligned}$$

where for the inequality we applied Corollary 4.27 with $\lambda = \sqrt{|X|/m} - O(1)$. We choose $m = c \cdot \epsilon t$ for some small constant $c > 0$, then:

$$= O\left( \frac{c \cdot \epsilon t n \log n}{t^2} \right) = O\left( \frac{c \cdot \epsilon t \epsilon^{-2} t \log(\epsilon^{-2} t)}{t^2} \right) = O(c \cdot \epsilon^{-1} \log t) = O(c).$$

Here we used the assumption $\log t \leq \epsilon^{-1} \leq \mathrm{poly}(t)$. By setting $c$ small enough, this probability becomes less than $\frac{1}{12}$. Then, by a union bound over the three possible values of $o - o'$ and by Markov's inequality, we conclude that the event in Equation (4) (and thereby the event in Equation (3)) happens with probability at least $\frac{1}{2}$.

We are ready to analyze the running time of Algorithm 4.1. Precomputing the primes $p, q$ runs in time $\mathrm{polylog}(t, \|A\|_\infty, \|B\|_\infty)$. As we just proved in the previous paragraph, with probability $1 - \frac{\delta}{4}$ the loop in Lines 4 and 8 runs for at most $O(\log(\delta^{-1}))$ iterations. Each execution of the loop body takes time $O(t)$, and thus the loop terminates in time $O(t \log(\delta^{-1}))$. Finally, we apply Lemma 4.5 three times running in time $O(t \log^2 w) = O(t \log^2(\epsilon^{-1}))$ and apply Lemma 4.6 running in time $O(D_{\delta/4}(t) + \log q)$ with error probability $\frac{\delta}{4}$. By boosting the $D_\delta$ algorithm a constant number of times, we obtain that $O(D_{\delta/4}(t)) = O(D_\delta(t))$. Summing over all contributions yields the bound from the lemma statement, with error probability at most $\frac{\delta}{4} + \frac{\delta}{4} \leq \frac{\delta}{2}$. ◀

**Lemma 4.8 (Correctness of Algorithm 4.1).** *With probability* $1 - 1/\mathrm{poly}(t)$*, Algorithm 4.1 correctly outputs a vector* $\widetilde{C}$ *with* $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$.

**Proof.** ▶ After the loop in Lines 4 and 8 has terminated, we have selected a hash function $g(x) = (ax+b) \bmod p$ and a set $\widetilde{Z} = g(Z) + \{0, p\}$ such that $F_m(\widetilde{Z}) \leq \frac{\epsilon t}{2}$. We correctly compute derivative representations $f(\partial^0 g(A)), \ldots, f(\partial^{w-1} g(A))$ of $g(A)$ and $f(\partial^0 g(B)), \ldots, f(\partial^{w-1} g(B))$ of $g(B)$ by Lemma 4.5. Writing $R = g(A) \star g(B)$, we compute the derivative representation $R_0 = f(\partial^0 R), \ldots, R_{w-1} = f(\partial^{w-1} R)$ of $R$ using Lemma 4.6. Finally, we recover an approximation $\widetilde{R}$ of $R$ using Lemma 4.5 with the guarantee that

$$\|\widetilde{R} - R\|_0 \leq w \cdot |\operatorname{supp}(R) \setminus \widetilde{Z}| + F_m(\widetilde{Z}) \leq w \cdot |\operatorname{supp}(R) \setminus \widetilde{Z}| + \frac{\epsilon t}{2}.$$

To bound $w \cdot |\operatorname{supp}(R) \setminus \widetilde{Z}|$, note that since $\operatorname{supp}(R) \subseteq g(\operatorname{supp}(A \star B)) + \{0, p\}$ and $\widetilde{Z} = Z + \{0, p\}$, we must have that $|\operatorname{supp}(R) \setminus \widetilde{Z}| \leq 2|\operatorname{supp}(A \star B) \setminus Z|$. It follows that

$$w \cdot |\operatorname{supp}(R) \setminus \widetilde{Z}| \leq 2w \cdot |\operatorname{supp}(A \star B) \setminus Z| \leq 2w \cdot \epsilon^3 t \leq O(\epsilon^2 t),$$

which becomes $\frac{\epsilon t}{2}$ for sufficiently large $t$. All in all, this shows that $\|R - \widetilde{R}\|_0 \leq \epsilon t$ as claimed.

The remaining steps are easy to analyze: The function $g(x) = (\sigma x + \tau) \bmod p$ is invertible on $[p]$ (assuming that $\sigma \neq 0$, which happens with high probability). As $g(A \star B) = R \bmod p$ and as $A \star B$ has length $2n < p$ it follows that $A \star B = g^{-1}(R)$. In the same way, we obtain for $\widetilde{C} = g^{-1}(\widetilde{R})$ that $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$. In the final step we use that $q$ is large enough (larger than any entry in the convolution $A \star B$), so we can safely cast $\widetilde{C}$ back to an integer vector. ◀

In combination, Lemmas 4.7 and 4.8 show that Algorithm 4.1 is correct and runs in the claimed running time with probability at least $1 - \frac{\delta}{2} - 1/\mathrm{poly}(t) \geq 1 - \delta$. This finishes the proof of Lemma 4.4.

## 4.3 Approximating the Support Set

This section is devoted to finding a set $Z$ which closely approximates $\operatorname{supp}(A \star B)$. To that end, our goal is to solve the following problem, which is later applied with $X = \operatorname{supp}(A)$ and $Y = \operatorname{supp}(B)$.

**Problem (TinyUniv-ApproxSupp).**

- Input: *An integer $t$ and sets $X, Y \subseteq [\epsilon^{-2} t]$, such that $|X + Y| \leq t$.*
- Task: *Compute a set $Z$ of size $O(t)$ such that $|(X + Y) \setminus Z| \leq \epsilon t$.*

**Algorithm 4.2.** Solves the TinyUniv-ApproxSupp problem. That is, given an integer $t$ and sets $X, Y \subseteq [n = \epsilon^{-2}t]$ such that $|X+Y| \leq t$, this algorithm approximates their sumset $Z \approx X + Y$.

```
1  procedure TinyUniv-ApproxSupp(X, Y, t)
2      Let m = 40t and pick a prime p ≥ n
3      Let L = ⌈log(ε^{-1})⌉
4      Z_L ← {0, 1, ..., ⌈n/2^ℓ⌉}
5      for ℓ ← L − 1, ..., 1, 0 do
6          X_ℓ ← Y div 2^ℓ
7          Y_ℓ ← Z div 2^ℓ
8          M_ℓ ← 2Z_{ℓ+1} + {0, 1, 2}
9          repeat R = Θ(log(ε^{-1}) + log(δ^{-1})) times
10             Randomly pick a linear hash function h with parameters p and m
11             Compute S_ℓ ≈ h(X_ℓ) + h(Y_ℓ) using Indyk's algorithm (Theorem 2.2)
                   with error probability 0.01
12             for x ∈ M_ℓ do
13                 if ∃o ∈ {−p, 0, p} s.t. (h(0) + h(x) + o) mod m ∈ (S_ℓ mod m) then
14                     Give a vote to x
15         Z_ℓ ← all elements in M that have gathered at least 3R/4 votes
16     return Z = Z_0
```

**Lemma 4.9 (TinyUniv-ApproxSupp).** *Let $\epsilon, \delta > 0$. The TinyUniv-ApproxSupp problem can be solved with error probability $\delta$ in time $O(k \log(\epsilon^{-1}) \log(\epsilon^{-1}\delta^{-1}))$.*

The algorithm claimed in Lemma 4.9 is given in Algorithm 4.2. For the remainder of this section, we will analyze this algorithm in several steps. We shall call the iterations of the outer loop *levels* and call an element $z$ a *witness at level $\ell$* if $z \in X_\ell + Y_\ell$. Otherwise, we say that $z$ is a non-witness. Fix a level $\ell$ and consider a single iteration of the inner loop (Lines 9 and 14). The *voting probability of $z$ at level $\ell$* is the probability that $z$ is given a vote in Line 14. Recall that in every such iteration, we pick a random linear hash function $h : [n] \to [m]$ using fresh randomness. The following lemmas prove that witnesses have large voting probability and non-witnesses have small voting probability.

**Lemma 4.10 (Witnesses have Large Voting Probability).** *At any level $\ell$, the voting probability of a witness $z$ is at least* 0.99.

**Proof.** ▶ Recall that if $z$ is a witness at level $\ell$, then $z = x + y$ for some $x \in X_\ell$ and $y \in Y_\ell$. By the almost-additiveness of linear hashing (Lemma 2.7), it holds that $h(x) + h(y) = h(z) + h(0) + o \pmod m$ for some offset $o \in \{-p, 0, p\}$. It follows that $(h(z) + h(0) + o) \bmod m$ is an element of the sumset $(h(X_\ell) + h(Y_\ell)) \bmod m$. However, in order for $z$ to gain a vote, this condition must be true for the set $S_\ell$ returned by Indyk's algorithm. By the guarantee of Theorem 2.2 (boosted to error probability 0.01), $S_\ell$ contains every element of $h(Y_\ell) + h(Z_\ell)$ with probability at least 0.99, which yields the claim. ◀

**Lemma 4.11 (Non-Witnesses have Small Voting Probability).** *At any level $\ell$, the voting probability of a non-witness $z$ is at most $\frac{1}{2}$.*

**Proof.** ▶ Given the fact that Indyk's algorithm never returns a false positive, it suffices to prove that none of the three values $(h(0) + h(z) + \{-p, 0, p\}) \bmod m$ is contained in the sumset $(h(X_\ell) + h(Y_\ell)) \bmod m$, with sufficiently large probability. By the almost-additiveness of $h$, we have

$$h(X_\ell) + h(Y_\ell) \bmod m \subseteq (h(0) + h(X_\ell + Y_\ell) + \{-p, 0, p\}) \bmod m.$$

So fix some offsets $o, o' \in \{-p, 0, p\}$ and some witness $z' \in X_\ell + Y_\ell$. As $z$ is not a witness, we must have $z \neq z'$. It suffices to bound the following probability:

$$\begin{aligned}\mathbf{P}(h(0) + h(z) + o = h(0) + h(z') + o' \bmod m) \\ = \mathbf{P}(h(z) = (h(z') + o' - o) \bmod m) \leq \frac{4}{m},\end{aligned}$$

where in the last step we applied the universality of $h$ (Lemma 2.7). By a union bound over the five possible values of $o' - o$ and over all witnesses $x'$, we conclude that the voting probability of $x$ is at most $20|X_\ell + Y_\ell|/m \leq 20t/m \leq \frac{1}{2}$. ◀

We are now ready to prove Lemma 4.9. We proceed in two steps: First we bound the running time and the number of false positives (i.e., $|Z \setminus (X + Y)|$), and second the number of false negatives (i.e., $|(X + Y) \setminus Z|$).

**Lemma 4.12 (Running Time of Algorithm 4.2).** *With probability $1 - \frac{\delta}{2}$, Algorithm 4.2 outputs a set $Z$ of size $O(t)$, and it runs in time $O(t \log(\epsilon^{-1}) \log(\epsilon^{-1}\delta^{-1}))$.*

**Proof.** ▶ Fix any level $\ell$. By Lemma 4.11 we know that the voting probability of any non-witness $z$ is at most $\frac{1}{2}$. Thus, by an application of Chernoff's bound, the probability that $z$ receives more than $\frac{3R}{4}$ votes over all $R = \Omega(\log L + \log(1/\delta))$ rounds is at most $2^{-\Omega(R)} \leq \delta/(12L)$ by appropriately choosing the constant in the definition of $R$ (in the upcoming Lemma 4.13 we will see why $R$ is even slightly larger). By Markov's inequality, we obtain that with probability $1 - \frac{\delta}{2L}$ the number of non-witness elements in $M_\ell$ which will be inserted in $X_\ell$ is at most $\frac{1}{6}|M_\ell| \leq \frac{1}{2}|X_{\ell+1}|$. By a union bound over all levels, with probability $1 - \frac{\delta}{2}$ we get that

$$|X_\ell| \leq k + \frac{1}{2}|X_{\ell+1}|,$$

for all $\ell \in [L]$. As initially $|X_L| \leq k$ it follows by induction that $|X_\ell| \leq \sum_{i=0}^{\infty} \frac{k}{2^i} = 2k$. In particular we have that $|X| = |X_0| = O(k)$, as claimed.

The total running time of the algorithm can be split into two parts—the time spent on running Indyk's algorithm in Line 11, and the time needed to iterate over all elements $x \in M_\ell$ across all levels and assign them votes (Line 14). The former is $O(mLR) = O(tLR)$ (recall that Indyk's algorithm runs for sets over the universe $[m]$) and also the latter is

$$\sum_{\ell \in [L]} O(|X_\ell|R) = \sum_{\ell \in [L]} O(tR) = O(tLR).$$

All in all, the time is bounded by $O(tLR) = O(t \log(\epsilon^{-1}) \log(\epsilon^{-1}\delta^{-1}))$. ◀

**Lemma 4.13 (Correctness of Algorithm 4.2).** *With probability $1 - \frac{\delta}{2}$, Algorithm 4.2 correctly outputs a set $Z$ with $|(X + Y) \setminus Z| \leq \epsilon t$.*

**Proof.** ▶ Fix any $x \in X$, $y \in Y$ and define $x_\ell = \lfloor \frac{x}{2^\ell} \rfloor$, $y_\ell = \lfloor \frac{y}{2^\ell} \rfloor$ and $z_\ell = x_\ell + y_\ell$. The first step is to prove that $y_\ell \in 2\{y_{\ell+1}\} + \{0, 1, 2\}$. Indeed, from the basic inequalities $2\lfloor a \rfloor \leq \lfloor 2a \rfloor \leq 2\lfloor a \rfloor + 1$, for all rationals $a$, it follows directly that

$$z_\ell - 2z_{\ell+1} = \left\lfloor \frac{x}{2^\ell} \right\rfloor + \left\lfloor \frac{y}{2^\ell} \right\rfloor - 2\left\lfloor \frac{x}{2^{\ell+1}} \right\rfloor - 2\left\lfloor \frac{y}{2^{\ell+1}} \right\rfloor \leq 2,$$

and in the same way $z_\ell - 2z_{\ell+1} \geq 0$.

Coming back to the algorithm, we claim that with probability $1 - \frac{\epsilon\delta}{2}$, $z = x + y$ will participate in $Z$. It suffices to show that with the claimed probability, for all levels $\ell$ the element $z_\ell$ belongs to $Z_\ell$. Note that trivially $z_L \in Z_L$. Fix a specific level $\ell$. Conditioning on $z_{\ell+1} \in Z_{\ell+1}$, it will be the case that $z_\ell$ is inserted into $M_\ell = 2Z_{\ell+1} + \{0, 1, 2\}$ in Line 8, by the fact that $z_\ell \in 2\{z_{\ell+1}\} + \{0, 1, 2\}$. Moreover, recall that $z_\ell$ is a witness at level $\ell$ and thus, by Lemma 4.10, its voting probability is at least 0.99. Therefore it receives more than $\frac{3R}{4}$ votes and is inserted

**Algorithm 4.3.** Solves the SmallUniv-Approx-SparseConv problem. Given an integer $t$ and nonnegative vectors $A, B$ of length $n \le \text{poly}(t)$ such that $\|A \star B\|_0 \le t$, this algorithm approximates the vector $A \star B$ by $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \epsilon k$.

**procedure** SmallUniv-Approx-SparseConv($A, B, t, \epsilon, \delta$)
  Let $p > U$ be a prime and let $m = \lceil 320\epsilon^{-1}\delta^{-1}t \rceil$
  Randomly pick a linear hash function with parameters $p$ and $m$
  *(Approximate $V = h(A) \star_m h(B)$)*
  $\widetilde{V}_1 \leftarrow$ TinyUniv-Approx-SparseConv$(h(A), h(B), t, \frac{\epsilon}{6}, \frac{\delta}{6})$
  $\widetilde{V} \leftarrow \widetilde{V}_1 \bmod m$
  *(Approximate $W = h(\partial A) \star_m h(B) + h(A) \star_m h(\partial B)$)*
  $\widetilde{W}_1 \leftarrow$ TinyUniv-Approx-SparseConv$(h(\partial A), h(B), t, \frac{\epsilon}{6}, \frac{\delta}{6})$
  $\widetilde{W}_2 \leftarrow$ TinyUniv-Approx-SparseConv$(h(A), h(\partial B), t, \frac{\epsilon}{6}, \frac{\delta}{6})$
  $\widetilde{W} \leftarrow (\widetilde{W}_1 + \widetilde{W}_2) \bmod m$
  $\widetilde{C} \leftarrow (0, \ldots, 0)$
  **for** $i \in \text{supp}(\widetilde{V})$ **do**
    $z \leftarrow \widetilde{W}[i] \,/\, \widetilde{V}[i]$
    **if** $z \in [2n-1]$ **then**
      $\widetilde{C}[z] \leftarrow \widetilde{C}[z] + \widetilde{V}[i]$
  **return** $\widetilde{C}$

into $Z_\ell$ with probability at least $1 - 2^{-\Omega(R)} \ge 1 - \frac{\epsilon\delta}{2L}$. Taking a union bound over all levels we obtain that $z$ is contained in $Z$ with probability $1 - \frac{\epsilon\delta}{2}$, and hence we can apply Markov's inequality to conclude that with probability $1 - \frac{\delta}{2}$ it is the case that $|(X+Y) \setminus Z| \le \epsilon t$. ◂

This finishes the proof of Lemma 4.9. Putting together the results from the previous section (Lemma 4.4) and this section (Lemma 4.9 with $\epsilon' = \epsilon^3$), we have established an efficient algorithm to approximate convolutions in a tiny universe:

**Lemma 4.14 (TinyUniv-Approx-SparseConv).** *Let $\epsilon, \delta > 0$. Then TinyUniv-Approx-SparseConv is in time $O(D_\delta(t) + t\log(\epsilon^{-1})\log(\epsilon^{-1}\delta^{-1}) + \text{polylog}(\|A\|_\infty, \|B\|_\infty))$ with error probability $\delta$, provided that $\log t \le \epsilon^{-1}, \delta^{-1} \le \text{poly}(t)$.*

## 4.4 Universe Reduction from Small to Tiny

The goal of this section is to prove that approximating convolutions in a small universe (that is, a universe of size $n = \text{poly}(t)$) reduces to approximating convolutions in a tiny universe.

**Problem (SmallUniv-Approx-SparseConv).**
- Input: *An integer $t$ and nonnegative vectors $A, B \in \mathbf{N}^{\text{poly}(t)}$ so that $\|A \star B\|_0 \le t$.*
- Task: *Compute $\widetilde{C}$ such that $\|A \star B - \widetilde{C}\|_0 \le \epsilon t$.*

**Lemma 4.15 (SmallUniv-Approx-SparseConv).** *Let $\epsilon, \delta > 0$. Then SmallUniv-Approx-SparseConv is in time $O(D_\delta(t) + t\log(\epsilon^{-1})\log(\epsilon^{-1}\delta^{-1}) + \text{polylog}(\|A\|_\infty, \|B\|_\infty))$ with error probability $\delta$, provided that $\log t \le \epsilon^{-1}, \delta^{-1} \le \text{poly}(t)$.*

Our goal is to drastically reduce the universe size from $\text{poly}(t)$ to $\epsilon^{-2}t$, while being granted to introduce up to $\epsilon t$ errors in the output. The idea is to use again to use linear hashing to reduce the universe size, as well as the derivative trick.

For the remainder of this section we analyze the procedure in Algorithm 4.3. Let $h$ be a random linear hash function with parameters $p$ and $m$. We say that an index $z \in \text{supp}(A \star B)$ is *isolated* if there is no other index $z' \in \text{supp}(A \star B)$ with hash value $h(z') \in (h(z) + \{-2p, -p, 0, p, 2p\}) \bmod m$.

**Lemma 4.16 (Most Indices are Isolated).** *With probability $1 - \frac{\delta}{2}$, the number of non-isolated indices $z \in \text{supp}(A \star B)$ is at most $\frac{\delta t}{8}$.*

**Proof.** ▸ For any fixed integers $z', o$, the probability that $h(z') = h(z) + o \pmod m$ is at most $4/m$ by the universality of linear hashing (Lemma 2.7). By taking a union bound over the five values $o \in \{-2p, -p, 0, p, 2p\}$ and the $|\operatorname{supp}(A \star B)| \leq t$ values of $z'$, the probability that $z$ is isolated is at least $1 - \frac{20t}{m}$. As $m = 320\epsilon^{-1}\delta^{-1}t$, any index $z \in \operatorname{supp}(A \star B)$ is isolated with probability at least $1 - \frac{\epsilon\delta}{16}$. The statement follows by an application of Markov's inequality. ◂

Recall that $V = h(A) \star_m h(B)$ and $W = h(\partial A) \star_m h(B) + h(A) \star_m h(\partial B)$.

**Lemma 4.17 (Isolated Indices are Recovered).** *For any isolated index $z$ we have $\sum_i V[i] = (A \star B)[z]$ where $i$ runs over $(h(0)+h(z)+\{-p, 0, p\}) \bmod m$. Furthermore, for any such $i$ it holds that $W[i] = z \cdot V[i]$.*

**Proof.** ▸ Let $z \in \operatorname{supp}(A \star B)$ be isolated and let $x \in \operatorname{supp}(A)$ and $y \in \operatorname{supp}(B)$ be such that $z \neq x + y$. We claim that $h(x) + h(y) \neq h(0) + h(z) + o \pmod m$ for all $o \in \{-p, 0, p\}$. Assume the contrary, then by almost-additiveness (Lemma 2.7) we have that $h(x)+h(y) = h(0)+h(x+y)+o' \pmod m$ for some $o' \in \{-p, 0, p\}$ and thus $h(x+y) = h(z)+o-o' \pmod m$. This is a contradiction as $x+y \in \operatorname{supp}(A \star B)$ and $o - o' \in \{-2p, -p, 0, p, 2p\}$ but $z$ is assumed to be isolated.

Recall that $V = h(A) \star_m h(B)$, and let $i \in (h(0) + h(x) + \{-p, 0, p\}) \bmod m$. For convenience, we write $\equiv$ to denote equality modulo $m$. From the previous paragraph it follows that

$$V[i] = \sum_{\substack{x,y \\ h(x)+h(y)\equiv i}} A[x] \cdot B[y] = \sum_{\substack{x+y=z \\ h(x)+h(y)\equiv i}} A[x] \cdot B[y].$$

By another application of almost-affinity it is immediate that $\sum_i V[i] = (A \star B)[z]$. Moreover, we can express $W[i]$ in a similar way: By repeating the previous argument twice, once with $\partial A$ in place of $A$ and once with $\partial B$ in place of $B$, we obtain that

$$\begin{aligned} W[i] &= \sum_{\substack{x+y=z \\ h(x)+h(y)\equiv i}} (\partial A)[x] \cdot B[y] + \sum_{\substack{x+y=z \\ h(x)+h(y)\equiv i}} A[x] \cdot (\partial B)[y] \\ &= \sum_{\substack{x+y=z \\ h(x)+h(y)\equiv i}} (x + y) \cdot A[x] \cdot B[z] = z \cdot V[i]. \end{aligned}$$ ◂

**Lemma 4.18 (Correctness of Algorithm 4.3).** *With probability $1 - \delta$, Algorithm 4.3 correctly outputs a vector $\widetilde{C}$ with $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$.*

**Proof.** ▸ We call an iteration $i \in \operatorname{supp}(\widetilde{V})$ *good* if the following three conditions hold: (i) $V_i = \widetilde{V}_i$, (ii) $W_i = \widetilde{W}_i$, and (iii) there is an isolated index $z \in \operatorname{supp}(A \star B)$ it holds that $i \in (h(0) + h(z) + \{-p, 0, p\}) \bmod m$. Otherwise, $i$ is *bad*. We start analyzing the algorithm with the unrealistic assumption that all iterations are good.

Focus on an arbitrary iteration $i$. By assumption (iii) there exists some isolated element $z \in \operatorname{supp}(A \star B)$ such that $i \in (h(0) + h(z) + \{-p, 0, p\}) \bmod m$. Moreover, by Lemma 4.17 and assumptions (i) and (ii) the algorithm correctly identifies $z = \widetilde{W}[i]/\widetilde{V}[i] = W[i]/V[i]$, and hence, over the course of the at most three iterations with $i \in (h(0)+h(z)+\{-p, 0, p\}) \bmod m$ the algorithm correctly assigns $\widetilde{C}[z] \leftarrow \sum_i V[i] = (A \star B)[z]$ in Line 13. Under the unrealistic assumption it follows that $\widetilde{C} = A \star B$ after all iterations.

We will now remove the unrealistic assumption. Clearly there is no hope of recovering the non-isolated elements, but Lemma 4.16 proves that there are at most $\frac{\epsilon t}{8}$ non-isolated elements with probability $1 - \frac{\delta}{2}$. Any isolated element will be recovered if it happens to show up in a good iteration as shown in the previous paragraph. It suffices to prove that the number of bad iterations is at most $\frac{7\epsilon t}{8}$. It then follows that $\|A \star B - \widetilde{C}\|_0 \leq \epsilon t$, as any iteration modifies $\widetilde{C}$ in at most one position.

**Algorithm 4.4.** Solves the SmallUniv-SparseConv problem. Given an integer $t$ and nonnegative vectors $A, B$ of length $n \le \text{poly}(t)$ such that $\|A \star B\|_0 \le t$, this algorithm computes their convolution $C = A \star B$.

**procedure** SmallUniv-SparseConv($A, B, t, \delta$)
  Let $m = 8t \log^{-2}(t) \log(n)$
  $C_0 \leftarrow$ SmallUniv-Approx-SparseConv($A, B, t, \frac{\delta}{2}, \frac{\delta}{2}$)
  **for** $\ell \leftarrow 1, \dots, L = O(\log \log t)$ **do**
    **repeat** $\lceil 2 \log(\frac{2L}{\delta}) / 1.5^{\ell-1} \rceil$ **times**
      Randomly pick a prime $p \in [m, 2m]$ and let $h(x) = x \bmod p$
      $V \leftarrow h(A) \star_p h(B) - h(C_{\ell-1})$ using FFT
      $W \leftarrow h(\partial A) \star_p h(B) + h(A) \star_p h(\partial B) - h(\partial C_{\ell-1})$ using FFT
    Keep $h, V, W$ for which $\|V\|_0$ is maximized
    $C_\ell \leftarrow C_{\ell-1}$
    **for** $i \in \text{supp}(V)$ **do**
      $z \leftarrow W[i] / V[i]$
      **if** $z \in [2n-1]$ **then**
        $C_\ell[x] \leftarrow C_{\ell-1}[x] + V[i]$
  **return** $C = C^L$

On the one hand, with probability $1 - \frac{\delta}{2}$, all three calls to the TinyUniv-Approx-SparseConv algorithm succeed and we have $\|V - \widetilde{V}\|_0 \le \frac{\epsilon t}{6}$ and $\|W - \widetilde{W}\|_0 \le \frac{\epsilon t}{3}$. So there can be at most $\frac{\epsilon t}{2}$ iterations for which either assumption (i) or (ii) fails. On the other hand, with probability $1 - \frac{\delta}{2}$ there are at most $\frac{\epsilon t}{8}$ non-isolated indices and any non-isolated index leads to at most three iterations for which (iii) fails. It follows that in total there are at most $\frac{\epsilon t}{2} + \frac{3\epsilon}{8} = \frac{7\epsilon}{8}$ bad iterations. ◂

It is easy to see that the running time of Algorithm 4.3 is dominated by the calls to TinyUniv-Approx-SparseConv and thus bounded as in Lemma 4.14. This completes the proof of Lemma 4.15.

## 4.5 Error Correction

In the previous sections, the goal was design algorithms to *approximate* convolutions. In this step, we show how to clean up the errors and turn the approximations into *exact* convolutions. Formally, we give an algorithm for the following problem:

**Problem (SmallUniv-SparseConv).**
- Input: *An integer $t$ and nonnegative vectors $A, B \in \mathbf{N}^{\text{poly}(t)}$ so that $\|A \star B\|_0 \le t$.*
- Task: *Compute $A \star B$.*

**Lemma 4.19 (SmallUniv-SparseConv).** *Let $\delta > 0$. The SmallUniv-SparseConv problem is in time $O(D_\delta(t) + t \log^2(\delta^{-1}) + \text{polylog}(\|A\|_\infty, \|B\|_\infty))$ with error probability $\delta$, provided that $\log^2 t \le \delta^{-1} \le \text{poly}(t)$.*

This is the only part of the reduction for which we cannot use linear hashing, as the recovery loop crucially relies on certain cancellations to take place. The problem is that linear hashing is only almost—not truly—additive. Instead, we use the simpler hash function $h(x) = x \bmod p$, where $p$ is a random prime in some specified range (see Lemma 2.6).

We analyze Algorithm 4.4. For the analysis, we refer to iterations of the outer loop as *levels* $\ell$. We say that an element $z \in \text{supp}(A \star B - C_{\ell-1})$ is *isolated at level* $\ell$ if there exists no $z' \in \text{supp}(A \star B - C_{\ell-1})$ with $z \ne z'$ and $h(x) = h(x')$, where $h$ is the function picked at the $\ell$-th level.

**Lemma 4.20 (Most Indices are Isolated).** *Let $\ell$ be any level. If*

$$\|A \star B - C_{\ell-1}\|_0 \le \frac{2^{-1.5^{\ell-1}} t}{\log^2(t)},$$

*then with probability* $1 - \frac{\delta}{2L}$*, there will be at most*

$$\frac{2^{-1.5^\ell}t}{2\log^2(t)}$$

*non-isolated elements at level* $\ell$*.*

**Proof.** ▶ We will prove the statement in three steps.

**1** *A random hash function* $h$ *achieves that there are at most* $2^{-1.5^\ell}t\log^{-2}(t)/4$ *non-isolated elements with probability at least* $1 - \sqrt{2^{-1.5^{\ell-1}}}$*:*
For any fixed index $z \in \operatorname{supp}(A \star B - C_{\ell-1})$ there are at most $\|A \star B - C_{\ell-1}\|_0$ other indices $z'$ that $z$ could collide with. And for any distinct $z, z'$ the collision probability is at most

$$\mathbf{P}(h(z) = h(z')) \leq \frac{2\log(n)}{m} \leq \frac{2\log(n)\log^2(t)}{8t\log(n)} = \frac{\log^2(t)}{4t}$$

by $O(\log n)$-universality, see Lemma 2.6. Taking a union bound over $z'$, we obtain that $z$ is non-isolated with probability at most $\|A \star B - C_{\ell-1}\|_0 \cdot \frac{\log^2(t)}{4t}$ and we expect at most $\|A \star B - C_{\ell-1}\|_0^2 \cdot \frac{\log^2(t)}{4t}$ non-isolated elements. By Markov's inequality, the probability that there are more than $2^{-1.5^\ell} \cdot \frac{t}{4\log^2(t)}$ non-isolated elements is at most

$$\frac{\|A \star B - C_{\ell-1}\|_0^2 \cdot \frac{\log^2(t)}{4t}}{2^{-1.5^\ell} \cdot \frac{t}{4\log^2(t)}} \leq \frac{(2^{-1.5^{\ell-1}})^2}{2^{-1.5^\ell}} = \sqrt{2^{-1.5^{\ell-1}}}.$$

**2** *With probability* $1 - \frac{\delta}{2L}$*, running* $\lceil 2\log(\frac{2L}{\delta}) \,/\, 1.5^{\ell-1} \rceil$ *independent trials will result in at least one function* $h$ *under which there are at most* $2^{-1.5^\ell}t\log^{-2}(t)/4$ *non-isolated elements:*
I.e., at some time during the execution of the inner loop in Lines 5 and 8 we find a good hash function $h$. Indeed, the failure probability is at most

$$\left(\sqrt{2^{-1.5^{\ell-1}}}\right)^{2\log(\frac{2L}{\delta}) \,/\, 1.5^{\ell-1}} = \frac{\delta}{2L}.$$

**3** *If there are at most* $r$ *non-isolated elements then* $\|V\|_0 \geq \|A \star B - C_{\ell-1}\|_0 - r$ *and conversely, if* $\|V\|_0 \geq \|A \star B - C_{\ell-1}\|_0 - r$ *then there can be at most* $2r$ *non-isolated elements:*
By additiveness (Lemma 2.6) the algorithm computes $V = h(A \star B - C_{\ell-1})$. As every isolated element $z$ is the unique element in its bucket $i = h(z)$ it follows directly that $\|V\|_0 \geq \|A \star B - C_{\ell-1}\|_0 - r$ without accounting for the non-isolated elements. For the converse direction we note that there is a way of "ignoring" $r$ elements $z \in \operatorname{supp}(A \star B - C_{\ell-1})$ such that all other elements become isolated. The number of non-isolated elements is thus at most $r$ (the ignored elements) plus $r$ (the number elements colliding with one of the ignored elements).

The lemma statement follows by combining the second and third intermediate claims: By the second claim, the inner loop (Lines 5 and 8) will eventually discover some hash function $h$ under which we have at most $2^{-1.5^\ell}t\log^{-2}(t)/4$ non-isolated elements and thus, by the third claim, $\|V\|_0 \geq \|A \star B - C_{\ell-1}\|_0 - 2^{-1.5^\ell}t\log^{-2}(t)/4$. As the algorithm selects the function which maximizes $\|V\|_0$, the third claim proves that whatever function is kept in Line 9 leads to at most $2^{-1.5^\ell}t\log^{-2}(t)/2$ non-isolated elements. ◀

**Lemma 4.21 (Isolated Indices are Recovered).** *Denoting the umber of non-isolated elements at level* $\ell$ *by* $r$*, we have* $\|A \star B - C_\ell\|_0 \leq 2r$*.*

**Proof.** ▶ Focus on arbitrary $\ell$, and assume that we already picked a hash function $h$ in Lines 5 and 9. By the additiveness of $h$ it holds that $V = h(A \star B - C_{\ell-1})$ and, by additionally using the product rule (Lemma 2.12), $W = h(\partial(A \star B - C^{\ell-1}))$.

Now focus on an arbitrary iteration $i \in \mathrm{supp}(V)$ of the second inner loop in Lines 11 and 14. There must exist some $z \in \mathrm{supp}(A \star B - C_{\ell-1})$ with $g(z) = i$. If $z$ is isolated then we will correctly set $C_\ell[z] = (A \star B)[z]$. Indeed, since $z$ is isolated it follows that $V[i] = (A \star B - C_{\ell-1})[z]$ and $W[i] = (\partial(A \star B - C^{\ell-1}))[z] = z \cdot V[i]$. Thus $z$ is correctly detected in Line 12 and in Line 14 we correctly assign $C_\ell[z] \leftarrow C_{\ell-1}[z] + V[i] = (A \star B)[z]$.

The previous paragraph shows that if at level $\ell$ all elements were isolated, we would compute $C_\ell = A \star B$. We analyze how this guarantee is affected by the *bad* iterations $i$ for which there exist non-isolated elements $z \in \mathrm{supp}(A \star B - C_{\ell-1})$ with $h(x) = i$. Clearly we cannot hope to correctly assign the $r$ entries $C_\ell[z]$ for which $z$ is non-isolated. Additionally, there are at most $r$ bad iterations, each of which possibly modifies $C_\ell$ in at most one position. All in all, we conclude that $\|A \star B - C_\ell\|_0 \leq 2r$. ◀

**Lemma 4.22 (Correctness of Algorithm 4.4).** *With probability $1 - \delta$, Algorithm 4.4 correctly outputs $C = A \star B$.*

**Proof.** ▶ We show that with probability $1 - \delta$ it holds that

$$\|A \star B - C_\ell\|_0 \leq 2^{-1.5^\ell} t \log^{-2}(t)$$

for all levels $\ell$. In particular, at the final level $L = \log_{1.5} \log k = O(\log \log k)$ we must have $\|A \star B - C_L\|_0 = 0$ and thus $A \star B = C_L = C$. The proof is by induction on $0 \leq \ell \leq L$.

For $\ell = 0$, the statement is true assuming that the SmallUniv-Approx-Sparse-Conv algorithm with parameter $\frac{\delta}{2} \leq \frac{1}{2} \log^{-2}(t)$ succeeds.

For $\ell \geq 1$, we appeal to the previous lemmas: By the induction hypothesis we assume that $\|A \star B - C^{\ell-1}\|_0 \leq 2^{-1.5^{\ell-1}} t \log^{-2}(t)$. Hence, by Lemma 4.20, the algorithm picks a hash function $h$ under which only $2^{-1.5^\ell} t \log^{-2}(t)/2$ elements are non-isolated at level $\ell$. It follows that $\|A \star B - C^\ell\|_0 \leq 2^{-1.5^\ell} t \log^{-2}(t)$ by Lemma 4.21, which is exactly what we intended to show.

Let us analyze the error probability: For $\ell = 0$, the error probability is $\frac{\delta}{2}$. For any other level (there are at most $L$ such), the error probability is $\frac{\delta}{2L}$ by Lemma 4.20. Taking a union bound over these contributions yields the claimed error probability of $1 - \delta$. ◀

**Lemma 4.23 (Running Time of Algorithm 4.4).** *The running time of Algorithm 4.4 is $O(D_\delta(t) + t \log^2(\delta^{-1}) + \mathrm{polylog}(\|A\|_\infty, \|B\|_\infty))$.*

**Proof.** ▶ We invoke SmallUniv-Approx-SparseConv once with parameter $\frac{\delta}{2}$ which takes time exactly as claimed, see Lemma 4.15. After that, the running is bounded by $O(t \log(\delta^{-1}))$ mostly due to the inner loop in Lines 5 and 8: A single execution of the loop body takes time $O(t)$ for hashing the six vectors $A, \partial A, B, \partial B, C_{\ell-1}, \partial C_{\ell-1}$ and for computing three convolutions of vectors of length $m = O(t/\log t)$ using FFT. It remains to bound the number of iterations:

$$\sum_{\ell=1}^{L} \left\lceil \frac{2\log(\frac{2L}{\delta})}{1.5^{\ell-1}} \right\rceil \leq L + \sum_{\ell=1}^{L} \frac{2\log(\frac{2L}{\delta})}{1.5^{\ell-1}} = O(L + \log(\tfrac{L}{\delta})) = O(\log(\delta^{-1})).$$ ◀

This finishes the proof of Lemma 4.19.

## 4.6 Universe Reduction from Large to Small

The final step in our chain of reductions is to reduce from an arbitrarily large universe to a small universe (that is, a universe of size $n = \mathrm{poly}(t)$). We thereby solve the SparseConv problem and almost complete the proof of Theorem 1.2—up to the assumption that the sparsity $t$ is known which we will remove in the next section.

**Algorithm 4.5.** Solves the SparseConv problem by computing the convolution $A \star B$ of two given vectors $A, B$ without any restriction on their length.

```
1  procedure SparseConv(A, B)
2      Let m = poly(t) and let p > nm be a prime
3      Randomly pick a linear hash function with parameters p and m
       (Compute V = h(A) ⋆_m h(B))
4      V_1 ← SparseUniv-SparseConv(h(A), h(B), δ/6)
5      V ← V_1 mod m
       (Compute W = h(∂A) ⋆_m h(B) + h(A) ⋆_m h(∂B))
6      W_1 ← SmallUniv-SparseConv(h(∂A), h(B), δ/6)
7      W_2 ← SmallUniv-SparseConv(h(A), h(∂B), δ/6)
8      W ← (W_1 + W_2) mod m
9      C ← (0, ..., 0)
10     for i ∈ supp(V) do
11         z ← W[i] / V[i]
12         if z ∈ [2n − 1] then
13             C[z] ← C[z] + V[i]
14     return C
```

**Problem (SparseConv).**

- Input: *Nonnegative vectors* $A, B \in \mathbf{N}^n$.
- Task: *Compute* $A \star B$.

**Lemma 4.24 (SparseConv).** *Let* $\delta > 0$. *The SparseConv problem can be solved in time* $O(D_\delta(t) + t \log^2(\delta^{-1}) + \text{polylog}(n, \|A\|_\infty, \|B\|_\infty))$ *with error probability* $\delta$, *provided that* $\log^2 t \le \delta^{-1} \le \text{poly}(t)$.

We will prove Lemma 4.24 by analyzing Algorithm 4.5, which in essence is a simpler version of Algorithm 4.3. For that reason, we will be brief in this section.

As in Section 4.4, we call an index $z \in \text{supp}(A \star B)$ *isolated* if there exists no other index $z' \in \text{supp}(A \star B)$ with $h(z') \in (h(z) + \{-2p, -p, 0, p, 2p\}) \bmod m$.

**Lemma 4.25 (All Indices are Isolated).** *With probability* $1 - 1/\text{poly}(t)$, *all indices* $z \in \text{supp}(A \star B)$ *are isolated.*

**Proof.** ▸ The probability that $h(z') = h(z) + o \pmod m$ is at most $\frac{4}{m}$, for any distinct keys $z, z'$ and any fixed integer $o$, by the universality of linear hashing (Lemma 2.7). By taking a union bound over the five values $o \in \{-2p, -p, 0, p, 2p\}$ and the $t^2$ values of $(z, z')$, the probability that all indices $z$ are isolated is at least $1 - \frac{20t^2}{m}$. The statement follows since $m = \text{poly}(\|A\|_0 \cdot \|B\|_0) \ge \text{poly}(t)$. ◂

**Proof of Lemma 4.24.** ▸ To use the SmallUniv-SparseConv algorithm, we only have to guarantee that the hashed vectors have length at most $\text{poly}(t)$, which is true by $m = \text{poly}(\|A\|_0 \cdot \|B\|_0) \le \text{poly}(t)$. Therefore, with probability $1 - \frac{\delta}{2}$ it holds that $V$ and $W$ are correctly computed. And, by Lemma 4.25, with probability $1 - 1/\text{poly}(t)$ we have that all indices $z$ are isolated. Both events happen simultaneously with probability at least $1 - \delta$, so for the rest for the proof we condition on both these events. By exactly the same proof as Lemma 4.17 we get that

$$(A \star B)[z] = \sum_i V[i],$$

where $i$ runs over $(h(0) + h(z) + \{-p, 0, p\}) \bmod m$, and, for any such $i$ it holds that $W[i] = z \cdot V[i]$. In particular, in Line 11 we correctly identify $z = W[i] / V[i]$ and thus correctly assign $C[z] = \sum_i V[i]$ over the course of the at most three iterations $i$.

The recovery loop in Lines 10 and 13 takes time $O(t)$. The running time is dominated by the sparse convolutions in a small universe that, as shown in Lemma 4.19,

**Algorithm 4.6.** For given sets $X, Y \subseteq [n]$, this algorithm estimates the size of their sumset $Z = X + Y$ up to a constant factor.

```
1  procedure EstimateSumsetSize(X, Y, δ)
2      Compute a prime p > n
3      for t* ← 2, 4, 8, ... do
4          m ← 0
5          repeat O(log(δ^{-1} log(t*))) times
6              Sample a linear hash function h with parameters p and m = t
7              Compute h(X) and h(Y)
8              Compute Z̃ ≈ h(X) + h(Y) using Indyk's algorithm (Theorem 2.2)
9              m ← max{m, |Z̃|}
10         if m ≤ t*/600 then
11             return t*
```

run in time $O(D_\delta(t) + t\log^2(\delta^{-1}) + \mathrm{polylog}(n, \|A\|_\infty, \|B\|_\infty))$. Here, in contrast to before, we cannot replace $n$ by $t$ in the additive term $\mathrm{polylog}(n, \|A\|_\infty, \|B\|_\infty)$, since the entries of $\partial A$ are as large as $n\|A\|_\infty$. ◀

## 4.7 Estimating the Sparsity $t$

In this section we remove the assumption that an estimate $t \geq \|A \star B\|_0$ is given as part of the input. Let us redefine the meaning of $t$ as $t = \|A \star B\|_0$ and refer to the estimate as $t^* \geq t$.

There are two natural approaches: The first and easier approach is to run the algorithm from the previous sections and guess the sparsity $t^* \leftarrow 1, 2, 4, \ldots$ via exponential search. Eventually we will reach the correct estimate $t^* \leq t \leq 2t$ and the algorithm will return the correct output with good probability. In order to not be fooled by the first iterations, we can use the verifier from Lemma 2.21. We take this approach in the paper version of this work, but it comes with annoying technical assumptions.[35]

[35] *The technical assumption is $\sum_{\ell=0}^{k} D_\delta(2^\ell) = O(D_\delta(2^k))$. Indeed, this assumption is necessary to bound the running time of the iterations before guessing a good estimate $t^*$. This is a rather mild assumption, but we managed to remove the assumption entirely in this write-up.*

We instead take another more direct approach here, and prove that we can estimate $t$ in advance. Consider the following lemma, applied with $X = \mathrm{supp}(A)$ and $Y = \mathrm{supp}(B)$.

**Lemma 4.26 (Estimating Sumset Size).** *Let $\delta > 0$. There is an algorithm that, given two sets $X, Y \subseteq [n]$ computes a constant-factor approximation $t^*$ of $t = |X + Y|$ in time $O(t\log(\delta^{-1}) + t\log\log t)$ with error probability $\delta$.*

The algorithm is given in Algorithm 4.6. The idea is to use linear hashing and Indyk's algorithm—similar to the algorithm in Section 4.3.

**Proof.** ▶ For the correctness of the algorithm, we argue that as soon as $t^* > 600t$, then the algorithm certainly terminates and returns $t^*$. Indeed, recall that by the almost-additiveness of linear hashing we have that $|h(X) + h(Y)| \leq 3|X + Y| = 3t$. Since Indyk's algorithm always returns a subset $\widetilde{Z} \subseteq X + Y$, in each iteration of the inner loop we have that $|\widetilde{Z}| \leq 3t \leq \frac{t^*}{600}$.

It remains to argue that the algorithm does not terminate prematurely, with probability at least $1 - \delta$. Consider an iteration $t^* \leq t$, and let $Z' \subseteq Z = X + Y$ be an arbitrary subset of size $\frac{t^*}{20}$. For any fixed elements $z, z' \in Z' = X + Y$, the event $h(z) = h(z')$ happens with probability at most $\frac{4}{t^*}$ by the universality of linear hashing (Lemma 2.7). It follows that the expected number of collisions in $Z'$ is at most $|Z'|^2 \cdot \frac{4}{t^*} = \frac{t^*}{100}$. By Markov's inequality, with probability at least $\frac{3}{4}$, the number of collisions is at most $\frac{t^*}{25}$. In this case the set $h(Z)$ has size at least $\frac{t^*}{20} - \frac{t^*}{25} \geq \frac{t^*}{100}$. Moreover, $|h(X) + h(Y)| \geq |h(Z)| \geq \frac{t^*}{100}$. Now let $\widetilde{Z}$ be a set as computed by Indyk's algorithm. Recall that each element $z \in h(X) + h(Y)$ has probability at most $\frac{1}{3}$ of not being included into $\widetilde{Z}$. By another application of Markov's inequality, with

probability $\frac{1}{2}$ at most half of the elements are missing in $\widetilde{Z}$. In this case it holds that $|\widetilde{Z}| \geq \frac{t^*}{200}$.

In summary, with error probability at most $\frac{1}{4} + \frac{1}{2} \leq \frac{3}{4}$, each iteration produces a set $\widetilde{Z}$ of size at least $\frac{t^*}{200}$. In particular, the probability that the algorithm stops prematurely in the $t^*$-th iteration is bounded by

$$(1 - \tfrac{3}{4})^{O(\log(\delta^{-1}\log(t^*)))} \leq \frac{\delta}{2\log^2(t^*)},$$

by choosing appropriate constants in the $O$-notation. Therefore, the total error probability across all iterations can be bounded as follows; let $t^* = 2^\ell$:

$$\sum_{\ell=1}^{\lceil \log t \rceil} \frac{\delta}{2\log^2(2^\ell)} \leq \delta \cdot \sum_{\ell=1}^{\infty} \frac{1}{2\ell^2} \leq \delta \cdot \frac{\pi^2}{12} \leq \delta.$$

To analyze the running time, we observe that each iteration $t^*$ of the outer loop runs in time $O(t^* \log(\delta^{-1}\log(t^*)))$. Summing over all iterations until $t^* > 600t$ yields total time

$$\sum_{\ell=1}^{\lceil \log(600t) \rceil} O(2^\ell \log(\delta^{-1}\log t)) = O(t\log(\delta^{-1}) + t\log\log t). \qquad \blacktriangleleft$$

Using Lemma 4.26, we can remove the implicit assumption that a constant-factor approximation of $t = \|A \star B\|_0$ is given as part of the input from the previous sections. The running time overhead $O(t\log(\delta^{-1}) + t\log\log t)$ is negligible in the running time from Lemma 4.24. All in all, this completes the proof Theorem 1.2.

## 4.8 Concentration Bounds for Linear Hashing

In this section we sharpen the best-known concentration bounds for the most classic textbook hash function

$$h(x) = ((\sigma x + \tau) \bmod p) \bmod m,$$

for a certain range of parameters. Here, $p$ is some (fixed) prime, $m \leq p$ is the (fixed) number of buckets and $\sigma, \tau \in [p]$ are chosen uniformly and independently at random. We say that $h$ is a *linear hash function* with parameters $p$ and $m$.

Our main goal is to prove the following Theorem 4.3, which is essential for the analysis of our sparse convolution algorithm and which we believe to be of independent interest. The result is based on the machinery established by Knudsen [139]. In that work [139], Knudsen gives the following improved concentration bounds for $h$, similarly (but also crucially different from) the ones achieved by three-wise independent hash functions. For completeness, we repeat some of Knudsen's proofs.

**Theorem 4.1 (Almost Three-Wise Independence [139, Theorem 5]).** *Let $X \subseteq [U]$ be a set of $t$ keys. Randomly pick a linear hash function $h$ with parameters $p > 4n^2$ and $m \leq n$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $y, z \in X$ be chosen independently and uniformly at random. Then:*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \leq \frac{1}{m^2} + \frac{2^{O(\sqrt{\log t \log\log t})}}{mt}. \tag{2}$$

Unfortunately, as we will prove soon, there is no hope of improving the overhead $2^{O(\sqrt{\log t \log\log t})}$ by much in general. Fortunately though, we prove that there is a loophole: In universes of small size tighter bounds are possible:

**Theorem 4.3 (Closer to Three-Wise Independence in Tiny Universes).** *Let $X \subseteq [n]$ be a set of $t$ keys. Randomly pick a linear hash function $h$ with parameters $p > 4n^2$ and $m \leq n$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $y, z$ be chosen independently and uniformly at random. Then:*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \leq \frac{1}{m^2} + O\left(\frac{n \log n}{mt^2}\right).$$

Our result improves upon Theorem 4.1 when $n \leq t \cdot 2^{o(\sqrt{\log t \log\log t})}$. For $n \leq t \operatorname{polylog} t$ and $m \approx t$ (which is the relevant case for us), Theorem 4.3 provides a bound which is worse only by a polylogarithmic factor in comparison to a truly three-wise independent hash function. (Indeed, for a truly three-wise independent hash function, the above probability is exactly $1/m^2$.) We apply Theorem 4.3 by means of the following corollary.

**Corollary 4.27 (Overfull Buckets).** *Let $X \subseteq [n]$ be a set of $t$ keys. Randomly pick a linear hash function $h$ with parameters $p > 4n^2$ and $m \leq n$, fix a key $x \notin X$ and buckets $a, b \in [m]$. Moreover, let $F = \sum_{y \in X} [h(y) = b]$. Then:*

$$\mathbf{E}(F \mid h(x) = a) = \mathbf{E}(F) = \frac{t}{m} \pm \Theta(1),$$

*and, for any $\lambda > 0$,*

$$\mathbf{P}(\,|F - \mathbf{E}(F)| \geq \lambda\sqrt{\mathbf{E}(F)} \mid h(x) = a) \leq O\left(\frac{n \log n}{\lambda^2 t}\right).$$

We remark that the same concentration bounds can be obtained for the related family of hash functions $x \mapsto \left\lfloor \frac{((\sigma x + \tau) \bmod p)m}{p} \right\rfloor$.

The remainder of this section is structured as follows: In Section 4.8.1 we recap some basic definitions from [139] and prove—as the key step—a certain number-theoretic bound. In Section 4.8.2 we then give proofs of Theorem 4.3 and Corollary 4.27 closely following Knudsen's proof outline. Finally, in Section 4.8.3 we prove that the concentration bound in Theorem 4.1 is almost optimal by giving an almost matching lower bound.

### 4.8.1 Heights

We start with some definitions. Let $p$ be a prime. There is a natural way to embed $\mathbf{Z}$ into $\mathbf{F}_p$ via $\iota(x) = x \bmod p$. As we often switch from $\mathbf{Z}$ to $\mathbf{F}_p$, we introduce some shorthand notation: For $x \in \mathbf{Z}$, we write $\bar{x} = \iota(x)$.

The central concept of this proof is an arithmetic measure called the *height* $H(x)$ of a nonzero rational number $x \in \mathbf{Q}$, which is defined as $\max(|a|, |b|)$ if $x$ can be written as $\frac{a}{b}$ for coprime integers $a, b$. We also define a similar height measure for $\mathbf{F}_p$: The height $H_p(\bar{x})$ of $\bar{x} \in \mathbf{F}_p^\times$ is defined as

$$H_p(\bar{x}) = \min\left\{ \max(\,|a|, |b|\,) : a, b \in \mathbf{Z}, \bar{x} = \bar{a}\bar{b}^{-1} \right\}.$$

**Lemma 4.28 (Equivalence of Heights [139, Lemma 2]).**

- *Let $x, y$ be nonzero integers with $|x|, |y| < \sqrt{p/2}$. Then $H(\frac{x}{y}) = H_p(\bar{x}\bar{y}^{-1})$.*
- *For all nonzero integers $x$, $H_p(\bar{x}) < \sqrt{p}$.*

**Proof.** ▸ We start with the first item. It is clear that $H_p(\bar{x}\bar{y}^{-1}) \leq H(\frac{x}{y})$ as whenever we can write $\frac{x}{y} = \frac{a}{b}$ for integers $a, b$, then we also have $\bar{x}\bar{y}^{-1} = \bar{a}\bar{b}^{-1}$. It remains to prove that $H(\frac{x}{y}) \leq H_p(\bar{x}\bar{y}^{-1})$. Recall that we have

$$H_p(\bar{x}\bar{y}^{-1}) \leq \max(|x|, |y|) < \sqrt{p/2}.$$

Take the pair $\bar{a}, \bar{b} \in \mathbf{F}_p^\times$ with $\bar{x}\bar{y}^{-1} = \bar{a}\bar{b}^{-1}$ which minimizes $\max(|a|, |b|)$ and thus satisfies that $\max(|a|, |b|) = H_p(\bar{x}\bar{y}^{-1}) < \sqrt{p/2}$. Then, on the one hand $xb - ya$ must be an integer divisible by $p$. On the other hand, since $|a|, |b|, |x|, |y| < \sqrt{p/2}$ it follows that $|xb - ya| < p$. In combination, we have that $xb - ya = 0$. Finally, observe that $\frac{x}{y} = \frac{a}{b}$ and thus $H(\frac{x}{y}) \le \max(|a|, |b|) = H_p(\bar{x}\bar{y}^{-1})$.

It remains to prove the second item. Fix any nonzero integer $x$, let $r = \lfloor \sqrt{p} \rfloor$ and let

$$\bar{S} = \{\bar{j}\bar{x} : j \in [0\,..\,r]\}.$$

Observe that $\bar{S}$ contains $r+1$ distinct elements. Let $0 = s_0 < \cdots < s_r < p$ denote the unique integers such that $\bar{S} = \{\bar{s}_0, \ldots, \bar{s}_r\}$, and let $s_{r+1} = p$. Then we have

$$\sum_{i=0}^{r} s_{i+1} - s_i = p,$$

and thus there exists some index $i$ with $s_{i+1} - s_i \le \frac{p}{r+1} < \sqrt{p}$. By the definition of $\bar{S}$ we can write $\bar{x} = (\bar{s}_{i+1} - \bar{s}_i)\bar{j}^{-1}$ for some $j \in \{1, \ldots, r\}$ and hence $H_p(\bar{x}) \le \max(s_{i+1} - s_i, r) < \sqrt{p}$. ◀

The next lemma constitutes the heart of our concentration bound. Knudsen [139, Corollary 4] shows that for any set $X$ of size $t$, the sum $\sum_{x,y \in X} 1/H(\frac{x}{y})$ can be bounded by $t \cdot 2^{O(\sqrt{\log t \log\log t})}$ regardless of the universe size $n$; in our setting (where $n$ is as small as $t \operatorname{polylog} t$) the following bound is significantly sharper.

**Lemma 4.29 (Sum of Inverse Heights).** *Let $X \subseteq [-n\,..\,n]$ be a set of nonzero integers. Then:*

$$\sum_{x,y \in X} \frac{1}{H(\frac{x}{y})} \le O(n \log n).$$

**Proof.** ▶ Let us assume that $X$ contains only positive integers; in the general case the sum can be at most four times larger since $H(\frac{x}{y}) = H(-\frac{x}{y})$. We start with the following simple observation: If $x, y$ are positive integers, then $H(\frac{x}{y}) = \max(x, y)/\gcd(x, y)$. Therefore, the goal is to bound

$$\sum_{x,y \in X} \frac{1}{H(\frac{x}{y})} = \sum_{x,y \in X} \frac{\gcd(x,y)}{\max(x,y)} \le 2 \sum_{x \in X} \frac{1}{x} \sum_{\substack{y \in X \\ x \ge y}} \gcd(x, y).$$

Fix $x$ and focus on the sum $\sum_y \gcd(x, y)$. Let $g$ be a divisor of $x$. Then there can be at most $x/g$ values $y \le x$ which are divisible by $g$. It follows that $|\{y \le x : \gcd(x, y) = g\}| \le x/g$. Thus:

$$\sum_{\substack{y \in X \\ x \ge y}} \gcd(x, y) \le \sum_{g | x} g \cdot \frac{x}{g} = x \cdot d(x),$$

where $d(x)$ denotes the number of divisors of $x$. Combining these previous equations, we obtain

$$\sum_{x,y \in X} \frac{1}{H(\frac{x}{y})} \le 2 \sum_{x \in X} d(x) \le 2 \sum_{x=1}^{n} d(x).$$

To bound the right-hand side by $O(n \log n)$, it suffices to check that the average number in $[n]$ has $O(\log n)$ divisors. More precisely: Any integer $g \in [n]$ divides at most $n/g$ elements in $[n]$ and therefore $\sum_x d(x) \le \sum_g n/g = O(n \log n)$. ◀

### 4.8.2 Proof of Theorem 4.3

We need some technical lemmas proved in [139]; for the sake of completeness we also give short proofs. Let us call a set $\bar{I} = \{\bar{a} + \overline{ib} : i \in [r]\} \subseteq \mathbf{F}_p$ an *arithmetic progression*, and if $\bar{b} = \bar{1}$ then we call $\bar{I}$ an *interval*. For a set $\bar{X} \subseteq \mathbf{F}_p$, we define the *discrepancy* as

$$\operatorname{disc}(\bar{X}) = \max_{\bar{I} \text{ interval}} \left| |\bar{X} \cap \bar{I}| - \frac{|\bar{X}|\,|\bar{I}|}{p} \right|.$$

**Lemma 4.30 ([139, Lemma 3]).** *Let $x, y$ be coprime integers with $|x|, |y| < \sqrt{p}$ and let $\bar{X}$ be an interval of length $|x|$. Then $\operatorname{disc}(\bar{y}\bar{x}^{-1}\bar{X}) \le 2$.*

**Proof.** ▶ For simplicity assume that $x, y$ are positive (the other cases are symmetric) and also assume that $\bar{X} = \{\bar{i} : i \in [x]\}$ (which is enough, since the discrepancy is invariant under shifts). We first show that $\bar{x}^{-1}\bar{X}$ is evenly distributed in the following strong sense: $\bar{x}^{-1}\bar{X} = \bar{Y}$, where $\bar{Y} = \{\overline{\lceil jp/x \rceil} : j \in [x]\}$. Since $\bar{x}^{-1}\bar{X}$ and $\bar{Y}$ are finite sets of the same size, it suffices to prove the inclusion $\bar{x}^{-1}\bar{X} \subseteq \bar{Y}$. So fix any $i \in [x]$; we show that $\bar{i}\bar{x}^{-1} \in \bar{Y}$. Let $j \in [x]$ be the unique integer such that $x$ divides $jp + i$. Then $\overline{(jp+i)/x} = \bar{i}\bar{x}^{-1}$ and $\lceil jp/x \rceil = (jp+i)/x$ and hence $\bar{i}\bar{x}^{-1} \in \bar{Y}$.

The next step is to show that also $\bar{y}\bar{x}^{-1}\bar{X}$ is distributed evenly. From the previous paragraph we know that $\bar{y}\bar{x}^{-1}\bar{X} = \{\overline{jpy/x + \epsilon_j y} : j \in [x]\}$ for some rational values $\epsilon_j = \lceil jp/x \rceil - jp/x < 1$. The key insight is that since $x$ and $y$ are coprime integers, the set $[x]$ is invariant under the dilation with $y$, that is, $\{jy \bmod x : j \in [x]\} = [x]$. It follows that $\bar{y}\bar{x}^{-1}\bar{X} = \{\overline{jp/x + \delta_j} : j \in [x]\}$ for some rationals $\delta_j$ with $0 \le \delta_j < y < p/x$.

We point out how to conclude that $\operatorname{disc}(\bar{y}\bar{x}^{-1}\bar{X}) \le 2$. First, it is obvious that all intervals of the form $\{\bar{i} : \lceil jp/x \rceil \le i < \lceil (j+1)p/x \rceil\}$ intersect $\bar{y}\bar{x}^{-1}\bar{X}$ in exactly one point. As every interval $\bar{I}$ can be decomposed into several such segments plus two smaller parts of size less than $p/x$ at the beginning and the end, respectively, a simple calculation confirms that $\operatorname{disc}(\bar{y}\bar{x}^{-1}\bar{X}) \le 2$. ◀

**Lemma 4.31 ([139, Lemma 4]).** *Let $\bar{x}, \bar{y} \in \mathbf{F}_p$ and let $\bar{I} \subseteq \mathbf{F}_p$ be an arithmetic progression. Then, for $\bar{\sigma} \in \mathbf{F}_p$ chosen uniformly at random:*

$$\mathbf{P}((\bar{\sigma}\bar{x}, \bar{\sigma}\bar{y}) \in \bar{I}^2) \le \frac{|\bar{I}|^2}{p^2} + O\left(\frac{1}{H_p(\bar{x}\bar{y}^{-1})} \cdot \frac{|\bar{I}|}{p} + \frac{1}{p}\right),$$

**Proof.** ▶ First, observe that $\bar{I} = \bar{z}\bar{J}$ for some interval $\bar{J} \subseteq \mathbf{F}_p$ and some nonzero scalar $\bar{z}$. As $(\bar{\sigma}\bar{x}, \bar{\sigma}\bar{y}) \in \bar{I}^2$ holds if and only if $(\bar{\sigma}\bar{x}\bar{z}^{-1}, \bar{\sigma}\bar{y}\bar{z}^{-1}) \in \bar{J}^2$, we may replace $\bar{x}, \bar{y}$ by $\bar{x}\bar{z}^{-1}, \bar{y}\bar{z}^{-1}$ and assume that $\bar{I}$ is an interval. Note that $H_p(\bar{x}\bar{y}^{-1})$ is invariant under this exchange. We may further scale and possibly swap $\bar{x}$ and $\bar{y}$ to ensure that $H_p(\bar{x}\bar{y}^{-1}) = x \ge |y|$, where $x, y$ are integers such that $\iota(x) = \bar{x}$ and $\iota(y) = \bar{y}$ with $x$ positive and $x, y$ coprime.

Pick $\bar{\sigma} \in \mathbf{F}_p$ uniformly at random, and let $\bar{S} = \{\bar{\sigma} + \bar{i}\bar{x}^{-1} : i \in [x]\}$. Instead of bounding the probability $\mathbf{P}((\bar{\sigma}\bar{x}, \bar{\sigma}\bar{y}) \in \bar{I}^2)$ directly, by linearity of expectation we may instead bound

$$\begin{aligned}
&\mathbf{P}((\bar{\sigma}\bar{x}, \bar{\sigma}\bar{y}) \in \bar{I}^2) \\
&\quad = \frac{1}{x}\,\mathbf{E}\left(\sum_{\bar{s} \in \bar{S}} [(\bar{s}\bar{x}, \bar{s}\bar{y}) \in \bar{I}^2]\right) \\
&\quad \le \frac{1}{x}\,\mathbf{E}(\min(\,|\bar{I} \cap \bar{x}\bar{S}|, |\bar{I} \cap \bar{y}\bar{S}|\,))
\end{aligned}$$

Recall that $\bar{I}$ is an interval and note that $\bar{y}\bar{S}$ is exactly of the form $\bar{y}\bar{x}^{-1}\bar{X}$ for coprime integers $x, y$ and an interval $\bar{X}$ of size $x$. Therefore, it follows from the previous Lemma 4.30 that $|\bar{I} \cap \bar{y}\bar{S}| \le \frac{|\bar{I}|\cdot|\bar{y}\bar{S}|}{p} + 2 = \frac{x|\bar{I}|}{p} + 2$:

$$\le \frac{1}{x}\mathbf{E}\left(\min\left(|\bar{I} \cap \bar{x}\bar{S}|, \frac{x|\bar{I}|}{p} + 2\right)\right)$$

Next, since both $\bar{I}$ and $\bar{x}\bar{S}$ are intervals, there are less than $|\bar{x}\bar{S}| + |\bar{I}| = x + |\bar{I}|$ choices of $\bar{\sigma}$ such that $\bar{I} \cap \bar{x}\bar{S}$ is non-empty. We conclude that:

$$\le \frac{x + |\bar{I}|}{px}\left(\frac{|\bar{I}|\,x}{p} + 2\right)$$
$$\le \frac{|\bar{I}|^2}{p^2} + \frac{3|\bar{I}|}{px} + \frac{2}{p},$$

recalling that $x = H_p(\bar{x}\bar{y}^{-1}) < \sqrt{p}$ (by Lemma 4.28). The claim follows. ◀

We are finally ready to prove Theorem 4.3 and Corollary 4.27. The proofs are analogous to [139, Theorems 5 and 6].

**Proof of Theorem 4.3.** ▶ Fix $y, z \in X$. We will later unfix $y$ and $z$ and consider them to be random variables. Let $I = \{i \in [p] : i \bmod m = a\}$ and define $J$ similarly with $b$ in place of $a$; clearly $\bar{I}$ and $\bar{J}$ are arithmetic progressions in $\mathbf{F}_p$. Let $\bar{h}(\bar{x}) = \bar{\sigma}\bar{x} + \bar{\tau}$ be a random linear function on $\mathbf{F}_p$. Then:

$$\begin{aligned}
&\mathbf{P}(h(y) = h(z) = b \mid h(x) = a)\\
&\quad = \mathbf{P}((\bar{h}(\bar{y}), \bar{h}(\bar{z})) \in \bar{J}^2 \mid \bar{h}(\bar{x}) \in \bar{I})\\
&\quad = \frac{1}{|I|}\sum_{\bar{u}\in\bar{I}} \mathbf{P}((\bar{h}(\bar{y}), \bar{h}(\bar{z})) \in \bar{J}^2 \mid \bar{h}(\bar{x}) = \bar{u}).
\end{aligned}$$

The last equality is by conditioning on $\bar{h}(\bar{x})$ taking some fixed value $\bar{u}$. As the random variables $\bar{h}(\bar{x})$ and $\bar{h}(\bar{y}) - \bar{h}(\bar{x}) = \bar{\sigma}(\bar{y} - \bar{x})$ are independent, we can omit the condition and apply Lemma 4.31:

$$\begin{aligned}
&\mathbf{P}((\bar{h}(\bar{y}), \bar{h}(\bar{z})) \in \bar{J}^2 \mid \bar{h}(\bar{x}) = \bar{u})\\
&\quad = \mathbf{P}((\bar{\sigma}(\bar{y} - \bar{x}), \bar{\sigma}(\bar{z} - \bar{x})) \in (\bar{J} - \bar{u})^2)\\
&\quad \le \frac{|J|^2}{p^2} + O\left(\frac{1}{H_p\left(\frac{\bar{y}-\bar{x}}{\bar{z}-\bar{x}}\right)}\cdot\frac{|J|}{p} + \frac{1}{p}\right)
\end{aligned}$$

By the definition of $\bar{J}$ we have $|J| \le \lceil \frac{p}{m} \rceil \le \frac{p}{m} + 1$ and thus:

$$\le \frac{1}{m^2} + O\left(\frac{1}{H_p\left(\frac{\bar{y}-\bar{x}}{\bar{z}-\bar{x}}\right)}\cdot\frac{1}{m} + \frac{1}{p}\right).$$

Now we unfix $y, z$ and consider $y, z \in X$ to be chosen uniformly at random. By averaging over the previous inequalities we get:

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) = \frac{1}{m^2} + O\left(\frac{1}{k^2 m}\sum_{y,z\in X}\frac{1}{H_p\left(\frac{\bar{y}-\bar{x}}{\bar{z}-\bar{x}}\right)} + \frac{1}{p}\right).$$

Finally, as $y - x$ and $z - x$ are nonzero integers of magnitude at most $n < \sqrt{p/2}$, we can apply Lemma 4.28 to replace $H_p$ by $H$. The remaining sum can be bounded using Lemma 4.29:

$$\sum_{y,z \in X} \frac{1}{H_p\left(\frac{\bar{y}-\bar{x}}{\bar{z}-\bar{x}}\right)} = \sum_{y,z \in X} \frac{1}{H(\frac{y-x}{z-x})} = O(n \log n),$$

and the claim follows. (The $+\frac{1}{p}$ term can be omitted assuming that $p = \Omega(n^2)$.) ◀

**Proof of Corollary 4.27.** ▶ Fix buckets $a, b \in [m]$, and let $F$ denote the number of keys in $X$ hashed to $b$. By the pairwise independence of $h$ (see Lemma 2.7), we have that

$$\mathbf{E}(F \mid h(x) = a) = \mathbf{E}(F) = \frac{t}{m} \pm \Theta\left(\frac{t}{p}\right) = \frac{t}{m} \pm \Theta(1).$$

In particular, since $p > m^2$ it holds that $\mathbf{E}(F) \geq t/m - O(t/p) \geq \Omega(t/m)$. By Theorem 4.3, we additionally have

$$\mathbf{E}(F^2 \mid h(x) = a) = \frac{t^2}{m^2} + O\left(\frac{n \log n}{m}\right).$$

It follows that

$$\mathbf{Var}(F \mid h(x) = a) = O\left(\frac{n \log n}{m}\right),$$

and finally, by an application of Chebyshev's inequality we have

$$\mathbf{P}(|F - \mathbf{E}(F)| \geq \lambda \sqrt{\mathbf{E}(F)} \mid h(x) = a) \leq \frac{\mathbf{Var}(F \mid h(x) = a)}{\lambda^2 \, \mathbf{E}(F)} = O\left(\frac{n \log n}{\lambda^2 t}\right),$$

for all $\lambda > 0$. ◀

### 4.8.3 An Almost-Matching Lower Bound for Theorem 4.1

In this section we prove the following statement which shows that Theorem 4.1 (Theorem 5 in [139]) is almost optimal in the case where $n$ is polynomial in $t$.

**Theorem 4.2 ([139, Theorem 5] is Almost Optimal).** *Let $t$ and $n$ be arbitrary parameters with $n \geq t^{1+\epsilon}$ for some constant $\epsilon > 0$, and let $h$ be a random linear hash function which hashes to $m$ buckets. Then there exists a set $X \subseteq [n]$ of $t$ keys, a fixed key $x \notin X$ and buckets $a, b \in [m]$ such that for uniformly random $y, z \in X$ we have*

$$\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \geq \frac{1}{mt} \exp\left(\Omega\left(\sqrt{\min\left(\frac{\log t}{\log \log t}, \frac{\log n}{\log^2 \log n}\right)}\right)\right).$$

We first describe the construction of $X$. By the Prime Number Theorem, for any $N \in \mathbf{N}$, there are $N$ primes $p_0, \ldots, p_{N-1}$ in the range $[N \log N \,..\, CN \log N)$ for some absolute constant $C > 1$. Let

$$N = \min\left(\frac{\epsilon \log n}{(1+\epsilon) C \log \log n}, \log t - 1\right),$$

and define

$$X' = \left\{ s \prod_{i \in I} p_i : I \subseteq [N], \, |I| = \frac{N}{2}, \, 1 \leq s \leq S \right\},$$

where $1 \leq S \leq t$ is chosen in such a way that $t/2 \leq |X'| \leq t$. There exists indeed such a value of $S$, as we can repeatedly increment $S$ (starting from $S \leftarrow 1$) until the

condition is satisfied; in each step the set grows by at most $2^N \le t/2$ elements. We then construct $X \supseteq X'$ by adding arbitrary (small) elements to $X$ until $|X| = t$. One can check that $X \subseteq [n]$ as the largest number in $X'$ has magnitude less than

$$S(CN\log N)^N \le t(\log n)^{\frac{\epsilon \log n}{(1+\epsilon)\log\log n}} = tn^{\frac{\epsilon}{1+\epsilon}} \le n^{\frac{1}{1+\epsilon}} n^{\frac{\epsilon}{1+\epsilon}} = n.$$

The first step towards proving that $X$ is an extreme instance is to give the following lower bound:

**Lemma 4.32.** *It holds that*

$$\sum_{x,y\in X} \frac{1}{H(\frac{x}{y})} = t\exp\left(\Omega\left(\sqrt{\min\left(\frac{\log t}{\log\log t}, \frac{\log n}{\log^2\log n}\right)}\right)\right).$$

**Proof.** ▸ We only need a lower bound, so we will ignore all elements in $X \setminus X'$. Fix any element $x = s\prod_{i\in I} p_i \in X'$; we prove a lower bound against $\sum_{y\in X'} 1/H(\frac{x}{y})$. We call an element $y$ *good* if it has the form $y = s\prod_{i\in J} p_i$, where both $x$ and $y$ have the same factor $s$ and the symmetric difference of $I$ and $J$ has size exactly $2r$ (i.e., $|I \setminus J| = |J \setminus I| = r$) for some parameter $r$ to be specified soon. In the fraction $\frac{x}{y}$ only the factors $s$ and $\prod_{i\in I\cap J} p_i$ cancel and therefore $H(\frac{x}{y}) = \Theta(N\log N)^r$. Moreover, the number of good elements $y$ is exactly $\binom{N/2}{r}^2$, and thus

$$\sum_{y\in X'} \frac{1}{H(\frac{x}{y})} \ge \frac{\binom{N/2}{r}^2}{O(N\log N)^r} \ge \frac{(\frac{N}{2r})^{2r}}{O(N\log N)^r} = \Omega\left(\frac{N}{r^2\log N}\right)^r.$$

Choosing $r = \Theta(\sqrt{N/\log N})$ yields

$$\sum_{y\in X'} \frac{1}{H(\frac{x}{y})} \ge \exp(\Omega(r)) = \exp\left(\Omega\left(\sqrt{\min\left(\frac{\log t}{\log\log t}, \frac{\log n}{\log^2\log n}\right)}\right)\right).$$

The claim follows by summing over all $x \in X'$. ◂

**Proof of Theorem 4.2.** ▸ Let $X$ be as before, choose $x = 0$ and choose $a = b = 0$. For now we also fix $y, z \in X$ but we will later in the proof unfix $y, z$ and treat them as random variables. The first steps are quite similar to the proof of Theorem 4.3. Let $I = \{i \in [p] : i \bmod m = 0\}$. Then $\bar{I}$ is an arithmetic progression in $\mathbf{F}_p$. Sample a random linear function on $\mathbf{F}_p$, $\bar{h}(\bar{x}) = \bar{\sigma}\bar{x} + \bar{\tau}$. Then:

$$\begin{aligned}
&\mathbf{P}(h(y) = h(z) = b \mid h(x) = a)\\
&\quad = \mathbf{P}((\bar{h}(\bar{y}), \bar{h}(\bar{z})) \in \bar{I}^2 \mid \bar{h}(\bar{x}) \in \bar{I})\\
&\quad = \frac{1}{|I|}\sum_{\bar{u}\in\bar{I}} \mathbf{P}((\bar{h}(y), \bar{h}(\bar{z})) \in \bar{I}^2 \mid \bar{h}(\bar{x}) = \bar{u}).
\end{aligned}$$

We continue to bound every term in the sum from below, so fix some value $\bar{u} \in \bar{I}$. As $\bar{h}(\bar{x}) = \bar{\tau}$ and $\bar{h}(\bar{y}) - \bar{h}(\bar{x}) = \bar{\sigma}\bar{y}$ are independent, we can omit the condition and it suffices to bound

$$\begin{aligned}
&\mathbf{P}((\bar{h}(\bar{y}), \bar{h}(\bar{z})) \in \bar{I}^2 \mid \bar{h}(\bar{x}) = \bar{u})\\
&\quad = \mathbf{P}((\bar{\sigma}\bar{y}, \bar{\sigma}\bar{z}) \in (\bar{I} - \bar{u})^2)
\end{aligned}$$

Let $T = \lfloor \frac{p}{2m} \rfloor$ and observe that either $\bar{J} = \{\bar{i}\bar{m} : i \in [T]\}$ or $\{-\bar{i}\bar{m} : i \in [T]\}$ is contained in $\bar{I} - \bar{u}$. In both cases we may replace $\bar{I} - \bar{u}$ by $\bar{J}$ (in the latter case we replace also $\bar{\sigma}$ by $-\bar{\sigma}$ which does not change the probability):

$$\begin{aligned}
&\geq \mathbf{P}((\bar{\sigma}\bar{y}, \bar{\sigma}\bar{z}) \in \bar{J}^2) \\
&= \frac{|\bar{y}^{-1}\bar{J} \cap \bar{z}^{-1}\bar{J}|}{p} \\
&= \frac{1}{p} \sum_{i,j \in [T]} \left[\bar{i}\bar{y}^{-1}\bar{m} = \bar{j}\bar{z}^{-1}\bar{m}\right] \\
&= \frac{1}{p} \sum_{i,j \in [T]} \left[\bar{i}\bar{j}^{-1} = \bar{y}\bar{z}^{-1}\right]
\end{aligned}$$

We claim that there are at least $\Omega(T/H(\frac{y}{z}))$ solutions $i, j \in [T]$ to the modular equation $\bar{i}\bar{j}^{-1} = \bar{y}\bar{z}^{-1}$. Indeed, consider the reduced fraction $\frac{y'}{z'} = \frac{y}{z}$ (i.e., $y'$ and $z'$ are coprime and $0 < y', z' \leq H(\frac{y}{z})$ by definition). For any $t < T/H(\frac{y}{z})$ we may pick $i = ty'$ and $j = tz'$. On the one hand, we have that $i$ and $j$ have the correct size since $i = ty' < (T/H(\frac{y}{z})) \cdot H(\frac{y}{z}) = T$. On the other hand, the equation is satisfied by $\bar{i}\bar{j}^{-1} = \bar{t}\bar{y}'\bar{t}^{-1}\bar{z}'^{-1} = \bar{y}\bar{z}^{-1}$. Hence:

$$\begin{aligned}
&\geq \Omega\left(\frac{T}{pH(\frac{y}{z})}\right) \\
&= \Omega\left(\frac{1}{mH(\frac{y}{z})}\right).
\end{aligned}$$

We now unfix $y, z$ and consider them as random variables. By averaging over the previous inequality and by applying Lemma 4.32 we finally obtain:

$$\begin{aligned}
&\mathbf{P}(h(y) = h(z) = b \mid h(x) = a) \\
&\quad \geq \Omega\left(\frac{1}{mt^2} \sum_{y,z \in X} \frac{1}{H(\frac{y}{z})}\right) \\
&\quad \geq \frac{1}{mt} \exp\left(\Omega\left(\sqrt{\min\left(\tfrac{\log t}{\log\log t}, \tfrac{\log n}{\log^2 \log n}\right)}\right)\right).
\end{aligned}$$

◂

# 5 Fine-Grained Complexity of Approximate Distance Oracles

This chapter is devoted to proving fine-grained lower bounds against distance oracles in graphs that are ultimately based on sparse convolution algorithms. The work presented in this chapter is based on the paper [2]. At the same time, Jin and Xu [130] have independently discovered the same main result. (Our papers differ in their focus points: While we focus in depth on the hardness of distance oracles in several regimes, Jin and Xu have more broadly focused on other fine-grained results such as the hardness of 4-variate linear degeneracy testing.)

2 Amir Abboud, Karl Bringmann, and Nick Fischer. "Stronger 3-SUM lower bounds for approximate distance oracles via additive combinatorics". In: *55th annual ACM symposium on theory of computing (STOC 2023)*. To appear. ACM, 2023. 10.48550/arXiv.2211.07058.

**Acknowledgements.** We would like to thank Merav Parter and Sebastian Forster for helpful discussions on the distance oracle upper bounds. We would also like to thank Seri Khoury and Or Zamir for collaboration on short cycle removal in graphs, and Marvin Künnemann and Karol Węgrzycki for collaboration on another related project, both of which inspired us to work on short cycle removal on numbers.

**Organization.** We start with a technical overview of our results in Section 5.1, explaining in detail how distance oracles connect to 3-SUM. We then give some necessary background on additive combinatorics in Section 5.2 and successively develop our reduction in Sections 5.3 to 5.7.

## 5.1 Overview

In this section, we give a high-level overview of our results. We start with some quick preliminaries about the 3-SUM problem and graphs.

**3-SUM.** The *monochromatic 3-SUM problem* is to determine whether in a given set $A$, there are $a, b, c \in A$ (not necessarily distinct) such that $a + b + c = 0$. We say that the instance has size $n = |A|$. The *trichromatic 3-SUM problem* is to determine whether in three given sets $A, B, C$, there are $a \in A, b \in B, c \in C$ such that $a + b + c = 0$. We say that the instance has size $n = |A| + |B| + |C|$. Both variants are equivalent in terms of subquadratic algorithms. We typically work under the well-established assumption that 3-SUM requires quadratic time [101].

**Graphs.** In this chapter all graphs are undirected and unweighted. The *distance* $d(u, v)$ of two vertices is the length of the shortest path from $u$ to $v$. We say that a graph is $\Theta(r)$-regular if there are constants $0 < c_1 < c_2$ such that every vertex has degree $\deg(v)$ satisfying $c_1 r \le \deg(v) \le c_2 r$.

### 5.1.1 Hardness Reductions from Triangle Listing Instances with Few Short Cycles

Our motivation is the observation that, if we could assume hardness of triangle listing in *random-like* graphs, we could rather easily conclude tight hardness of 4-cycle listing. Under the same assumption and with some more work, we can also show the promised hardness of distance oracles. More specifically, assume that it is $n^{2-o(1)}$-hard to list $O(n^{3/2})$ triangles in a $\Theta(n^{1/2})$-regular graph which contains at most $O(n^2)$ 4-cycles—this is indeed the number of 4-cycles we expect in a random $\Theta(n^{1/2})$-regular graph.

**Hardness of Listing 4-Cycles.** It is easy to conclude the fine-grained hardness of listing all 4-cycles in a given graph

**Theorem 1.12 (Hardness of Listing 4-Cycles).** *For any $\epsilon > 0$, there is no algorithm listing all 4-cycles in time $\widetilde{O}(n^{2-\epsilon} + t)$ or in time $\widetilde{O}(m^{4/3-\epsilon} + t)$ (where $t$ is the number of 4-cycles), unless the 3-SUM conjecture fails.*

First, by a simple subsampling trick we can reduce the number of 4-cycles in the given triangle instance a tiny bit further: We randomly split the vertex set into $n^\delta$ many groups and list all triangles in each triple of groups. In this way we incur an overhead of $n^{3\delta}$ to the running time. However, we have reduced the total number of 4-cycles (across all triples of groups) to $O(n^{2-\delta})$. Indeed, each 4-cycle falls into a fixed triple of groups only with probability $n^{-4\delta}$, and thus the total number of 4-cycles is $n^{3\delta} \cdot O(n^{2-4\delta}) = O(n^{2-\delta})$.

We follow a natural approach on the smaller instances $G = (V, E)$ (i.e., for each triple of groups): We create a new graph consisting of four copies $V_1, V_2, V_3, V_4$ of $V$ (i.e., each vertex $v \in V$ now has four copies $v_1, v_2, v_3, v_4$). We add all edges from $E$ between the parts $V_1$ and $V_2$, between $V_2$ and $V_3$ and between $V_3$ and $V_4$. Finally, we connect all matching vertices in $V_1$ and $V_4$ (i.e., for all $v \in V$ we add the edge $(v_1, v_4)$).

With this construction, each triangle $(u, v, w)$ in the original graph can now be found as a 4-cycle $(u_1, v_2, w_3, u_4)$. However: There might be many more 4-cycles which do not correspond to triangles in the original instance, e.g., 4-cycles which only zigzag between the vertex parts $V_1$ and $V_2$. These 4-cycles must be part of the original graph though, and thus the total number of 4-cycles in the instance is bounded by $O(n^{2-4\delta})$. It follows that if there is an algorithm listing all $t$ 4-cycles in a graph in time $O(n^{2-\epsilon} + t)$, then we could list all triangles in time $O(n^{2-\epsilon} + n^{2-4\delta})$. The total time across all triples of groups is bounded by $O(n^{2-\epsilon+3\delta} + n^{2-\delta})$, which is subquadratic by setting $\delta > 0$ small enough.

**Hardness of Approximate Distance Oracles.** In a similar spirit we derive hardness results for distance oracles. We achieve results for several settings (see Theorems 1.9 to 1.11), but in this overview we will only focus on the simplest version to get the idea across. We demonstrate how to rule out distance oracles with stretch $k$, constant query time and preprocessing time $O(m^{1+\frac{1}{2k+1}-\epsilon})$. (This is a weaker bound than in Theorem 1.9, where we even rule distance oracles with preprocessing time $O(m^{1+\frac{1}{k}-\epsilon})$).

We again start from an instance $G$ of listing $O(n^{3/2})$ in a $\Theta(n^{1/2})$-regular $n$-vertex graph, and assume that the graph contains at most $O(n^{k/2})$ $k$-cycles, for all $k$. Without loss of generality assume that the instance is a tripartite graph $G = (X, Y, Z, E)$ with vertex parts $X, Y, Z$. We will uniformly subsample all vertex parts with sampling rate $\rho$ (which we determine later) to obtain a smaller graph $G'$ with vertices $X' \subseteq X, Y' \subseteq Y, Z' \subseteq Z$. This graph is $\Theta(\rho n^{1/2})$-regular, has $O(\rho n)$ vertices and has $O(\rho^2 n^{3/2})$ edges. Most interestingly though, the number of $k$-cycles in $G'$ is at most $O(\rho^k n^{k/2})$, as every $k$-cycle survives the subsampling only with probability $\rho^k$.

We will now use the distance oracle to efficiently list all triangles in $G'$. To this end, let $G''$ be a duplicate of $G'$ where we delete the edges between $X'$ and $Z'$; any pair of vertices $(x, z) \in X' \times Z'$ which was part of a triangle has distance $d(x, z) \leq 2$ in $G''$. We preprocess $G''$ with the distance oracle and query each pair $(x, z) \in (X' \times Z') \cap E$ to get distance estimates $d(x, z) \leq \widetilde{d}(x, z) \leq k \cdot d(x, z)$. We say that a pair $(x, z)$ is a *candidate* if its distance estimate is $\widetilde{d}(x, z) \leq 2k$. The idea is that only the candidate pairs can possibly be part of a triangle—as all other pairs must have distance more than 2 in $G''$. However, note that among the candidate pairs there may be many pairs which do not form a triangle. Our listing algorithm now enumerates all candidate pairs $(x, z)$ and all neighbors $y \in Y'$ of $x$ and tests whether $(x, y, z)$ forms a triangle. It should be clear that the algorithm cannot miss any triangle in $G'$. And by repeating the subsampling $\widetilde{O}(\rho^{-3})$ times,

with good probability every triangle in $G$ occurs in at least one instance $G'$ and will therefore eventually be detected.

The running time is dominated by two major contributions: The preprocessing time of the distance oracle and the enumeration step (for this setting of parameters the query time can be ignored). The total preprocessing time across all $\widetilde{O}(\rho^{-3})$ repetitions is bounded by

$$\widetilde{O}(\rho^{-3} \cdot (\rho^2 n^{3/2})^{1+\frac{1}{2k+1}-\epsilon}). \tag{5}$$

Next we deal with the contribution of the enumeration step. The key in the analysis is to get a good bound on the number of candidate pairs $(x, z)$. Observe that as any candidate pair $(x, z)$ has distance $d(x, z) \leq 2k$ in $G''$, it must be part of a cycle of length at most $2k + 1$ in $G'$. We can thus control the number of candidate pairs by controlling the number of cycles in $G'$—as argued before, there are at most $O(\rho^{2k+1} n^{\frac{2k+1}{2}})$ cycles of length at most $2k + 1$. Dealing with a single candidate pair takes time $O(\rho n^{1/2})$ (to list all neighbors $y$ of $x$), and therefore the total running time of the enumeration step is bounded by

$$O(\rho^{-3} \cdot \rho^{2k+1} n^{\frac{2k+1}{2}} \cdot \rho n^{1/2}) = O(\rho^{2k-1} n^{k+1}) \tag{6}$$

By optimizing $\rho$ in Equations (5) and (6), we find that the running is indeed subquadratic (for $\rho = n^{-\frac{k-1}{2k-1}-\delta}$ and some tiny $\delta > 0$). This completes the proof outline of the weaker lower bound. For the improved lower bound from Theorem 1.9, we find better trade-off between the size of the preprocessed graph and the number of queries to the distance oracle.

**Revisiting Hardness of Listing Triangles.** The main message is that if miraculously the given triangle instance contains few 4-cycles, then we would obtain interesting hardness results. We therefore investigated whether this variant of triangle listing is conditionally hard, and managed to prove the desired result:

**Theorem 5.1 (Hardness of Triangle Listing).** *For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$-time algorithm listing all triangles in a $\Theta(n^{1/2})$-regular $n$-vertex graph which contains at most $O(n^{k/2})$ $k$-cycles for all $k \geq 3$, unless the 3-SUM conjecture fails.*

There are known lower bounds against listing $O(n^{3/2})$ triangles in $\Theta(n^{1/2})$-regular graphs (without the assumption that the graph has few short cycles) under the 3-SUM conjecture by Pătraşcu [172] with refinements by Kopelowitz, Pettie and Porat [143] and under the All-Pairs Shortest Paths conjecture by Vassilevska Williams and Xu [204]. We specifically focused on the 3-SUM hardness and as a first step significantly simplified the known reduction (see Section 5.5). We then raised the question: In this reduction from 3-SUM to triangle listing, *what makes the constructed triangle instance have many 4-cycles?*

It turns out that the number of 4-cycles in the triangle instance is controlled by the number of solutions to the equation $a_1 + a_2 = a_3 + a_4$, where $a_1, a_2, a_3, a_4 \in A$, in the 3-SUM instance $A$. In the additive combinatorics literature this quantity is commonly referred to as the *additive energy* $E(A)$ of $A$. Note that $E(A)$ ranges from $n^2$ (as there are at least $n^2$ trivial solutions with $a_1 = a_3$ and $a_2 = a_4$) to $n^3$ (as any fixed values $a_1, a_2, a_3$ uniquely determine $a_4$). A set with additive energy close to $n^2$ is considered *unstructured*—for instance a random set has expected energy $O(n^2)$. A set with energy close to $n^3$ is considered structured—examples include intervals and arithmetic progressions.

In summary: To obtain a triangle listing instance containing few 4-cycles, we have to start from a 3-SUM instance with very small additive energy $E(A)$ (in Section 5.5 we prove this statement in detail).

### 5.1.2 Energy Reduction for 3-SUM

We manage to show a self-reduction for 3-SUM which reduces the energy down to $O(n^2)$. We will refer to this type of reduction as an *energy reduction* for 3-SUM. Our outline for the energy reduction is as follows: First, we reduce the additive energy by a tiny bit, say to $n^{2.9999}$, using several tools from additive combinatorics. Second, we apply a randomized 3-SUM self-reduction (which can be seen as an efficient way of *subsampling* the instance) to amplify the tiny improvement to an arbitrarily large improvement. We will now describe both steps in more detail.

**First Step: Energy Reduction via Additive Combinatorics.** The precise result we obtain in this step is as follows. Here, and in fact throughout the whole paper, we will set $K = n^{0.0001}$. Moreover, throughout let the group $G$ be either $G = \mathbf{Z}$ or $G = \mathbf{F}_p^d$.

**Lemma 5.2 (Energy Reduction via Additive Combinatorics).** *Let $K \geq 1$. There is a fine-grained reduction from a 3-SUM instance $A$ of size $n$ to an equivalent 3-SUM instance $A^* \subseteq A$, where $E(A^*) \leq |A^*|^3/K$. The reduction runs in time $\widetilde{O}(K^{314}n^{7/4})$.*

A key ingredient for this step is the seminal Balog-Szemerédi-Gowers theorem (in short: the BSG theorem). Intuitively, the theorem states that every set $A$ with large additive energy $E(A)$ must contain a large subset $A'$ which behaves like an interval or an arithmetic progression in the sense that its sumset $A+A = \{a_1+a_2 : a_1, a_2 \in A\}$ has very small size (we also say that $A$ has small *doubling*). The theorem can be formally stated as follows:

**Theorem 5.3 (Balog-Szemerédi-Gowers).** *Let $A \subseteq G$. If $E(A) \geq |A|^3/K$, then there is a subset $A' \subseteq A$ such that*

- $|A'| \geq \Omega(K^{-2}|A|)$, *and*
- $|A' + A'| \leq O(K^{24}|A|)$.

*Moreover, we can compute $A'$ in time $\widetilde{O}(K^{12}|A|)$ by a randomized algorithm.*

The existential part of the theorem (without the claimed running time bounds) was originally proved by Balog and Szemerédi [30] and Gowers [114]. The efficient algorithm to compute $A'$ was later devised by Chan and Lewenstein [71] based on a proof of the BSG theorem which was independently discovered by Balog [29] and Sudakov, Szemerédi and Vu [195].

The BSG theorem suggests the following algorithmic idea: As long as $A$ has large additive energy, apply the BSG theorem to extract a highly structured subset $A' \subseteq A$, and efficiently solve 3-SUM on that set. More specifically, we have to solve the trichromatic 3-SUM instance $(A', A, A)$, where we can assume that $|A' + A'| \leq O(K^{24}|A|) \leq O(K^{26}|A'|)$. Indeed, either there exists a solution contained in $A \setminus A'$ in which case we can simply discard $A'$, or part of the 3-SUM solution is contained in $A'$ in which case this will be a valid solution in $(A', A, A)$. One can prove that after at most $\widetilde{O}(K)$ extractions, we have either found a 3-SUM solution or the remaining set has small additive energy as required.

It remains to solve the 3-SUM instances $(A', A, A)$. There are some known results about structured 3-SUM instances: For instance, using sparse convolution algorithms we can solve 3-SUM instances $(A, B, C)$ in subquadratic time whenever $A + B$ has subquadratic size. Another result by Chan and Lewenstein [71] is that 3-SUM admits subquadratic-time algorithms whenever one of the sets is *clustered*, that is, if it can be covered by a subquadratic number of size-$n$ intervals. Unfortunately, neither of these algorithms can be applied in our context and to the best of our knowledge no algorithm is known for the case when one of the input sets has small doubling. It is one of our key technical contributions to design an algorithm for this problem:

**Theorem 5.4 (3-SUM for Structured Inputs).** *Let $(A, B, C)$ be a 3-SUM instance of size $n$ with $A, B, C \subseteq G$ and $|A + A| \leq K|A|$. Then we can solve $(A, B, C)$ in time $\widetilde{O}(K^{12}n^{7/4})$.*

We omit the description of this algorithm for now and continue with the energy reduction. Later in the overview, in Section 5.1.3, we give the main ideas and in Section 5.3 we provide the detailed proof of Theorem 5.4.

**Second Step: Amplification via Hashing.** In the previous step we have reduced a worst-case 3-SUM instance to another instance with a tiny improvement in additive energy. In this step, we will amplify this improvement by means of the following reduction:

**Lemma 5.5 (Energy Reduction via Hashing).** *Let $K \geq 1$. There is a fine-grained reduction from a 3-SUM instance $A$ with $E(A) \leq |A|^3/K$ to $g = O(|A|^2/K^2)$ 3-SUM instances $A_1, \ldots, A_g$ of size $O(K)$ and with expected energy $\mathbf{E}(E(A_i)) \leq O(K^2)$. The reduction runs in time $\widetilde{O}(|A|^2/K)$.*

The rough idea behind Lemma 5.5 is to create many randomly subsampled instances from $A$. An efficient way to implement such a self-reduction is to not subsample $A$ uniformly, but instead make use of linear hashing. This general idea is not new and has appeared several times before in the context of 3-SUM [32, 172, 143, 70], but we have to pay closer attention than usual in order to analyze the additive energy.

We describe the simplified idea, glimpsing over several problems: Sample a linear hash $h$ to $m$ buckets and create the instance $B = \{a \in A^* : h(a) = 0\}$. Here, *linear* means that $h$ satisfies the condition $h(a) + h(b) = h(a + b)$ for all inputs $a, b$. What is the probability that a fixed 3-SUM solution $a + b + c = 0$ survives? The probability is at least $1/m^2$, since $1/m^2$ is the probability that $h(a) = h(b) = 0$, which entails that also $h(c) = 0$ by the linearity of the hash function. This means that we have to repeat this reduction $\widetilde{\Omega}(m^2)$ times until a 3-SUM solution survives.

In contrast, what is the probability that a 4-tuple satisfying $a_1 + a_2 = a_3 + a_4$ survives? By the same linearity argument, we can only use the randomness for three of the four variables as the hash value of the remaining variable is fixed. We therefore expect each solution to survive with probability $1/m^3$. Since this probability is smaller by a factor $m$ compared to the survival probability of a 3-SUM solution, only a $1/m$-fraction of solutions $a_1 + a_2 = a_3 + a_4$ survives and appears in one the small instances. In particular, by setting $m = n/K$ we create $n^2/K^2$ instances of size $n/m = K$ and with additive energy bounded by $E(A)/m^3 \leq (n/m)^3/K = K^2$.

However, there is a serious issue with this approach: In order to argue that each solution to the equation $a_1 + a_2 = a_3 + a_4$ survives with probability at most $1/m^3$ we have assumed that three elements, say, $a_1$, $a_2$ and $a_3$, are hashed *independently*. Unfortunately there are no hash functions which are linear and 3-wise independent at the same time. We will ignore this issue for now, and explain later in Section 5.1.4 how to overcome this challenge.

By combining both steps of the energy reduction, we obtain the following theorem:

**Theorem 5.6 (Energy Reduction).** *For any $\epsilon, \delta > 0$, there is no $O(n^{2-\epsilon})$-time algorithm solving the 3-SUM problem on instances $A$ with size $n$ and additive energy $E(A) \leq O(|A|^{2+\delta})$, unless the 3-SUM conjecture fails.*

**Comparison to Abboud, Bringmann, Khoury and Zamir [4].** We remark that our approach for an energy reduction is conceptually similar to the work of Abboud, Bringmann, Khoury and Zamir [4]: Their goal was also to reduce the number of 4-cycles in a triangle instance. They achieved this by first reducing the number of 4-cycles by a little bit (by identifying and removing dense pieces in the graph, which contain many 4-cycles), and then subsample the remaining instance to amplify the 4-cycle reduction. In contrast to our setting, working on the triangle in-

stances directly has the disadvantage that sparse triangle problems are not known to admit efficient self-reductions. As a result, their subsampling step is lossy and leads to non-matching lower bounds.

This completes the description of the energy reduction. In the following subsections we describe what we left out in the previous overview—how to efficiently solve 3-SUM for structured inputs and how to deal with the hashing issue.

### 5.1.3 3-SUM for Structured Inputs

In this section we describe a subquadratic-time algorithm for 3-SUM instances $(A, B, C)$ in which the set $A$ has doubling $|A+A| \leq K|A|$. We first describe a simple toy algorithm to build some intuition.

**Warm-Up: $A$ Is Contained in an Interval.** We give a simple algorithm that works whenever $A$ is contained in a small interval, say $I = [10n]$ (this is indeed an example of a set with small doubling). Our approach is to *cover* $B$ and $C$ by translates of $I$. That is, we split $B$ into a collection of disjoint subsets $B_1, \dots, B_\ell$ each of which is obtained by intersecting $B$ with a translate of $I$. Note that we need at most $|B|$ translates to cover the full set $B$. We similarly cover $C$ by disjoint subsets $C_1, \dots, C_m$. The insight is that $A + B_i$ is contained in an interval of size $20n$. Therefore if there is a 3-SUM solution $(a, b, c) \in A \times B \times C$ with $b \in B_i$, there are at most three sets $C_j$ which could possibly contain $c$. Calling a pair $(i, j)$ *relevant* if there could possibly be a 3-SUM solution in $A \times B_i \times C_j$, we have argued that the number of relevant pairs is at most $O(n)$.

We iterate over all relevant pairs $(i, j)$, and use a heavy-light approach: If both sets $B_i$ and $C_j$ have size at most $n^{1/3}$, then we brute-force over all $(b, c) \in B_i \times C_j$ and test whether they constitute a 3-SUM solution. Otherwise, we compute $B_i + C_j$ using FFT, and test for each element in the sumset whether it is part of a 3-SUM solution with $A$. The total time of the light case is bounded by $O(n)$ (the number of relevant pairs) times $O(n^{2/3})$ (the number of pairs $(b, c)$ we explicitly test). The total time of the heavy case is bounded by $O(n^{2/3})$ (there can be at most that many relevant pairs $i, j$ for which either $B_i$ or $C_j$ has size larger than $n^{1/3}$) times $\widetilde{O}(n)$ (running FFT on sets of universe size $20n$). The total time is $O(n^{5/3})$, which is subquadratic.

The take-away message is that when we know that $A$ is contained in a small interval, we can benefit from the structure by pruning the search space in $B \times C$ (i.e., we do not compare every element in $B$ to every element in $C$). The question is: *What is the appropriate generalization of an interval?*

**Full Algorithm: $A$ Is Contained in an Approximate Group.** For us, the appropriate generalization are *approximate groups.* A set $H$ is a $K$-approximate group if (i) $H = -H$ and (ii) $H + H$ can be covered by at most $K$ translates of $H$. The key ingredient to our algorithm is yet another result from additive combinatorics: Ruzsa's covering lemma. More specifically, we exploit the following consequence of Ruzsa's covering lemma which states that any set with small doubling can be covered by a small approximate group.

**Lemma 5.7 (Covering by Approximate Groups).** *Let $A \subseteq G$ be a set with $|A + A| \leq K|A|$. Then there is a set $H \subseteq G$ with the following properties:*

- $|H| \leq K^2|A|$,
- *$H$ is a $K^5$-approximate group, that is, there is some set $X \subseteq G$ of size $|X| \leq K^5$ such that $H = -H$ and $H + H \subseteq H + X$, and*
- *there is some $a_0 \in A$ such that $A - a_0 \subseteq H$.*

*Moreover, we can compute $H$, $X$ and $a_0$ in time $\widetilde{O}(K^{12}|A|)$.*

The existential result is well-known in additive combinatorics (see for instance the book by Tao and Vu [196]), but for our purposes it is also important to have an efficient algorithm to compute $H$. We derive an algorithm based on computing sparse convolutions, see the proof in Section 5.2.

For our 3-SUM algorithm, thanks to Lemma 5.7 we can assume that $A$ is contained in (a translate of) a small approximate group $H$. We mimic the warm-up algorithm with the same idea: Cover $B$ and $C$ by translates of $H$, say $B_1, \dots, B_\ell$ and $C_1, \dots, C_m$. For each set $B_i$, there are only few sets $C_j$ which are candidates to contain a 3-SUM solution, namely at most $K^5$ many. Therefore, we can apply a similar heavy-light approach as outlined before, and either enumerate all pairs in $B_i \times C_j$ if both sets are sparse, or efficiently compute $B_i + C_j$ using a sparse sumset algorithm (in place of FFT) if one of the sets is dense.

An additional difficulty is that we cannot simply cover $B$ and $C$ by translates of $H$ in linear time. (For intervals this is easy, but we have no information about $H$ other than that is an approximate group.) We therefore sample a set $S$ of *random* shifts and attempt to cover $B$ by the sets $B \cap (H + s)$ for $s \in S$ (similarly for $C$). However, computing the sets $B \cap (H+s)$ is not easy (in fact, this is again an instance of 3-SUM). We deal with this new obstacle by combining the above algorithm with a universe reduction to a universe of subquadratic size. The detailed proof can be found in Section 5.3.

We remark that we have not attempted to improve the dependence on $K$ as it is immaterial for our reduction. It is likely possible to drastically reduce the $K$ term in the running time of Theorem 5.4.

### 5.1.4 Hashing—Additive and Independent?

A major technical issue that we are facing in the energy reduction (and in fact also in the reduction from 3-SUM to listing triangles) is that we need hash functions which are both *additive* and sufficiently *independent*. More specifically, recall that we hash a given 3-SUM instance $A$ to a smaller instance $B = \{a \in A : h(a) = 0\}$ hoping that thereby the number of solutions to the equation $a_1 + a_2 = a_3 + a_4$ reduces by a factor of $1/m^3$, where $m$ is the number of buckets $h$ hashes to. By the reasons outlined before, the hash function *must* be additive.

Recall from the collection of (almost-)additive hash functions in Section 2.2.1 that there is no family of hash functions satisfying both properties simultaneously—in fact, this it is simply impossible that $h(a_1), h(a_2), h(a_3)$ are always independent for an additive hash function. For example, $h(1)$, $h(2)$ and $h(5)$ cannot be independent, since the latter can be expressed as $h(1 + 2 + 2) = h(1) + h(2) + h(2)$. This is similarly true for almost-additive hash functions such as "linear hashing". As outlined in Section 4.8, Knudsen proved that linear hashing is *almost* 3-wise independent [139] but the independence guarantees are not strong enough for our purposes here.

We propose the following solution: Instead of working over the integers, we instead work over the group $G = \mathbf{F}_p^d$ for some constant (or slightly superconstant) $p$. All the tools from additive combinatorics mentioned before work just as well over $\mathbf{F}_p^d$, we can compute sparse convolutions over $\mathbf{F}_p^d$ (see Theorem 2.24) and also from the perspective of fine-grained complexity, the 3-SUM problem over the integers reduces to the 3-SUM problem over $\mathbf{F}_p^d$:

**Lemma 5.8 (Integer 3-SUM to Vector 3-SUM, [8]).** *For any $\epsilon > 0$, there is some prime $p$ such there is no $O(n^{2-\epsilon})$-time algorithm for 3-SUM over $\mathbf{F}_p^d$ (with $d = O(\log n)$), unless the 3-SUM conjecture fails.*

Working over finite field vector spaces $\mathbf{F}_p^d$ has the advantage that we have access to a nicer family of hash functions: Projections to random subspaces via random linear maps $h : \mathbf{F}_p^d \to \mathbf{F}_p^{d'}$. For this family of hash functions, we can easily characterize the degree of independence: The hash values $h(a_1), h(a_2), h(a_3)$

are independent if and only if $a_1, a_2, a_3$ are linearly independent vectors. Of course the same counterexamples as above still apply, however, the number of bad triples $a_1, a_2, a_3 \in A$ is now very small: For each $a_1 \in A$, there are only $p = O(1)$ vectors $a_2$ which are linearly dependent on $a_1$, and similarly there are only $p^2 = O(1)$ vectors $a_3$ which are linearly dependent on $a_1, a_2$. This kind of reasoning is a recurring theme in several of our proofs (see Lemma 5.5 and Theorem 5.1).

## 5.2 Background on Additive Combinatorics

Additive combinatorics is the theory of additive structure in sets. In this section we summarize the basics from additive combinatorics which are needed throughout the paper. For a more thorough treatment, we refer to the book by Tao and Vu [196]. In contrast to the classical theory, however, we need many of the tools to work as efficient algorithms.

### 5.2.1 Sumsets

Let $G$ be an additive group, and let $A, B \subseteq G$. Recall that the sumset $A+B$ is defined as $\{a+b : a \in A, b \in B\}$. We write $r_{A,B}(x) = \#\{(a,b) \in A \times B : a+b = x\}$ to denote the *multiplicities* in the sumset. In Chapter 2 we have proven that we can compute sumsets $A+B$ along with the multiplicities $r_{A,B}$ in input- plus output-sensitive time over the integers $G = \mathbf{Z}$ (Theorem 2.13) or over finite-field vector spaces $G = \mathbf{F}_p^d$ (Theorem 2.24).

### 5.2.2 Additive Energy

An important definition for us is the *additive energy* $E(A)$, defined as the number of solutions $(a_1, a_2, a_3, a_4) \in A^4$ to the equation $a_1 + a_2 = a_3 + a_4$. An equivalent definition is that $E(A) = \sum_{x \in G} r_{A,A}(x)^2$. Intuitively, the additive energy measures how structured a set $A$ behaves with respect to addition. Consider two extremes: For very structured sets $A$ such as intervals or arithmetic progressions, we expect small additive energy $E(A) \approx |A|^2$, whereas for very unstructured sets such as (pseudo-)random sets or Sidon[36] sets we expect additive energy $E(A) \approx |A|^3$. The following lemma proves these quantitative bounds:

[36] *A Sidon set is a set with maximum possible additive energy, that is, a set without any nontrivial solution to the equation $a_1 + a_2 = a_3 + a_4$.*

**Lemma 5.9 (Basic Bounds for Additive Energy).** *Let $A, B \subseteq G$. Then:*

$$|A|^2 \leq \frac{|A|^4}{|A+A|} \leq E(A) \leq |A|^3.$$

**Proof.** ▶ The first inequality is obvious, and the last inequality is also easy seeing that any choice of $a_1, a_2, a_3$ uniquely determines $a_4$. For the second inequality, note that by the Cauchy-Schwartz inequality we have

$$|A|^2 = \sum_{x \in A+A} r_{A,A}(x) \leq \sqrt{\sum_{x \in A+A} 1} \sqrt{\sum_{x \in A+A} r_{A,A}(x)^2} = \sqrt{|A+A| \cdot E(A)}. \quad \blacktriangleleft$$

From a computational perspective we often need to compute the additive energy. Using the identity $E(A) = \sum_{x \in G} r_{A,A}(x)^2$, and using the efficient algorithms to compute $r_{A,A}(x)$ in Theorems 2.13 and 2.24 we can compute $E(A)$ in time $\widetilde{O}(|A+A|)$. However, for unstructured sets this becomes quadratic in the size of $A$ which is prohibitive in most cases. Therefore, we typically settle for the following approximation algorithm for $E(A)$.

**Lemma 5.10 (Approximating Additive Energy).** *Let $A \subseteq G$. For any constant $\epsilon > 0$, we can compute a $(1+\epsilon)$-approximation of $E(A)$ in time $\widetilde{O}(|A|)$ by a randomized algorithm.*

**Proof.** ▶ Sample $R = 100\epsilon^{-2}|A|\log|A|$ triples $a_1, a_2, a_3 \in A$, and test for each triple whether $a_1 + a_2 - a_3 \in A$. Return as an estimate $|A|^3/R$ times the number of successful tests.

For the analysis, let $X_i$ be the random variable indicating whether the $i$-th test was successful, and let $X = \sum_{i=1}^{R} X_i$. We have that $\mathbf{P}(X_i = 1) = E(A)/|A|^3$, and therefore $\mathbf{E}(X) = R \cdot E(A)/|A|^3$. In other words, our estimator is indeed unbiased. To prove that it returns an accurate estimate with high probability, we apply Chernoff's bound:

$$\mathbf{P}\left(\left|X \cdot \frac{|A|^3}{R} - E(A)\right| \geq \epsilon E(A)\right) = \mathbf{P}\left(\left|X - \frac{R \cdot E(A)}{|A|^3}\right| \geq \epsilon \frac{R \cdot E(A)}{|A|^3}\right)$$
$$\leq 2\exp\left(-\epsilon^2 \frac{R \cdot E(A)}{3|A|^3}\right) \leq 2\exp\left(-\epsilon^2 \frac{100\epsilon^{-2}|A|\log|A| \cdot |A|^2}{3|A|^3}\right) \leq |A|^{-10}. \quad \blacktriangleleft$$

Most of the time we will apply Lemma 5.10 and pretend that the output is perfect without paying too much attention to the approximation error. In all occurrences in this chapter, one can easily replace the bound by, say, a 1.1-approximation and still get the correct algorithms.

### 5.2.3 Fourier Analysis

A useful application of additive energy is that it offers some control over the number of solutions to *any* linear equation (not only $a_1 + a_2 = a_3 + a_4$):

**Lemma 5.11 (Low Energy Means Few Solutions to Linear Equations).** *Let $A \subseteq \mathbf{F}_p^d$. For any $k \geq 4$ and $\alpha_1, \ldots, \alpha_k \in \mathbf{F}_p$ with $\alpha_1, \ldots, \alpha_k \neq 0$ we have that:*

$$\#\{(a_1, \ldots, a_k) \in A^k : \alpha_1 a_1 + \cdots + \alpha_k a_k = \beta\} \leq E(A) \cdot |A|^{k-4}.$$

We remark that this lemma implies the same statement over the integers by setting $d = 1$ and setting $p$ to be a sufficiently large prime $p$. For the proof of this lemma we need some more background on Fourier analysis.

**Definition 5.12 (Fourier Transform).** *For any complex-valued function $f : \mathbf{F}_p^d \to \mathbf{C}$, we define its* Fourier transform $\widehat{f} : \mathbf{F}_p^d \to \mathbf{C}$ *via*

$$\widehat{f}(\xi) = \sum_{x \in G} f(x) \cdot \overline{e(x, \xi)},$$

*where*

$$e(x, \xi) = \exp\left(\frac{2\pi i}{p} \cdot \sum_{i=1}^{d} x_i \cdot \xi_i\right).$$

For a set $A \subseteq \mathbf{F}_p^d$, let $\mathbf{1}_A : \mathbf{F}_p^d \to \{0, 1\}$ denote the indicator function of $A$. We need the following lemma:

**Lemma 5.13 (Counting Solutions to Linear Equations).** *Let $A \subseteq \mathbf{F}_p^d$. For any $k \geq 1$ and $\alpha_1, \ldots, \alpha_k \in \mathbf{F}_p$ we have that:*

$$\#\{(a_1, \ldots, a_k) \in A^k : \alpha_1 a_1 + \cdots + \alpha_k a_k = 0\} = \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \widehat{\mathbf{1}}_A(\alpha_1 \xi) \cdot \ldots \cdot \widehat{\mathbf{1}}_A(\alpha_k \xi).$$

**Proof.** ▶ The proof is a simple calculation. We express the number of solutions $(a_1, \ldots, a_k) \in A^k$ satisfying $\alpha_1 a_1 + \cdots + \alpha_k a_k = 0$ as

$$\sum_{a_1, \ldots, a_k \in \mathbf{F}_p^d} \mathbf{1}_A(a_1) \cdot \ldots \cdot \mathbf{1}_A(a_k) \cdot \mathbf{I}(\alpha_1 a_k + \cdots + \alpha_k a_k = 0),$$

where $\mathbf{I}(\cdot) \in \{0,1\}$ denotes the truth value of the parenthesized expression. We use the following simple claim: $\sum_{\xi \in \mathbf{F}_p^d} e(x,\xi) = p^d \cdot \mathbf{I}(x=0)$, for any fixed $x \in \mathbf{F}_p^d$. Therefore:

$$
\begin{aligned}
&= \sum_{a_1,\ldots,a_k \in \mathbf{F}_p^d} \mathbf{1}_A(a_1) \cdot \ldots \cdot \mathbf{1}_A(a_k) \cdot \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \overline{e(\alpha_1 a_k + \cdots + \alpha_k a_k, \xi)} \\
&= \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \left( \sum_{a_1 \in A} \mathbf{1}_A(a_1) \cdot \overline{e(\alpha_1 a_1, \xi)} \right) \cdot \ldots \cdot \left( \sum_{a_k \in A} \mathbf{1}_A(a_k) \cdot \overline{e(\alpha_k a_k, \xi)} \right) \\
&= \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \left( \sum_{a_1 \in A} \mathbf{1}_A(a_1) \cdot \overline{e(a_1, \alpha_1 \xi)} \right) \cdot \ldots \cdot \left( \sum_{a_k \in A} \mathbf{1}_A(a_k) \cdot \overline{e(a_k, \alpha_k \xi)} \right) \\
&= \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \widehat{\mathbf{1}}_A(\alpha_1 \xi) \cdot \ldots \cdot \widehat{\mathbf{1}}_A(\alpha_k \xi). \qquad \blacktriangleleft
\end{aligned}
$$

**Proof of Lemma 5.11.** ▸ By the previous Lemma 5.13, we can express the additive energy as

$$
\begin{aligned}
E(A) = \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \widehat{\mathbf{1}}_A(\xi) \cdot \widehat{\mathbf{1}}_A(\xi) \cdot \widehat{\mathbf{1}}_A(-\xi) \cdot \widehat{\mathbf{1}}_A(-\xi) \\
= \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \widehat{\mathbf{1}}_A(\xi)^2 \cdot \overline{\widehat{\mathbf{1}}_A(\xi)}^2 = \frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\xi)|^4.
\end{aligned}
$$

By another application of Lemma 5.13, the number of solutions $(a_1, \ldots, a_k) \in A^k$ satisfying $\alpha_1 a_1 + \cdots + \alpha_k a_k = 0$ is

$$
\frac{1}{p^d} \sum_{\xi \in \mathbf{F}_p^d} \widehat{\mathbf{1}}_A(\alpha_1 \xi) \cdot \ldots \cdot \widehat{\mathbf{1}}_A(\alpha_k \xi),
$$

which, by repeated applications of the Cauchy-Schwartz inequality can be upper bounded by

$$
\begin{aligned}
&\leq \frac{1}{p^d} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_1 \xi)|^2 \cdot |\widehat{\mathbf{1}}_A(\alpha_2 \xi)|^2 \right)^{1/2} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_3 \xi)|^2 \cdot \ldots \cdot |\widehat{\mathbf{1}}_A(\alpha_k \xi)|^2 \right)^{1/2} \\
&\leq \frac{|A|^{k-4}}{p^d} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_1 \xi)|^2 \cdot |\widehat{\mathbf{1}}_A(\alpha_2 \xi)|^2 \right)^{1/2} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_3 \xi)|^2 \cdot |\widehat{\mathbf{1}}_A(\alpha_4 \xi)|^2 \right)^{1/2} \\
&\leq \frac{|A|^{k-4}}{p^d} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_1 \xi)|^4 \right)^{1/4} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_2 \xi)|^4 \right)^{1/4} \\
&\qquad \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_3 \xi)|^4 \right)^{1/4} \cdot \left( \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\alpha_4 \xi)|^4 \right)^{1/4}
\end{aligned}
$$

Since $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \neq 0$, this becomes:

$$
\begin{aligned}
&\leq \frac{|A|^{k-4}}{p^d} \cdot \sum_{\xi \in \mathbf{F}_p^d} |\widehat{\mathbf{1}}_A(\xi)|^4 \\
&\leq |A|^{k-4} \cdot E(A),
\end{aligned}
$$

using the previously proved identity for the additive energy $E(A)$. ◂

### 5.2.4 Plünnecke-Ruzsa Inequality

We often rely on the following inequality due to Plünnecke [174] and Ruzsa [186] to control the size of iterated sum- and difference sets (see also [196, Corollary 6.29]). Here, we abbreviate $nB = B + \cdots + B$ with $n$ terms in the sum.

**Lemma 5.14 (Plünnecke-Ruzsa Inequality).** *Let $A, B \subseteq G$. If $|A + B| \leq K|A|$, then $|nB - mB| \leq K^{n+m}|A|$ for all nonnegative integers $n, m$.*

Most of the time we will apply this lemma with $A = B$,in which case the inequality is more commonly known as just *Plünnecke's inequality*.

### 5.2.5 Ruzsa's Covering Lemma

Next, we present Ruzsa's covering lemma [186] and the relevant consequence that sets with small doubling can be covered by small approximate groups. We provide proofs because—even though the existential results are well-known—there has been no work on turning the results into efficient algorithms, to the best of our knowledge. As before, let $G = \mathbf{Z}$ or $G = \mathbf{F}_p^d$ with polylogarithmic $p, d$.

**Lemma 5.15 (Ruzsa's Covering Lemma).** *Let $A, B \subseteq G$. Then there is a subset $X \subseteq B$ with the following properties:*

- $B \subseteq A - A + X$,
- $|X| \leq \frac{|A+B|}{|A|}$.

*Moreover, we can compute $X$ in time $\widetilde{O}(\frac{|A-A+B| \cdot |A+B|}{|A|}) \leq \widetilde{O}(\frac{|A-A+B|^2}{|A|})$.*

**Proof.** ▸ We start with a recap of the well-known existential proof. The proof is already algorithmic: We initialize the set $X \leftarrow \emptyset$. While there exists some $b \in B$ such that $A + b$ is disjoint from $A + X$, add $b$ to $X$.

We prove that after the algorithm has terminated, $X$ is as desired. Indeed, after the algorithm has terminated, the sets $A+b$ and $A+X$ are not disjoint for any $b \in B$. Or equivalently, $b \in A-A+X$. Moreover, note that the size of $A+X$ increases by $|A|$ with every step of the algorithm and that ultimately $|A + X| \leq |A + B|$. It follows that the algorithm runs for at most $\frac{|A+B|}{|A|}$ iterations. Since each iteration adds exactly one element to $X$, we obtain the claimed size bound $|X| \leq \frac{|A+B|}{|A|}$.

While this proof is already algorithmic, it is a priori not clear how to efficiently find $b$. Our approach is as follows: Compute the sets $C \leftarrow (A+B) \setminus (A+X)$ and $C-A$, and additionally compute the multiplicities $r_{C,-A}(x)$ for all $x \in C - A$. We now take any $b \in B$ satisfying $r_{C,-A}(b) = |A|$, and if no such $b$ exists we terminate the algorithm. Recall that $r_{C,-A}(b)$ is equal to the number of witnesses $(c, a) \in C \times A$ with $a + b = c$. There are $|A|$ such witnesses (the maximum number) if and only if $A + b \subseteq C$. By the way we assigned $C$, this in turn is equivalent to the desired condition that $A + b$ is disjoint from $A + X$.

It remains to analyze the running time of this algorithm. Finding a single $b$ amounts to computing the sets $C \subseteq A+B$ and $C-A \subseteq A-A+B$. Using Theorems 2.13 and 2.24 we can compute both sets in output-sensitive time $\widetilde{O}(|A-A+B|)$ along with the multiplicities $r_{C,-A}$. Finally, recall that the algorithm runs for a total of $\frac{|A+B|}{|A|}$ iterations. The claimed time bound follows. ◂

**Lemma 5.7 (Covering by Approximate Groups).** *Let $A \subseteq G$ be a set with $|A + A| \leq K|A|$. Then there is a set $H \subseteq G$ with the following properties:*

- $|H| \leq K^2|A|$,
- *$H$ is a $K^5$-approximate group, that is, there is some set $X \subseteq G$ of size $|X| \leq K^5$ such that $H = -H$ and $H + H \subseteq H + X$, and*
- *there is some $a_0 \in A$ such that $A - a_0 \subseteq H$.*

*Moreover, we can compute $H$, $X$ and $a_0$ in time $\widetilde{O}(K^{12}|A|)$.*

**Proof.** ▶ We first apply Ruzsa's covering lemma with $A$ and $B = 2A-2A$. We thereby obtain a subset $X \subseteq 2A - 2A$ which satisfies that $B \subseteq A - A + X$. By choosing $H = A - A$, we have that $H+H = B \subseteq A-A+X = H+X$. Ruzsa' covering lemma further guarantees that

$$|X| \leq \frac{|A+B|}{|A|} = \frac{|3A-2A|}{|A|} \leq \frac{K^5|A|}{|A|} = K^5,$$

where for the latter inequality we have applied Plünnecke's inequality. Therefore, $H$ satisfies the second property. The first property is easy by another application of Plünnecke's inequality. For the third property take an arbitrary $a_0 \in A$. Then by definition $A - a_0 \subseteq A - A = H$.

The running time to compute $H$ and $X$ is dominated by the call to Ruzsa's covering lemma, which runs in time

$$O\left(\frac{|A-A+B|^2}{|A|}\right) = O\left(\frac{|3A-3A|^2}{|A|}\right) \leq O(K^{12}|A|),$$

where have again used Plünnecke's inequality. ◀

### 5.2.6 Balog-Szemerédi-Gowers Theorem

In this subsection we summarize two important results in additive combinatorics which are crucial ingredients to our algorithms. First, we recall the BSG theorem:

**Theorem 5.3 (Balog-Szemerédi-Gowers).** *Let $A \subseteq G$. If $E(A) \geq |A|^3/K$, then there is a subset $A' \subseteq A$ such that*

- $|A'| \geq \Omega(K^{-2}|A|)$, *and*
- $|A' + A'| \leq O(K^{24}|A|)$.

*Moreover, we can compute $A'$ in time $\widetilde{O}(K^{12}|A|)$ by a randomized algorithm.*

For an existential proof see for instance [29, Theorem 5]. An efficient algorithm was later devised by Chan and Lewenstein [71], however, they designed their algorithm for the following two-set version of the theorem:

**Theorem 5.16 (Theorem 2.1 and Lemma 7.2 in [71]).** *Let $A, B \subseteq G$ and $E \subseteq A \times B$. Suppose that $|A| \cdot |B| = \Theta(n)^2$, $|\{a+b \mid (a,b) \in E\}| \leq tn$, and $|E| \geq \alpha n^2$. Then there exist subsets $A' \subseteq A$ and $B' \subseteq B$ such that:*

- $|A' + B'| \leq O((1/\alpha)^5 t^3 n)$ *and*
- $|E \cap (A' \times B')| \geq \Omega(\alpha|A'||B|) \geq \Omega(\alpha^2 n^2)$.

*Given $A, B$ and query access to $E$, such sets $A', B'$ can be computed by a randomized algorithm in time $\widetilde{O}((1/\alpha)^6(|A|+|B|))$.*

**Proof of Theorem 5.3.** ▶ We quickly prove how to derive our version of the BSG from Chan and Lewenstein's algorithm. Let $C \subseteq A + A$ be the subset containing all elements $x$ with multiplicity $r_{A,A}(x) \geq \frac{|A|}{2K}$. We claim that $|C|$ has size at least $\frac{|A|}{2K}$ as otherwise we would have

$$E(A) = \sum_{x \in G} r_{A,A}(x)^2 = \sum_{\substack{x \in G \\ r_{A,A}(x) \leq |A|/2K}} r_{A,A}(x)^2 + \sum_{\substack{x \in G \\ r_{A,A}(x) > |A|/2K}} r_{A,A}(x)^2$$
$$< \frac{|A|}{2K} \cdot \sum_{\substack{x \in G \\ r_{A,A}(x) \leq |A|/2K}} r_{A,A}(x) + \frac{|A|}{2K} \cdot |A|^2 = \frac{|A|^3}{K}.$$

Let $C_0 \subseteq C$ be an arbitrary subset of size exactly $\frac{|A|}{2K}$. We will apply Theorem 5.16 with the bipartite graph with vertex parts $A$ and $B = A$ and edges

$$E = \{(a,b) \in A^2 : a + b \in C_0\}.$$

Since $|C_0| = \frac{|A|}{2K}$ and since each element in $C_0$ contributes at least $\frac{|A|}{2K}$ edges to the graph, we conclude that $|E| \geq \frac{|A|^2}{4K^2}$. We can therefore apply Theorem 5.16 with parameters $n = |A| = |B|$, $\alpha = \frac{1}{4K^2}$ and $t = 1$. In this way we obtain $A', B' \subseteq A$, and we claim that the set $A'$ is as desired.

We first check that $A'$ and $B'$ are sufficiently large. Theorem 5.16 immediately implies that $|A'| \cdot |B'| \geq \Omega(\alpha|A'|n) \geq \Omega(\alpha^2 n^2)$. In particular, it follows that $|A'| \geq \Omega(\alpha n) = \Omega(K^{-2}n)$ and $|B'| \geq \Omega(\alpha n) \geq \Omega(K^{-2}n)$.

To see that the sumset $|A' + A'|$ is sufficiently small, we first note that the theorem implies that $|A' + B'| \leq O(\alpha^{-5}t^3 n) = O(K^{10}n) = O(K^{12}|B'|)$. We apply the Plünnecke-Ruzsa inequality (Lemma 5.14 with inputs $A \leftarrow B'$ and $B \leftarrow A'$) to conclude that $|A' + A'| \leq O(K^{24}|B'|) \leq O(K^{24}n)$. Finally, the running is bounded by $\widetilde{O}(K^{12}|A|)$ as claimed. ◂

## 5.3 3-SUM for Structured Inputs

The purpose of this section is to prove the following theorem. We focus on the group $G = \mathbf{F}_p^d$, but the theorem holds for $G = \mathbf{Z}$ as well, using the same proof with minor modifications (such as substituting an appropriate linear hash function for integers).

**Theorem 5.4 (3-SUM for Structured Inputs).** *Let $(A, B, C)$ be a 3-SUM instance of size $n$ with $A, B, C \subseteq G$ and $|A + A| \leq K|A|$. Then we can solve $(A, B, C)$ in time $\widetilde{O}(K^{12}n^{7/4})$.*

**Universe Reduction.** As the first step, we will hash all sets to a smaller group $G'$ of size $\ll n^2$ via some hash function $h : G \to G'$. Under the hashing we are bound to introduce several *false positives*, that is, triples $a \in A, b \in B, c \in C$ where $a+b+c \neq 0$ but $h(a)+h(b)+h(c) = 0$. To deal with these false positives, we have to list several 3-SUM solutions in the smaller group instead of merely determining the existence of one solution. The following problem definition and lemma make this precise.

**Definition 5.17 (3-SUM Listing).** *Given sets $A, B, C \subseteq G$ and a parameter $t$, compute for each $a \in A$ a list of $t$ distinct pairs $(b, c) \in B \times C$ with $a + b + c = 0$ (or if there are less than $t$ solutions, a list containing all of them).*

**Lemma 5.18 (Reduction to 3-SUM Listing in Small Groups).** *Let $G = \mathbf{F}_p^d, G' = \mathbf{F}_p^{d'}$, and let $h : G \to G'$ be a random linear map. There is a fine-grained reduction from a 3-SUM instance $A, B, C \subseteq G$ to one 3-SUM listing instance $h(A), h(B), h(C) \subseteq G'$ (where $h(A) = \{h(a) : a \in A\}$) with parameter $t = O(n^2/|G'|)$. The reduction runs in time $O(nt^2)$ and succeeds with constant probability $\frac{8}{10}$.*

**Proof.** ▸ Let $t = 10n^2|G'|^{-1}$. We may assume that $|G'| \geq 10n$, since otherwise we can simply solve the given instance in time $O(n^2) = O(nt)$. We precompute a lookup table to find, given a hash value $a' \in h(A)$, all $a \in A$ with $h(a) = x$ (and similarly for $B$ and $C$). Then we run the listing algorithm with parameter $t$ on the instance $(h(A), h(B), h(C))$, and for every reported solution $a', b', c'$ we use the lookup table to check whether these correspond to some $a \in A, b \in B, c \in C$ with $a + b + c = 0$.

It is clear that the reduction cannot report "yes" unless the given 3-SUM instance $(A, B, C)$ is a "yes" instance. We argue that the reduction misses a "yes" instance with probability at most $\frac{1}{10}$. To this end we fix any $a \in A$ which is part of a 3-SUM solution and prove that with probability at least $\frac{9}{10}$, there are less than $t$ many false positives $(b^*, c^*) \in B \times C$ with $h(a) + h(b^*) + h(c^*) = 0$. In this case it follows that the listing algorithm will return at least one proper solution $(h(a), h(b), h(c))$ where $a + b + c = 0$, and we will recover $(a, b, c)$ using the lookup table. And indeed, for any fixed pair $(b^*, c^*) \in B \times C$ with $a + b^* + c^* \neq 0$, we have that $h(a) + h(b^*) + h(c^*) = 0$ with probability at most $|G'|^{-1}$. Hence the expected number of false positives is $|B| \cdot |C| \cdot |G'|^{-1} \leq n^2|G'|^{-1}$. By Markov's inequality the number of false positives exceeds $t = 10n^2|G'|^{-1}$ with probability at most $\frac{1}{10}$. This completes the correctness argument.

**Algorithm 5.1.** Lists $t$ 3-SUM solutions for a given instance $A, B, C \subseteq G$ where $|G| \leq O(n^2/t)$ and $|A + A| \leq K|A|$ in subquadratic time.

```
1  Apply Lemma 5.7 on A to compute H, X and a_0
2  Subsample a set S ⊆ G with rate 100 log n / |H|
3  for each s ∈ S, x ∈ X do
4      Compute the sets B_s = B ∩ (H + s) and C_{s,x} = C ∩ (H − s − x − a_0)
5  for each s ∈ S, x ∈ X do
6      if |B_s| ≤ Δ and |C_{s,x}| ≤ Δ then
7          List all pairs (b, c) ∈ B_s × C_{s,x} and whenever b + c ∈ −A, report the
               corresponding 3-SUM solution (−(b + c), b, c)
8      else
9          Compute the sumset B_s + C_{s,x} using Theorems 2.13 and 2.24 and list t
               witnesses (b, c) for each a ∈ B_s + C_{s,x} using Theorem 2.25
10         for each a ∈ (B_s + C_{s,x}) ∩ −A do
11             Report the solution (−a, b, c) for each witness (b, c) of a
```

Before analyzing the running time, we first analyze the maximum *bucket load* $L$ of the hashing, i.e., the maximum number of elements in $A$ (or similarly in $B$ or $C$) hashing to the same value under $h$. Note that $\binom{L}{2}$ is at most the number of *collisions* of the hash function (i.e., the number of distinct pairs $a, a' \in A$ with $h(a) = h(a')$), as any two elements in the same bucket cause a collision. For any fixed $a, a' \in A$, the collision probability is at most $|G'|^{-1}$ and thus the expected number of collisions is at most $n^2|G'|^{-1}$. Using again Markov's inequality, the hashing causes at most $t = 10n^2|G'|^{-1}$ collisions with probability $\frac{9}{10}$, and in that case we can bound $L = O(t^{1/2})$.

We are finally ready to bound the running time. Constructing $h(A), h(B), h(C)$, the hashing and the lookup table takes linear time. After that, we check all the $nt$ listed solutions. For each solution $(x, y, z)$ we have to enumerate all pairs $(a, b) \in A \times B$ with $h(a) = x$ and $h(b) = y$ which takes time $O(L^2)$. Hence, the total time is $O(nt^2)$. ◂

**The Algorithm.** Using Lemma 5.18, we may assume that $A, B, C \subseteq G$ where $G$ has size $O(n^2/t)$, and we have to list $t$ solutions for each $a \in A$. Moreover, by the assumption in Theorem 5.4 we can assume that $|A + A| \leq K|A|$ (this property is preserved under the linear hashing in Lemma 5.18).

The 3-SUM algorithm is given in Algorithm 5.1. First cover $A$ by a translate of an approximate group $H$ (that is, a small set $H$ satisfying $H + H \subseteq H + X$ where $X$ is small) using Lemma 5.7. Then sample a set $S \subseteq G$ (with rate $\frac{10 \log n}{|H|}$) such that $H + S$ covers the whole universe $G$. We precompute the sets $B_s = B \cap (H + s)$ and $C_{s,x} = C \cap (H - s - x)$ for all shifts $s \in S$ and $x \in X$. The crucial insight is that we only have to look for 3-SUM solutions in $A \times B_s \times C_{s,x}$. For each such group we apply a heavy-light approach: Either the sets $B_s, C_{s,x}$ are sparse (with size smaller than some parameter $\Delta$ to be determined later), and we can afford to enumerate all pairs. Or the sets are dense, in which case we compute $B_s + C_{s,x}$ in linear time, but this case cannot happen too often. We analyze these steps in more detail, starting with the proof that $H + S$ indeed covers the whole universe $G$:

**Lemma 5.19 (Random Cover).** *With high probability, for any $z \in G$ there are $\Theta(\log n)$ shifts $s \in S$ such that $z \in H + s$ (in short: $r_{H,S}(z) = \Theta(\log n)$).*

**Proof.** ▸ In expectation, each element $z \in G$ is contained in $|H| \cdot \frac{100 \log n}{|H|} = 100 \log n$ sets of the form $H + s, s \in S$. By Chernoff's bound, the probability that we hit less than $50 \log n$ sets or more than $150 \log n$ sets is at most $2 \exp(-\frac{100 \log n}{12}) \leq n^{-8}$. Taking a union bound over the $|G| \leq n^2$ elements $z$, the statement is correct with probability at least $1 - n^{-6}$. ◂

**Lemma 5.20 (Correctness of Algorithm 5.1).** *Algorithm 5.1 is correct, that is, it reports a list of $t$ witnesses for each $a \in A$ (or a list of all witnesses if there are less than $t$ many).*

**Proof.** ▶ Focus on any 3-SUM solution $a+b+c = 0$. The key is to prove that there are shifts $s \in S, x \in X$ such that $(a, b, c) \in A \times B_s \times C_{s,x}$. In this case it is easy to check that the algorithm will either report $(a, b, c)$ (in Line 7 or Line 11) or it already reported $t$ other witnesses for $a$ (if the list of $t$ witnesses computed in Line 9 does not contain the particular witness $(b, c)$).

To see that $s, x$ exist as claimed, invoke the previous lemma to find some $s \in S$ such that $b \in H + s$, that is, there is some $v \in H$ such that $b = v + s$. Then, since $a \in A \subseteq H + a_0$ by Lemma 5.7 we have that $a + v \in H + H + a_0 \subseteq H + X + a_0$. Thus, there is some $w \in H$ and $x \in X$ such that $a + v = w + x + a_0$. It follows that $c = -(a + b) = -(w + s + x + a_0) \in -H - s - x - a_0 = H - s - x - a_0$. Using the definitions of $B_s$ and $C_{s,x}$, we conclude that $b \in B_s$ and $c \in C_{s,x}$ as stated. ◀

**Lemma 5.21 (Running Time of Algorithm 5.1).** *Algorithm 5.1 runs in time*

$$\widetilde{O}\left(K^{12}\left(\frac{n^2}{t} + \frac{n\Delta^2}{t} + \frac{n^2 t}{\Delta}\right)\right).$$

**Proof.** ▶ Computing $H$ and $X$ in Line 1 takes time $O(K^{12}n)$ by Lemma 5.7, and the same lemma guarantees that $|H| \leq K^2 n$ and $|X| \leq K^5$. Sampling $S$ in a naive way in Line 2 takes time $O(|G|) = O(n^2/t)$ and with high probability, $S$ has size $\widetilde{O}(\frac{|G|}{|H|}) = \widetilde{O}(n/t)$. In Line 4, it takes linear time to compute each set $B_s$ and $C_{s,x}$, so the total time is $O(n|S||X|) \leq \widetilde{O}(K^5 n^2/t)$.

For the loop over pairs $s \in S, x \in X$ (in Line 5) we split the analysis into two cases: The *light* pairs $s, x$ with $|B_s|, |C_{s,x}| \leq \Delta$ and the remaining *heavy* pairs. There are up to $|S| \cdot |X| \leq \widetilde{O}(K^5 n/t)$ light pairs, and for each such pair we spend time $O(\Delta^2)$ in Line 7. The total time spent on light pairs is thus $O(K^5 n\Delta^2/t)$.

The number of heavy pairs is bounded by $\widetilde{O}(K^5 n/\Delta)$. Indeed, recall that each element $b$ occurs in at most $O(\log n)$ sets $B_s$ by Lemma 5.19. Hence, there are at most $\widetilde{O}(n/\Delta)$ heavy sets $B_s$. Similarly, each element $c$ occurs in at most $O(|X| \log n) = O(K^5 \log n)$ sets $C_{s,x}$ and therefore the number of heavy sets $C_{s,x}$ is at most $\widetilde{O}(K^5 n/\Delta)$. For each heavy pair, we spend time $\widetilde{O}(t \cdot |B_s + C_{s,x}|)$ to list $t$ witnesses for each element in the sumset $B_s + C_{s,x}$ by Theorem 2.25. Recall that $B_s + C_{s,x} \subseteq H + H - x - a_0 \subseteq H + X - x - a_0$, and thus $|B_s + C_{s,x}| \leq |H| \cdot |X| \leq K^7 n$. Hence, the heavy pairs amount to time $O((K^5 n/\Delta) \cdot (tK^7 n)) = O(K^{12} n^2 t/\Delta)$. Summing over all these contributions gives the claimed time bound. ◀

**Proof of Theorem 5.4.** ▶ We proceed as outlined before: First apply Lemma 5.18 to reduce the given 3-SUM instance to a 3-SUM listing instance with parameter $t$ in a universe of size $|G'| = O(n^2/t)$ and then run Algorithm 5.1 on that instance. This algorithm is correct by Lemma 5.20, and runs in the claimed running time by setting $t = n^{1/4}$ and $\Delta = n^{1/2}$. Since the universe reduction succeeds only with constant probability $\frac{8}{10}$, we need to repeat this whole process $O(\log n)$ times to achieve high success probability. ◀

## 5.4 Energy Reduction for 3-SUM

In this section we prove the self-reduction for 3-SUM to instances with small additive energy. As outlined before, the proof consists of two key lemmas, see Lemmas 5.2 and 5.5.

**Algorithm 5.2.** The energy reduction via additive combinatorics. Given a 3-SUM instance $A \subseteq G$, this algorithm either detects a 3-SUM solution or constructs an equivalent instance $A^* \subseteq A$ with additive energy $E(A^*) \leq |A^*|^3/K$.

```
repeat
    Estimate E(A) using Lemma 5.10
    if E(A) ≤ |A|^3/K then
        return A* ← A
    else
        Apply the Balog-Szemerédi-Gowers theorem on A to obtain A' ⊆ A
        Solve the 3-SUM instance (A', A, A) using Theorem 5.4
        if (A', A, A) is a "yes" instance then return "yes"
        A ← A \ A'
```

### 5.4.1 Energy Reduction via Additive Combinatorics

**Lemma 5.2 (Energy Reduction via Additive Combinatorics).** *Let $K \geq 1$. There is a fine-grained reduction from a 3-SUM instance $A$ of size $n$ to an equivalent 3-SUM instance $A^* \subseteq A$, where $E(A^*) \leq |A^*|^3/K$. The reduction runs in time $\widetilde{O}(K^{314}n^{7/4})$.*

**Proof.** ▶ The reduction is given in Algorithm 5.2. We repeatedly estimate the additive energy of $A$ using Lemma 5.10 and as long as $E(A) \geq |A|^3/K$, we apply the BSG theorem to obtain a structured subset $A' \subseteq A$. This set has large size $|A'| \geq \Omega(K^{-2}|A|)$ and small doubling $|A' + A'| \leq O(K^{24}|A|) = O(K^{26}|A'|)$. We solve the 3-SUM instance $(A', A, A)$ using Theorem 5.4; if a solution is found in this step we report "yes". Otherwise continue the process with $A \setminus A'$ in place of $A$. As soon as the additive energy of $A$ drops below the desired threshold $|A|^3/K$, we stop and return $A^* \leftarrow A$.

The correctness is easy to prove: In each step we split off a subset $A'$. If there is a 3-SUM solution involving an element from $A'$, we detect the solution by calling Theorem 5.4 and correctly report "yes". Otherwise it is safe to discard $A'$.

To analyze the running time, first observe that in every step the size of $A$ reduces by at least $\Omega(K^{-2}|A|)$. Therefore, after at most $O(K^2)$ steps the size of $A$ must have halved and thus the total number of steps is bounded by $O(K^2 \log n)$. In each step, computing $A'$ via the BSG theorem takes time $O(K^{12}|A|)$ and solving the structured 3-SUM instance $(A', A, A)$ takes time $\widetilde{O}((K^{26})^{12}n^{7/4}) = \widetilde{O}(K^{312}n^{7/4})$. In total we spend time $\widetilde{O}(K^{314}n^{7/4})$ as claimed. ◀

### 5.4.2 Amplification via Hashing

**Lemma 5.5 (Energy Reduction via Hashing).** *Let $K \geq 1$. There is a fine-grained reduction from a 3-SUM instance $A$ with $E(A) \leq |A|^3/K$ to $g = O(|A|^2/K^2)$ 3-SUM instances $A_1, \ldots, A_g$ of size $O(K)$ and with expected energy $\mathbf{E}(E(A_i)) \leq O(K^2)$. The reduction runs in time $\widetilde{O}(|A|^2/K)$.*

The reduction is summarized in Algorithm 5.3. We sample a linear hash function $h : G \to G'$ and construct the instances $A_{x,y} = \{a \in A : h(a) \in \{x, y, -(x+y)\}\}$, for all $x, y \in G'$. We solve all instances by brute-force which exceed their expected size by a constant factor, and pass the other instance to the reduction. If we find a 3-SUM solution in one of the constructed instances, we report "yes".

The analysis involves several steps, but the correctness argument is simple: Since all sets $A_{x,y}$ are subsets of $A$, we can never return "yes" unless $A$ is a "yes" instance. On the other hand, whenever there is a 3-SUM solution $a+b+c=0$ in $A$, we can pick $x = h(a)$ and $y = h(b)$ so that $A_{x,y}$ is a "yes" instance (by the linearity of the hash function).

**Algorithm 5.3.** The energy reduction via subsampling. Given a 3-SUM instance $A \subseteq G = \mathbf{F}_p^d$ with bounded additive energy $E(A) \leq |A|^3/K$, this algorithm constructs $O(|A|^2/K^2)$ smaller 3-SUM instances of size $O(K)$ and with expected additive energy $O(K^2)$.

```
1 Let d' = ⌈log_p(|A|/K)⌉ and let G' = F_p^{d'}
2 Sample a linear hash function h : G → G'
3 for each x, y ∈ G' do
4     Construct the 3-SUM instance A_{x,y} = {a ∈ A : h(a) ∈ {x, y, −(x + y)}}
5     if |A_{x,y}| ≤ 6K then
6         Solve the 3-SUM instance A_{x,y} by means of the reduction
7     else
8         Solve the 3-SUM instance A_{x,y} by brute-force
9 return "yes" if and only if one of the instances A_{x,y} is a "yes" instance
```

We continue with the analysis of the running time of the reduction, which mainly involves proving that most instances have size $O(K)$ and therefore do not have to be brute-forced.

**Lemma 5.22 (Running Time of Algorithm 5.3).** *Algorithm 5.3 runs in expected time* $\widetilde{O}(n^2/K)$.

**Proof.** ▸ For most steps of the algorithm it is easy to bound the running time. In particular, we can construct the instances $A_{x,y}$ in time $O(n^2/K)$ by first precomputing the hash values $h(a)$ for all $a \in A$. The interesting part is to bound the running time of the brute-force step in Line 8. To this end, we analyze the sizes of the constructed instances $A_{x,y}$.

Fix any $x, y \in G'$. We compute the expectation and variance of $|A_{x,y}|$ as follows. For ease of notation, write $X = \{x, y, -(x+y)\}$:

$$\mathbf{E}(|A_{x,y}|) = \sum_{a \in A} \mathbf{P}(h(a) \in X) \leq \sum_{a \in A} \frac{3}{|G'|} = \frac{3n}{|G'|} \leq 3K.$$

Next, we compute the variance:

$$\begin{aligned}
&\mathbf{Var}(|A_{x,y}|) \\
&\quad = -\mathbf{E}(|A_{x,y}|)^2 + \mathbf{E}(|A_{x,y}|^2) \\
&\quad = -\left(\sum_{a \in A} \mathbf{P}(h(a) \in X)\right)^2 + \sum_{a,b \in A} \mathbf{P}(h(a), h(b) \in X)
\end{aligned}$$

Here, we distinguish two cases for $a, b$: If $a$ and $b$ are linearly independent, then the random variables $h(a)$ and $h(b)$ are independent. If $a, b$ are linearly dependent, then there are at most $np = O(n)$ choices for $a, b$ (fix $a$ arbitrarily, then there are at most $p$ choices for $b$ in the span $\langle a \rangle$). It follows that the above expression can be bounded as follows:

$$\begin{aligned}
&\leq -\left(\sum_{a \in A} \mathbf{P}(h(a) \in X)\right)^2 + \left(\sum_{a,b \in A} \mathbf{P}(h(a) \in X) \cdot \mathbf{P}(h(b) \in X)\right) + O\left(\frac{n}{|G'|}\right) \\
&\leq O\left(\frac{n}{|G'|}\right) \\
&\leq O(K).
\end{aligned}$$

We are now ready to bound the expected running time of Line 8 using Chebyshev's inequality:

$$\begin{aligned}
&\sum_{x,y\in G'}\sum_{i=0}^{\log n}\mathbf{P}(|A_{x,y}|\geq 2^i\cdot 6K)\cdot O((2^iK)^2)\\
&\quad\leq \sum_{x,y\in G'}\sum_{i=0}^{\log n}\mathbf{P}\left(|A_{x,y}|-\mathbf{E}(|A_{x,y}|)\geq \Omega(2^i\,\mathbf{Var}(|A_{x,y}|))\right)\cdot O(2^{2i}K^2)\\
&\quad\leq \sum_{x,y\in G'}\sum_{i=0}^{\log n}O\left(\frac{1}{2^{2i}K}\cdot 2^{2i}K^2\right)\\
&\quad\leq \widetilde{O}(n^2/K).
\end{aligned}$$

This completes the running time analysis. ◀

**Lemma 5.23 (Bounded Energy).** *Fix $x, y \in G'$ and let $A_{x,y}$ be as in Algorithm 5.3. Then $\mathbf{E}(E(A_{x,y})) \leq O(K)$.*

**Proof.** ▶ We bound the expected energy as follows:

$$\begin{aligned}
&\mathbf{E}(E(A_{x,y}))\\
&\quad=\sum_{\substack{a_1,a_2,a_3,a_4\in A\\ a_1+a_2=a_3+a_4}}\mathbf{P}(h(a_1),h(a_2),h(a_3),h(a_4)\in\{x,y,-(x-y)\})\\
&\quad=\sum_{s=0}^{3}\sum_{\substack{a_1,a_2,a_3,a_4\in A\\ a_1+a_2=a_3+a_4\\ \dim\langle a_1,a_2,a_3,a_4\rangle=s}}\mathbf{P}(h(a_1),h(a_2),h(a_3),h(a_4)\in\{x,y,-(x-y)\})
\end{aligned}$$

For fixed elements $a_1, a_2, a_3, a_4$ spanning a subspace of dimension $s$, there are at least $s$ hash values in $h(a_1), h(a_2), h(a_3), h(a_4)$ which are independent and therefore the probability can be upper bounded by $1/|G'|^s$.

$$\leq\sum_{s=0}^{3}\frac{1}{|G'|^s}\cdot\sum_{\substack{a_1,a_2,a_3,a_4\in A\\ a_1+a_2=a_3+a_4\\ \dim\langle a_1,a_2,a_3,a_4\rangle=s}}1$$

We now distinguish two cases: For $s = 3$ we use that the inner sum is at most $E(A)$ by definition. For $s \leq 2$ we bound the inner sum by the weaker bound $O(|A|^s)$. (Indeed, any tuple $a_1, a_2, a_3, a_4$ spanning a subspace of dimension $s$ can be obtained by first picking $s$ arbitrary elements from $A$ and expressing the others as one out of $p^4 = O(1)$ possible linear combinations.)

$$\begin{aligned}
&\leq\sum_{s=0}^{2}O\left(\frac{|A|^s}{|G'|^s}\right)+O\left(\frac{E(A)}{|G'|^3}\right)\\
&\leq O\left(\frac{|A|^2}{|G'|^2}+\frac{E(A)}{|G'|^3}\right)\\
&\leq O(K^2).
\end{aligned}$$

In the final step we have used that $|G'| \geq |A|/K$ and that $E(A) \leq |A|^3/K$. ◀

### 5.4.3 Putting Both Parts Together

By concatenating both energy reductions we obtain the following result.

**Theorem 5.6 (Energy Reduction).** *For any $\epsilon, \delta > 0$, there is no $O(n^{2-\epsilon})$-time algorithm solving the 3-SUM problem on instances $A$ with size $n$ and additive energy $E(A) \leq O(|A|^{2+\delta})$, unless the 3-SUM conjecture fails.*

**Proof.** ▶ Suppose that there are $\epsilon, \delta > 0$ and an algorithm $\mathcal{A}$ solving 3-SUM on instances $A$ of size $n$ with additive energy $O(|A|^{2+\delta})$ in time $O(n^{2-\epsilon})$.

We reduce a given 3-SUM instance $A$ to this problem. Let $K = |A|^{0.0001}$. We first apply Lemma 5.2 with parameter $K$ to either detect a 3-SUM solution in $A$ or to find an equivalent instance $A^* \subseteq A$ with additive energy bounded by $|A^*|^3/K$. Next, apply the reduction from Lemma 5.5 to obtain $g = O(|A|^2/K^2)$ instances $A_1, \ldots, A_g$ of size $O(K)$ with expected additive energy $O(K^2)$. By Markov's bound, each such instance has additive energy more than $K^{2+\delta}$ with probability at most $O(K^{-\delta})$. We may therefore use Lemma 5.10 to estimate the additive energies of the constructed instances, and brute-force all instances with energy exceeding $K^{2+\delta}$. We solve the remaining instances using the efficient algorithm $\mathcal{A}$.

It remains to analyze the running time. Lemma 5.2 runs in time $O(K^{314}n^{7/4}) = O(n^{1.7814})$ and Lemma 5.5 runs in time $O(n^2/K) = O(n^{1.9999})$. Since we only solve a $K^{-\epsilon}$-fraction of the instances by brute-force, the total expected running time of brute-forcing instances with exceptionally large additive energy takes time $O(K^{-\epsilon}n^2/K^2 \cdot K^2) = O(n^{2-0.0001\epsilon})$. Finally, solving the remaining instances using $\mathcal{A}$ amounts for time $O(n^2/K^2 \cdot K^{2-2\delta}) = O(n^{2-0.0001\delta})$. All in all, the running time is subquadratic as claimed. ◀

## 5.5 Reducing 3-SUM to Triangle Listing

The first reduction from 3-SUM to triangle listing is by Pătraşcu [172], and this reduction was later generalized by Kopelowitz, Pettie and Porat [143]. It is also known how to adapt the reduction to 3-XOR [128] (i.e., the $G = \mathbf{F}_2^d$ version of 3-SUM).

In this section we revisit this reduction. We present a modified (and arguably simplified) version of the known constructions. As before, we consider 3-SUM instances over the group $G = \mathbf{F}_p^d$, where $p$ is a constant prime and $d = O(\log n)$. Our goal is to prove the following theorem:

**Theorem 5.1 (Hardness of Triangle Listing).** *For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$-time algorithm listing all triangles in a $\Theta(n^{1/2})$-regular $n$-vertex graph which contains at most $O(n^{k/2})$ $k$-cycles for all $k \geq 3$, unless the 3-SUM conjecture fails.*

For the remainder of this subsection, we will prove Theorem 5.1. We start with the construction in Section 5.5.1. In Section 5.5.2 we analyze the number of $k$-cycles and in Section 5.5.3 we justify the assumption that the graph is $\Theta(n^{1/2})$-regular. We summarize the proof of Theorem 5.1 in Section 5.5.4. Throughout, let $A$ be the given 3-SUM instance. By the energy reduction in Theorem 5.6 (applied with $\delta = \frac{1}{2}$, say) we can assume that $E(A) \leq O(|A|^{5/2})$.

### 5.5.1 The Construction

We start with the construction of the triangle listing instance. Let $G' = \mathbf{F}_p^{d'}$ be a subspace of $G$ with prescribed size $|G'| \leq n$ which we will set later. We randomly sample linear maps $h_1, h_2, h_3 : G \to G'$, and let $h : G \to (G')^3$ be defined by $h(a) = (h_1(a), h_2(a), h_3(a))$. Let

$$X = G' \times G' \times \{0\}$$
$$Y = G' \times \{0\} \times G'$$
$$Z = \{0\} \times G' \times G'$$

be the vertex parts in the constructed tripartite graph. Observe that each set $X, Y, Z$ is a subgroup of $(G')^3$. We now add edges to the graph: For each $a \in A$, add an edge between $x \in X$ and $y \in Y$ whenever $y = x + h(a)$. We say that this edge $(x, y)$ is *labeled* with $a$. Similarly, add an edge between $y \in Y$ and $z \in Z$ whenever $z = y + h(a)$ and add an edge between $z \in Z$ and $x \in X$ whenever $x = z + h(a)$.

We remark that for the analysis we view the instance as a labeled (multi-)graph with labels as just described, but for the actual reduction we forget about the edge labels (and multiple edges) and treat the constructed instance as a simple graph; this notation is purely for convenience.

We introduce some more notation. As before, we say that $(a,b,c) \in A^3$ is a *solution* if $a+b+c=0$. We say that $(a,b,c) \in A^3$ is a *pseudo-solution* if $h(a)+h(b)+h(c)=0$. As a first step, we argue that there is a one-to-one correspondence between triangles in the constructed instance and pseudo-solutions.

**Lemma 5.24 (Pseudo-Solutions Are Triangles).** *The labels $a,b,c$ of any triangle in the constructed instance form a pseudo-solution. Moreover, for every pseudo-solution $a,b,c$ there are at most six triangles in the instance labeled with $a,b,c$.*

**Proof.** ▸ The first claim is easy: By construction, the edge labels $a,b,c$ of any triangle $(x,y,z)$ (in fact, of any closed walk) must satisfy that $h(a)+h(b)+h(c)=0$. By definition, $a,b,c$ constitutes a pseudo-solution.

For the other direction, let $a,b,c$ be a pseudo-solution. There are six ways to assign the edge labels to the edge parts; we will focus on one case and prove that there is a unique triangle $(x,y,z) \in X \times Y \times Z$ where $(x,y)$ is labeled with $a$, $(y,z)$ is labeled with $b$, and $(z,x)$ is labeled with $c$. Writing $x=(x_1,x_2,0)$, $y=(y_1,0,y_3)$ and $z=(0,z_2,z_3)$, we obtain the following constraints:

$$\begin{aligned} y_1 &= x_1 + h_1(a) & 0 &= x_2 + h_2(a) & y_3 &= h_3(a) \\ 0 &= y_1 + h_1(b) & z_2 &= h_2(b) & z_3 &= y_3 + h_3(b) \\ x_1 &= h_1(c) & x_2 &= z_2 + h_2(c) & 0 &= z_3 + h_3(c) \end{aligned}$$

This equation system (with indeterminates $x_1, x_2, y_1, y_3, z_2, z_3$) is uniquely solvable by $x_1 = h_1(c), x_2 = -h_2(a), y_1 = -h_1(b), y_3 = h_3(a), z_2 = h_2(b), z_3 = -h_3(c)$. ◂

By this characterization it is easy to complete the reduction: By listing all triangles in the constructed instance, in particular we list all pseudo-solutions of the 3-SUM instance. We check whether one of these pseudo-solutions forms a proper solution and return "yes" in this and only this case. Moreover, we obtain the following bound on the number of triangles in the constructed instance:

**Lemma 5.25 (Number of Triangles).** *Either we can find a 3-SUM solution in time $\widetilde{O}(|G'|^3/n)$, or the expected number of triangles in the constructed instance is $O(n^3|G'|^{-3})$.*

**Proof.** ▸ By the previous lemma, the number of triangles is bounded by six times the number of pseudo-solutions. First, focus on a pseudo-solution $a,b,c$ that is not a proper solution (i.e., $a+b+c \neq 0$). The probability that $h(a)+h(b)+h(c)=0$, or equivalently that $h(a+b+c)=0$, is at most $|G'|^{-3}$. It follows that the expected number of non-proper pseudo-solutions is at most $n^3|G'|^{-3}$.

Next, focus on the proper solutions. We distinguish two cases: On the one hand, if there are at most $n^3|G'|^{-3}$ proper solutions, then the total number of pseudo-solutions and therefore the total number of triangles is $O(n^3|G'|^{-3})$, as claimed. On the other hand, if there are at least $n^3|G'|^{-3}$ solutions, then it suffices to sample $\widetilde{O}(n^2/(n^3|G'|^{-3})) = \widetilde{O}(|G'|^3/n)$ pairs $(a,b) \in A^2$ to detect at least one 3-SUM solution $(a,b,-(a+b)) \in A^3$ with high probability. ◂

Finally, the instance can be constructed in time $O(n|G'|)$ as follows: We precompute the hash values $h_1(a), h_2(a), h_3(a)$ for all $a \in A$. For each vertex in the instance, say, $x=(x_1,x_2,0)$, we then check only those $a$'s with hash values satisfying $x_2 + h_2(a) = 0$ (or $x_1 - h_1(a) = 0$) and add the respective edges.

### 5.5.2 Counting the Number of $k$-Cycles

The most interesting part in our setting is to bound the number of $k$-cycles in the constructed instance (for $k \geq 4$). To this end, we introduce some notation. We say

that a length-$k$ walk is *labeled* by $a_1, \dots, a_k$ whenever the edges in the walk are labeled with $\pm a_1, \dots, \pm a_k$. More specifically, we fix an order of the vertex parts (say the *clockwise* order is $X, Y, Z$) and require that the edge in the $i$-th step is labeled with $a_i$ if the walk takes a step in clockwise direction (that is, from $X$ to $Y$, from $Y$ to $Z$ or from $Z$ to $X$) and labeled with $-a_i$ if the walk takes a step in counter-clockwise direction (that is, from $Y$ to $X$, $X$ to $Z$ or from $Z$ to $Y$). For example, the walk $a_1, -a_2, a_3, a_4$ for elements $a_1, a_2, a_3, a_4 \in A$ takes one step in clockwise direction, takes one step in counter-clockwise direction (to the same part where it started from) and takes two more steps in clockwise direction. Here we assume for simplicity that $A$ and $-A$ are disjoint, so that the label of a walk uniquely determines its directions.[37]

[37] *More generally, we should use pairs $(s_i, a_i)$ with $s_i = \pm 1$ and $a_i \in A$ to label paths, but we stick to the simpler version described in the text.*

We distinguish between two types of $k$-cycles: A $k$-cycle labeled with $a_1, \dots, a_k$ is called a *pseudo-$k$-cycle* if $a_1 + \cdots + a_k \neq 0$, and a *zero-$k$-cycle* otherwise. The analysis differs for these two types of cycles: For pseudo-$k$-cycles we can exploit more randomness since all labels $a_1, \dots, a_k$ can be expected to produce independent hash values $h(a_1), \dots, h(a_k)$. For zero-$k$-cycles, one of the hash values is determined by the others and we therefore have a smaller degree of independence. But we have the advantage that the 3-SUM instance has small additive energy, and therefore the number of solutions to $a_1 + \cdots + a_k = 0$ is small.

**Lemma 5.26 (Rate of Zero-$k$-Cycles).** *Fix a vertex $v$ and labels $a_1, \dots, a_k \in \pm A$ satisfying $a_1 + \cdots + a_k = 0$. Then there is a cycle starting from and ending at $v$ labeled with $a_1, \dots, a_k$ with probability at most $|G'|^{-s}$, where $s = \dim\langle a_1, \dots, a_k\rangle$.*

**Proof.** ▸ First observe that any walk with labels $a_1 + \cdots + a_k = 0$ that starts at $v$ also ends at $v$. We therefore bound the probability that there is a walk starting from $v$ which is labeled with $a_1, \dots, a_k$ by $|G'|^{-s}$. The proof is by induction on $k$. For the base $k = 0$ we have $s = 0$ and can trivially bound the probability by 1.

For the inductive case assume that $k \geq 1$. By induction, there is a walk of length $k-1$ with probability at most $|G'|^{s'}$ where $s' = \dim\langle a_1, \dots, a_{k-1}\rangle$. We distinguish two cases: If $s' = s$, then we are done. If $s' = s - 1$ (which is indeed the only other case), then the vector $a_k$ is linearly independent from $a_1, \dots, a_{k-1}$ and thus the random variable $h(a_k)$ is independent from the other random variables $h(a_1), \dots, h(a_{k-1})$. Now suppose that the walk after $k-1$ steps has reached some vertex, say, $x = (x_1, x_2, 0)$ and we move in clockwise direction. Then the target vertex $y = (y_1, 0, y_3)$ is uniquely determined by $y_1 = x_1 + h_1(a_k)$ and $y_3 = h_3(a_k)$. In addition, we induce the constraint $0 = x_2 + h_2(a_k)$ which is satisfied with probability at most $|G'|^{-1}$. By the aforementioned independence, the total probability is at most $|G'|^{-s'}|G'|^{-1} = |G'|^{-s}$. ◂

**Lemma 5.27 (Rate of Pseudo-$k$-Cycles).** *Fix a vertex $v$ and labels $a_1, \dots, a_k \in \pm A$ satisfying $a_1 + \cdots + a_k \neq 0$. Then there is a cycle starting from and ending at $v$ labeled with $a_1, \dots, a_k$ with probability at most $|G'|^{-s-2}$, where $s = \dim\langle a_1, \dots, a_k\rangle$.*

The proof of this lemma is a bit more involved than the previous one. Our strategy is to prove the following more technical generalization (see Lemma 5.28). The proof of Lemma 5.27 then follows by setting $a_0 = a_1 + \cdots + a_k$. Indeed, any cycle labeled with $a_1, \dots, a_k$ is in particular a walk and because it is closed we must have $h(a_1) + \cdots + h(a_k) = 0$.

**Lemma 5.28.** *Fix a vertex $v$ and labels $a_1, \dots, a_k \in \pm A$ and any non-zero $a_0 \in G$. Then the probability of the combined events (i) there is a walk starting from $v$ labeled with $a_1, \dots, a_k$ and (ii) $h(a_0) = 0$, is at most $|G'|^{-s-2}$, where $s = \dim\langle a_0, a_1, \dots, a_k\rangle$.*

**Proof.** ▸ The proof is by induction on $k$. We start with the base case $k = 0$. Since we assume that $a_0 \neq 0$, we have that $s = \langle a_0\rangle = 1$. Moreover, the probability that $h(a_0) = 0$ is exactly $|G'|^{-3}$.

Next consider the inductive case $k \geq 1$, and let $s' = \dim\langle a_0, \dots, a_{k-1}\rangle$. If $s' = s$, then we are done by induction. Otherwise, we have $s' = s - 1$ and $a_k$ is linearly independent from the other vectors $a_0, \dots, a_{k-1}$. Suppose that after $k-1$ steps the

walk has reached some vertex, say $z = (0, z_2, z_3)$, and we are moving in counter-clockwise direction. Then the target vertex $y = (y_1, 0, y_3)$ is uniquely determined by $y_1 = h_1(a_k)$ and $y_3 = z_3 + h_3(a_k)$, but moving to $y$ is only possible if the new constraint $0 = z_2 + h_2(a_k)$ is satisfied. This constraint is satisfied with probability $|G'|^{-1}$ and since $h(a_k)$ is independent from the randomness in previous steps, the overall probability is at most $|G'|^{-s-1} \cdot |G'|^{-1} \leq |G'|^{-s-2}$. ◀

**Lemma 5.29 (Number of $k$-Cycles).** *For any constant $k \geq 4$, the expected number of $k$-cycles in the constructed instance is $O(E(A) \cdot n^{k-4}|G'|^{-k+3} + n^{k-2}|G'|^{-k+4} + n^k|G'|^{-k})$.*

**Proof.** ▶ We first compute the expected number of pseudo-$k$-cycles:

$$\begin{aligned}
&\sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k \neq 0}} \sum_{v \in V} \mathbf{P}(\exists \text{cycle starting from and ending at } v \text{ labeled with } a_1, \dots, a_k) \\
&\quad \leq \sum_{s=0}^{k} \sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k \neq 0 \\ \dim\langle a_1,\dots,a_k\rangle = s}} \sum_{v \in V} |G'|^{-s-2} \\
&\quad \leq \sum_{s=0}^{k} O(n^s \cdot |G'|^2 \cdot |G'|^{-s-2}) \\
&\quad = O(n^k |G'|^{-k}).
\end{aligned}$$

Here, for the first inequality we have applied Lemma 5.27 and we have bounded the number of tuples $(a_1, \dots, a_k)$ with $\dim\langle a_1, \dots, a_k\rangle = s$ by $O(n^s)$ (indeed, after fixing $s$ linearly independent vectors from $\pm A$, each remaining vector can be expressed as one out of $p^k \leq O(1)$ possible linear combinations).

Next, we compute the number of zero-$k$-cycles:

$$\begin{aligned}
&\sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k = 0}} \sum_{v \in V} \mathbf{P}(\exists \text{cycle starting from and ending at } v \text{ labeled with } a_1, \dots, a_k) \\
&\quad \leq \sum_{s=0}^{k-1} \sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k = 0 \\ \dim\langle a_1,\dots,a_k\rangle = s}} \sum_{v \in V} |G'|^{-s} \\
&\quad = \sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k = 0 \\ \dim\langle a_1,\dots,a_k\rangle = k-1}} \sum_{v \in V} |G'|^{-k+1} + \sum_{s=0}^{k-2} \sum_{\substack{a_1,\dots,a_k \in \pm A \\ a_1+\dots+a_k = 0 \\ \dim\langle a_1,\dots,a_k\rangle = s}} \sum_{v \in V} |G'|^{-s} \\
&\quad \leq O(E(A) \cdot n^{k-4} \cdot |G'|^2 \cdot |G'|^{-k+1}) + \sum_{s=0}^{k-2} O(n^s \cdot |G'|^2 \cdot |G'|^{-s}) \\
&\quad = O(E(A) \cdot n^{k-4}|G'|^{-k+3} + n^{k-2}|G'|^{-k+4}).
\end{aligned}$$

For the first inequality we applied Lemma 5.26. For the second inequality, we have bounded the number of tuples $a_1, \dots, a_k$ with $\dim\langle a_1, \dots, a_k\rangle = s$ by $O(n^s)$ as before. In addition, we have bounded the number of solutions $a_1, \dots, a_k \in \pm A$ to the linear equation $a_1 + \dots + a_k = 0$ using Lemma 5.11 by $E(A) \cdot |A|^{k-4}$. ◀

We can use the previous lemmas to bound the number of $k$-cycles for any constant $k$ (but simultaneously for all $k \geq 4$). Using the following trick by [130] which states that the number of $k$-cycles is determined by the number of 4-cycles, we can obtain this stronger claim.

**Lemma 5.30 (Relation of $k$-Cycles and 4-Cycles, [130]).** *Let $G$ be an undirected graph with maximum degree $d$. For any $k \geq 4$, it holds that $C_k(G) \leq d^{k-4} \cdot C_4(G)$, where $C_k(G)$ is the number of length-$k$ closed walks in $G$.*

**Proof.** ▶ Let $A$ be the adjacency matrix of $G$. As $G$ is undirected, the matrix $A$ is symmetric and its eigenvalues $\lambda_1, \ldots, \lambda_n$ are real. By the Gershgorin circle theorem, all eigenvalues are equal to some diagonal entry $A[i,i] = 0$ up to additive error $\sum_{j \neq i} |A[i,j]| \leq d$. In particular, all eigenvalues have magnitude at most $d$.

Observe that $C_k(G) = \operatorname{tr}(A^k) = \sum_{i=1}^n \lambda_i^k$. It follows that

$$C_k(G) = \sum_{i=1}^n \lambda_i^k \leq d^{k-4} \cdot \sum_{i=1}^n |\lambda|^4 = d^{k-4} \cdot \sum_{i=1}^n \lambda^4 = d^{k-4} \cdot C_4(G). \qquad \blacktriangleleft$$

### 5.5.3 Making the Graph Regular

The next step is to enforce the assumption that the constructed is $\Theta(r)$-regular, where $r = 2n/|G'|$. To this end, we first analyze the *expected degrees* in the instance constructed in the previous Section 5.5.1.

**Lemma 5.31.** *Fix a vertex $v$. Then* $\mathbf{E}(\deg(v)) = r \pm O(1)$ *and* $\mathbf{Var}(\deg(v)) \leq O(r)$.

**Proof.** ▶ Focus on an arbitrary vertex, say, $x = (x_1, x_2, 0) \in X$ (the proof is similar for vertices in $Y$ and $Z$). We write $\deg(x) = \deg_Y(x) + \deg_Z(x)$, where $\deg_Y(x)$ denotes the number of edges from $x$ to $Y$, and $\deg_Z(x)$ denotes the number of edges from $x$ to $Z$. We focus on the analysis of $\deg_Y(x)$, the same treatment applies to $\deg_Z(x)$. For each edge label $a$, there is only a unique candidate $y = (y_1, 0, y_3) \in Y$ which is reachable by an edge from $x$ (indeed, $y_1$ and $y_3$ are determined by $x_1$ and $a$). There is an edge to that unique candidate $y$ if and only if $x_2 + h_2(a) = 0$. Therefore, the expected degree is:

$$\mathbf{E}(\deg_Y(x)) = \sum_{a \in A} \mathbf{P}(x_2 + h_2(a) = 0) = n|G'|^{-1} \pm O(1).$$

(The $\pm O(1)$ term stems from the element 0 which may or may not be present in $A$ but always hashes to 0 under a linear hash function.) To bound the variance, we compute

$$\begin{aligned}
&\mathbf{Var}(\deg_Y(x)) \\
&\quad = \mathbf{E}(\deg_Y(x)^2) - \mathbf{E}(\deg_Y(x))^2 \\
&\quad \leq \left( \sum_{a,b \in A} \mathbf{P}(x_2 + h_2(a) = x_2 + h_2(b) = 0) \right) - \left( \sum_{a \in A} \mathbf{P}(x_2 + h_2(a) = 0) \right)^2
\end{aligned}$$

To bound the first sum, we consider two cases: Either $a$ and $b$ are linearly independent, in which case the random variables $h_2(a)$ and $h_2(b)$ are independent. Or $a$ and $b$ are linearly dependent, in which case there are at most $pn = O(n)$ such pairs (we can pick $a$ arbitrarily and there are only $p$ choices for $b$ in the span $\langle a \rangle$). It follows that:

$$\begin{aligned}
&\leq O(n|G'|^{-1}) + \left( \sum_{a,b \in A} \mathbf{P}(x_2 + h_2(a) = 0) \cdot \mathbf{P}(x_2 + h_2(b) = 0) \right) \\
&\qquad - \left( \sum_{a \in A} \mathbf{P}(x_2 + h_2(a) = 0) \right)^2 \\
&= O(n|G'|^{-1}).
\end{aligned}$$

Recall that $\deg(x) = \deg_Y(x) + \deg_Z(x)$. Since the random variables $\deg_Y(x)$ and $\deg_Z(x)$ depend on the independent hash functions $h_2$ and $h_1$, the random variables $\deg_Y(x)$ and $\deg_Z(x)$ are independent. It follows that $\mathbf{E}(\deg(x))$ and $\mathbf{Var}(\deg(x))$ are as claimed. ◀

**Algorithm 5.4.** Turns the triangle listing instance from Section 5.5.1 into a $\Theta(n/|G|)$-regular graph (by removing some vertices, and listing all triangles involving at least one of the removed vertices).

```
1 Let (V_0, E_0) be the instance constructed in Section 5.5.1
2 Let V ← V_0, E ← E_0 and let r ← 2n/|G'|
3 while there is a vertex v in (V, E) with degree < (1/4)r or > 2r do
4     Enumerate all pairs of neighbors u, w ∈ V of v and report (u, v, w) if it is
        a triangle
5     Remove v from V and its incident edges from E
6 return (V, E)
```

Given the previous lemma, most vertices in the constructed instance have degree $\Theta(r)$. However, we want that *every* vertex has degree $\Theta(r)$. We will therefore select an (induced) subgraph of the constructed instance, in which the degree bound is satisfied. Note that by selecting a subgraph, we cannot increase the number of $k$-cycles, and the analysis from the previous Section 5.5.2 remains intact.

We use the algorithm described in Algorithm 5.4. It is easiest to describe using some terminology: We call a vertex $v$ *high-degree* if it has degree more than $2r$, *low-degree* if it has degree less than $\frac{1}{2}r$ and *tiny-degree* if it has degree less than $\frac{1}{8}r$. As long as there is a high-degree or tiny-degree vertex $v$ in the graph, we remove $v$ and all its incident edges. In order to not miss the triangles involving the removed vertices $v$, we list all pairs of neighbors $u, w$ of $v$ and report all triangles $(u, v, w)$ found in this way. It is obvious that the remaining graph is $\Theta(r)$-regular, and moreover we have not missed any triangle by pruning the graph in this way. It remains to analyze the running time of Algorithm 5.4.

**Lemma 5.32 (Running Time of Algorithm 5.4).** *Algorithm 5.4 runs in expected time* $\widetilde{O}(n|G'|)$.

**Proof.** ▶ As in Algorithm 5.4, we denote by $(V_0, E_0)$ the graph constructed in Section 5.5.1. We split the analysis in two parts: First, we bound the time spend in iterations removing a high-degree vertex and second, we bound the time spend in iterations removing tiny-degree vertices. Since the running time per iteration is dominated by enumerating all pairs of neighbors of the vertex $v$ to be removed, we can bound the expected time to remove all high-degree vertices as follows:

$$\begin{aligned}
&\sum_{v\in V_0} \deg(v)^2 \cdot \mathbf{P}(\deg(v) \geq 2r) \\
&\quad\leq \sum_{v\in V_0} \sum_{i=1}^{\log|V_0|} 2^{2i+2}r^2 \cdot \mathbf{P}(\deg(v) \geq 2^i r) \\
&\quad\leq \sum_{v\in V_0} \sum_{i=1}^{\log|V_0|} 2^{2i+2}r^2 \cdot \mathbf{P}(|\deg(v) - \mathbf{E}(\deg(v))| \geq \Omega(2^i \cdot \mathbf{Var}(\deg(v)))) \\
&\quad\leq \sum_{v\in V_0} \sum_{i=1}^{\log|V_0|} 2^{2i+2}r^2 \cdot O\left(\frac{1}{2^{2i}r}\right) \\
&\quad\leq \sum_{v\in V_0} \sum_{i=1}^{\log|V_0|} O(r) \\
&\quad\leq \widetilde{O}(n|G'|).
\end{aligned}$$

We now focus on the time spent on iterations removing tiny-degree vertices. Each such iteration runs in time $O(r^2)$, and we therefore aim to bound the number of iterations. The first step is to show that in the original graph $(V_0, E_0)$, the expected number of edges incident to high-degree or low-degree vertices is at

most $O(n)$. Indeed, by Chebyshev's inequality and again using the previously obtained bounds, the expected number of edges incident to high-degree vertices is at most

$$\begin{aligned}
&\sum_{v\in V_0} \deg(v)\cdot \mathbf{P}\left(\deg(v)\geq 2r\right)\\
&\quad\leq \sum_{v\in V_0}\sum_{i=1}^{\infty} 2^{i+1}r\cdot \mathbf{P}(\deg(v)\geq 2^i r)\\
&\quad\leq \sum_{v\in V_0}\sum_{i=1}^{\infty} 2^{i+1}r\cdot \mathbf{P}(|\deg(v)-\mathbf{E}(\deg(v))|\geq \Omega(2^i\cdot \mathbf{Var}(\deg(v))))\\
&\quad\leq \sum_{v\in V_0}\sum_{i=1}^{\infty} 2^{i+1}r\cdot O\left(\frac{1}{2^{2i}r}\right)\\
&\quad\leq \sum_{v\in V_0} O(1)\\
&\quad= O(|G'|^2).
\end{aligned}$$

Using the same idea we can bound the number of edges incident to low-degree vertices by $O(|G'|^2)$, too. Moreover, we can bound the numbers $L$ and $H$ of low-degree and high-degree vertices in the original graph by $L, H = O(|G'|^2/r)$.

We now again turn to Algorithm 5.4 and bound the number of iterations. There are up to $H$ iterations removing the high-degree vertices, and the remaining iterations remove tiny-degree vertices. However, observe after removing $e$ edges from the original graph, we can create at most $L+6e/r$ tiny-degree vertices: Up to $L$ vertices which are low-degree in the original graph plus at most $2e/(\frac{1}{2}r-\frac{1}{8}r)\leq 6e/r$ vertices which were not low-degree in the original graph but which turned tiny-degree by losing edges. Since every iteration removing a tiny-degree vertex removes at most $\frac{1}{4}r$ edges, the total number of edges removed after $i$ iterations is at most $O(|G'|^2)+\frac{i}{4}r$. Consequently, if the algorithm reaches the $i$-th iteration, it has witnessed at least $i-H$ tiny-degree vertices and we therefore have

$$i-H\leq L+\frac{6\cdot(O(|G'|^2)+\frac{i}{8}r)}{r}\leq L+O(|G'|^2/r)+\frac{3i}{4}.$$

It follows that $i\leq O(L+H+|G'|^2/r)=O(|G'|^2/r)$, and therefore Algorithm 5.4 runs for at most $O(|G'|^2/r)$ iterations. Recall that each iteration removing a tiny-degree vertex takes time $O(r^2)$, and therefore the total time of all iterations removing tiny-degree vertices is $O(|G'|^2/r\cdot r^2)=O(n|G'|)$. ◀

### 5.5.4 Putting the Pieces Together

We are ready to prove Theorem 5.1.

**Proof of Theorem 5.1.** ▶ Recall that we start from a 3-SUM instance with additive energy $E(A)\leq O(n^{5/2})$, by the energy reduction in Theorem 5.6 applied with $\delta=\frac{1}{2}$. We set $|G'|=n^{1/2}$ (that is, we set $d'=\lceil\frac{1}{2}\log_p(n)\rceil$ and $G'=\mathbf{F}_p^{d'}$) and construct the triangle listing instance $(V_0,E_0)$ as described in Section 5.5.1. This step takes time $O(n|G'|)=O(n^{3/2})$. We then run Algorithm 5.4 as described in Section 5.5.3 to obtain an induced subgraph $(V_1,E_1)$ which is regular with degree $\Theta(n/|G'|)=\Theta(n^{1/2})$. This step again takes time $\widetilde{O}(n|G'|)=O(n^{3/2})$ in expectation, see Lemma 5.32.

We next bound the (expected) number of $k$-cycles for all $k\geq 3$. By Lemma 5.25, the expected number of triangles in $(V_0,E_0)$ is at most $O(n^3|G'|^{-3})$ (alternatively, we can immediately find a 3-SUM solution in time $\widetilde{O}(|G'|^3/n)=\widetilde{O}(n^{1/2})$). For $k=4$, by Lemma 5.29 the expected number of 4-cycles in $(V_0,E_0)$ is at most

$$O(E(A)\cdot n^{4-4}|G'|^{-4+3}+n^{4-2}|G'|^{-4+4}+n^4|G'|^{-4})=O(n^2).$$

Using Markov's bound, this number is at most ten times its expected value with probability at least $\frac{9}{10}$. In this case, for any constant $k > 4$, the number of $k$-cycles is at most $O(n^{1/2})^{k-4} \cdot O(n^2) = O(n^{k/2})$ by Lemma 5.30.

Now suppose that we can list all triangles in $(V_1, E_1)$ in time $O(n^{2-\epsilon})$. Adding the triangles detected by Algorithm 5.4, we can compute a list of all triangles in $(V_0, E_0)$. Recall that by Lemma 5.24, every triangle corresponds to a pseudo-solution in the 3-SUM instance. Therefore, it suffices to test whether there exists a proper solution among the pseudo-solutions and to return "yes" in this case. The total expected running time is $\widetilde{O}(n^{3/2} + n^{2-\epsilon})$ and we succeed with constant error probability. ◀

**Listing Hardness in Graphs with Smaller Degrees.** For one of our corollaries of the reduction we need denser graphs than the $\Theta(n^{1/2})$-regular graphs constructed before. It is easy to obtain the following generalization of our reduction to graphs which are $\Theta(r)$-regular.

**Lemma 5.33 (Hardness of Listing Triangles in $\Theta(r)$-Regular Graphs).** *For any $\epsilon > 0$ and any parameter $N^{1/2} \leq r \leq N^{1-\Omega(1)}$, there is no $O((Nr^2)^{1-\epsilon})$-time algorithm listing all triangles in a $\Theta(r)$-regular $N$-vertex graph which contains as most $O(r^k)$ $k$-cycles for all $3 \leq k \leq O(1)$, unless the 3-SUM conjecture fails.*

**Proof.** ▶ We redo the proof of Theorem 5.1 with a different choice of the group size $G'$. Specifically, start from a 3-SUM instance of size $n = N^{1/2}r$ and with additive energy $E(A) \leq O(n^{5/2})$ and set $|G'| = N^{1/2}$. The constructions in Sections 5.5.1 and 5.5.3 construct a graph with at most $|G'|^2 = N$ vertices, and the degree of every vertex is $\Theta(n/|G'|) = \Theta(r)$. The running time of these steps is bounded by $\widetilde{O}(n|G'| + |G'|^3/n)$ (by Lemmas 5.25 and 5.32). By Lemma 5.25 the expected number of triangles is $O(n^3|G'|^{-3}) = O(r^3)$ and by Lemma 5.29, the expected number of 4-cycles is bounded

$$O(E(A) \cdot n^{4-4}|G'|^{-4+3} + n^{4-2}|G'|^{-4+4} + n^4|G'|^{-4}) \\ = O(N^{3/4}r^{5/2} + Nr^2 + r^4) = O(r^4).$$

For last step we have used the assumption $N^{1/2} \leq r$. By Lemma 5.30 it follows that the number of $k$-cycles is bounded by $O(r^k)$. Finally, an algorithm in time $O((Nr^2)^{1-\epsilon})$ would imply an algorithm in time $O(n^{2-2\epsilon} + n|G'| + |G'|^3/n)$ for the 3-SUM instance we started from. As $n^{1/2} \leq |G'| \leq n^{1-\Omega(1)}$, this is subquadratic and contradicts the 3-SUM conjecture. ◀

**All-Edges Triangle.** Many reductions starting from triangle listing can be phrased in a nicer way by starting instead from the *All-Edges Triangle* problem: Given a graph, determine for each edge whether it is part of a triangle. Using our reduction and in addition some known tricks to turn detection algorithms into witness-finding algorithms, we also obtain the following conditional lower bound:

**Lemma 5.34 (Hardness of All-Edges Triangle).** *For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$-time algorithm for the All-Edges Triangle problem in $\Theta(n^{1/2})$-regular $n$-vertex graphs which contain at most $O(n^{k/2})$ $k$-cycles for all $k \geq 3$, unless the 3-SUM conjecture fails.*

## 5.6 Hardness of 4-Cycle Listing

This section is devoted to proving the following Theorem 1.12.

**Theorem 1.12 (Hardness of Listing 4-Cycles).** *For any $\epsilon > 0$, there is no algorithm listing all 4-cycles in time $\widetilde{O}(n^{2-\epsilon} + t)$ or in time $\widetilde{O}(m^{4/3-\epsilon} + t)$ (where $t$ is the number of 4-cycles), unless the 3-SUM conjecture fails.*

**Algorithm 5.5.** The reduction from listing triangles in a $\Theta(n^{1/2})$-regular tripartite graph $G = (V, E)$ to listing 4-cycles.

```
1 Randomly split V into V_1, ..., V_s
2 for each (i, j, ℓ) ∈ [s]^3 do
3     Let V_{i,j,ℓ} = V_i ∪ V_j ∪ V_ℓ
4     Let G_{i,j,ℓ} be the graph with vertices {x_1, x_2, x_3, x_4 : x ∈ V_{i,j,ℓ}} and edges
          {(x_1, y_2), (x_2, y_3), (x_3, y_4) : (x, y) ∈ E}
5     Run the fast 4-cycle listing algorithm on G_{i,j,ℓ}, and for each 4-cycle of the
          form (x_1, y_2, z_3, x_4) report the triangle (x, y, z) (unless already
          reported)
```

Suppose that for some $\epsilon > 0$, there is an algorithm listing all 4-cycles in a graph in time $O(n^{2-\epsilon} + t)$. We give a reduction from listing triangles as described in Theorem 5.1 to listing 4-cycles. That is, we are given an $\Theta(n^{1/2})$-regular $n$-vertex graph $G = (V, E)$ which contains at most $O(n^2)$ 4-cycles, and the goal is to list $O(n^{3/2})$ triangles in subquadratic time. The reduction is summarized in Algorithm 5.5.

The algorithm randomly splits the vertex set into $s$ groups, and for each triple $(i, j, \ell) \in [s]^3$ of groups, constructs a new graph $G_{i,j,\ell}$. This graph is obtained from $G$ by copying each vertex $x$ four times $x_1, x_2, x_3, x_4$, and we add edges $(x_1, y_2), (x_2, y_3), (x_3, y_4)$ as in the original graph, and additionally add all edges $(x_1, x_4)$. We list all 4-cycles in the graph $G_{i,j,\ell}$ and for each 4-cycle of the form $(x_1, y_2, z_3, x_4)$ we report the triangle $(x, y, z)$. Our first claim is that the algorithm correctly reports all triangles in $G$.

**Lemma 5.35 (Correctness of Algorithm 5.5).** *Algorithm 5.5 correctly lists all triangles in $G$.*

**Proof.** ▶ First, observe that by the construction of $G_{i,j,\ell}$ every triple $(x, y, z)$ reported by the algorithm indeed forms a triangle in $G$. Moreover, each triangle $(x, y, z)$ in $G$ can be found as the 4-cycle $(x_1, y_2, z_3, x_4)$ in a graph $G_{i,j,\ell}$, where $x \in V_i, y \in V_j, z \in Z_\ell$. (In addition, there are five other 4-cycles which correspond to $(x, y, z)$.) ◀

**Lemma 5.36 (Number of 4-Cycles).** *The expected total number of 4-cycles across all graphs $G_{i,j,\ell}$ is at most $O(n^2/s + n^{3/2})$.*

**Proof.** ▶ Each 4-cycle using an edge $(x_1, x_4)$ must take the form $(x_1, y_2, z_3, x_4)$. In this case, $(x, y, z)$ is a triangle in the original graph $G$. As each triangle appears as a four cycle in all six possible permutations, the contribution from 4-cycles using an edge $(x_1, x_4)$ is therefore bounded by six times the number of triangles in $G$. By Theorem 5.1, $G$ contains at most $O(n^{3/2})$ many triangles.

In each graph $G_{i,j,\ell}$ there are five types of 4-cycles which do not use edges of the form $(x_1, x_4)$, namely $(x_1, y_2, z_1, w_2)$, $(x_2, y_3, z_2, w_3)$, $(x_3, y_4, z_3, w_4)$, $(x_1, y_2, z_3, w_2)$ and $(x_2, y_3, z_4, w_3)$. In all five cases, $(x, y, z, w)$ forms a 4-cycle in the original graph—more specifically, in the subgraph induced by $V_{i,j,\ell}$. In particular, the contribution of these 4-cycles is five times the number of 4-cycles in $G[V_{i,j,\ell}]$. Recall that in $G$ there are only $O(n^2)$ 4-cycles, and each 4-cycle *survives* only if all of its four vertices are sampled into some set $V_{i,j,\ell}$. This happens with probability $s^3 \cdot s^{-4} = s^{-1}$. Hence, the expected total number of surviving 4-cycles is $O(n^2/s)$. ◀

**Lemma 5.37 (Running Time of Algorithm 5.5).** *For $s = n^{\epsilon/4}$, Algorithm 5.5 runs in expected time $O(n^{2-\epsilon/4})$.*

**Proof.** ▶ The total running time is dominated by the running time of the fast 4-cycle listing algorithm. Assume that this algorithm runs in time $\widetilde{O}(n^{2-\epsilon} + t_{i,j,\ell})$ where $t_{i,j,\ell}$ is the number of 4-cycles in the respective instance. By the previous Lemma 5.36

**Algorithm 5.6.** The reduction from listing triangles in a $\Theta(n^{1/2})$-regular tripartite graph $G = (X, Y, Z, E)$ to approximate distance oracles with stretch $k$.

```
1  Randomly split X, Y, Z into X_1, ..., X_s, Y_1, ..., Y_t, Z_1, ..., Z_s
2  for each (i, j, ℓ) ∈ [s] × [t] × [s] do
3      Let G_{i,j,ℓ} be the subgraph of G induced by X_i, Y_j, Z_ℓ where all edges
         between X_i and Z_ℓ are deleted
4      Preprocess G_{i,j,ℓ} with the approximate distance oracle
5      for each (x, z) ∈ (X_i × Z_ℓ) ∩ E do
6          Query the distance oracle to get an estimate d̃(x, z) satisfying
             d(x, z) ≤ d̃(x, z) ≤ k · d(x, z)
7          if d̃(x, z) ≤ 2k then
8              for each y ∈ Y_j with (x, y) ∈ E do
9                  if (y, z) ∈ E then
10                     Report the triangle (x, y, z)
```

we have that $\sum_{i,j,\ell} t_{i,j,\ell} \leq O(n^2/s) = O(n^{2-\epsilon/4})$. Hence, the total running time of Algorithm 5.5 is

$$\sum_{i,j,\ell \in [s]} \widetilde{O}(n^{2-\epsilon} + t_{i,j,\ell}) \leq \widetilde{O}(n^{3\epsilon/4} \cdot n^{2-\epsilon}) + \widetilde{O}(n^{2-\epsilon/4}) = \widetilde{O}(n^{2-\epsilon/4}).$$

Similarly, if the fast 4-cycle listing algorithm runs in time $\widetilde{O}(m^{4/3-\epsilon} + t_{i,j,\ell})$, then the running time becomes

$$\sum_{i,j,\ell \in [s]} \widetilde{O}((n^{3/2})^{4/3-\epsilon} + t_{i,j,\ell}) \leq \widetilde{O}(n^{3\epsilon/4} \cdot n^{2-\epsilon}) + \widetilde{O}(n^{2-\epsilon/4}) = \widetilde{O}(n^{2-\epsilon/4}). \quad \blacktriangleleft$$

The proof of Theorem 1.12 is complete by Lemmas 5.35 and 5.37. If necessary we can further let the algorithm terminate with high probability in time $\widetilde{O}(n^{2-\epsilon/4})$ by repeating the reduction $O(\log n)$ times and interrupting each execution which takes too long.

## 5.7 Hardness of Distance Oracles

In this section we prove our conditional hardness results for approximate distance oracles. We start with the stretch-$k$ regime (in Section 5.7.1), followed by the stretch-$\alpha$ regime for $2 \leq \alpha < 3$ (in Section 5.7.2), and the improved hardness for dynamic approximate distance oracles (in Section 5.7.3).

### 5.7.1 Stretch $k$

The goal of this section is to prove the following theorem:

**Theorem 1.9 (Hardness of Distance Oracles with Stretch $k$).** *For any integer constant $k \geq 2$, there is no approximate distance oracle for sparse graphs with stretch $k$, preprocessing time $\widetilde{O}(m^{1+p})$ and query time $\widetilde{O}(m^q)$ with $kp + (k+1)q < 1$, unless the 3-SUM conjecture fails.*

Assume that we have access to an approximate distance oracle with stretch $k$, preprocessing time $\widetilde{O}(m^{1+p})$ and query time $\widetilde{O}(m^q)$. We prove hardness starting from an instance of listing $O(n^{3/2})$ triangles in a $\Theta(n^{1/2})$-regular tripartite $n$-vertex graph $G = (X, Y, Z, E)$ which contains at most $O(n^{k'/2})$ $k'$-cycles for all $k' \geq 4$ (that is, we apply Theorem 5.1, and the additional assumption that $G$ be tripartite is without loss of generality). We let $s, t \leq n^{1/2-\Omega(1)}$ be two parameters to be set later and give the reduction in Algorithm 5.6.

We first arbitrarily split the vertex parts $X, Y, Z$ into $s, t, s$ many groups $X_i, Y_j, Z_\ell$, respectively. Then consider all graphs $G_{i,j,\ell}$ induced by $X_i \cup Y_j \cup Z_\ell$, where we have deleted all edges between $X_i$ and $Z_\ell$. We preprocess $G_{i,j,\ell}$ with the distance oracle, and query the oracle for estimates $d(x,z) \leq \widetilde{d}(x,z) \leq k \cdot d(x,z)$ for all pairs $(x,z) \in (X_i \times Z_\ell) \cap E$. We call a pair $(x,z)$ with estimate $\widetilde{d}(x,z) \leq 2k$ a *candidate* pair. The algorithm enumerates all candidate pairs $(x,z)$ and all neighbors $y$ of $x$, tests whether $(x,y,z)$ forms a triangle (in the original graph) and reports the triangle in the positive case. It is easy to see that the reduction is correct:

**Lemma 5.38 (Correctness of Algorithm 5.6).** *The reduction in Algorithm 5.6 correctly lists all triangles in the given graph $G = (X, Y, Z, E)$.*

**Proof.** ▶ First note that whenever the algorithm reports a triangle $(x,y,z)$, we have verified in Lines 5, 8 and 9 that all edges $(x,y)$, $(y,z)$, $(x,z)$ are present.

Next, focus on any triangle $(x,y,z)$ in $G$; we prove that it is reported by the algorithm. Clearly there exist $i \in [s]$, $j \in [t]$, $\ell \in [s]$ such that $x \in X_i$, $y \in Y_j$, $z \in Z_\ell$. Focus on the iteration of the loop in Line 2 with $(i,j,\ell)$ and on the iteration of the inner loop in Line 5 with $(x,z)$. The distance oracle is queried to obtain a distance estimate $\widetilde{d}(x,z) \leq k \cdot d(x,z)$ for the distance of $x$ and $z$ in $G_{i,j,\ell}$. Note that $x$ and $z$ are connected by a 2-path via $y$, hence the distance estimate satisfies $\widetilde{d}(x,z) \leq 2k$ (that is, $(x,z)$ is indeed a candidate pair). It follows that we enter the loop in Line 8 and report $(x,y,z)$ in Line 10. ◀

The more interesting part of the proof is to bound the running time of the reduction. For the analysis, we first analyze the sizes and degrees of the graphs $G_{i,j,\ell}$. It is easy to see that all bounds are true in expectation, and the high probability bounds follow from Chernoff's bound.

**Lemma 5.39 (Size of $G_{i,j,\ell}$).** *With high probability the following bounds hold for all $(i,j,\ell) \in [s] \times [t] \times [s]$:*

- $|X_i|, |Z_\ell| \leq O(n/s)$ *and* $|Y_j| \leq O(n/t)$.
- $|(X_i \times Y_j) \cap E|, |(Y_j \times Z_\ell) \cap E| \leq O(n^{3/2}/st)$ *and* $|(X_i \times Z_\ell) \cap E| \leq O(n^{3/2}/s^2)$.
- *The degree of any vertex $x \in X_i$ in $G_{i,j,\ell}$ is $O(n^{1/2}/t)$.*

**Lemma 5.40 (Few Candidates).** *Fix $i, j, \ell \in [s] \times [t] \times [s]$. In expectation, the number of candidate pairs $(x,z) \in (X_i \times Z_\ell) \cap E$ is at most*

$$\widetilde{O}\left(\frac{n^{k+1/2}}{s^{k+1}t^k}\right).$$

**Proof.** ▶ Since each candidate pair $(x,z)$ has distance $d(x,z) \leq 2k$ in $G_{i,j,\ell}$, $x$ and $z$ must be connected by a path of length $2k' \leq 2k$ in $G_{i,j,\ell}$. It follows that $(x,z)$ is part of a cycle of (odd) length $2k'+1$ in the induced subgraph $G[X_i \cup Y_j \cup Z_\ell]$. So fix any cycle in $G$ of length $2k'+1$ which uses exactly one edge between $X$ and $Z$. In this case the cycle has exactly $k'+1$ vertices in $X \cup Z$ and exactly $k'$ vertices in $Y$. The probability that this cycle is also contained in $G[X_i \cup Y_j \cup Z_\ell]$ is therefore at most $(1/s)^{k'+1}(1/t)^{k'}$. Since the total number of $(2k'+1)$-cycles in $G$ is at most $O(n^{k'+1/2})$, we obtain the claimed bound on the expected number of candidate pairs $(x,z)$:

$$\sum_{k'=1}^{k} O\left(\frac{n^{k'+1/2}}{s^{k'+1}t^{k'}}\right) \leq O\left(\frac{n^{k+1/2}}{s^{k+1}t^k}\right),$$

where the last inequality holds by $s, t \leq n^{1/2}$. ◀

**Lemma 5.41 (Running Time of Algorithm 5.6).** *With high probability, Algorithm 5.6 runs in expected time*

$$\widetilde{O}\left(s^2 t \cdot \left(\left(\frac{n^{3/2}}{st}\right)^{1+p} + \frac{n^{3/2}}{s^2} \cdot \left(\frac{n^{3/2}}{st}\right)^{q} + \frac{n^{k+1}}{s^{k+1}t^{k+1}}\right)\right).$$

*Moreover, if $kp+(k+1)q < 1$ then we can optimize $s$ and $t$ such that the time bound becomes truly subquadratic.*

**Proof.** ▶ We can construct the partitions $X_1, \ldots, X_s, Y_1, \ldots, Y_t$ and $Z_1, \ldots, Z_s$ in time $O(n)$ and prepare the graphs $G_{i,j,\ell}$ in time $O(n^{3/2}s + n^{3/2}t)$ by a single pass over the edge set.

The algorithm runs for $s^2t$ iterations of the outer loop; focus on one such iteration $i, j, \ell$. Preprocessing $G_{i,j,\ell}$ with the distance oracle takes time $\widetilde{O}((n^{3/2}/st)^{1+p})$. Then we issue $O(n^{3/2}/s^2)$ queries, each running in time $\widetilde{O}((n^{3/2}/st)^q)$. There are at most $O(n^{k+1/2}s^{-k-1}t^{-k})$ candidate pairs in expectation by Lemma 5.40, and only for those we pass the condition Line 7. Executing the inner-most loop in Lines 8 to 10 takes time proportional to the degre of $x$ in $G_{i,j,\ell}$, that is, time $O(n^{1/2}/t)$. Summing all contributions, the expected running time becomes:

$$\widetilde{O}\left(s^2t \cdot \left(\left(\frac{n^{3/2}}{st}\right)^{1+p} + \frac{n^{3/2}}{s^2} \cdot \left(\frac{n^{3/2}}{st}\right)^q + \frac{n^{k+1}}{s^{k+1}t^{k+1}}\right)\right).$$

We now prove that if $kp + (k+1)q < 1$, then the running time becomes subquadratic for some appropriate choice of $s$ and $t$. Let $\epsilon > 0$ be a small constant to be specified later, and set

$$s = n^{1/2 - \frac{p}{2-2p-2q} - \epsilon},$$
$$t = n^{1/2 - \frac{q}{2-2p-2q} - \epsilon}.$$

We analyze the three contributions of the running time in isolation. The first term (i.e., the contribution of the preprocessing time) is

$$\widetilde{O}(n^{2 - \frac{p}{2-2p-2q} - \epsilon + p(1/2 + \frac{p+q}{2-2p-2q} + \epsilon)}) = \widetilde{O}(n^{2 - \frac{p}{2-2p-2q} - \epsilon + p(\frac{1}{2-2p-2q} + \epsilon)}) = \widetilde{O}(n^{2-\epsilon(1-p)}).$$

This is subquadratic for any choice of $\epsilon > 0$ as we assume that $p < 1$. The second term (i.e., the contribution of the query time) similarly becomes subquadratic:

$$\widetilde{O}(n^{2 - \frac{q}{2-2p-2q} - \epsilon + q(1/2 + \frac{p+q}{2-2p-2q} + \epsilon)}) = \widetilde{O}(n^{2 - \frac{q}{2-2p-2q} - \epsilon + q(\frac{1}{2-2p-2q} + \epsilon)}) = \widetilde{O}(n^{2-\epsilon(1-q)}).$$

For the third term (i.e., the contribution of testing all candidate pairs) we obtain the following bound:

$$\widetilde{O}(n^{3/2 + \frac{p(k-1)}{2-2p-2q} + \frac{qk}{2-2p-2q} + \epsilon(2k-1)}) = \widetilde{O}(n^{3/2 + \frac{kp+(k-1)q-p-q}{2-2p-2q} + \epsilon(2k-1)}).$$

By the same assumption that $kp + (k+1)q < 1$, the exponent becomes strictly smaller than 2 when ignoring the contribution of $\epsilon$. Therefore, a sufficiently small choice of $\epsilon > 0$ achieves truly subquadratic running time. ◀

### 5.7.2 Stretch $2 \leq \alpha < 3$

In this section we prove the following theorem:

**Theorem 1.10 (Hardness of Distance Oracles with Stretch $2 \leq \alpha < 3$).** *For any $2 \leq \alpha < 3$ and $\epsilon > 0$, in sparse graphs there is no distance oracle with stretch $\alpha$, query time $n^{o(1)}$ and preprocessing time $O(m^{1+\frac{2}{1+\alpha}-\epsilon})$, unless the 3-SUM conjecture fails.*

We use a powerful gadget which was already used in the conditional space lower bounds by Pătraşcu, Roditti and Thorup [171]: *Butterfly graphs*. We first define the butterfly graph and then give quick proofs for the properties relevant for our reduction.

**Algorithm 5.7.** The reduction from listing triangles in a $\Theta(r)$-regular tripartite graph $G = (X, Y, Z, E)$ to approximate distance oracles with stretch $\alpha$.

```
1  Randomly split Y into Y_1, ..., Y_t
2  for each j ∈ [t] do
3      Let G_j be the following graph: Add the vertices X_i and Z_ℓ, and add a copy
         of the butterfly graph with alphabet σ and dimension d for each vertex
         in Y_j. For each (x, y) ∈ (X × Y_j) ∩ E, add an edge from x to a random
         vertex in the left layer of the butterfly graph corresponding to y, and
         similarly for each (y, z) ∈ (Y_j × Z) ∩ E, add an edge from z to a random
         vertex in the right layer of the butterfly graph corresponding to y
4      Preprocess G_j with the approximate distance oracle
5      for each (x, z) ∈ (X × Z) ∩ E do
6          Query the distance oracle to get an estimate d̃(x, z) satisfying
             d(x, z) ≤ d̃(x, z) ≤ α · d(x, z)
7          if d̃(x, z) ≤ α · (d + 2) then
8              for each y ∈ Y_j with (x, y) ∈ E do
9                  if (y, z) ∈ E then
10                     Report the triangle (x, y, z)
```

**Definition 5.42 (Butterfly Graph).** *The butterfly graph with alphabet $\sigma$ and dimension $d$ is the $(d+1)$-partite graph with vertex sets $[\sigma]^d \times [d+1]$, and edges*

$$\Big\{ \big( (s, i), (s', i+1) \big) : s[j] = s'[j] \text{ for all } j \in [d], j \neq i \Big\}.$$

*That is, two vertices $(s, i)$ and $(s', i+1)$ are connected by an edge if and only if the length-$d$ string $s$ equals $s'$ in all positions except $i$ (where it might be or might not be equal). We call the vertices $(s, i)$ the $i$-*th layer*, and we occasionally call the 1-st layer the* left layer *and the $(d+1)$-st layer the* right layer*.*

In particular, we remark that the butterfly graph with alphabet $\sigma$ and dimension $d$ has $(d+1)\sigma^d$ vertices and $d\sigma^{d+1}$ edges.

**Lemma 5.43 (Butterfly Graph).** *Focus on the butterfly graph with alphabet $\sigma$ and dimension $d$. Then:*

- Left to right: *The distance from any vertex in the left layer to any vertex in the right layer is exactly $d$.*
- Left to left: *The probability that two* random *vertices in the left layer have distance at most $2d - 2\ell$ is at most $\sigma^{-\ell}$.*

**Proof.** ▶ Observe that in the butterfly graph, exactly the edges from the $i$-th to the $(i+1)$-st layer can change the $i$-th position of the strings. This makes the first property obvious: For any two vertices $(s, 1)$ and $(s', d+1)$, follow the unique path which corrects the mismatches between $s$ and $s'$ in positions $1, 2, \ldots, d$.

For the second property, let $(s, 1)$ and $(s', 1)$ be two random vertices in the left layer, i.e., let $s$ and $s'$ be random strings in $[\sigma]^d$. The distance between $(s, 1)$ and $(s', 1)$ is exactly two times the largest $i$ for which $s[i] \neq s'[i]$, as we have to reach the $i$-th layer in the butterfly in order to change $s[i]$ into $s'[i]$. Hence, they have distance at most $2d - 2\ell$ only if $s$ equals $s'$ in the last $\ell$ positions. Since $s$ and $s'$ are random strings, this happens with probability at most $\sigma^{-\ell}$. ◀

With this gadget in mind, we are ready to state the reduction, see Algorithm 5.7. Let $d = \lceil \max(32/\epsilon, \frac{4}{3-\alpha}) \rceil$. Using Lemma 5.33 we start from a $\Theta(r)$-regular $n$-vertex graph (for some parameter $r \geq n^{1/2}$ to be fixed later) which contains at most $O(n^{k/2})$ $k$-cycles, for all $k \geq 4$, and will list $O(nr)$ triangles in time $O(nr^{2-\delta})$. Let $t \leq r^{1-\Omega(1)}$ be another parameter, and let $\sigma = r^{1/d}$.

The reduction is very similar to the one in the previous section, except that we only split the vertex set $Y$ (in the language of the previous section we have $s = 1$)

and that we construct the graphs $G_j$ differently: The difference is that we replace every vertex in $Y_j$ by a copy of the butterfly gadget. The edges from $X$ are connected to a random vertex in the left layer, and the edges from $Z$ are connected to a random vertex in the right layer. Notice that thereby two vertices $x, z$ which are connected by a 2-path via some vertex $y$ in the original graph, are now connected via a $(d+2)$-path which traverses the butterfly gadget from left to right.

We preprocess each graph $G_j$ with the distance oracle, and then query all edges $(x,z) \in (X \times Z) \cap E$ to get a distance estimate $d(x,z) \leq \widetilde{d}(x,z) \leq \alpha d(x,z)$. We say that a pair $(x,z)$ is a *candidate* pair if the $\widetilde{d}(x,z) \leq \alpha(d+2)$. Note that only candidate pairs can be part of a triangle, and we therefore enumerate all candidate pairs $(x,z)$ and all neighbors $y \in Y_j$ of $x$ and test whether $(x,y,z)$ forms a triangle.

We start to analyze the size of the graphs $G_j$. Note that we have to take care of the additional vertices and edges added by the butterfly gadgets.

**Lemma 5.44 (Size of $G_j$).** *With high probability, the following bounds hold for all $j \in [t]$: The graph $G_j$ has $O(n + nr/t)$ vertices and $O(nr^{1+1/d}/t)$ edges, and the degree of any vertex $x \in X$ is bounded by $O(r/t)$.*

**Proof.** ▶ For the degree bound the butterfly gadgets play no role and the proof is the same as in the last section using Chernoff's bound. For the number of vertices, first recall that with high probability there are $O(n/t)$ vertices in $Y_j$. Since each vertex in $Y_j$ is replaced by a butterfly graph of size $O(d\sigma^d) = O(r)$, the bound on the vertices is correct. Moreover, each butterfly graph contributes $d\sigma^{d+1} = O(r^{1+1/d})$ additional edges and therefore also the bound on the edges is as claimed. ◀

**Lemma 5.45 (Few Candidates).** *Fix $j \in [t]$. In expectation, the expected number of candidate pairs $(x,z) \in (X \times Z) \cap E$ is at most*

$$O\left(\frac{r^{3d+\frac{3+\alpha}{2}+\frac{4}{d}}}{t^{3d}}\right).$$

**Proof.** ▶ First note that there is a natural correspondence between paths from $x$ to $z$ in the original graph (which we will call *original paths*) and paths in the constructed graph $G_j$ which take the shortest route through the butterfly gadgets (which we will call *inflated paths*).

Observe that any inflated path from $x$ to $z$ of length at most $\alpha \cdot (d+2)$ must be separable into a path which zigzags between $X$ and the butterfly gadgets, followed by a path which zigzags between the butterfly gadgets and $Z$. Any other inflated path would pass through at least three butterfly gadgets (once by traveling from $X$ to $Z$, once by traveling back to $X$ and once more by traveling to $Z$ to reach the final destination $z$) which would require length $3d+2$. Since we set $d > \frac{4}{3-\alpha}$, we have the inequality $\alpha \cdot (d+2) < 3d+2$ which leads to a contradiction.

Any such inflated path originates from a $2k$-path in the original graph $G$ (for some $k \leq 3d$) that first zigzags between $X$ and $Y_j$, and then zigzags between $Y_j$ and $Z$. (In particular, for exactly $k-1$ times the path reaches the vertex part $Y_j$ without crossing to the other side from $X$ to $Z$ or vice versa.) Since the edge from $x$ to $z$ is also present by assumption, this closes a cycle of length $2k+1$ in the original graph.

As we have a good bound on the number of such cycles (namely, $O(r^{2k+1})$ many), our strategy is to prove that each cycle becomes a short inflated path only with small probability. First of all, any original $2k$-path as the one described survives only with probability at most $t^{-k}$ in the induced graph $G[X \cup Y_j \cup Z]$. But even if a path survives, we claim that it leads to a short inflated path only with small probability. Since the path has to traverse $k-1$ butterfly gadgets from left to left (or right to right), we expect the path to have length $2k + d + (k-1) \cdot 2d$ ($2k$ steps in the original path plus $d$ steps to cross through one butterfly gadget plus $(k-1) \cdot 2d$ because of the remaining butterfly gadgets). Using Lemma 5.43,

the probability that it has length at most $2k + d + (k-1) \cdot 2d - 2\ell$ is therefore at most

$$\sum_{\substack{\ell_1,\dots,\ell_{k-1} \in \mathbf{Z} \\ \ell_1+\dots+\ell_{k-1}=\ell}} \sigma^{-\ell_1} \cdot \ldots \cdot \sigma^{-\ell_{k-1}} = \sum_{\substack{\ell_1,\dots,\ell_{k-1} \in \mathbf{Z} \\ \ell_1+\dots+\ell_{k-1}=\ell}} \sigma^{-\ell} = O(\sigma^{-\ell}).$$

Here we hide in the $O$-notation a constant which only depends on $k$ and $\ell$, both of which are functions of $d$ and thereby constants for us.

Hence, for $\ell = \lfloor k + \frac{d}{2} + (k-1)d - \frac{\alpha \cdot (d+2)}{2} \rfloor$ the probability that the inflated path has length at most $\alpha \cdot (d+2) \leq 2k + d + (k-1) \cdot 2d - 2\ell$ is at most

$$\begin{aligned}
&O(\sigma^{-\ell}) \\
&\quad \leq O(\sigma^{-k-\frac{d}{2}-(k-1)d+\frac{\alpha\cdot(d+2)}{2}+1}) \\
&\quad \leq O(\sigma^{-d(\frac{1}{2}+k-1-\frac{\alpha}{2})+4}) \\
&\quad \leq O(\sigma^{-d(k-\frac{\alpha+1}{2})+4}) \\
&\quad \leq O(r^{-k+\frac{\alpha+1}{2}+\frac{4}{d}}).
\end{aligned}$$

By combining the arguments from the previous paragraphs, we obtain that each $2k+1$ cycle survives only with probability $t^{-k}$ and (independently) becomes a short inflated path with probability at most $O(r^{-k+\frac{\alpha+1}{2}+\frac{4}{d}})$. Since the total number of $(2k+1)$-cycles in the original graph is $O(r^{2k+1/2})$, we obtain the claimed bound on the expected number of candidate pairs:

$$O\left(\sum_{k=2}^{3d} \frac{r^{2k+1} r^{-k+\frac{\alpha+1}{2}+\frac{4}{d}}}{t^k}\right) = O\left(\sum_{k=2}^{3d} \frac{r^{k+\frac{3+\alpha}{2}+\frac{4}{d}}}{t^k}\right) \leq O\left(\frac{r^{3d+\frac{3+\alpha}{2}+\frac{4}{d}}}{t^{3d}}\right).$$

For the last inequality we have used that $t \leq r$. ◀

**Proof of Theorem 1.10.** ▶ We pick $r = n^{\frac{2}{1+\alpha}\cdot(1-\delta)}$ and $t = r^{1-\gamma}$, for some $\gamma, \delta > 0$ to be picked later. Recall that we set $d = \lceil \max(32/\epsilon, \frac{4}{3-\alpha}) \rceil$ and $\sigma = r^{1/d}$. The correctness proof should be clear from the in-text explanations. It remains to analyze the running time with respect to this choice of parameters. Recall that we aim for a running time of the form $(nr^2)^{1-\Omega(1)}$.

First, consider the contribution of querying the distance oracle: Issuing $O(tnr)$ queries, each running in subpolynomial time, takes time $O(nr^{2-\gamma+o(1)})$. Next, consider the contribution of explicitly testing whether an edge $(x, z)$ is part of a triangle, that is, the running time of the inner-most loop Line 8. By the previous lemma we pass the condition in Line 7 at most

$$t \cdot O\left(\frac{r^{3d+\frac{3+\alpha}{2}+\frac{4}{d}}}{t^{3d}}\right)$$

times and each call runs in time $O(r/t)$. Therefore, the total time for this step becomes

$$\begin{aligned}
&t \cdot O\left(\frac{r^{3d+\frac{3+\alpha}{2}+\frac{4}{d}}}{t^{3d}} \cdot \frac{r}{t}\right) \\
&\quad = O\left(r^{\frac{1+\alpha}{2}} \cdot r^{2+3d\gamma+\frac{4}{d}}\right) \\
&\quad = O\left(n^{1-\delta} r^{2+3d\gamma+\frac{4}{d}}\right) \\
&\quad = O\left(nr^{2+3d\gamma+\frac{4}{d}-\delta}\right).
\end{aligned}$$

Finally, we need to consider the preprocessing time of the distance oracles. Recall that each graph $G_j$ has $O(nr/t)$ vertices and $O(nr^{1+1/d}/t)$ edges. Assuming that the

**Algorithm 5.8.** The reduction from listing triangles in a $\Theta(n^{1/2})$-regular tripartite graph $G = (X, Y, Z, E)$ to dynamic approximate distance oracles with stretch $2k - 1$.

```
1  Randomly split Y, Z into Y_1, ..., Y_t, Z_1, ..., Z_s
2  for each (j, ℓ) ∈ [t] × [s] do
3      Let G_{j,ℓ} be the subgraph of G induced by Y_j ∪ Z_ℓ, where we subdivide each
         edge into a path of length 10k (equivalently, think of this path as an
         edge of weight 10k), and add an isolated vertex v
4      Preprocess G_{j,ℓ} with the dynamic approximate distance oracle (i.e., add
         the edges one by one)
5      for each x ∈ X do
6          Add an edge (v, y) for each neighbor y ∈ Y_j of x
7          for each z ∈ Z_ℓ with (x, z) ∈ E do
8              Query the distance d(v, z) ≤ d̃(v, z) ≤ (2k − 1) · d(v, z)
9              if d̃(v, z) ≤ (2k − 1) · (10k + 1) then
10                 for each y ∈ Y_j with (x, y) ∈ E do
11                     if (y, z) ∈ E then
12                         Report the triangle (x, y, z)
13         Delete all edges incident to v
```

preprocesing time of the distance oracle is $O(m^{1+\frac{2}{1+\alpha}-\epsilon})$ as in the theorem statement, the total preprocessing time is bounded by

$$\begin{aligned}
&t \cdot O\left(\left(\frac{nr^{1+1/d}}{t}\right)^{1+\frac{2}{1+\alpha}-\epsilon}\right) \\
&\quad \leq O(r(nr^{1/d+\gamma})^{1+\frac{2}{1+\alpha}-\epsilon}) \\
&\quad \leq O(r^{1+(\frac{1}{d}+\gamma)\cdot(1+\frac{2}{1+\alpha}-\epsilon)} n^{1+\frac{2}{1+\alpha}-\epsilon}) \\
&\quad \leq O(r^{1+\frac{2}{d}+2\gamma} r^{\frac{1}{1-\delta}} n^{1-\epsilon}) \\
&\quad \leq O(r^{2+\frac{2}{d}+2\gamma+2\delta-\epsilon} n).
\end{aligned}$$

We pick $\delta = \epsilon/4$, $d = \lceil \max(32/\epsilon, \frac{4}{3-\alpha}) \rceil$ (as announced before), and let $\gamma > 0$ be tiny enough. Then both contributions to the running time become $O(nr^{2-\Omega(1)})$. This contradicts the 3-SUM hypothesis by Lemma 5.33. ◀

### 5.7.3 Dynamic Distance Oracles

In contrast to the previous sections, we now consider *dynamic* distance oracles. That is, we expect the distance oracle to compute distance estimates while the graph undergoes edge insertions and deletions.

**Theorem 1.11 (Hardness of Dynamic Distance Oracles).** *For any integer constant $k \geq 2$, there is no dynamic approximate distance oracle with stretch $2k - 1$, update time $O(m^u)$ and query time $O(m^q)$ with $ku + (k + 1)q < 1$, unless the 3-SUM conjecture fails.*

We again prove the theorem by a reduction from listing $O(n^{3/2})$ triangles in a $\Theta(n^{1/2})$-regular $n$-vertex graph $G = (X, Y, Z, E)$ which contains at most $O(n^{k'/2})$ $k'$-cycles for all $k' \geq 4$ (that is, we use the conditional hardness result from Theorem 5.1).

Let $s, t \leq n^{1/2-\Omega(1)}$ be two parameters. The reduction is given in Algorithm 5.8. Our analysis is very similar to the analysis in the previous two sections, and we will therefore omit some details. It is easy to prove that the algorithm reports all triangles in $G$ and is therefore correct. The critical part is to analyze the running time. To this end, we first check the size of the graphs $G_{j,\ell}$. Note that the number

of vertices in $G_{j,\ell}$ is dominated by the vertices edges added to the graph by the subdivision of edges into paths.

**Lemma 5.46 (Size of $G_{i,j}$).** *With high probability the following bounds hold for all $(j,\ell) \in [t] \times [s]$:*

- *The graph $G_{j,\ell}$ has $O(n^{3/2}/st)$ vertices and edges.*
- *The degree of any vertex $z \in Z_\ell$ in $G_{j,\ell}$ is $O(n^{1/2}/t)$.*

We call a pair $(x,z)$ a *candidate* pair if, in the $x$-iteration of the loop in Line 5, the distance estimate $\widetilde{d}(v,z)$ satisfies $\widetilde{d}(v,z) \leq (2k-1)(10k+1)$. That is, the condition in Line 9 is satisfied only for candidate pairs.

**Lemma 5.47 (Few Candidates).** *Fix $j, \ell \in [t] \times [s]$. In expectation, the number of candidate pairs $(x,z) \in (X \times Z_\ell) \cap E$ is at most*

$$O\left(\frac{n^{k+1/2}}{s^k t^k}\right).$$

**Proof.** ▶ Focus on a candidate pair $(x,z)$. There must be a neighbor $y \in Y_j$ of $X$ (in the original graph) such that $y$ and $z$ have distance $d(y,z) \leq (2k-1)(10k+1)-1$ in $G_{j,\ell}$. A shortest $y$-$z$-path can therefore zigzag at most $2k-1$ times between $Y_j$ and $Z_\ell$, as otherwise it would have length at least $2k \cdot 10k > (2k-1)(10k+1)-1$.

Therefore, any candidate pair $(x,z)$ is part of a cycle of length at most $2k'+1 \leq 2k+1$ in the original graph $G$. For fixed $j, \ell$, the probability that any $(2k'+1)$-cycle survives in the induced subgraph $G[X \cup Y_j \cup Z_\ell]$ is at most $s^{-k'}t^{-k'}$. Therefore, using that in $G$ there are at most $O(n^{k'+1/2})$ cycles of length $2k'+1$, we obtain the claimed bound on the number of candidate pairs:

$$\sum_{k'=1}^{k} O\left(\frac{n^{k'+1/2}}{s^{k'}t^{k'}}\right) \leq O\left(\frac{n^{k+1/2}}{s^k t^k}\right),$$

where the last inequality holds by $s, t \leq n^{1/2}$. ◀

**Proof of Theorem 1.11.** ▶ We run the reduction in Algorithm 5.8. We omit the correctness proof which is similar to the previous sections, and focus on the running time. We set

$$s = n^{\frac{1}{2} - \frac{u}{2-2u-2q} - \gamma},$$
$$t = n^{1/2 - \frac{q}{2-2u-2q} - \gamma},$$

for some small $\gamma > 0$ to be determined later. There are three major contributions to the running time.

First, the time to preprocess the graphs $G_{j,\ell}$ (via adding all edges one by one) is bounded by $O(st \cdot n^{3/2}/st)$ times the time to perform a single update and therefore negligible. The time to perform the edge insertions and deletions in Lines 6 and 13 is bounded by

$$\begin{aligned} &O\left(st \cdot n \cdot \frac{n^{1/2}}{t} \cdot \left(\frac{n^{3/2}}{st}\right)^u\right) \\ &\quad = O(n^{2 - \frac{u}{2-2u-2q} - \gamma + u \cdot (\frac{1}{2} + \frac{u+q}{2-2u-2q}) + u\gamma}) \\ &\quad = O(n^{2 + \frac{-u+u-u^2-uq+u^2+uq}{2-2u-2q} - \gamma(1-u)}) \\ &\quad = O(n^{2-\gamma(1-u)}), \end{aligned}$$

which is subquadratic for an arbitrarily small $\gamma > 0$. Similarly, the total query time can be bounded by

$$O\left(st \cdot n \cdot \frac{n^{1/2}}{s} \cdot \left(\frac{n^{3/2}}{st}\right)^q\right) = O(n^{2-\gamma(1-q)}).$$

It remains to bound the time spend in the inner-most loop in Line 10. By the previous lemma we pass the condition in Line 9 at most

$$st \cdot O\left(\frac{n^{k+1/2}}{s^k t^k}\right)$$

times, and each execution of the loop body takes time $O(n^{1/2}/t)$. Therefore, the total time spent in the loop is

$$\begin{aligned} & st \cdot O\left(\frac{n^{k+1}}{s^k t^{k+1}}\right) \\ & \quad = O(n^{3/2+\frac{(k-1)u+kq}{2-2u-2q}+(2k-1)\gamma}) \\ & \quad \leq O(n^{3/2+\frac{ku+(k+1)q-u-q}{2-2u-2q}+(2k-1)\gamma}). \end{aligned}$$

By the assumption that $ku + (k+1)q < 1$, the first terms in the exponent is strictly less than 2, and therefore we can set $\gamma > 0$ sufficiently small to achieve subquadratic running time. ◀

# 6 Fast Minimization of Tardy Processing Time via Partition-and-Convolve

In this comparably small chapter we apply the partition-and-convolve design paradigm to design faster-than-brute-force algorithms for a scheduling problem, summarizing the results from the paper [58].

58 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. "Faster minimization of tardy processing time on a single machine". In: *47th international colloquium on automata, languages, and programming (ICALP 2020)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pages 19:1–19:12. 10.4230/LIPIcs.ICALP.2020.19.

**Organization.** We start with an overview in Section 6.1. Our algorithm is split into two parts which we present in Sections 6.2 and 6.3.

## 6.1 Overview

We study the $1||\sum p_jU_j$ problem: In this problem we are given $n$ jobs with *processing times* $p_j \in \mathbf{N}$ and *due dates* $d_j \in \mathbf{N}$. A (single-machine) *schedule* $\sigma$ is a permutation $\sigma : \{0,\dots,n-1\} \to \{0,\dots,n-1\}$, where the *completion time* $C_j$ of a job $j$ under $\sigma$ is given by $C_j = \sum_{\sigma(i)\le\sigma(j)} p_i$, that is, the total processing time of jobs preceding $j$ in $\sigma$ (including $j$ itself). Job $j$ is *tardy* in $\sigma$ if $C_j > d_j$, and *early* otherwise. Our goal is to find a schedule with minimum total processing time of tardy jobs. If we assign a binary indicator variable $U_j$ to each job $j$, where $U_j = 1$ if $j$ is tardy and otherwise $U_j = 0$, our objective function can be written as $\sum p_jU_j$.

The main result of this chapter is the following theorem. Here we set $P = \sum_j p_j$ as the total processing time of all jobs and assume that $n \le P$ (as we can ignore all jobs with processing time 0).

**Theorem 1.13 (Faster Minimization of Tardy Processing Time).** *The* $1||\sum p_jU_j$ *problem can be solved in time* $\widetilde{O}(P^{7/4})$.

To this end, we employ the partition-and-convolve paradigm: We first prove that there is a divide-and-conquer algorithm for $1||\sum p_jU_j$ that partitions the problem into smaller subtasks and recombines the recursive solutions via the following convolution-style problem:

**Definition 1.14 (Skewed Convolution).** *Let* $A, B, S \in \mathbf{Z}^n$. *The* $(\max,\min)$-skewed convolution *of* $A, B, S$ *is defined as the vector* $C \in \mathbf{Z}^{2n-1}$ *with entries*

$$C[k] = \max_{\substack{i,j\in[n]\\ i+j=k}} \min\{A[i], B[j] + S[k]\}.$$

Note that in contrast to the more classical $(\max,\min)$-convolution, we *skew* the right side of the minimum by adding a value $S[k]$ depending on $k$. This property is crucially necessary for us to obtain the following fine-grained reduction:

**Lemma 6.1 (Reduction to Skewed Convolutions).** *If the* $(\max,\min)$*-skewed convolution of length-*$n$ *vectors with maximum entry* $\Delta$ *can be computed in time* $T(n,\Delta)$, *then the* $1||\sum p_jU_j$ *problem can be solved in time* $O(T(P,P)\log n)$.

As the second step, we design an efficient algorithm for computing $(\max,\min)$-skewed convolutions. We remark that the follow-up work by Klein, Polak and Rohwedder directly improves this result to time $\widetilde{O}(n^{5/3})$ [137].

**Lemma 6.2 (Fast Skewed Convolutions).** *The* $(\max,\min)$*-skewed convolution of length-*$n$ *vectors with maximum entry* $\Delta$ *can be computed in time* $\widetilde{O}(n^{7/4}\log\Delta)$.

We prove the Lemmas 6.1 and 6.2 in the next Sections 6.2 and 6.3.

## 6.2 Reduction to Skewed Convolutions

In this section we prove Lemma 6.1. We start with a key observation about the $1||\sum p_j U_j$ problem, used already by Lawler and Moore. Any instance of the problem always has an optimal schedule of a specific type, namely an *Earliest Due Date (EDD)* schedule. An Earliest Due Date schedule is a schedule such that

- any early job precedes all tardy jobs, and
- any early job precedes all early jobs with later due dates.

In other words, in an EDD schedule all early jobs are scheduled before all tardy jobs, and all early jobs are scheduled in non-decreasing order of due dates. The first step in the algorithm is therefore to sort the jobs according to their due dates so that we can assume $d_1 \leq \cdots \leq d_n$ in the following.

Let $J \subseteq [n]$ be a subset of jobs. Write $P(J) = \sum_{j \in J} P_j$. We define a vector $M(J)$, where $M(J)[x]$ equals the latest (that is, maximum) time point $x_0$ for which there is a subset of jobs in $J$ with total processing time equal to $x$ that can be scheduled early in an EDD schedule starting at $x_0$. If no such subset of jobs exists, we define $M(J)[x] = -\infty$. Here we assume that $x$ ranges from 0 to $P(J)$. In order to solve the whole problem, it suffices to compute the length-$P$ vector $M([n])$. We can read off the maximum processing time of early jobs (and thereby the minimum processing time of tardy jobs) as largest $x$ for which $M([n])[x] \geq 0$.

We use a divide-and-conquer approach to compute $M(J)$. As the base case it is easy to compute $M(J)$ for singleton sets $J = \{j\}$: We have $M(\{j\})[x] = d_j - p_j$ if $x = p_j$, and $M(\{j\})[x] = -\infty$ otherwise. For larger sets $J$, we have the following lemma.

**Lemma 6.3 (Divide-and-Conquer Rule).** *Let $J \subseteq [n]$ be an interval, with a partition into subintervals $J = J_1 \cup J_2$. Then, for each $x$ we have*

$$M(J_1 \cup J_2)[x] = \max_{x_1 + x_2 = x} \min\{M(J_1)[x_1], M(J_2)[x_2] - x_1\}.$$

**Proof.** ▶ $M(J)[x]$ is the latest time point after which a subset of jobs $J^* \subseteq J$ of total processing time $x$ can be scheduled early in an EDD schedule. Let $x_1$ and $x_2$ be the total processing times of jobs in $J_1^* = J^* \cap J_1$ and $J_2^* = J^* \cap J_2$, respectively. Then $x = x_1 + x_2$. Clearly, $M(J)[x] \leq M(J_1)[x_1]$, since we have to start scheduling the jobs in $J_1^*$ at time $M(J_1)[x_1]$ by latest. Similarly, it holds that $M(J)[x] \leq M(J_2)[x_2] - x_1$ since the jobs in $J_2^*$ are scheduled at time $M(I_2)[x_2]$ by latest and the jobs in $J_1^*$ have to be processed before that time point in an EDD schedule. In combination, we have shown that LHS $\leq$ RHS in the equation of the lemma.

To prove that LHS $\geq$ RHS, we construct a feasible schedule for jobs in $J$ starting at RHS. Let $x_1$ and $x_2$ be the two values with $x_1 + x_2 = x$ that maximize RHS. Then there is a schedule which schedules some jobs $J_1^* \subseteq J_1$ of total processing time $x_1$ beginning at time $\min\{M(J_1)[x_1], M(J_2)[x_2] - x_1\} \leq M(J_1)[x_1]$, followed by another subset of jobs $J_2^* \subseteq J_2$ of total processing time $x_2$ starting at time $\min\{M(J_1)[x_1], M(J_2)[x_2] - x_1\} + x_1 \leq M(J_2)[x_2]$. This is a feasible schedule starting at time RHS for a subset of jobs in $J$ which has total processing time $x$. ◀

Note that the equation given in Lemma 6.3 is close but not precisely the equation defined in Definition 1.14 for the $(\min, \max)$-Skewed-Convolution problem. Nevertheless, the next lemma shows that we can easily translate between these two concepts.

**Lemma 6.4.** *Assume that we can compute the $(\max, \min)$-skewed convolution of length-$n$ vectors with maximum entry $\Delta$ in time $T(n, \Delta)$. Let $J = J_1 \cup J_2$ be as in Lemma 6.3. Given $M(J_1)$ and $M(J_2)$, we can compute $M(J)$ in time $O(T(P(J), P(J)))$.*

**Algorithm 6.1.** Computes the vector $M(J)$ for a given interval $J \subseteq [n]$.

**if** $J = \{j\}$ **then**
    **return** the vector with $M(\{j\})$ with

$$M(\{j\})[x] = \begin{cases} d_j - p_j & \text{if } x = p_j, \\ -\infty & \text{otherwise.} \end{cases}$$

**else**
    Partition $J = J_1 \cup J_2$ into subintervals of equal size (up to $\pm 1$)
    Recursively compute $M(J_1)$ and $M(J_2)$
    **return** $M(J)$ as computed by Lemma 6.4

**Proof.** ▶ We construct auxiliary vectors $A, B$ and $S$ defined by $A[x] = M(J_2)[x] + x$ and $B[x] = M(J_1)[x]$ and $S[x] = x$ for each entry $x$. Compute the $(\max, \min)$-skewed convolution of $A, B, S$, and let $C$ denote the resulting vector. We return the vector $M(J)[x] = C[x] - x$ and claim that this is correct. Indeed, we have

$$\begin{aligned} C[x] - x &= \max_{x_1+x_2=x} \min\{A[x_1], B[x_2] + S[x]\} - x \\ &= \max_{x_1+x_2=x} \min\{M(J_2)[x_1] + x_1, M(J_1)[x_2] + x\} - x \\ &= \max_{x_1+x_2=x} \min\{M(J_2)[x_1] - x_2, M(J_1)[x_2]\} \\ &= \max_{x_1+x_2=x} \min\{M(J_1)[x_1], M(J_2)[x_2] - x_1\} \\ &= M(J_1 \cup J_2)[x], \end{aligned}$$

where in the second step we expanded the definition of $A, B, S$, in the second-to-last step we used the symmetry of $x_1$ and $x_2$, and in the last step we have applied Lemma 6.3.

For the running time, observe that the vectors $A, B, S$ have entries $0, \ldots, P(J)$ or $-\infty$. We can replace $-\infty$ be a sufficiently negative number of magnitude $O(P)$, without changing the relevant outputs. The computation of $C$ therefore amounts to computing the $(\max, \min)$-skewed convolution of length-$P(J)$ vectors with maximum entry $O(P(J))$, which runs in time $O(T(P(J), P(J)))$. ◀

**Proof of Lemma 6.1.** ▶ We compute the vector $M([n])$ as outlined before. For singleton sets, we can prepare $M(\{j\})$ in one shot. For larger intervals $J$, we partition $J$ into equal-sized intervals $J = J_1 \cup J_2$, recur to compute $M(J_1)$ and $M(J_2)$ and recombine these vectors into $M(J)$ using Lemma 6.4. The pseudocode is given in Algorithm 6.1. After computing $M([n])$, we return the largest value $0 \le x \le P$ such that $M([n])[x] \ge 0$. The correctness of this algorithm is immediate.

The running time is dominated by Algorithm 6.1. We argue that this algorithm runs in time $O(T(P, P) \log n)$. Indeed, the recursion tree reaches depth $O(\log n)$, and on each level of the recursion the running time is dominated by the calls to Lemma 6.4. Indeed, as each level induces as a partition $[n] = J_1 \cup \cdots \cup J_k$, the total running time is $\sum_{i=1}^{k} O(T(P(J_i), P(J_i))) \le O(T(P, P))$. ◀

We remark that this reduction from $1||\sum p_j U_j$ to computing $(\max, \min)$-skewed convolutions was improved by Schieber and Sitaraman in a recent arXiv preprint [188]. They prove that if the latter problem can be solved in time $\widetilde{O}(n^\alpha)$, the former problem can be solved in time $\widetilde{O}(P^{2-1/\alpha})$.

## 6.3 Fast Skewed Convolutions

In the following section we present our algorithm for $(\max, \min)$-skewed convolution, and provide a proof of 6.2. Throughout, let $A, B, S$ denote the input vectors of length $n$. Recall we wish to compute the vector $C$ defined by

$$C[k] = \max_{i+j=k} \min\{A[i], B[j] + S[k]\}.$$

We present a version of our algorithm that slightly differs from conference version, but for which it is easier to see how Klein, Polak and Rohwedder obtained their improved algorithm.

**Step 0: Ensure Distinct Entries.** As a preliminary step, we will ensure that the vectors $A$ and $B$ do not contain duplicate entries. We can achieve this by replacing the three vectors by $A_0[i] = n \cdot A[i] + i$, $B_0[j] = n \cdot B[j] + j$ and $S_0[k] = n \cdot S[k]$. Clearly $A'$ and $B'$ have distinct entries, and moreover we can read off the $(\max, \min)$-skewed convolution of $A, B, S$ from the $(\max, \min)$-skewed convolution of $A_0, B_0, C_0$ by dividing each coordinate by $n$ (without remainders).

**Step 1: Rank-Approximation.** Let $a_0 < \cdots < a_{n-1}$ denote the entries of $A_0$ in sorted order. For an integer $x$, we fix the *rank* $\operatorname{rank}(x)$ as the smallest index $\ell$ such that $x \leq a_\ell$. Moreover, let $r$ be a parameter and let $p_0 = a_0, p_1 = a_r, p_2 = a_{2r}, \ldots$; we call these elements the *pivot* elements.

The idea is to compute the vector $A_1$ obtained from $A_0$ by replacing each entry $A_0[i]$ by the smallest pivot larger than $A_0[i]$. This results in a vector where for each coordinate $i$, $A_0[i]$ and $A_1[i]$ have distance at most $r$ in terms of their ranks, and $A_1$ has at most $\lceil n/r \rceil$ distinct entries. We compute the $(\max, \min)$-skewed convolution of $A_1, B_0, C_0$, and argue in the following lemma that is possible to recover from the errors introduced in this way.

**Lemma 6.5 (Correcting the Rank-Approximation).** *Given the* $(\max, \min)$*-skewed convolution* $C_1$ *of* $A_1, B_0, S_0$*, we can compute the* $(\max, \min)$*-skewed convolution* $C_0$ *of* $A_0, B_0, S_0$ *in time* $O(nr)$*.*

**Proof.** ▸ We compute each coordinate $C_0[k]$ separately in time $O(r)$. There are two cases: If the value $C_0[k] = \max_{i+j=k} \min\{A_0[i], B_0[j] + S_0[k]\}$ is attained as the RHS of the minimum, then we have $C_0[k] = C_1[k]$ (as $A_0[i] \leq A_1[i]$ for all coordinates $i$). So assume that $C_0[k]$ is attained as the LHS of the minimum. In this case, we claim that $\operatorname{rank}(C_1[k]) - \operatorname{rank}(C_0[k]) \leq r$. Indeed, recall that for each fixed $i, j$ going from $\min\{A_0[i], B_0[j] + S_0[k]\}$ to $\min\{A_1[i], B_0[j] + S_0[k]\}$ can increase the rank by at most $\operatorname{rank}(A_1[i]) - \operatorname{rank}(A_0[i]) \leq r$. Therefore, it suffices to compute $C_0[k]$ via:

$$C_0[k] = \max\left\{ C_1[k], \max_{\substack{i \in [n] \\ |\operatorname{rank}(A_0[i]) - \operatorname{rank}(C_1[k])| \leq r}} \min\{A_0[i], B[k-i] + S[k]\} \right\}.$$

After precomputing the ranks for all entries of $A$ in linear time, evaluating this term takes time $O(r)$. ◂

**Step 2: Reduction to $(\max, \min)$-convolution.** The final step of our algorithm is to reduce the computation of the $(\max, \min)$-*skewed* convolution of $A_1, B_0, C_0$ to few computations of $(\max, \min)$-convolutions (unskewed). Here we exploit that $A$ contains only few distinct entries.

**Lemma 6.6 (Reduction to $(\max, \min)$-Convolution).** *The* $(\max, \min)$*-skewed convolution* $C_1$ *of* $A_1, B_0, C_0$ *can be computed in time* $\widetilde{O}(n^{5/2}/r)$*.*

**Proof.** ▶ We iterate over all pivots $p$. Let $X_p$ denote the lenght-$n$ indicator-like vector satisfying $X_p[i] = \infty$ if $A[i] = p$ and $X_p[i] = -\infty$ otherwise. We compute the $(\max, \min)$-convolution $Y_p$ of $X_p$ and $B$, that is,

$$Y_p[k] = \max_{i+j=k} \min\{X_p[i], B[j]\} = \max_{\substack{j \in [n] \\ A[k-j]=p}} B[j].$$

We compute and return $C_1$ via

$$C_1[k] = \max_{\text{pivot } p} \min\{p, Y_p[k] + S[k]\}.$$

The correctness of the algorithm follows from a simple calculation:

$$\begin{aligned} C_1[k] &= \max_{\substack{i,j \in [n] \\ i+j=k}} \min\{A[i], B[j] + S[k]\} \\ &= \max_{\text{pivot } p} \max_{\substack{j \in [n] \\ A[k-j]=p}} \min\{p, B[j] + S[k]\} \\ &= \max_{\text{pivot } p} \min\{p, \max_{\substack{j \in [n] \\ A[k-j]=p}} B[j] + S[k]\} \\ &= \max_{\text{pivot } p} \min\{p, Y_p[k] + S[k]\}. \end{aligned}$$

For the running time analysis, recall that the $(\max, \min)$-convolution of length-$n$ vectors can be computed in time $\widetilde{O}(n^{3/2})$ [144]. Hence, the running time to compute the vectors $Y_p$ is bounded by $\widetilde{O}(n/p \cdot n^{3/2}) = \widetilde{O}(n^{5/2}/r)$ (as there are $\lceil n/r \rceil$ pivots $p$), and the time to compute $C_1$ is bounded by $O(n \cdot n/r) = O(n^2/r)$. ◀

The proof of Lemma 6.2 is now immediate by combining the previous two lemmas, and by choosing $r = n^{3/4}$.

We remark that Klein, Polak and Rohwedder [137] obtain their improvement essentially by also approximating the vector $B$ by pivots. In their version we have to enumerate all *pairs of pivots*, but instead of computing a $(\max, \min)$-convolution it suffices to compute a Boolean convolution running in near-linear time.

# 7 Sublinear-Time Edit Distance Approximation

In this chapter we present an algorithm to approximate the edit distance of two strings in sublinear time. The work presented in this chapter is based on the paper [56]. I have contributed an equal share of work compared to the other authors, and 50% of the write-up.

56 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Almost-optimal sublinear-time edit distance in the low distance regime". In: *54th annual ACM symposium on theory of computing (STOC 2022)*. ACM, 2022, pages 1102–1115. 10.1145/3519935.3519990.

**Organization.** This chapter is structured as follows. In Section 7.1 we introduce the necessary preliminaries on strings. We then start with an overview of the Andoni-Krauthgamer-Onak algorithm in Section 7.2 from which we borrow in our sublinear-time algorithm. We give a high-level overview of our result in Section 7.3 and provide a detailed proof in Section 7.4. In the remaining Sections 7.5 to 7.8 we provide some missing proofs.

## 7.1 Preliminaries

For integers $i, j$ we write $[i\,..\,j] = \{i, i{+}1, \dots, j\}$ and similarly define the (half-)open intervals $[i\,..\,j), (i\,..\,j], (i\,..\,j)$. As before, we set $[j] = [0\,..\,j)$.

**Strings.** Let $\Sigma$ be an *alphabet*. A string $X$ is a sequence of characters from $\Sigma$. The length of $X$ is denoted by $|X|$ and we write $X \circ Y$ to denote the *concatenation* of $X$ and $Y$. For a nonnegative integer $i$, we denote by $X[i]$ the $i$-th character in $X$ (starting with index zero). For integers $i, j$ we denote by $X[i\,..\,j]$ the substring of $X$ with indices in $[i\,..\,j]$. In particular, if the indices are out-of-bounds, then we set $X[i\,..\,j] = X[\max(i,0)\,..\,\min(j,|X|-1)]$. For (half-)open intervals, we similarly define $X[i\,..\,j), X(i\,..\,j], X(i\,..\,j)$.

For two strings $X, Y$ with equal length, we say that $X$ is a *rotation* of $Y$ if $X[i] = Y[(i+s) \bmod |Y|]$ for some integer shift $s$. We say that $X$ is *primitive* if all of the nontrivial rotations of $X$ are not equal to $X$. For a string $P$, we denote by $P^*$ the infinite-length string obtained by repeating $P$. We say that $X$ is *periodic with period* $P$ if $X = P^*[0\,..\,|X|)$. We also say that $X$ is *$p$-periodic* if it is periodic with some period of length at most $p$.

**Hamming and Edit Distance.** For two strings $X, Y$ with equal length, we define their *Hamming distance* $\mathrm{HD}(X,Y)$ as the number of non-equal characters—in other words, $\mathrm{HD}(X,Y) = |\{i : X[i] \neq Y[i]\}|$. For two strings $X, Y$ (with possibly different lengths), we define their *edit distance* $\mathrm{ED}(X,Y)$ as the smallest number of edit operations necessary to transform $X$ into $Y$; here, an edit operation means *inserting*, *deleting* or *substituting* a character.

We also define an *optimal alignment*, which is a basic object in several of the forthcoming proofs. For two strings $X, Y$, an *alignment* between $X$ and $Y$ is a monotonically non-decreasing function $A : [0\,..\,|X|] \to [0\,..\,|Y|]$ such that $A(0) = 0$ and $A(|X|) = |Y|$. We say that $A$ is an *optimal alignment* if additionally

$$\mathrm{ED}(X,Y) = \sum_{i=0}^{|X|-1} \mathrm{ED}(X[i], Y[A(i)\,..\,A(i+1)]).$$

This definition is slightly non-standard (compare for instance to the definition in [116]), but more convenient for our purposes. Note that the alignments between $X$ and $Y$ correspond to the paths through the standard edit distance dynamic program. In that correspondence, an optimal alignment corresponds to a minimum-cost path.

The $(k, K)$-*gap edit distance* problem is formally defined as follows, for two given length-$n$ strings $X, Y$: The task is to return Close if $\mathrm{ED}(X, Y) \leq k$, to return Far if $\mathrm{ED}(X, Y) > K$ and to return either answer if $k < \mathrm{ED}(X, Y) \leq K$.

**Trees.** In the following we will implicitly refer to trees $T$ where each node has an ordered list of children. A node is a *leaf* if it has no children and otherwise an *internal* node. The *depth* of a node $v$ is defined as the number of ancestors of $v$, and the depth of a tree $T$ is the length of the longest root-leaf path. We refer to the subset of nodes with depth $i$ as the *$i$-th level* in $T$.

## 7.2 Andoni-Krauthgamer-Onak Algorithm

In this section we reinterpret the algorithm of Andoni, Krauthgamer and Onak [20] as a *framework* consisting of a few fundamental ingredients. In the next Section 7.3 we line out our results and how to improve their algorithm to sublinear time, with details provided in Section 7.4.

### 7.2.1 First Ingredient: Tree Distance

The first crucial ingredient for the framework is a way to split the computation of the edit distance into smaller, independent subtasks. A natural approach would be to divide the two strings into equally sized blocks, compute the edit distances of the smaller blocks recursively, and combine the results. The difficulty in doing this is that the edit distance might depend on a *global alignment*, which determines how the blocks should align and therefore the subproblems are not independent (e.g. the optimal alignment of one block might affect the optimal alignment of the next block). However, this can be overcome by computing the edit distances of one block in one string with several shifts of its corresponding block in the other string, and combining the results smartly. This type of *hierarchical decomposition* appeared in previous algorithms for approximating edit distance [24, 20, 37, 167]. In particular, Andoni, Krauthgamer and Onak [20] define a string similarity measure called the *tree distance*[38] which gives a good approximation of the edit distance and cleanly splits the computation into independent subproblems.

[38] *In fact, Andoni et al. [20] call the measure the $\mathcal{E}$-*distance*. However, in a talk by Robert Krauthgamer he recoined the name to* tree distance*, and we decided to stick to this more descriptive name.*

We will define the tree distance for an underlying tree $T$ which we sometimes refer to as the *partition tree*.

**Definition 7.1 (Partition Tree).** *Let $T$ be a tree where each node $v$ is labeled with a non-empty interval $I_v$. We call $T$ a* partition tree *if*

- *for the root node $v$ we have $I_v = [n]$, and*
- *for any node $v$ with children $v_0, \ldots, v_{B-1}$, $I_v$ is the concatenation of $I_{v_0}, \ldots, I_{v_{B-1}}$.*

For the original Andoni-Krauthgamer-Onak algorithm, we will use a complete $B$-ary partition tree $T$ with $n$ leaves. In particular, the depth of $T$ is bounded by $\log_B(n)$. There is a unique way to label $T$ with intervals $I_v$: For the $i$-th leaf (ordered from left to right) we set $I_v = \{i\}$; this choice determines the intervals for all internal nodes. For our algorithm we will later focus on the subtree of $T$ with depth bounded by $O(\log_B(k))$ (this is again a partition tree, as can easily be checked).

The purpose of the partition tree is that it determines a decomposition of two length-$n$ strings $X$ and $Y$. For a node $v$ labeled with interval $I_v = [i \,..\, j]$ we define the following three substrings:

- $X[v] := X[i \,..\, j]$ (the substring of $X$ relevant at $v$),

- $Y[v,s] := Y[i+s\,..\,j+s]$ (the substring of $Y$ relevant at $v$ for a specific shift $s$),
- $Y[v] := Y[i-K\,..\,j+K]$ (the entire substring of $Y$ relevant at $v$).

In particular, the leaf labels in the partition tree determine a partition into consecutive substrings. With this definition in hand, we can define the tree distance:

**Definition 7.2 (Tree Distance).** *Let $X, Y$ be length-$n$ strings and let $T$ be a partition tree. For any node $v$ in $T$ and any shift $s \in \mathbf{Z}$, we set:*

- *If $v$ is a leaf, then $\mathrm{TD}_T(X,Y,v,s) = \mathrm{ED}(X[v], Y[v,s])$.*
- *If $v$ is an internal node with children $v_0, \dots, v_{B-1}$, then*

$$\mathrm{TD}_T(X,Y,v,s) = \sum_{i\in[B]} \min_{s'\in\mathbf{Z}} (\mathrm{TD}_T(X,Y,v_i,s') + 2\cdot|s-s'|). \tag{7}$$

*We write $\mathrm{TD}_T(X,Y) = \mathrm{TD}_T(X,Y,r,0)$ where $r$ is the root node in $T$, and we may omit the subscript $T$ when it is clear from the context.*

The following lemma (which slightly generalizes the analogous result by [20]) shows that the tree distance is a useful measure to approximate the edit distance of two strings. We repeat the proof in Section 7.5.

**Lemma 7.3 (Equivalence of Edit Distance and Tree Distance).** *Let $X, Y$ be strings and let $T$ be a partition tree with degree at most $B$ and depth at most $D$. Then $\mathrm{ED}(X,Y) \le \mathrm{TD}_T(X,Y) \le 2BD\cdot\mathrm{ED}(X,Y)$.*

In light of this lemma, we now focus on approximating the tree distance of two strings. The idea behind the Andoni-Krauthgamer-Onak algorithm is to approximately evaluate Definition 7.2 for all nodes $v$ in the partition tree: For the leaves we directly evaluate the edit distance and for the internal nodes we use Equation (7) to combine the recursive computations. However, notice that in Equation (7) we minimize over an *infinite* number of shifts $s' \in \mathbf{Z}$.

### 7.2.2 Capped Distances

To remedy this situation, recall that we anyways only want to solve a gap problem: Whenever the tree distance exceeds some value $K$ we will immediately report *far*. We therefore restrict our attention to approximating the *capped distances.*

**Definition 7.4 (Capped Edit Distance).** *For strings $X, Y$ and $K \ge 0$, we define the $K$-capped edit distance $\mathrm{ED}^{\le K}(X,Y) = \min(\mathrm{ED}(X,Y), K)$.*

**Definition 7.5 (Capped Tree Distance).** *Let $X, Y$ be strings, let $K \ge 0$ and let $T$ be a partition tree. For any node $v$ in $T$ and any shift $s \in [-K\,..\,K]$, we define the $K$-capped tree distance $\mathrm{TD}_T^{\le K}(X,Y,v,s)$ as follows:*

- *If $v$ is a leaf, then $\mathrm{TD}_T^{\le K}(X,Y,v,s) = \mathrm{ED}^{\le K}(X[v], Y[v,s])$.*
- *If $v$ is an internal node with children $v_0, \dots, v_{B-1}$, then*

$$\mathrm{TD}_T^{\le K}(X,Y,v,s) = \min\left(\sum_{i\in[B]} \min_{-K\le s'\le K} (\mathrm{TD}_T^{\le K}(X,Y,v_i,s') + 2\cdot|s-s'|), K\right). \tag{8}$$

*We write $\mathrm{TD}_T^{\le K}(X,Y) = \mathrm{TD}_T^{\le K}(X,Y,r,0)$ where $r$ is the root node in $T$, and we omit the subscript $T$ when it is clear from the context.*

It is easy to prove that for computing the $K$-capped tree distance $\mathrm{TD}^{\le K}(X,Y)$ can be expressed as $\min(\mathrm{TD}(X,Y), K)$, see the following lemma. (The same statement does not apply to $\mathrm{TD}(X,Y,v,s)$ for all nodes in the tree.) We provide a formal proof in Section 7.5.

**Lemma 7.6 (Equivalence of Capped Distances).** $\mathrm{TD}^{\le K}(X,Y) = \min(\mathrm{TD}(X,Y), K)$.

### 7.2.3 Tree Distance Problem

At this point we are ready to define precisely what we want to compute. We will associate a computational problem to each node in the partition tree $T$. Roughly speaking, the goal for a node $v$ is to approximate the tree distance $\mathrm{TD}_T^{\le K}(X, Y, v, s)$ for all shifts $s$. We now make the details of this approximation precise: To each node $v$, we associate a *multiplicative accuracy* $\alpha_v \ge 1$ and an *additive accuracy* $\beta_v \ge 0$. We define the *Tree Distance Problem* as the dynamic-programming task defined as follows:

**Definition 7.7 (Tree Distance Problem).** *For any node $v$ in the partition tree $T$, compute an array of integers $D[v, -K \,..\, K]$ such that for all $s \in [-K \,..\, K]$:*

$$\frac{1}{\alpha_v} \mathrm{ED}^{\le K}(X[v], Y[v, s]) - \beta_v \le D[v, s] \le \alpha_v \mathrm{TD}^{\le K}(X, Y, v, s) + \beta_v. \tag{9}$$

Given a node $v$ of the partition tree, we will sometimes refer to computing the array $D[v, \cdot]$ as simply *solving* $v$. As a sanity check, let us confirm that an algorithm for the Tree Distance Problem can distinguish edit distances with a polylogarithmic gap:

**Lemma 7.8 (Reduction to the Tree Distance Problem).** *Let $T$ be a partition tree with degree $B$, depth $D$ and root node $r$. Given a solution $D[r, 0]$ to the tree distance problem with root accuracies $\alpha_r \le 10$ and $\beta_r \le 0.001K$, we can infer the correct answer to $(k, K)$-gap edit distance for $k = K/(1000BD)$.*

**Proof.** ▶ The algorithm is simple: We report Close if $D[r, 0] \le 0.05K$ and Far otherwise. To prove that this answer is correct, we first observe that by Equation (9) we have $0.1 \cdot \mathrm{ED}^{\le K}(X, Y) - 0.001K \le D[r, 0] \le 10 \cdot \mathrm{TD}^{\le K}(X, Y) + 0.001K$. Hence, we can easily distinguish the Close and Far cases: On the one hand, if $\mathrm{ED}(X, Y) \le k$ then Lemmas 7.3 and 7.6 imply that $D[r, 0] \le 0.02K + 0.001K = 0.021K$. On the other hand, if $\mathrm{ED}(X, Y) \ge K$ then $D[r, 0] \ge 0.1K - 0.001K = 0.099K$. ◀

### 7.2.4 Second Ingredient: Precision Sampling Lemma

The next step is to assign appropriate accuracies $\alpha_v, \beta_v$ to all nodes in the partition tree. Intuitively, the additive accuracy $\beta_v$ should govern the number of characters accessed in the computation of the subtree below $v$. Indeed, in the analogous approximation of Hamming distances we can afford to sample with rate $\approx 1/\beta_v$ to obtain an additive $\beta_v$-approximation. We aim for a similar dependence, and therefore set $\beta_r = 0.001K$ at the root $r$, so that we read $O(n/K)$ characters in total.

But how to assign the other accuracies $\beta_v$? Starting at the root, the challenge is the following: We want to assign rates to the $B$ children $v_1, \ldots, v_B$ of the current node $v$ in such a way that we can obtain a good approximation of the tree distance at $v$ after combining the results from the children. Naively assigning the same rate to all children incurs an additive error of up to $\beta_{v_1} + \cdots + \beta_{v_B}$. We would therefore have to set $\beta_{v_1} = \cdots = \beta_{v_B} = \beta_v/B$, which is too large as we would be reading more than the $|X[v]|/\beta_v$ characters we are aiming for. A more sophisticated approach to assign the parameters $\beta_v$ is needed.

**Precision Sampling.** Roughly speaking we have an instance of the following problem: There are unknown numbers $A_1, \ldots, A_n \in \mathbf{R}$. We can specify *precisions* $u_1, \ldots, u_n$ and obtain estimates $\widetilde{A}_i$ such that $|A_i - \widetilde{A}_i| \le u_i$, where the *cost* of each estimate is $1/u_i$ (in our setting the cost corresponds to the number of characters we read). The goal is to set the precisions appropriately to be able to distinguish whether $\sum_i A_i < 0.1$ or $\sum_i A_i > 10$, say, and minimize the total cost $\sum_i 1/u_i$. If we set the precisions equally, we would need to have $u_i < 10/n$ (otherwise we cannot distinguish the case where $A_i = 10/n$ for all $i$ from the case $A_i = 0$ for all $i$), which incurs in total cost $\Omega(n^2)$. Andoni, Krauthgamer

and Onak [20] give a very elegant randomized solution to this problem with total cost $\widetilde{O}(n)$ and good error probability, called the *Precision Sampling Lemma* (see Lemma 7.9). This lemma was first shown in [20] and later refined and simplified in [21, 19]. For completeness, we give a full proof in Section 7.6. In the statement, we say that $\widetilde{x}$ is an $(\alpha, \beta)$-*approximation of* $x$ if $\alpha^{-1}x - \beta \leq \widetilde{x} \leq \alpha x + \beta$.

**Lemma 7.9 (Precision Sampling).** *Let $\epsilon, \delta > 0$. There is a distribution $\mathcal{D} = \mathcal{D}(\epsilon, \delta)$ supported over the real interval $(0, 1]$ and an algorithm* Recover *with the following guarantees:*

1. Accuracy: *Fix reals $A_0, \ldots, A_{n-1}$ and independently sample $u_0, \ldots, u_{n-1} \sim \mathcal{D}$. Then, given $(\alpha, \beta \cdot u_i)$-approximations $\widetilde{A_i}$ of $A_i$, the algorithm* Recover *computes an $((1+\epsilon)\cdot\alpha, \beta)$-approximation of $\sum_i A_i$ with success probability $1 - \delta$, for any parameters $\alpha \geq 1$ and $\beta \geq 0$.*
2. Running Time: *Recover runs in time $O(n \cdot \epsilon^{-2} \log(\delta^{-1}))$.*
3. Efficiency: *Sample $u \sim \mathcal{D}$. Then, for any $N \geq 1$ there is an event $E = E(u)$ such that:*
   - *$E$ happens with probability at least $1 - 1/N$, and*
   - *$\mathbf{E}_{u\sim\mathcal{D}}(1/u \mid E) \leq \widetilde{O}(\epsilon^{-2} \log(\delta^{-1}) \log N)$.*

### 7.2.5 Third Ingredient: Range Minima

The final ingredient is an efficient algorithm to combine the recursively computed tree distances. Specifically, the following subproblem can be solved efficiently:

**Lemma 7.10 (Range Minima).** *There is an $O(K)$-time algorithm that, given an integer array $A[-K\,..\,K]$, computes the integer array $B[-K\,..\,K]$ specified by:*

$$B[s] = \min_{-K \leq s' \leq K} A[s'] + 2 \cdot |s - s'|.$$

In [20], the authors use efficient Range Minimum queries (for instance implemented by segment trees) to *approximately* solve this problem with a polylogarithmic overhead. We present a simpler, faster and *exact* algorithm; see Section 7.7 for the pseudocode and proof.

### 7.2.6 Putting the Pieces Together

We are ready to assemble the three ingredients; see Algorithm 7.1 for the pseudocode. The algorithm fills the dynamic programming table $D[v, s]$ for nodes $v$ from leafs to root.

In Lines 1 and 2 we test whether the node $v$ can be solved trivially: If $v$ is a leaf, then we have reached the base case of Definition 7.2 and computing the tree distances boils down to computing the edit distance. In Lines 3 and 4 we solve the instance by another base case: If $|X[v]| \leq \beta_v$, then the edit distance $\mathrm{ED}(X[v], Y[v, s])$ is also upper-bounded by $|X[v]| \leq \beta_v$. We can therefore safely return 0 which is an additive $\beta_v$-approximation to the correct edit distance.

The interesting case happens in Lines 5 to 13 we use the approximations recursively computed by $v$'s children to solve $v$. The idea is to approximately evaluate the following expression for the tree distance (which is equivalent to Equation (8)):

$$\mathrm{TD}^{\leq K}(X, Y, v, s) = \min\left(\sum_{i \in [B]} A[i, s], K\right),$$

where

$$A[i, s] = \min_{s' \in [-K\,..\,K]} (\mathrm{TD}^{\leq K}(X, Y, v_i, s') + 2 \cdot |s - s'|).$$

**Algorithm 7.1.** The Andoni-Krauthgamer-Onak algorithm, reinterpreted in our language. That is, for a node $v$ in the partition tree $T$, this algorithm solves the Tree Distance Problem by computing the array $D[v, -K\,..\,K]$ as in Equation (9).

1 **if** $v$ is a leaf **then**
2 **return** $D[v,s] = \mathrm{ED}(X[v], Y[v,s])$ for all $s \in [-K\,..\,K]$
3 **if** $|X[v]| \le \beta_v$ **then**
4 **return** $D[v,s] = 0$ for all $s \in [-K\,..\,K]$
5 Initialize a two-dimensional array $\widetilde{A}[0\,..\,B-1,\ -K\,..\,K]$
6 **for each** $i \in [B]$ **do**
7 Let $v_i$ be the $i$-th child of $v$
8 Sample $u_{v_i} \sim \mathcal{D}(\frac{1}{2\log n}, \frac{1}{100Kn})$ and let $\alpha_{v_i} = \alpha_v(1 - \frac{1}{2\log n})$, $\beta_{v_i} = \beta_v \cdot u_{v_i}$
9 Recursively compute $D[v_i, -K\,..\,K]$ with accuracies $\alpha_{v_i}, \beta_{v_i}$
10 Compute $\widetilde{A}[i,s] = \min_{s' \in [-K\,..\,K]} D[v_i, s'] + 2 \cdot |s - s'|$ using Lemma 7.10
11 **for each** $s \in [-K\,..\,K]$ **do**
12 Let $D[v,s]$ be the result of the recovery algorithm (Lemma 7.9) applied to $\widetilde{A}[0,s], \ldots, \widetilde{A}[B-1,s]$ with precisions $u_{v_0}, \ldots, u_{v_{B-1}}$
13 **return** $\min(D[v,s], K)$ for all $s \in [-K\,..\,K]$

For each child $v_i$ of $v$ the algorithm first recursively solves the tree distance problem and computes an approximation $D[v_i, s']$ in Line 9. Then, in Line 10, we exactly evaluate

$$\widetilde{A}[i,s] = \min_{s' \in [-K\,..\,K]} D[v_i, s'] + 2 \cdot |s - s'|$$

for all $s$ using Lemma 7.10. In the next step, the Precision Sampling Lemma comes into play. The recursive call returns approximations $D[v_i, s']$ with multiplicative error $\alpha_{v_i} = \alpha_v$ and additive error $\beta_{v_i}$. Hence, $\widetilde{A}[i,s]$ is also an approximation of $A[i,s]$ with multiplicative error $\alpha_{v_i}$ and additive error $\beta_{v_i}$. We pick the parameters $\alpha_{v_i} = \alpha_v(1 - \frac{1}{2\log n})$ and $\beta_{v_i} = \beta_v \cdot u_{v_i}$ where $u_{v_i} \sim \mathcal{D}(\frac{1}{2\log n}, \frac{1}{100Kn})$. In this situation, the Precision Sampling Lemma (Lemma 7.9) allows to approximate the sum $\sum_i A[i,s]$ (for some fixed $s$) with multiplicative error $(1 + \frac{1}{2\log n}) \cdot \alpha_{v_i} \le \alpha_v$ and additive error $\beta_v$—just as required. Hence, the values $D[v,s]$ computed in Line 12 are as claimed (up to taking the minimum with $K$ in order to obtain estimates for the capped tree distance, see Line 13). Moreover, as the recursion depth reaches $\ll \log n$, the multiplicative approximation factors $\alpha_v$ remain at least 1.

In terms of the error probability, note that the only source of randomness in the algorithm is the Precision Sampling Lemma, which, by our choice of parameters, errs with probability $\frac{1}{100Kn}$ per execution. As we execute this lemma $2K+1$ times per node in the computation tree, by a union bound all executions succeed with probability at least $1 - \frac{(2K+1)n}{100Kn} \ge 0.9$. In summary, we obtain the following correctness lemma:

**Lemma 7.11 (Correctness of Algorithm 7.1).** *Let $X, Y$ be length-$n$ strings. Given any node $v$ in a partition tree, Algorithm 7.1 correctly solves the Tree Distance Problem at $v$, with constant probability* 0.9*.*

**Running Time.** It remains to check that the algorithm runs in almost-linear time. In this regard the analysis differs quite substantially from our new sublinear-time algorithm. A single execution of Algorithm 7.1 (ignoring the recursive calls and lower-order factors such as $B$) takes roughly time $O(K)$. However, note that the recursive calls do not necessarily reach every node in the partition tree: Some nodes $v$ are trivially solved by the base case in Lines 3 and 4 and thus their children are never explored. Let us call a node $v$ *active* if the recursive computation reaches $v$. One can bound the number of active nodes by roughly $n/K$ as described in the following lemma:

**Lemma 7.12 (Number of Active Nodes).** *Set $\beta_r = 0.001K$ to be the additive accuracy at the root $r$, and generate the remaining accuracies $\beta_v$ as in Algorithm 7.1. Then the number of nodes $v$ with $|X[v]| > \beta_r$ is at most $\frac{n}{K} \cdot (\log n)^{O(\log_B(n))}$ with probability at least 0.9.*

**Proof.** ▶ We first analyze the additive accuracy $\beta_v$ at a node $v$. More specifically we apply Lemma 7.9 with $N = 100n$ to obtain that: For any sample $u_v$ from the distribution $\mathcal{D}(\epsilon = \frac{1}{2\log n}, \delta = \frac{1}{100Kn})$, there exists an event $E_v$ with $\mathbf{P}(E_v) \geq 1 - \frac{1}{100n}$ and

$$\mathbf{E}(1/u_v \mid E_v) \leq \widetilde{O}(\epsilon^{-2} \log(\delta^{-1}) \log n) = \text{polylog}(n).$$

Using a union bound over all nodes $v$ in the tree, we can assume that all events $E_v$ simultaneously happen with probability at least 0.99. Hence, from now on we condition on $E = \bigwedge_v E_v$. Recall that $\beta_v = 0.001K \cdot u_{v_1} \cdot \ldots \cdot u_{v_d}$ where $v_0, v_1, \ldots, v_d = v$ is the root-to-node path leading to $v$. Therefore, and using that the $u_v$'s are sampled independently:

$$\mathbf{E}(1/\beta_v \mid E) = \frac{1000}{K} \cdot \prod_{i=1}^{d} \mathbf{E}(1/u_{v_i} \mid E_{v_i}) \leq \frac{1}{K} \cdot (\log n)^{O(d)}.$$

Recall that a node $v$ is active only if $|X[v]| > \beta_v$, or equivalently $1/\beta_v > 1/|X[v]|$. Using Markov's inequality we obtain that

$$\mathbf{P}(v \text{ is active} \mid E) = \mathbf{P}(1/\beta_v > 1/|X_v| \mid E) \leq |X_v| \cdot \mathbf{E}(1/\beta_v \mid E) \leq \frac{|X_v|}{K} \cdot (\log n)^{O(d)}.$$

Therefore, the number of active nodes at depth $d$ is $\sum_v |X_v|/K \cdot (\log n)^{O(d)} = n/K \cdot (\log n)^{O(d)}$ (where the sum is over all nodes $v$ at depth $d$, hence $\sum_v |X_v| = |X| = n$). We apply this bound at the deepest level $d = \log_B(n)$, and obtain by another application of Markov's inequality that the total number of active nodes is bounded by $100n/K \cdot (\log n)^{O(\log_B(n))}$ with probability at least 0.99. The total success probability is $0.98 \geq 0.9$. ◀

It is now easy to conclude that the algorithm runs in almost-linear time. Combining the previous lemma with the observation that each active node runs in time $O(KB\,\text{polylog}(n))$, we obtain:

**Lemma 7.13 (Running Time of Algorithm 7.1).** *Algorithm 7.1 runs in time $nB \cdot (\log n)^{O(\log_B(n))}$ with constant probability 0.9.*

It remains to pick the parameter $B$. By setting $B = (\log n)^{O(1/\epsilon)}$ for some $\epsilon > 0$, the running time becomes $O(n^{1+\epsilon})$ (by Lemma 7.13) and the approximation factor becomes $(\log n)^{O(1/\epsilon)}$ (by Lemmas 7.8 and 7.11).

## 7.3 Going Sublinear—An Overview

We are finally ready to describe the pruning rules leading to our sublinear-time algorithm. In contrast to the previous section, our pruning rules will allow us to bound the number of active nodes by $\text{poly}(K)$. We will however spend more time for each active node: In the Andoni-Krauthgamer-Onak algorithm the running time per node is essentially $K$, whereas in our version we run some more elaborate tests per node spending time proportional to $|X_v|/\beta_v + \text{poly}(K)$. We will now progressively develop the pruning rules; the pseudocode is given at the end of this section.

### 7.3.1 Structural Insights

**First Insight: Matching Substrings.** The first insight is that if $\mathrm{ED}(X,Y) \le K$, then we can assume that for almost all nodes $v$ there exists a shift $s^* \in [-K\,..\,K]$ for which $X[v] = Y[v,s^*]$. In this case, we say that the node $v$ is *matched*. The benefit is that if we know that $v$ is matched with shift $s^*$, then instead of approximating the edit distances between $X[v]$ and all shifts $Y[v,s]$, we can instead approximate the edit distance between $Y[v,s^*]$ and all shifts $Y[v,s]$. That is, it suffices to compute the edit distances between a string and *a shift of itself*.

To see that almost all nodes are matched, consider an optimal alignment between $X$ and $Y$ and recall that each level of the partition tree induces a partition of $X$ into substrings $X[v]$. The number of misaligned characters is bounded by $\mathrm{ED}(X,Y)$, thus there are at most $\mathrm{ED}(X,Y)$ parts $X[v]$ containing a misalignment. For all other parts, the complete part $X[v]$ is perfectly matched to some substring $Y[v,s]$. We prove the claim formally in Lemma 7.26. In light of this insight, we can assume that there are only $K$ unmatched nodes—otherwise, the edit distance and thereby also the tree distance between $X$ and $Y$ exceeds $K$ and we can stop the algorithm. In the following we will therefore focus on matched nodes only.

For now we assume that we can efficiently test whether a node is matched. We justify this assumption soon (see the *Matching Test* in Lemma 7.19) by giving a property tester for this problem in sublinear time.

**Second Insight: Structure versus Randomness.** The second idea is to exploit a structure versus randomness dichotomy on strings: As the two extreme cases, a string is either periodic or random-like. The hope is that whenever $Y[v]$ falls into one of these extreme categories, then we can approximate $\mathrm{ED}(Y[v,s^*], Y[v,s])$ directly, without expanding $v$'s children. Concretely, we use the following measure to interpolate between periodic and random-like:

**Definition 7.14 (Block Periodicity).** *Let $Y$ be a string. The $K$-*block periodicity $\mathrm{BP}_K(Y)$ *of $Y$ is the smallest integer $L$ such that $Y$ can be partitioned into $Y = \bigcirc_{\ell=1}^{L} Y_\ell$, where each substring $Y_\ell$ is $K$-periodic (i.e., $Y_\ell$ is periodic with period length at most $K$).*

Suppose for the moment that we could efficiently compute the block periodicity of a string $Y$. Under this assumption, we first compute $\mathrm{BP}_{4K}(Y[v])$ for each matched node $v$ in the tree and distinguish three regimes depending on whether the block periodicity is small, large or intermediate. In the following section we discuss the pruning rules that we apply in these regimes.

### 7.3.2 Pruning Rules

**The Periodic Regime: $\mathrm{BP}_{4K}(Y[v]) = 1$.** For this case, we can approximate the edit distances via the following lemma (think of $Y = Y[v]$ and $Y_s = Y[v,s]$).

**Lemma 7.15 (Periodic Rule).** *For a string $Y$, write $Y_s = Y[K+s\,..\,|Y|-K+s)$. If $Y$ is periodic with primitive period $P$ and $|Y| \ge |P|^2 + 2K$, then for all $s, s' \in [-K\,..\,K]$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot \min_{j \in \mathbf{Z}} \big| s - s' + j|P| \big|.$$

Note that if some node $v$ that is matched and we know that $Y[v]$ is periodic, then Lemma 7.15 allows us to return $D[v,s] = 2 \cdot \min_{j\in\mathbf{Z}} | s - s^* + j|P| |$ as the desired estimates for each shift $s$. In this way we do not recur on $v$'s children and thereby prune the entire subtree below $v$.

To get some intuition for why Lemma 7.15 holds, note that $2 \cdot \min_{j\in\mathbf{Z}} | s - s^* + j|P| |$ is exactly the cost of aligning both $Y_s$ and $Y_{s'}$ in such a way that all occurrences of the period $P$ match (i.e., we shift both strings to the closest-possible occurrence

of $P$). The interesting part is to prove that this alignment is best-possible. We give the complete proof in Section 7.4.2.

**The Random-Like Regime: $\mathbf{BP_{4K}(Y[v]) > 10K}$.** Next, we give the analogous pruning rule for the case when the block periodicity of $Y[v]$ is large. We show that in this case, the best possible way to align any two shifts $Y[v,s]$ and $Y[v,s']$ is to insert and delete $|s-s'|$ many characters. In this sense, $Y[v]$ behaves like a *random* string.

**Lemma 7.16 (Random-Like Rule).** *For a string $Y$, write $Y_s = Y[K+s\,..\,|Y|-K+s)$. If $\mathrm{BP}_{4K}(Y) > 10K$, then for all $s, s' \in [-K\,..\,K]$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot |s - s'|.$$

Again, this rule can be used to solve a matched node $v$ directly, pruning the subtree below $v$. We give the proof in Section 7.4.2.

**The Intermediate Regime: $\mathbf{1 < BP_{4K}(Y[v]) \leq 10K}$.** We cannot directly approximate the edit distance $\mathrm{ED}(Y[v,s^*], Y[v,s])$ in this case. Instead, we exploit the following lemma to argue that the branching procedure below $v$ is computationally cheap:

**Lemma 7.17 (Intermediate).** *Let $v$ be a node in the partition tree. Then, in any level in the subtree below $v$, for all but at most $2\,\mathrm{BP}_{4K}(Y[v])$ nodes $w$ the string $Y[w]$ is $4K$-periodic.*

Indeed, since any node $v$ for which $Y[w]$ is $4K$-periodic can be solved by the Periodic Rule, this lemma implies that there are at most $20K$ active nodes on any level in the partition subtree below any matched node $v$.

### 7.3.3 String Property Testers

Next, we describe how to remove the assumptions that we can efficiently test whether a node is matched and that we can compute block periodicities. We start with the second task.

**Computing Block Periodicities?** The most obvious approach would be to show how to compute (or appropriately approximate) the block periodicity. This is indeed possible, but leads to a more complicated and slower algorithm (in terms of $\mathrm{poly}(K)$).

Instead, we twist the previous argument: We first show how to detect whether a string is periodic (in the straightforward way, see the following Lemma 7.18). For any matched node $v$ we then run the following procedure: If $Y[v]$ is $4K$-periodic, then we solve $v$ according to Lemma 7.15. Otherwise, we continue to explore $v$'s children—with the following constraint: If at some point there are more than $20K$ active nodes which are not $4K$-periodic on any level in the recursion tree below $v$, then we interrupt the computation of this subtree and immediately solve $v$ according to Lemma 7.16. This approach is correct, since by Lemma 7.17 witnessing more than $20K$ active nodes which are not $4K$-periodic on any level serves as a certificate that $\mathrm{BP}_{4K}(Y[v])$ is large. To test whether $Y[v]$ is $4K$-periodic, we use the following tester.

**Lemma 7.18 (Periodicity Test).** *Let $X$ be a string, and let $\beta, \delta > 0$. There is an algorithm which returns one of the following two outputs:*

- *Close$(P)$, where $P$ is a primitive string with $|P| \leq K$ and $\mathrm{HD}(X, P^*[0\,..\,|X|)) \leq \beta$.*
- *Far, in which case $X$ is not $K$-periodic.*

*The algorithm runs in time $O(\beta^{-1}|X|\log(\delta^{-1}) + K)$ and errs with probability $\delta$.*

Recall that as we are shooting for a sublinear-time algorithm we have to resort to a property tester which can only distinguish between *close* and *far* properties (in this case: periodic or *far from periodic*). The proof of Lemma 7.18 is simple, see Section 7.4.3 for details.

**Testing for Matched Nodes.** We need another property tester to test whether a node is matched. Again, since our goal is to design a sublinear-time algorithm, we settle for the following algorithm which distinguishes whether $v$ is matched or *far from matched*.

**Lemma 7.19 (Matching Test).** *Let $X, Y$ be strings such that $|Y| = |X| + 2K$, and let $\beta, \delta > 0$. There is an algorithm which returns one of the following two outputs:*

- *Close$(s^*)$, where $s^* \in [-K \,..\, K]$ satisfies $\mathrm{HD}(X, Y[K + s^* \,..\, |X| + K + s^*)) \leq \beta$.*
- *Far, in which case there is no $s^* \in [-K \,..\, K]$ with $X = Y[K + s^* \,..\, |X| + K + s^*)$.*

*The algorithm runs time $O(\beta^{-1}|X| \log(\delta^{-1}) + K \log |X|)$ and errs with probability $\delta$.*

The proof of Lemma 7.19 is non-trivial. It involves checking whether $X$ and $Y$ follow a common period—in this case we can simply return any shift respecting the periodic pattern. If instead we witness errors to the periodic pattern, then we try to identify a shift under which also the errors align. See Section 7.4.3 for the details.

### 7.3.4 Putting the Pieces Together

We finally assemble our complete algorithm. The pseudocode is given in Algorithm 7.2. In this section we will sketch that Algorithm 7.2 correctly and efficiently solves the Tree Distance Problem. The formal analysis is deferred to Section 7.4.

**Correctness.** We first sketch the correctness of Algorithm 7.2. The recursive calls in Lines 10 to 18 are essentially copied from Algorithm 7.1 (except for differences in the parameters which are not important here) and correct by the same argument as before (using the Precision Sampling Lemma as the main ingredient). The interesting part happens in Lines 1 to 9. In Lines 1 and 2 we test whether the strings are short enough so that we can afford to compute the edit distances $\mathrm{ED}(X[v], Y[v, s])$ by brute-force. If not, we continue to run the Matching Test for $X[v]$ and $Y[v]$ and the Periodicity Test for $Y[v]$.

There are two interesting cases—both assume that the Matching Test reports Close$(s^*)$ and therefore $X[v] \approx Y[v, s^*]$ where $\approx$ denotes equality up to $\beta_v/3$ Hamming errors. If also the Periodicity Test returns Close$(P)$ then we are in the situation that $X[v] \approx Y[v, s^*] \approx P^*$. Moreover, $Y_{v,s}$ is $\approx$-approximately equal to a shift of $P^*$. Lemma 7.15 implies that the edit distance between $X[v]$ and $Y[v, s]$ is approximately $2 \cdot \min_{j \in \mathbf{Z}} |s - s^* + j|P||$. We lose an additive error of $\beta_v/3$ for each of the three $\approx$ relations, hence the total additive error is $\beta_v$ as hoped, and we do not introduce any multiplicative error.

Next, assume that the Periodicity Test reports Far. Then, as stated in Line 9 we continue the recursive computation, but with an exception: If at some point during the recursive computation there are more than $20K$ active nodes on any level for which the Periodicity Test reports Far, then we interrupt the computation and return $D[v, s] = 2 \cdot |s - s^*|$. Suppose that indeed this exception occurs. Then there are more than $20K$ nodes $w$ on one level of the partition tree below $v$ for which $Y[w]$ is *not* $4K$-periodic. Using Lemma 7.17 we conclude that $\mathrm{BP}_{4K}(Y[v]) > 10K$, and therefore returning $D[v, s] = 2 \cdot |s - s^*|$ is correct by Lemma 7.16.

**Running Time.** We will first think of running Algorithm 7.2 with $T$ being a balanced $B$-ary partition tree with $n$ leaves, just as as in the Andoni-Krauthgamer-Onak algorithm (we will soon explain why we need to modify this). To bound the running time of Algorithm 7.2 we first prove that the number of active nodes

**Algorithm 7.2.** Solves the tree distance problem for two given strings $X, Y$. I.e., given a node in the partition tree, this algorithm computes an array $D[v, -K\,..\,K]$ as specified by Equation (9).

1 **if** $v$ is a leaf **or** $|X_v| \le 100K^2$ **then**
2  **return** $D[v,s] = \mathrm{ED}^{\le K}(X[v], Y[v,s])$ for all $s \in [-K\,..\,K]$ (alternatively compute 2-approximations using Theorem 7.20)
3 Run the Matching Test (Lemma 7.19) for $X_v, Y_v$ (with $\beta = \beta_v/3$, $\delta = \frac{1}{100K^{100}}$)
4 Run the $4K$-Periodicity Test (Lemma 7.18) for $Y_v$ (with $\beta = \beta_v/3$, $\delta = \frac{1}{100K^{100}}$)
5 **if** the Matching Test returns $\textsc{Close}(s^*)$ **then**
6  **if** the Periodicity Test returns $\textsc{Close}(P)$ **then**
7  **return** $D[v,s] = 2 \cdot \min_{j \in \mathbf{Z}} |\, s - s^* + j|P|\,|$
8  **else**
9  Continue in Line 11 with the following exception: If at some point during the recursive computation there is some level containing more than $20K$ active nodes below $v$ for which the Periodicity Test (in Line 4) reports $\textsc{Far}$, then interrupt and **return** $D[v,s] = 2 \cdot |s - s^*|$
10 Initialize a two-dimensional array $\widetilde{A}[0\,..\,B-1,\ -K\,..\,K]$
11 **for each** $i \in [B]$ **do**
12  Let $v_i$ be the $i$-th child of $v$
13  Sample $u_{v_i} \sim \mathcal{D}(\frac{1}{200 \log K}, \frac{1}{100K^{101}})$, let $\alpha_{v_i} = \alpha_v(1 - \frac{1}{200 \log K})$, $\beta_{v_i} = \beta_v \cdot u_{v_i}$
14  Recursively compute $D[v_i, -K\,..\,K]$ with accuracies $\alpha_{v_i}, \beta_{v_i}$
15  Compute $\widetilde{A}[i,s] = \min_{s' \in [-K\,..\,K]} D[v_i, s'] + 2 \cdot |s - s'|$ using Lemma 7.10
16 **for each** $s \in [-K\,..\,K]$ **do**
17  Let $D[v,s]$ be the result of the recovery algorithm (Lemma 7.9) applied to $\widetilde{A}[0,s], \ldots, \widetilde{A}[B-1,s]$ with precisions $u_{v_0}, \ldots, u_{v_{B-1}}$
18 **return** $\min(D[v,s], K)$ for all $s \in [-K\,..\,K]$

in the partition tree is bounded by $\mathrm{poly}(K)$. One can show that there are only $\mathrm{poly}(K)$ unmatched nodes in the tree (see Lemma 7.26), so we may only focus on the matched nodes. Each matched node, however, is either solved directly (in Line 2 or in Line 7) or continues the recursive computation with at most $\mathrm{poly}(K)$ active nodes (in Line 9). In Section 7.4 we give more details.

Knowing that the number of active nodes is small, we continue to bound the total expected running time of Algorithm 7.2. It is easy to check that a single execution of Algorithm 7.2 (ignoring the cost of recursive calls) is roughly in time $|X_v|/\beta_v + \mathrm{poly}(K)$. Using the Precision Sampling Lemma, we first bound $1/\beta_v$ by $1/K \cdot (\log n)^{O(\log_B(n))}$ in expectation, for any node $v$. Therefore, the total running time can be bounded by

$$\sum_{v \text{ active}} (|X_v|/\beta_v + \mathrm{poly}(K)) \le \frac{1}{K} \cdot (\log n)^{O(\log_B(n))} \cdot \sum_v |X_v| + \mathrm{poly}(K)$$
$$\le \frac{n}{K} \cdot (\log n)^{O(\log_B(n))} + \mathrm{poly}(K).$$

Here we used that $\sum_w |X[w]| = n$ where the sum is over all nodes $w$ on any fixed level in the partition tree. It follows that $\sum_v |X[v]| \le n \log n$, summing over *all* nodes $v$.

**Optimizing the Lower-Order Terms.** This running time bound does not match the claimed bound in Theorem 7.29: The overhead $(\log n)^{O(\log_B(n))}$ should rather be $(\log K)^{O(\log_B(K))}$ and not depend on $n$. If $n \le K^{100}$, say, then both terms match. But we can also reduce the running time in the general case by "cutting" the partition tree at depth $\log_B(K^{100})$. That is, we delete all nodes below that depth from the partition tree and treat the nodes at depth $\log_B(K^{100})$ as leaves in the

algorithm. The remaining tree is still a partition tree, according to Definition 7.1. The correctness argument remains valid, but we have to prove that the running time of Line 2 does not explode. For each leaf at depth $\log_B(K^{100})$ we have that $|X[v]| \leq n/K^{100}$ and for that reason computing $\mathrm{ED}^{\leq K}(X[v], Y[v,s])$ for all shifts $s$ (say with the Landau-Vishkin algorithm) takes time $O(n/K^{99} + K^3)$. Since there are much less than $K^{99}$ active nodes, the total contribution can again be bounded by $O(n/K + \mathrm{poly}(K))$.

**Optimizing the Polynomial Dependence on $K$.** Finally, let us pinpoint the exponent of the polynomial dependence on $K$. In the current algorithm we can bound the number of active nodes by roughly $K^2$ (there are at most $K$ nodes per level for which the matching test fails, and the subtrees rooted at these nodes contain at most $K$ active nodes per level). The most expensive step in Algorithm 7.2 turns out to be the previously mentioned edit distance computation in Line 2. Using the Landau-Vishkin algorithm (with cap $K$) for $O(K)$ shifts $s$, the running time of Line 2 incurs a cubic dependence on $K$, and therefore the total dependence on $K$ becomes roughly $K^5$.

We give a simple improvement to lower the dependence to $K^4$ and leave further optimizations of the $\mathrm{poly}(K)$ dependence as future work. The idea is to use the following result on approximating the edit distances *for many shifts s:*

**Theorem 7.20 (Edit Distance Approximations for Many Shifts).** *Let $X, Y$ be strings with $|Y| = |X| + 2K$. We can 2-approximate $\mathrm{ED}^{\leq K}(X, Y[K+s\,..\,|X|+K+s))$, for all shifts $s \in [-K\,..\,K]$, in time $O(|X| + K^2)$.*

This result can be proven by a modification of the Landau-Vishkin algorithm [148]; we provide the details in Section 7.8. It remains to argue that computing a 2-approximation in Line 2 does not mess up the correctness proof. This involves the multiplicative approximation guarantee of our algorithm which we entirely skipped in this overview, and for this reason we defer the details to Section 7.4.

**Implementation Details.** We finally describe how to implement the interrupt condition in Line 9. Note that the order of the recursive calls in Line 14 is irrelevant. In the current form the algorithm explores the partition tree in a depth-first search manner, but we might as well use breadth-first search. For any node $v$ which reaches Line 9 we may therefore continue to compute all recursive computations using breadth-first search. If at some point we encounter one search level containing more than $200K$ active nodes for which the Periodicity Test reports FAR, we stop the breadth-first search and jump back to Line 9. With this modification the algorithm maintains one additional counter which does not increase the asymptotic time complexity.

## 7.4 Going Sublinear—In Detail

In this section we give the formal analysis of Algorithm 7.2. We split the proof into the following parts: In Section 7.4.1 we prove some lemmas about periodic and block-periodic strings. In Section 7.4.2 we give the structural lemmas about edit distances in special cases. In Section 7.4.3 we prove the correctness of the string property testers (the "Matching Test" and "Periodicity Test"). In Section 7.4.4 we finally carry out the correctness and running time analyses for Algorithm 7.2, and in Section 7.4.5 we give a formal proof of our main theorem.

In the following proofs we will often use the following simple proposition.

**Proposition 7.21 (Alignments Have Small Stretch).** *Let $X, Y$ be strings of equal length. If $A$ is an optimal alignment between $X$ and $Y$, then $|i - A(i)| \leq \frac{1}{2}\mathrm{ED}(X,Y)$ for all $0 \leq i \leq |X|$.*

**Proof.** ▸ Since $A$ is an optimal alignment between $X$ and $Y$, we can write the edit distance $\mathrm{ED}(X,Y)$ as $\mathrm{ED}(X[0\,..\,i),Y[0\,..\,A(i))) + \mathrm{ED}(X[i\,..\,|X|),Y[A(i)\,..\,|Y|))$. Both edit distances are at least $|i - A(i)|$ which is the length difference of these strings, respectively. It follows that $\mathrm{ED}(X,Y) \geq 2 \cdot |i - A(i)|$, as claimed. ◂

### 7.4.1 Facts about Periodicity

We prove the following two lemmas, both stating roughly that *if a string $X$ closely matches a shift of itself, then $X$ is close to periodic.* The first lemma is easy and well-known. The second lemma is new.

**Lemma 7.22 (Self-Alignment Implies Periodicity).** *Let $X$ be a string. For any shift $s > 0$, if $X[0\,..\,|X| - s) = X[s\,..\,|X|)$ then $X$ is $s$-periodic (with period $X[0\,..\,s)$).*

**Proof.** ▸ By assumption we have that $X[j] = X[s + j]$ for each $j \in [|X| - s]$. It follows that for $P = X[0\,..\,s)$ we have $X[j] = P[j \bmod s]$ for all indices $j \in [|X|]$ and thus $X = P^*[0\,..\,|X|)$. ◂

**Lemma 7.23 (Self-Alignment Implies Small Block Periodicity).** *Let $X$ be a string. For any shift $s > 0$, if $\mathrm{ED}(X[0\,..\,|X| - s), X[s\,..\,|X|)) < 2s$ then $\mathrm{BP}_{2s}(X) \leq 4s$.*

**Proof.** ▸ Let $Y = X[0\,..\,|X| - s)$, $Z = X[s\,..\,|X|)$ and let $A$ denote an optimal alignment between $Y$ and $Z$. We greedily construct a sequence $0 = i_0 < \cdots < i_L = |Y|$ as follows: Start with $i_0 = 0$. Then, having assigned $i_\ell$ we next pick the smallest index $i_{\ell+1} > i_\ell$ for which $Y[i_\ell\,..\,i_{\ell+1}) \neq Z[A(i_\ell)\,..\,A(i_{\ell+1}))$. Using that $A$ is an optimal alignment, we have constructed a sequence of $L < 2s$ indices, since

$$2s > \mathrm{ED}(Y,Z) = \sum_{\ell=0}^{L-1} \mathrm{ED}(Y[i_\ell\,..\,i_{\ell+1}), Z[A(i_\ell)\,..\,A(i_{\ell+1}))) \geq L.$$

Moreover, by Proposition 7.21 we have that $|i - A(i)| < s$ for all $i$. The greedy construction guarantees that $Y[i_\ell\,..\,i_{\ell+1}-1) = Z[A(i_\ell)\,..\,A(i_{\ell+1}-1))$. Therefore, and since $Y, Z$ are substrings of $X$, we have $X[i_\ell\,..\,i_{\ell+1}-1) = X[s+A(i_\ell)\,..\,s+A(i_{\ell+1}-1))$. We will now apply Lemma 7.22 to these substrings of $X$ with shift $s' = s+A(i_\ell)-i_\ell$. Note that $0 < s' < 2s$ (which satisfies the precondition of Lemma 7.22) and thus $X[i_\ell\,..\,i_{\ell+1} - 1)$ is $2s$-periodic.

Finally, consider the following partition of $X$ into $2L + 1$ substrings

$$X = \left(\bigcirc_{\ell=0}^{L-1} X[i_\ell\,..\,i_{\ell+1} - 1) \circ X[i_{\ell+1} - 1]\right) \circ X[|X| - s\,..\,|X|).$$

We claim that each of these substrings is $2s$-periodic: For $X[i_\ell\,..\,i_{\ell+1} - 1)$ we have proved this in the previous paragraph, and the remaining strings $X[i_{\ell+1} - 1]$ and $X[|X| - s\,..\,|X|)$ have length less than $2s$ and are thus trivially $2s$-periodic. This decomposition certifies that $\mathrm{BP}_{2s}(X) \leq 2L + 1 \leq 4s$. ◂

### 7.4.2 Edit Distances between Periodic and Random-Like Strings

The goal of this section is to prove the structural Lemmas 7.15 and 7.16 which determine the edit distance between certain structured strings.

**Lemma 7.15 (Periodic Rule).** *For a string $Y$, write $Y_s = Y[K + s\,..\,|Y| - K + s)$. If $Y$ is periodic with primitive period $P$ and $|Y| \geq |P|^2 + 2K$, then for all $s, s' \in [-K\,..\,K]$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot \min_{j \in \mathbf{Z}} \big|\, s - s' + j|P| \,\big|.$$

**Proof.** ▸ For simplicity set $D = 2 \cdot \min_{j\in\mathbf{Z}} |\, s - s' + j|P| \,|$ and $p = |P|$. We will argue that $D$ is both an upper bound and lower bound for $\mathrm{ED}(Y_s, Y_{s'})$. The upper bound is simple: Note that due to the periodicity of $Y$, we can transform $Y_{s'}$ into $Y_s$ by deleting and inserting $\min_{j\in\mathbf{Z}} |\, s-s'+j|P| \,|$ many characters. Thus, $\mathrm{ED}(Y_s, Y_{s'}) \leq D$.

Next, we prove the lower bound. Suppose that $\mathrm{ED}(Y_s, Y_{s'}) < D$. We assumed that $|Y_s| = |Y| - 2K \geq p^2$, and we can therefore split $Y_s$ into $p$ parts of length $p$ plus some rest. Let $i_\ell = \ell \cdot p$ for all $i \in [0\,..\,p)$ and let $i_{p+1} = |Y_s|$; we treat $Y_s[i_\ell\,..\,i_{\ell+1})$ as the $\ell$-th part. Let $A$ denote an optimal alignment between $Y_s$ and $Y_{s'}$; we have

$$\mathrm{ED}(Y_s, Y_{s'}) = \sum_{\ell=0}^{p} \mathrm{ED}(Y_s[i_\ell\,..\,i_{\ell+1}), Y_{s'}[A(i_\ell)\,..\,A(i_{\ell+1}))).$$

We assumed that $\mathrm{ED}(Y_s, Y_{s'}) < D \leq |P|$ (the latter inequality is by the definition of $D$) and therefore at least one of the first $p$ terms in the sum must be zero, say the $\ell$-th one, $\ell < p$. It follows that the two strings $Y_s[i_\ell\,..\,i_{\ell+1})$ and $Y_{s'}[A(i_\ell)\,..\,A(i_{\ell+1}))$ are equal. Both are length-$p$ substrings of $Y$ and thus rotations of the global period $P$. We assumed that $P$ is primitive (i.e., $P$ is not equal to any of its non-trivial rotations) and therefore $s + i_\ell \equiv s' + A(i_\ell) \pmod{p}$. By the definition of $D$ we must have that $|A(i_\ell) - i_\ell| \geq D/2$. But this contradicts Proposition 7.21 which states that $|A(i_\ell) - i_\ell| \leq \mathrm{ED}(Y_s, Y_{s'})/2 < D/2$. ◀

**Lemma 7.16 (Random-Like Rule).** *For a string $Y$, write $Y_s = Y[K + s\,..\,|Y| - K + s)$. If $\mathrm{BP}_{4K}(Y) > 10K$, then for all $s, s' \in [-K\,..\,K]$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot |s - s'|.$$

**Proof.** ▶ Let $D = 2 \cdot |s - s'|$. First note that $\mathrm{ED}(Y_s, Y_{s'}) \leq D$ since we can transform $Y_s$ into $Y_{s'}$ by simply inserting and deleting $|s - s'|$ symbols. For the lower bound, suppose that $\mathrm{ED}(Y_s, Y_{s'}) < D$. Therefore, we can apply Lemma 7.23 for an appropriate substring of $Y$: Assume without loss of generality that $s \leq s'$ and let $Z = Y[K + s\,..\,|Y| - K + s')$. Then we clearly have $Y_s = Z[0\,..\,|Z| - s' + s)$ and $Y_{s'} = Z[s' - s\,..\,|Z|)$, so Lemma 7.23 applied to $Z$ with shift $s' - s$ yields the bound $\mathrm{BP}_{2(s'-s)}(Y') \leq 4 \cdot (s' - s)$. We conclude that $\mathrm{BP}_{4K}(Y') \leq 8K$, using the trivial bound $s' - s \leq 2K$. We can obtain $Y$ by adding at most $K$ characters to the start and end of $Z$. It follows that $\mathrm{BP}_{4K}(Y) \leq \mathrm{BP}_{4K}(Z) + 2 \leq 10K$. This contradicts the assumption in the lemma statement, and therefore $\mathrm{ED}(Y_s, Y_{s'}) \geq D$. ◀

**Lemma 7.17 (Intermediate).** *Let $v$ be a node in the partition tree. Then, in any level in the subtree below $v$, for all but at most $2\,\mathrm{BP}_{4K}(Y[v])$ nodes $w$ the string $Y[w]$ is $4K$-periodic.*

**Proof.** ▶ Focus on some level of the computation subtree below $v$. We will bound the number of nodes $w$ in this level for which $Y[w]$ is not $4K$-periodic. Note that if $Y[v]$ was partitioned into $Y[v] = \bigcirc_w Y[w]$, then by the definition of block periodicity we would immediately conclude that at most $\mathrm{BP}_{4K}(Y[v])$ many parts $Y[w]$ are not $4K$-periodic. However, recall that for a node $w$ with associated interval $I_w = [i\,..\,j]$, we defined $Y[w]$ as $Y[w] = Y[i - K\,..\,j + K]$. This means that the substrings $Y[w]$ overlap with each other and therefore do *not* partition $Y[v]$.

To deal with this, note that we can assume that $|Y[w]| > 4K$, since otherwise $Y[w]$ is trivially $4K$-periodic. Hence, each $Y[w]$ can overlap with at most two neighboring nodes (since the intervals $I_w$ are disjoint). Therefore, we can divide the $w$'s in two groups such that the $Y[w]$'s in each group do not overlap with each other. For each group, we apply the argument from above to derive that there are at most $\mathrm{BP}_{4K}(Y[v])$ many $Y[w]$'s which are not $4K$-periodic. In this way, we conclude that there are at most $2\,\mathrm{BP}_{4K}(Y[v])$ nodes in the level which are not $4K$-periodic, as desired. ◀

### 7.4.3 Some String Property Testers

The main goal of this section is to formally prove Lemmas 7.18 and 7.19, that is, the Matching Test and Periodicity Test. As a first step, we need the following simple lemma about testing equality of strings.

**Lemma 7.24 (Equality Test).** *Let $X, Y$ be strings of the same length, and let $\beta, \delta > 0$. There is an algorithm which returns one of the following two outputs:*

- *Close, in which case $\mathrm{HD}(X, Y) \leq \beta$.*
- *Far$(i)$, in which case $X[i] \neq Y[i]$.*

*The algorithm runs in time $O(\beta^{-1}|X| \log(\delta^{-1}))$ and errs with probability $\delta$.*

**Proof.** ▸ The idea is standard: For $\beta^{-1}|X| \ln(\delta^{-1})$ many random positions $i \in [|X|]$, test whether $X[i] = Y[i]$. If no error is found, then we report Close. This equality test is clearly sound: If $X = Y$, then it will never fail. It remains to argue that if $\mathrm{HD}(X, Y) > \beta$ then the test fails with probability at least $1 - \delta$. Indeed, each individual sample finds a Hamming error with probability $\beta/|X|$. Hence, the probability of not finding any Hamming error across all samples is at most

$$\left(1 - \frac{\beta}{|X|}\right)^{\beta^{-1}|X| \ln(\delta^{-1})} < \exp(-\ln(\delta^{-1})) = \delta.$$

The running time is bounded by $O(\beta^{-1}|X| \log(\delta^{-1}))$. ◂

**Lemma 7.18 (Periodicity Test).** *Let $X$ be a string, and let $\beta, \delta > 0$. There is an algorithm which returns one of the following two outputs:*

- *Close$(P)$, where $P$ is a primitive string with $|P| \leq K$ and $\mathrm{HD}(X, P^*[0..|X|)) \leq \beta$.*
- *Far, in which case $X$ is not $K$-periodic.*

*The algorithm runs in time $O(\beta^{-1}|X| \log(\delta^{-1}) + K)$ and errs with probability $\delta$.*

**Proof.** ▸ We start analyzing the length-$2K$ prefix $Y = X[0..2K)$. In time $O(K)$ we can compute the smallest period $P$ such that $Y = P^*[0..|Y|)$ by searching for the first match of $Y$ in $Y \circ Y$, e.g. using the Knuth-Morris-Pratt pattern matching algorithm [140]. If no such match exists, we can immediately report Far. So suppose that we find a period $P$. It must be primitive (since it is the smallest such period) and it remains to test whether $X$ globally follows the period. For this task we use the Equality Test (Lemma 7.24) with inputs $X$ and $P^*$ (of course, we cannot write down the infinite-length string $P^*$, but we provide oracle access to $P^*$ which is sufficient here). On the one hand, if $X$ is indeed periodic with period $P$, then the Equality Test reports Close. On the other hand, if $X$ is $\beta$-far from any periodic string, then in particular $\mathrm{HD}(X, P^*) > \beta$ and therefore the Equality Test reports Far. The only randomized step is the Equality Test. We therefore set the error probability of the Equality Test to $\delta$ and achieve total running time $O(\beta^{-1}|X| \log(\delta^{-1}) + K)$. ◂

**Lemma 7.19 (Matching Test).** *Let $X, Y$ be strings such that $|Y| = |X| + 2K$, and let $\beta, \delta > 0$. There is an algorithm which returns one of the following two outputs:*

- *Close$(s^*)$, where $s^* \in [-K..K]$ satisfies $\mathrm{HD}(X, Y[K + s^*..|X| + K + s^*)) \leq \beta$.*
- *Far, in which case there is no $s^* \in [-K..K]$ with $X = Y[K + s^*..|X| + K + s^*)$.*

*The algorithm runs time $O(\beta^{-1}|X| \log(\delta^{-1}) + K \log |X|)$ and errs with probability $\delta$.*

**Proof.** ▸ For convenience, we write $Y_s = Y[K + s..|X| + K + s)$. Our goal is to obtain a single *candidate shift* $s^*$ (that is, knowing $s^*$ we can exclude all other shifts from consideration). Having obtained a candidate shift, we can use the Equality Test (Lemma 7.24 with parameters $\beta$ and $\delta/3$) to verify whether we indeed have $X = Y_{s^*}$. In the positive case, Lemma 7.24 implies that $\mathrm{HD}(X, Y_{s^*}) \leq \beta$, hence returning $s^*$ is valid. The difficulty lies in obtaining the candidate shift. Our algorithm proceeds in three steps:

1. *Aligning the Prefixes:* We start by computing the set $S$ consisting of all shifts $s$ for which $X[0..2K) = Y_s[0..2K)$. One way to compute this set in linear time $O(K)$ is by using a pattern matching algorithm with pattern $X[0..2K)$ and text $Y[0..4K)$ (like the Knuth-Morris-Pratt algorithm [140]). It is clear that $S$ must contain any shift $s$ for which globally $X = Y_s$. For that reason we

can stop if $|S| = 0$ (in which case we return FAR) or if $|S| = 1$ (in which case we test the unique candidate shift $s^* \in S$ and report accordingly).

**2** *Testing for Periodicity:* After the previous step we can assume that $|S| \geq 2$. Take any pair $s < s'$ from $S$; we have that $X[0\,..\,2K) = Y_s[0\,..\,2K) = Y_{s'}[0\,..\,2K)$. It follows that $X[0\,..\,2K - s' + s) = X[s' - s\,..\,2K)$, and thus by Lemma 7.22 we conclude that $X[0\,..\,2K)$ is periodic with period $P = X[0\,..\,s' - s)$; the period length is $|P| = s' - s \leq 2K$. Obviously the same holds for $Y_s[0\,..\,2K)$ and $Y_{s'}[0\,..\,2K)$.

We will now test whether $X$ and $Y_s$ are also globally periodic with this period $P$. To this end, we apply the Equality Test two times (each time with parameters $\beta/2$ and $\delta/3$) to check whether $X = P^*[0\,..\,|X|)$ and $Y_s = P^*[0\,..\,|Y_s|)$. If both tests return CLOSE, then Lemma 7.24 yields that $\mathrm{HD}(X, P^*[0\,..\,|X|)) \leq \beta/2$ and $\mathrm{HD}(Y_s, P^*[0\,..\,|Y_s|)) \leq \beta/2$. We conclude that $\mathrm{HD}(X, Y_s) \leq \beta$ by the triangle inequality. Note that we have witnessed a matching shift $s^* = s$.

**3** *Aligning the Leading Mismatches:* Assuming that the previous step did not succeed, one of the Equality Tests returned $\mathrm{FAR}(i_0)$ for some position $i_0 > 2K$ with $X[i_0] \neq P^*[i_0]$ or $Y_s[i_0] \neq P^*[i_0]$. Let us refer to these indices as *mismatches*. Moreover, we call a mismatch $i$ a *leading mismatch* if the $2K$ positions to the left of $i$ are not mismatches. We continue in two steps: First, we find a leading mismatch. Second, we turn this leading mismatch into a candidate shift.

**3a** *Finding a Leading Mismatch:* To find a leading mismatch, we use the following binary search-style algorithm: Initialize $L \leftarrow 0$ and $R \leftarrow i_0$. We maintain the following two invariants: (i) All positions in $[L\,..\,L + 2K)$ are not mismatches, and (ii) $R$ is a mismatch. Both properties are initially true. We will now iterate as follows: Let $M \leftarrow \lceil (L + R)/2 \rceil$ and test whether there is a mismatch $i \in [M\,..\,M + 2K)$. If there is such a mismatch $i$, we update $R \leftarrow i$. Otherwise, we update $L \leftarrow M$. It is easy to see that in both cases both invariants are maintained. Moreover, this procedure is guaranteed to make progress as long as $L + 4K < R$. If at some point $R \leq L + 4K$, then we can simply check all positions in $[L\,..\,R]$—one of these positions must be a leading mismatch $i$.

**3b** *Finding a Candidate Shift:* Assume that the previous step succeeded in finding a leading mismatch $i$. Then we can produce a single candidate shift as follows: Assume without loss of generality that $X[i] \neq P^*[i]$, and let $i \leq j$ be the smallest position such that $Y_s[j] \neq P^*[j]$. Then $s^* = s + j - i$ is the only candidate shift (if it happens to fall into the range $[-K\,..\,K]$).

Indeed, for any $s'' > s^*$ we can find a position where $X$ and $Y_{s''}$ differ. To see this, assume that $s''$ respects the period (i.e., $P^* = P^*[K + s''\,..\,\infty)$), since otherwise we find a mismatch in the length-$2K$ prefix. But then

$$\begin{aligned} Y_{s''}[j + s - s''] &= Y_s[j] && (10)\\ &\neq P^*[j] && (11)\\ &= P^*[j + s - s''] && (12)\\ &= X[j + s - s''], && (13) \end{aligned}$$

which proves that $X \neq Y_{s''}$ and thereby disqualifies $s''$ as a feasible shift. Here we used (10) the definition of $Y_s$, (11) the assumption that $Y_s[j] \neq P^*[j]$, (12) the fact that both $s$ and $s''$ respect the period $P$ and (13) the assumption that $i$ was a leading mismatch which implies that $X$ matches $P^*$ at the position $j + s - s'' < i$.

A similar argument works for any shift $s'' < s^*$. In this case one can show that $X[i] \neq P^*[i] = Y_{s''}[i]$ which also disqualifies $s''$ as a candidate shift.

We finally bound the error probability and running time of this algorithm. We only use randomness when calling the Equality Test which runs at most three times. Since each time we set the error parameter to $\delta/3$, the total error probability is $\delta$ as claimed. The running time of the Equality Tests is bounded by $O(\beta^{-1}|X|\log(\delta^{-1}))$ by Lemma 7.24. In addition, steps 1 and 2 take time $O(K)$. Step 3 iterates at most $\log|X|$ times and each iteration takes time $O(K)$. Thus, the total running time is $O(\beta^{-1}|X|\log(\delta^{-1}) + K\log|X|)$. ◀

### 7.4.4 Putting The Pieces Together

In this section we give a formal analysis of Algorithm 7.2.

**Setting the Parameters.** Throughout this section we assume that $T$ is a balanced $B$-ary partition tree with $\min(n, K^{100})$ leaves, where each leaf $v$ is labeled with an interval $I_v$ of length $|I_v| \approx \max(1, n/K^{100})$. In particular there are at most $2 \cdot K^{100}$ nodes in the tree and its depth is bounded by $\lceil \log_B \min(n, K^{100}) \rceil$.

We also specify the root accuracies $\alpha_r = 10$ and $\beta_v = 0.001K$. Note that the algorithm assigns the smallest multiplicative accuracy to $\alpha_v = 10 \cdot (1 - \frac{1}{200\log K})^d$, where $d$ is the depth of the partition tree. It follows that $\alpha_v \geq 5$.

**Correctness.** We start with the correctness proof.

**Lemma 7.25 (Correctness of Algorithm 7.2).** *Let $X, Y$ be strings. Given any node $v$ in the partition tree, Algorithm 7.2 correctly solves the Tree Distance Problem at $v$, with constant probability* 0.9*.*

**Proof.** ▶ The analysis of Lines 11 to 18 (that is, combining the recursive computations) is precisely as in Lemma 7.11. We therefore omit the details an assume that these steps succeed. In this proof we show that Lines 1 to 9 are correct as well. (We postpone the error analysis to the end of the proof.) There are three possible cases:

- *The Strings are Short:* First assume that $|X[v]| \leq 100K^2$ or that $v$ is a leaf node, in which case the condition in Line 1 triggers. The algorithm computes and returns a multiplicative 2-approximation $D[v, s]$ of $\mathrm{ED}(X[v], Y[v, s])$ for all shifts $s$ using Theorem 7.20. We need to justify that $2 \leq \alpha_v$ so that $D[v, s]$ is a valid approximation in the sense of Equation (9), but recall that using the parameter setting in the previous paragraph we have $\alpha_v \geq 5$.

If the algorithm does not terminate in this first case, we may assume from now on that $|X[v]| \geq 100K^2$. The algorithm continues running and applies the Matching Test (Lemma 7.19) to $X[v], Y[v]$ and the $4K$-Periodicity Test (Lemma 7.18) to $Y[v]$, both with rate parameter $\beta = \beta_v/3$ and error parameter $\delta = \frac{1}{100K^{100}}$. We again postpone the error analysis and assume that both tests returned a correct answer. We continue analyzing the remaining two cases:

- *The Strings are Periodic:* Assume that the Matching Test reports $\textsc{Close}(s^*)$ and that the $4K$-Periodicity Test reports $\textsc{Close}(P)$, where $P$ is a primitive string with length $|P| \leq 4K$. The algorithm reaches Line 7 and returns $D[v, s] = 2 \cdot \min_{j \in \mathbf{Z}} |s - s^* - j|P||$. We argue that this approximation is valid using Lemma 7.15 and by applying the triangle inequality three times. To this end we define $Z = P^*[0\,..\,|Y_v|)$ (that is, $Z$ is equal to $Y_v$ after "correcting" the periodicity errors) and $Z_s = Z[K + s\,..\,|Z| - K + s)$. By Lemma 7.15 we have that

$$\mathrm{ED}(Z_{s^*}, Z_s) = 2 \cdot \min_{j \in \mathbf{Z}} \left| s - s^* - j|P| \right| = D[v, s],$$

for all shifts $s$. Here we use the assumption that $|X[v]| \geq 100K^2 \geq |P|^2$ and its consequence $|Z| = |Y[v]| \geq |P|^2 + 2K$ to satisfy the precondition of Lemma 7.15. Since the Periodicity Test reported $\textsc{Close}(P)$, we get $\mathrm{HD}(Y[v, s], Z_s) \leq \beta_v/3$ for

all shifts $s$, and using that the Matching Test reported $\textsc{Close}(s^*)$ we obtain $\mathrm{HD}(X[v], Y[v, s^*]) \leq \beta_v/3$. By applying the triangle inequality three times we conclude that

$$\begin{aligned} D[v,s] &= \mathrm{ED}(Z_{s^*}, Z_s) \\ &\leq \mathrm{ED}(Z_{s^*}, Y[v,s^*]) + \mathrm{ED}(Y[v,s^*], X[v]) \\ &\qquad + \mathrm{ED}(X[v], Y[v,s]) + \mathrm{ED}(Y[v,s], Z_s) \\ &\leq \mathrm{ED}(X[v], Y[v,s]) + \beta_v, \end{aligned}$$

and similarly $D[v,s] \geq \mathrm{ED}(X_v, Y_{v,s}) - \beta_v$. It follows that $D[v,s]$ is an additive $\beta_v$-approximation of $\mathrm{ED}(X[v], Y[v,s])$, as required in Equation (9). (Here, we do not suffer any multiplicative error.)

*The Strings are Random-Like:* Finally assume that the Matching Test reports $\textsc{Close}(s^*)$, but the $4K$-Periodicity Test reports $\textsc{Far}$. In this case the algorithm reaches Line 9 and continues with the recursive computation (in Line 11). However, if at any level in the computation subtree rooted at $v$ there are more than $20K$ active nodes for which the Periodicity Test (in Line 4) reports $\textsc{Far}$, then the recursive computation is interrupted and we return $D[v,s] = 2 \cdot |s - s^*|$. We have already argued that the unrestricted recursive computation is correct, but it remains to justify why interrupting the computation makes sense.

So suppose that the recursive computation is interrupted, i.e., assume that there are more than $20K$ descendants $w$ of $v$ at some level for which the Periodicity Test reported $\textsc{Far}$. Assuming that all Periodicity Tests computed correct outputs, we conclude that for all these descendants $w$ the strings $Y_v$ are not $4K$-periodic. From Lemma 7.17 we learn that necessarily $\mathrm{BP}_{4K}(Y_v) > 10K$. Hence Lemma 7.16 applies and yields that

$$\mathrm{ED}(Y[v,s^*], Y[v,s]) = 2 \cdot |s - s^*| = D[v,s].$$

Using again the triangle inequality and the assumption that $\mathrm{ED}(Y[v,s^*], X[v]) \leq \beta_v$ (by the Matching Test), we derive that

$$\begin{aligned} D[v,s] &= \mathrm{ED}(Y[v,s^*], Y[v,s]) \\ &\leq \mathrm{ED}(Y[v,s^*], X[v]) + \mathrm{ED}(X[v], Y[v,s]) \\ &\leq \mathrm{ED}(X[v], Y[v,s]) + \beta_v. \end{aligned}$$

The lower bound can be proved similarly and therefore $D[v,s]$ is an additive $\beta_v$-approximation of $\mathrm{ED}(X[v], Y[v,s])$.

We finally analyze the error probability of Algorithm 7.2. There are three sources of randomness in the algorithm: The Matching and Periodicity Tests in Lines 3 and 4 and the application of the Precision Sampling Lemma. For each node, therefore have three error events: With probability at most $2\delta = \frac{2}{100K^{-100}}$ one of the property tests fails. We apply the Precision Sampling Lemma with $\delta = \frac{1}{100K^{101}}$ for $2K+1$ shifts in every node, hence the error probability is $\frac{2K+1}{100K^{101}} \leq \frac{3}{100K^{100}}$. In total, the error probability per node is $\frac{5}{100K^{100}}$. Recall that there are at most $2K^{100}$ nodes in the partition tree, and thus the total error probability is bounded by 0.1. ◀

**Running Time.** This concludes the correctness part of the analysis and we continue bounding the running time of Algorithm 7.2. We proceed in two steps: First, we give an upper bound on the number of active nodes in the partition tree (see Lemmas 7.26 and 7.27). Second, we bound the expected running time of a single execution of Algorithm 7.2 (ignoring the cost of recursive calls). The expected running time is bounded by their product.

Recall that call a node $v$ *matched* if there is some shift $s^* \in [-K \,..\, K]$ such that $X[v] = Y[v, s^*]$. Moreover, we say that $v$ is *active* if the recursive computation of Algorithm 7.2 reaches $v$.

**Lemma 7.26 (Number of Unmatched Nodes).** *Assume that* $\mathrm{ED}(X, Y) \le K$. *If the partition tree has depth D, then there are at most KD nodes which are not matched.*

**Proof.** ▸ Focus on any level in the partition tree and let $0 = i_0 < \cdots < i_w = n$ denote the partition induced by that level, i.e., let $[i_\ell \,..\, i_{\ell+1}) = I_v$ where $v$ is the $\ell$-th node in the level (from left to right). Let $A$ be an optimal alignment between $X$ and $Y$, then:

$$\mathrm{ED}(X, Y) = \sum_{\ell=0}^{w-1} \mathrm{ED}(X[i_\ell \,..\, i_{\ell+1}), Y[A(i_\ell) \,..\, A(i_{\ell+1}))).$$

Since we assumed that $\mathrm{ED}(X, Y) \le K$, there can be at most $K$ nonzero terms in the sum. For any zero term we have that $X[i_\ell \,..\, i_{\ell+1}) = Y[A(i_\ell) \,..\, A(i_{\ell+1}))$ and therefore the $\ell$-th node in the current level is matched with shift $A(i_\ell) - i_\ell$. By Proposition 7.21 we have that $|A(i_\ell) - i_\ell| \le \mathrm{ED}(X, Y) \le K$. This completes the proof. ◂

**Lemma 7.27 (Number of Active Nodes).** *Assume that* $\mathrm{ED}(X, Y) \le K$. *If the partition tree has depth D, then there are at most* $O((KDB)^2)$ *active nodes, with probability* 0.98.

**Proof.** ▸ Recall that (unconditionally) there are at most $2K^{100}$ nodes in the partition tree. Hence, by a union bound, all Matching Tests in Line 3 succeed with probability at least $1 - 0.02 = 0.98$. We will condition on this event throughout the proof. We distinguish between three kinds of nodes $v$:

1. $v$ itself and all of $v$'s ancestors are not matched,
2. $v$ itself is matched, but all of $v$'s ancestors are not matched,
3. some ancestor of $v$ (and therefore also $v$ itself) is matched.

By the previous lemma we know that there are at most $KD$ nodes which are not matched. It follows that there are at most $KD$ nodes of the first kind.

It is also easy to bound the number of nodes $v$ of the second kind: Observe that $v$'s parent is a node of the first kind. Hence, there can be at most $KD \cdot B$ nodes of the second kind.

Finally, we bound the number of nodes of the third kind. Any such node $v$ has a unique ancestor $w$ of the second kind. There are two cases for $w$: Either the condition in Line 6 succeeds and the algorithm directly solves $w$. This is a contradiction since we assumed that $v$ (a descendant of $w$) is active. Or this condition fails, and the algorithm continues branching with the exception that if in the subtree below $w$ there are more than $20K$ active nodes per level for which the Periodicity Test in Line 4 reports Far, then we interrupt the recursive computation. We claim that consequently in the subtree below $w$ (consisting only of nodes of the third kind), there are at most $20KB$ active nodes per level. Indeed, suppose there were more than $20KB$ active nodes on some level. Then consider their parent nodes; there must be more than $20K$ parents. For each such parent $u$ the Matching Test reported Close (since $u$ is a matched node, and we assumed that all Matching Tests succeed) and the Periodicity Test reported Far (since otherwise the condition in Line 4 triggers and solves $u$ directly, but we assumed that $u$ is the parent of some other active node). Note that we have witnessed more than $20K$ nodes on one level below $w$ for which the Periodicity Test reported Far. This is a contradiction.

In total the number of active nodes below $w$ is bounded by $20KB \cdot D$. Recall that there are at most $KD \cdot B$ nodes $w$ of the second kind, hence the total number of active nodes of the third kind is $20(KDB)^2$. Summing over all three kinds, we obtain the claimed bound. ◂

We are ready to bound the total running time. In the following lemma we prove that the algorithm is efficient *assuming that the edit distance between $X$ and $Y$ is small.* This assumption can be justified by applying Algorithm 7.2 with the following modification: We run Algorithm 7.2 with a time budget and interrupt the computation as soon as the budget is depleted. In this case we can immediately infer that the edit distance between $X$ and $Y$ must be large.

**Lemma 7.28 (Running Time of Algorithm 7.2).** *Let $X, Y$ be length-$n$ strings with $\mathrm{ED}(X,Y) \le K$. Then Algorithm 7.2 runs in time*

$$\frac{n}{K} \cdot (\log K)^{O(\log_B(K))} + \widetilde{O}(K^4 B^2),$$

*with constant probability* 0.9*.*

**Proof.** ▶ We first bound the expected running time of a single execution of Algorithm 7.2 where we ignore the cost of recursive calls. Let $D$ denote the depth of the partition tree. We proceed in the order of the pseudocode:

- Lines 1 and 2: If the test in Line 1 succeeds, then Line 2 takes time $O(|X[v]|+K^2)$ by Theorem 7.20, where $|X[v]| \le 100K^2$ or $|X[v]| \le O(n/K^{100})$. The total time of this step is therefore bounded by $O(K^2 + n/K^{100})$.
- Lines 3 and 4: Running the Matching and Periodicity Tests (Lemmas 7.18 and 7.19) takes time $O(\beta_v^{-1}|X[v]| \log K + K \log |X[v]|)$. To match the claimed time bound, we want to replace the $\log |X[v]|$ by $\log K$ here. So assume that the second term dominates, i.e., $\beta_v^{-1}|X[v]| \log K \le K \log |X[v]|$. At any node $v$ the additive accuracy $\beta_v$ is always at most $0.001K$ (since at the root we exactly have accuracy $0.001K$ and below the root the additive error never increases), hence $|X[v]|/\log |X[v]| \le \mathrm{poly}(K)$. It follows that we can bound the total time of this step indeed by $O(\beta_v^{-1}|X[v]| \log K + K \log K)$.
- Lines 5 to 9: Here we merely produce the output according to some fixed rules. The time of this step is bounded by $O(K)$.
- Lines 11 to 18: It is easy to check that these steps run in time $\widetilde{O}(KB)$. This analysis is exactly as in Lemma 7.13.

In total, the time of a single execution is $O(\beta_v^{-1}|X[v]| \log K + K^2 + n/K^{100})$. We will simplify this term by plugging in the (expected) value $1/\beta_v$ for any node $v$ (in a way similar to Lemma 7.13).

Recall that $\beta_v = 0.001K \cdot u_{v_1} \cdot \ldots \cdot u_{v_d}$ where $v_0, v_1, \ldots, v_d = v$ is the root-to-node path leading to $v$ and each $u_i$ is sampled from $\mathcal{D}(\epsilon = \frac{1}{200 \log K}, \delta = \frac{1}{100K^{101}})$, independently. Using Lemma 7.9 there are events $E_w$ happening each with probability $1 - 1/N$ such that

$$\mathbf{E}(1/u_w \mid E_w) \le \widetilde{O}(\epsilon^{-2} \log(\delta^{-1}) \log N) \le \mathrm{polylog}(K).$$

In the last step we set $N = 100K^{100}$. Taking a union bound over all active nodes $w$ (there are at most $2K^{100}$ many), the event $E = \bigwedge_w E_w$ happens with probability at least 0.98, and we will condition on $E$ from now on. Under this condition we have:

$$\mathbf{E}(1/\beta_v \mid E) = \frac{1000}{K} \prod_{i=1}^{d} \mathbf{E}(1/u_{v_i} \mid E_{v_i}) \le \frac{(\log K)^{O(d)}}{K} \le \frac{(\log K)^{O(\log_B(K))}}{K}.$$

Finally, we can bound the total expected running time (conditioned on $E$) as follows, summing over all active nodes $v$:

$$\sum_v O\left(|X[v]| \cdot \frac{(\log K)^{O(\log_B(K))}}{k} + K^2 + \frac{n}{K^{100}}\right).$$

Using that $\sum_w |X[w]| = n$ whenever $w$ ranges over all nodes on a fixed level in the partition tree, and thus $\sum_v |X[v]| \le n \cdot D$ where $v$ ranges over all nodes, we

can bound the first term in the sum by $n/K \cdot (\log K)^{O(\log_B(K))}$. The second term can be bounded by $K^2$ times the number of active nodes. By Lemma 7.27 this becomes $O(K^4D^2B^2) = \widetilde{O}(K^4B^2)$. By the same argument the third term becomes at most $n/K^{90}$ and is therefore negligible.

We conditioned on two events: The event $E$ and the event that the number of active nodes is bounded by $O(K^2D^2B^2)$ (Lemma 7.27). Both happen with probability at least 0.98, thus the total success probability is $0.96 \geq 0.9$. ◀

### 7.4.5 Main Theorems

We finally recap and formally prove our two main results on sublinear edit distance. We start with the most general statement we obtain:

**Theorem 7.29 (Sublinear-Time Gap Edit Distance).** *Let $2 \leq B \leq k$ be a parameter. The $(k, \Theta(kB\log_B(k)))$-gap edit distance problem can be solved in time*

$$\frac{n}{k} \cdot (\log k)^{O(\log_B(k))} + \widetilde{O}(k^4 \operatorname{poly}(B)).$$

**Proof.** ▶ Let $\bar{k}, \bar{K}$ be parameters to be set later. By Lemma 7.8, a solution of the tree distance problem with root accuracies $\alpha_r \leq 10$ and $\beta_r \leq 0.001\bar{K}$ translates to an answer of the $(\bar{k}, \bar{K})$-gap edit problem for $\bar{k} = \bar{K}/(1000BD)$, where $B$ and $D$ are the degree and depth of the partition tree, respectively. By Lemma 7.25, Algorithm 7.2 correclty solves the tree distance problem and thereby $(\bar{k}, \bar{K})$-gap edit distance. To bound the running time by

$$\frac{n}{\bar{K}} \cdot (\log \bar{K})^{O(\log_B(\bar{K}))} + \widetilde{O}(\bar{K}^4 \cdot \operatorname{poly}(B)),$$

we interrupt the algorithm after it exceeds this time budget and return FAR. Indeed, by Lemma 7.28 the algorithm runs in this time budget whenever the edit distance of the given strings $X, Y$ is $\mathrm{ED}(X, Y) \leq \bar{K}$.

We finally pick $\bar{k} = k$ and $\bar{K} = 1000BDk$. Noting that the partition tree has depth $D = \log_B(\bar{K}) = O(\log_B(k))$, the claimed statement follows. ◀

**Theorem 1.15 (Subpolynomial Gap Edit Distance).** *The $(k, k \cdot 2^{\widetilde{O}(\sqrt{\log k})})$-gap edit distance problem can be solved in time $O(n/k + k^{4+o(1)})$.*

**Proof.** ▶ Simply plugging $B = 2^{\sqrt{\log k}}$ into Theorem 7.29 leads to the correct gap, but we suffer a factor $k^{o(1)}$ in the running time. For that reason, let $\bar{k}$ be a parameter to be specified later and apply Theorem 7.29 with parameter $\bar{k}$ and $B = 2^{\sqrt{\log \bar{k}}}$. In that way we can distinguish the gap $\bar{k}$ versus $\bar{k} \cdot 2^{\widetilde{\Theta}(\sqrt{\log \bar{k}})}$ in time

$$\frac{n}{\bar{k}} \cdot f(\bar{k}) + \bar{k}^{4+o(1)},$$

where

$$f(\bar{k}) = (\log \bar{k})^{O(\log_B(\bar{k}))} = k^{o(1)}.$$

We set $\bar{k} = k \cdot f(k)^2$. For sufficiently large $k$, we have that $f(\bar{k}) \leq f(k)^2$ and therefore the running time becomes $O(n/k + k^{4+o(1)})$ as claimed. (For small constant $k$, we can exactly compute the $k$-capped edit distance in linear time $O(n)$ using the Landau-Vishkin algorithm [148].) Summarizing, our algorithm distinguishes the gap $\bar{k}$ versus $\bar{k} \cdot 2^{O(\sqrt{\log k})} = k \cdot 2^{\widetilde{\Theta}(\sqrt{\log k})}$. Since $k \leq \bar{k}$, this is sufficient to prove the claim. ◀

**Theorem 1.16 (Polylogarithmic Gap Edit Distance).** *The $(k, k \cdot (\log k)^{O(1/\epsilon)})$-gap edit distance problem can be solved in time $O(n/k^{1-\epsilon} + k^{4+o(1)})$, for any $\epsilon \in (0, 1)$.*

**Proof.** ▶ Let $c$ be the constant so that the time bound in Theorem 7.29 becomes

$$\frac{n}{k} \cdot (\log k)^{c \log_B(k)} + \widetilde{O}(k^4 \operatorname{poly}(B)).$$

We apply Theorem 7.29 with parameter $B = (\log k)^{c/\epsilon}$. Then the gap is indeed $k$ versus $k \cdot (\log k)^{O(1/\epsilon)}$ as claimed. Since $(\log k)^{c \log_B(k)} = k^\epsilon$ the running time bound becomes $O(n/k^{1-\epsilon} + k^{4+o(1)})$. ◀

## 7.5 Equivalence of Edit Distance and Tree Distance

In this section we prove that the tree distance closely approximates the edit distance. The proof is an easy generalization of [20, Theorem 3.3]. At the end of the section, we also provide a proof of Lemma 7.6.

**Lemma 7.3 (Equivalence of Edit Distance and Tree Distance).** *Let $X, Y$ be strings and let $T$ be a partition tree with degree at most $B$ and depth at most $D$. Then* $\mathrm{ED}(X,Y) \le \mathrm{TD}_T(X,Y) \le 2BD \cdot \mathrm{ED}(X,Y)$.

We prove the lower and upper bounds on $\mathrm{TD}(X,Y)$ in two separate steps.

**Lower Bound.** We show that $\mathrm{ED}(X[v], Y[v,s]) \le \mathrm{TD}(X,Y,v,s)$. The proof is by induction on the depth of $T$. If $v$ is a leaf we have $\mathrm{ED}(X[v], Y[v,s]) = \mathrm{TD}(X,Y,v,s)$ by definition. So focus on an internal node $v$ with children $v_0, \dots, v_{B-1}$. In this case we have:

$$\begin{aligned}
\mathrm{ED}(X[v], Y[v,s]) &\le \sum_{\ell=0}^{B-1} \mathrm{ED}(X[v_\ell], Y[v_\ell, s]) && (14)\\
&\le \sum_{\ell=0}^{B-1} \min_{s' \in \mathbf{Z}} \mathrm{ED}(X[v_\ell], Y[v_\ell, s']) + \mathrm{ED}(Y[v_\ell, s'], Y[v_\ell, s]) && (15)\\
&\le \sum_{\ell=0}^{B-1} \min_{s' \in \mathbf{Z}} \mathrm{ED}(X[v_\ell], Y[v_\ell, s']) + 2 \cdot |s - s'| && (16)\\
&\le \sum_{\ell=0}^{B-1} \min_{s' \in \mathbf{Z}} \mathrm{TD}(X,Y,v_\ell,s') + 2 \cdot |s - s'| && (17)\\
&= \mathrm{TD}(X,Y,v,s). && (18)
\end{aligned}$$

Here we used (14) the facts that $X[v] = \bigcirc_\ell X[v_\ell]$ and $Y[v,s] = \bigcirc_\ell Y[v_\ell, s]$, (15) the triangle inequality, (16) the observation that $\mathrm{ED}(Y[w,s], Y[w,s']) \le 2 \cdot |s - s'|$ for all shifts $s, s' \in \mathbf{Z}$ (by deleting $|s - s'|$ in the beginning and inserting $|s - s'|$ characters at the end of the strings), (17) the induction hypothesis, and finally (18) the definition of the tree distance.

**Upper Bound.** Because of some technical complications we do not prove the upper bound by induction. Instead, we will unfold the definition of tree distance and prove the upper bound "globally". Specifically, it is easy to prove (by induction on the depth of the computation tree) that

$$\mathrm{TD}(X,Y) = \min_{\substack{s \in \mathbf{Z}^T \\ s_r = 0}} \left( \sum_{\substack{\text{internal} \\ \text{nodes } v}} \sum_{\substack{\text{children } w \\ \text{of } v}} 2 \cdot |s_v - s_w| \right) + \sum_{\text{leaves } v} \mathrm{ED}(X[v], Y[v, s_v]),$$

where we write $r$ for the root node. For clarity: Here, we minimize over all shifts $s_v \in \mathbf{Z}$ for all nodes $v$ in the computation tree—except for the root node $r$ for which we fix $s_r = 0$.

To prove an upper bound on this expression, we first specify the values $s_v$. For this purpose, we let $A$ denote an optimal alignment between $X$ and $Y$. Recall that the computation tree determines an interval $I_v$ for each node $v$; let us

write $I_v = [i_v \,..\, j_v)$. We can now pick the value $s_v = A(i_v) - i_v$. Slightly abusing notation, we also define the substring $Y'[v] = Y[A(i_v) \,..\, A(j_v))$. By the way we defined optimal alignments $A$, we have that $\mathrm{ED}(X,Y) = \sum_v \mathrm{ED}(X[v], Y'[v])$, where the sum is over all nodes $v$ in a cut through the partition tree (e.g., all leaves or all nodes at one specific level).

**Claim 7.30.** $\mathrm{ED}(X[v], Y[v, s_v]) \leq 2\,\mathrm{ED}(X[v], Y'[v])$.

**Proof.** ▶ By the triangle inequality, we can bound $\mathrm{ED}(X[v], Y[v, s_v])$ by the sum of $\mathrm{ED}(X[v], Y'[v])$ and $\mathrm{ED}(Y'[v], Y[v, s_v])$. But observe that both strings $Y'[v]$ and $Y[v, s_v]$ are substrings of $Y$ starting at the same position $A(i_v)$. Hence, one is a prefix of the other string and we can bound their edit distance by their difference in lengths, $\mathrm{ED}(Y'[v], Y[v, s_v]) \leq ||Y'[v]| - |Y[v, s_v]|| = ||Y'[v]| - |X[v]|| \leq \mathrm{ED}(X[v], Y'[v])$. ◀

**Claim 7.31.** *Let $w$ be a child of $v$. Then* $|s_v - s_w| \leq \mathrm{ED}(X[v], Y'[v])$.

**Proof.** ▶ Recall that $s_v = A(i_v) - i_v$ and $s_w = A(i_w) - i_w$. Using that the edit distance of two strings is at least their difference in length, we have that $|s_v - s_w| \leq \mathrm{ED}(X[i_v \,..\, i_w), Y[A(i_v) \,..\, A(i_w)))$. Since $w$ is a child of $v$ we have that $i_w \leq j_v$. Thus we obtain $|s_v - s_w| \leq \mathrm{ED}(X[i_v \,..\, i_w), Y[A(i_v) \,..\, A(i_w))) = \mathrm{ED}(X[v], Y'[v])$, using again that $A$ is an optimal alignment. ◀

Using the two claims, we can finally give an upper bound on the tree distance. In the following calculation we will use (19) the tree distance characterization from before (plugging in our values $s_v$), (20) the two claims, (21) the assumption that each node has at most $B$ children and the (obvious) fact that each node is either a leaf or internal, and finally (22) that $\sum_v \mathrm{ED}(X[v], Y'[v]) = \mathrm{ED}(X,Y)$ whenever the sum is over all nodes $v$ at one specific level of the computation tree, hence $\sum_v \mathrm{ED}(X[v], Y'[v]) = D \cdot \mathrm{ED}(X,Y)$ if the sum is over all nodes $v$:

$$\mathrm{TD}(X,Y) \leq \left( \sum_{\substack{\text{internal} \\ \text{nodes } v}} \sum_{\substack{\text{children} \\ \text{of } v}} 2 \cdot |s_v - s_w| \right) + \sum_{\text{leaves } v} \mathrm{ED}(X_v, Y_{v,s_v}) \tag{19}$$

$$\leq \left( \sum_{\substack{\text{internal} \\ \text{nodes } v}} \sum_{\substack{\text{children} \\ \text{of } v}} 2 \cdot \mathrm{ED}(X_v, Y'_v) \right) + \sum_{\text{leaves } v} 2 \cdot \mathrm{ED}(X_v, Y'_v) \tag{20}$$

$$\leq 2B \sum_{\substack{\text{all} \\ \text{nodes } v}} \mathrm{ED}(X_v, Y'_v) \tag{21}$$

$$\leq 2BD \cdot \mathrm{ED}(X,Y). \tag{22}$$

This completes the proof of Lemma 7.3. It remains to prove Lemma 7.6.

**Lemma 7.6 (Equivalence of Capped Distances).** $\mathrm{TD}^{\leq K}(X,Y) = \min(\mathrm{TD}(X,Y), K)$.

**Proof.** ▶ It is easy to prove that $\mathrm{TD}^{\leq K}(X,Y,v,s) \geq \min(\mathrm{TD}(X,Y,v,s), K)$ for all nodes $v$ and all shifts $s$, by induction. However, the other direction is not necessarily true for all nodes. For the other direciton, we thus directly prove $\mathrm{TD}^{\leq K}(X,Y) \leq \min(\mathrm{TD}(X,Y), K)$. We may assume that $\mathrm{TD}(X,Y) < K$ as otherwise the statement is clear. Let $s_v$ denote the optimal shifts picked at all nodes $v$ in the tree distance definition (7). That is, we have $s_r = 0$ for the root node $r$ and for all nodes $v$ with children $v_1, \ldots, v_B$ the following equation is satisfied:

$$\mathrm{TD}(X,Y,v,s_v) = \sum_{i \in [B]} (\mathrm{TD}(X,Y,v_i,s_{v_i}) + 2 \cdot |s_v - s_{v_i}|).$$

By induction one can easily verify that $|s_v| \leq \mathrm{TD}(X,Y) < K$ for all nodes $v$. But this implies that the same shifts can be picked in the $K$-capped tree distance definition (8) to certify that $\mathrm{TD}^{\leq K}(X,Y,r,0) \leq \mathrm{TD}(X,Y,r,0)$, and thus $\mathrm{TD}^{\leq K}(X,Y) \leq \mathrm{TD}(X,Y)$. ◀

## 7.6 Precision Sampling Lemma

In this section, we give a proof of the Precision Sampling Lemma.

**Lemma 7.9 (Precision Sampling).** *Let $\epsilon, \delta > 0$. There is a distribution $\mathcal{D} = \mathcal{D}(\epsilon, \delta)$ supported over the real interval $(0, 1]$ and an algorithm* Recover *with the following guarantees:*

1. Accuracy: *Fix reals $A_0, \dots, A_{n-1}$ and independently sample $u_0, \dots, u_{n-1} \sim \mathcal{D}$. Then, given $(\alpha, \beta \cdot u_i)$-approximations $\widetilde{A}_i$ of $A_i$, the algorithm* Recover *computes an $((1+\epsilon)\cdot\alpha, \beta)$-approximation of $\sum_i A_i$ with success probability $1 - \delta$, for any parameters $\alpha \geq 1$ and $\beta \geq 0$.*
2. Running Time: *Recover runs in time $O(n \cdot \epsilon^{-2} \log(\delta^{-1}))$.*
3. Efficiency: *Sample $u \sim \mathcal{D}$. Then, for any $N \geq 1$ there is an event $E = E(u)$ such that:*
   - *$E$ happens with probability at least $1 - 1/N$, and*
   - *$\mathbf{E}_{u\sim\mathcal{D}}(1/u \mid E) \leq \widetilde{O}(\epsilon^{-2} \log(\delta^{-1}) \log N)$.*

Although originally used and stated by Andoni, Krauthgamer and Onak in [20], our formulation is essentially taken from their follow-up paper [21]. We give a simpler proof using the exponential distribution, as suggested in a later paper by Andoni [19].

Recall that the *exponential distribution* $\mathrm{Exp}(\lambda)$ with rate $\lambda > 0$ is a continuous distribution over the positive reals with probability density function $f(x) = \lambda e^{-\lambda x}$. We will use the following facts:

**Fact 7.32 (Properties of the Exponential Distribution).** *Let $\lambda, \lambda_1, \dots, \lambda_n > 0$.*

- Scaling: *Sample $u \sim \mathrm{Exp}(\lambda)$ and let $\alpha > 0$. Then $u/\alpha$ is distributed as $\mathrm{Exp}(\alpha \cdot \lambda)$.*
- Min-Stability: *Independently sample $u_1 \sim \mathrm{Exp}(\lambda_1), \dots, u_n \sim \mathrm{Exp}(\lambda_n)$. Then their minimum $\min\{u_1, \dots, u_n\}$ is distributed as $\mathrm{Exp}(\sum_i \lambda_i)$.*

**Proof of Lemma 7.9.** ▸ We pick the distribution $\mathcal{D} = \mathcal{D}(\epsilon, \delta) = \mathrm{Exp}(\lambda)$ for some integer $\lambda = \lambda(\epsilon, \delta)$. Note that by the min-stability property, sampling $u_i \sim \mathcal{D}$ is equivalent to sampling $\lambda$ values $u_{i,1}, \dots, u_{i,\lambda} \sim \mathrm{Exp}(1)$ and returning their minimum. For simplicity, we assume that we can keep track of these $\lambda$ original values $u_{i,j}$ for each sample $u_i$. With this distribution in mind, we prove the three properties:

1. *Accuracy:* Fix reals $A_1, \dots, A_n$ and independently sample $u_1, \dots, u_n \sim \mathcal{D}$ (while keeping track of the $u_{i,j}$'s). Then an adversary provides us with approximations $\widetilde{A}_1, \dots, \widetilde{A}_n$ satisfying that $\widetilde{A}_i$ is an $(\alpha, \beta \cdot u_i)$-approximation of $A_i$. Our goal is to design an algorithm Recover that $(\alpha(1+\epsilon), \beta)$-approximates $\sum_i A_i$: The algorithm is simple:
   - Compute $\widetilde{M}_j = \max_{i\in[n]} \widetilde{A}_i/u_{i,j}$ for all $j \in [\lambda]$.
   - Return $\ln(2) \cdot \mathrm{median}_{j\in[\lambda]} \widetilde{M}_j$.

   For the analysis, we define the quantities $M_j = \max_{i\in[n]} A_i/u_{i,j}$. Note that $M_j$ is a random variable and does not depend on the adversary's choice. We argue in two steps: First, note that $\widetilde{M}_j$ is an $(\alpha, \beta)$-approximation of $M_j$ as

$$\widetilde{M}_j = \max_{i\in[n]} \frac{\widetilde{A}_i}{u_{i,j}} \leq \max_{i\in[n]} \frac{\alpha A_i + \beta \cdot u_{i,j}}{u_{i,j}} \leq \alpha\left(\max_{i\in[n]} \frac{A_i}{u_{i,j}}\right) + \beta = \alpha M_j + \beta,$$

   and similarly $\widetilde{M}_j \geq \alpha^{-1} M_j - \beta$.

   Second, we prove that $\ln(2)\cdot\mathrm{median}_j M_j$ is a $(1+\epsilon)$-approximation of $\sum_i A_i$, with probability at least $1 - \delta$. From this statement, the correctness of the algorithm follows immediately. To show that this statement holds, first note that by the scaling and min-stability properties from Fact 7.32, each random variable $M_j$ is distributed like the inverse of an exponentially distributed random

variable with rate $\sum_i A_i$. And thus, $(\sum_i A_i)/M_j$ is distributed as $\mathrm{Exp}(1)$. It follows that

$$\begin{aligned}
&\mathbf{P}\left(\ln(2)\cdot M_j > (1+\epsilon)\sum_{i\in[n]} A_i\right)\\
&\quad= \mathbf{P}\left(\frac{\sum_{i\in[n]} A_i}{M_j} < \frac{\ln(2)}{1+\epsilon}\right)\\
&\quad= \int_{u=0}^{\frac{\ln(2)}{1+\epsilon}} e^{-u}\,du\\
&\quad= 1 - e^{-\frac{\ln(2)}{1+\epsilon}}\\
&\quad= 1 - (\tfrac{1}{2})^{\frac{1}{1+\epsilon}}\\
&\quad= \tfrac{1}{2} - \Theta(\epsilon),
\end{aligned}$$

and similarly,

$$\mathbf{P}\left(\ln(2)\cdot M_j > (1+\epsilon)\sum_{i\in[n]} A_i\right) = \tfrac{1}{2} - \Theta(\epsilon).$$

Using Chernoff's bound, with probability $1 - \exp(-\Omega(\lambda\epsilon^2))$, we have that (i) the number of $j$'s with $\ln(2)\cdot M_j > (1+\epsilon)\sum_i A_i$ is less than $\lambda/2$ and (ii) the number of $j$'s with $\ln(2)\cdot M_j < (1-\epsilon)\sum_i A_i$ is less than $\lambda/2$. If both (i) and (ii) are simultaneously true, then $\ln(2)\cdot \mathrm{median}_j M_j$ is as claimed. To achieve the desired success probability $1-\delta$, we set $\lambda(\epsilon,\delta) = \Theta(\epsilon^{-2}\log(\delta^{-1}))$.

2 *Running Time:* The algorithm clearly runs in time $O(\lambda) = O(\epsilon^{-2}\log(\delta^{-1}))$.

3 *Efficiency:* Now we prove the efficiency property. Fix any $N \geq 1$ and sample $u \sim \mathcal{D}$. Let $E$ be the event that $u > \frac{1}{\lambda N}$. This event happens with probability

$$\mathbf{P}(E) = \int_{u=\frac{1}{\lambda N}}^{\infty} \lambda e^{-\lambda u}\,du = e^{-\frac{1}{N}} > 1 - \frac{1}{N}.$$

Next, we analyze the conditional expectation $\mathbf{E}(1/u \mid E)$ where $u$ is sampled from $\mathcal{D} = \mathrm{Exp}(\lambda)$. Using that $\mathbf{P}(E) \geq e^{-1}$ we have that

$$\begin{aligned}
\mathbf{E}(1/u \mid E) &= \frac{1}{\mathbf{P}(E)}\int_{u=\frac{1}{\lambda N}}^{\infty} \frac{1}{u}\cdot\lambda e^{-\lambda u}\,du\\
&\leq e\int_{u=\frac{1}{\lambda N}}^{1} \frac{1}{u}\cdot\lambda e^{-\lambda u}\,du + e\int_{u=1}^{\infty} \frac{1}{u}\cdot\lambda e^{-\lambda u}\,du\\
&\leq e\lambda\int_{u=\frac{1}{\lambda N}}^{1} \frac{1}{u}\,du + e\\
&= O(\lambda\log(\lambda N))\\
&= \widetilde{O}(\lambda\log N).
\end{aligned}$$

Plugging in our choice of $\lambda$, we obtain that $\mathbf{E}(1/u \mid E) = \widetilde{O}(\epsilon^{-2}\log(\delta^{-1})\log N)$, as claimed.

Finally, we enforce that the distribution $\mathcal{D}$ is supported over $(0,1]$, as the lemma states. We can simply transform a sample $u \sim \mathcal{D}$ into $u' \leftarrow \min(u,1)$. Since the samples $u$ only become smaller in this way, the estimates $\widetilde{A}_i$ obtained by the adversary have only smaller additive error, and we therefore preserve the accuracy property. For the efficiency property, note that $\mathbf{E}(1/u' \mid E) \leq \mathbf{E}(1/u \mid E) + 1$, and therefore the bound on the conditional expectation remains valid asymptotically. ◀

**Algorithm 7.3.** Given an integer array $A[-K\,..\,K]$, this algorithm computes the integer array $B[-K\,..\,K]$ specified by $B[s] = \min_{-K \le s' \le K} A[s'] + 2 \cdot |s - s'|$.

```
1 Let B^L[−K..K] and B^R[−K..K] be integer arrays
2 Initialize B^L[−K] ← A[−K] and B^R[K] ← A[K]
3 for s ← −K + 1, ..., K do
4 |   B^L[s] ← min(B^L[s − 1] + 2, A[s])
5 for s ← K − 1, ..., −K do
6 |   B^R[s] ← min(B^R[s + 1] − 2, A[s])
7 return the array B[−K..K] computed by B[s] ← min(B^L[s], B^R[s])
```

**Implementation in the word RAM Model.** Throughout we implicitly assumed that we can sample from the continuous distribution $\mathrm{Exp}(\lambda)$. We briefly comment on how to implement the algorithm in the word RAM model. An easy way is to discretize the samples to multiples of $1/\mathrm{poly}(n)$. The resulting distribution is a geometric distribution which we can efficiently sample from [62]. Alternatively, one can prove the Precision Sampling Lemma directly in terms of a discrete probability distribution, see for instance the original paper [20].

## 7.7 Range Minima

In this section we give a simple algorithm to efficiently combine the recursive results from the children at every node of the computation tree.

**Lemma 7.10 (Range Minima).** *There is an $O(K)$-time algorithm that, given an integer array $A[-K\,..\,K]$, computes the integer array $B[-K\,..\,K]$ specified by:*

$$B[s] = \min_{-K \le s' \le K} A[s'] + 2 \cdot |s - s'|.$$

**Proof.** ▶ We give the pseudocode in Algorithm 7.3. We claim that the algorithm correctly computes

$$B^L[s] = \min_{-K \le s' \le s} A[s'] - 2s' + 2s,$$
$$B^R[s] = \min_{s \le s' \le K} A[s'] + 2s' - 2s,$$

for all $s$. This claim implies that we correctly output $B[s] = \min(B^L[s], B^R[s])$. We prove that $B^L[s]$ is computed correctly; the other statement is symmetric. As the base case, the algorithm correctly assigns $B^L[-K] = A[-K]$. We may therefore inductively assume that $B^L[s-1]$ is assigned correctly, and have to show that $B^L[s] \leftarrow \min(B^L[s-1] + 2, A[s])$ is a correct assignment. There are two cases: Either $B^L[s]$ attains its minimum for $s' < s$, and thus $B^L[s] = B^L[s-1] + 2$. Or $B^L[s]$ attains the minimum for $s' = s$, in which case $B^L[s] = A[s]$. The running time is bounded by $O(K)$. ◀

## 7.8 2-Approximating Edit Distance for Many Shifts

In this section we show how to modify the Landau-Vishkin algorithm [148] to give a constant-factor approximation of the edit distance of a string $X$ and several consecutive shifts of another string $Y$. We use this routine at the leaves of our algorithm (Line 2 of Algorithm 7.2).

**Theorem 7.20 (Edit Distance Approximations for Many Shifts).** *Let $X, Y$ be strings with $|Y| = |X| + 2K$. We can 2-approximate $\mathrm{ED}^{\le K}(X, Y[K+s\,..\,|X|+K+s))$, for all shifts $s \in [-K\,..\,K]$, in time $O(|X| + K^2)$.*

**Proof.** ▶ Let $m = |X|$ and $n = |Y|$. We will write $Y_s = Y[K+s \,..\, m+K+s)$ for short. Consider the following table $S[i, j]$ defined as

$$S[i,j] = \min_{0 \le s' \le 2K} \mathrm{ED}(X[0\,..\,i), Y[s'\,..\,j)), \tag{23}$$

where $i \in [m]$ and $j \in [n]$. We prove the statement in two steps: First, we will demonstrate how to compute the values $S[m, m{+}K{+}s]$ in the claimed running time. Second, we show that these values constitute 2-approximations of $\mathrm{ED}^{\le K}(X, Y_s)$.

**Computing the Values $S[m, m+K+s]$.** It is not hard to see that Equation (23) follows the same recursive formulation as the dynamic program for edit distance, but with a twist in the base case. That is, we initialize $S[0, j] = 0$ for all $j \in [0\,..\,2K]$ (which accounts for the possible starting shifts of $Y$) and the rest of the entries follow the recurrence

$$S[i,j] = \min\{1 + S[i-1, j], 1 + S[i, j-1], c_{i,j} + S[i-1, j-1]\},$$

where $c_{i,j}$ is the cost of matching $X[i]$ with $Y[j]$ (that is, $c_{i,j} = 1$ if $X[i] \neq Y[j]$, and otherwise $c_{i,j} = 0$) and where we set the out-of-bounds entries $S[-1, j], S[i, -1]$ to $\infty$.

Since we want to approximate *capped* edit distances, we are only interested in computing values $\le K$ in this table. This can be done using the classic Landau-Vishkin algorithm [148] in time $O(|X| + K^2)$. We now briefly sketch how this algorithm works. The idea is to iterate over the edit distance values $d = 0, \ldots, K$. For each value $d$ and for each upper-left to bottom-right diagonal in the dynamic programming table, we maintain the furthest position to the right along each diagonal which contains the value $d$. We call this set of positions the *frontier*. Since moving away from a diagonal incurs cost 1 and since we initialize $K$ consecutive diagonals with $d = 0$ in the base case, the frontier consists of $O(K)$ values at any iteration of the algorithm (i.e., we only need to keep track of $O(K)$ diagonals). Given the frontier for some edit distance value $d$, we can obtain the frontier for $d+1$ by performing a *longest common extension query* for each diagonal and updating the corresponding position along each diagonal. Longest common extension queries can be answered using suffix trees in $O(1)$ time after preprocessing $X$ and $Y$ in linear time [148, 94], and updating the position along each diagonal also takes constant time. Thus, each of the $K$ iterations takes time $O(K)$ and the total running time is $O(|X| + K^2)$.

**Approximation Quality.** We claim that for every $s \in [-K\,..\,K]$, the following bounds hold:

$$S[m, m+K+s] \le \mathrm{ED}(X, Y_s) \le 2 \cdot S[m, m+K+s].$$

This claim implies the lemma statement. Indeed, the previous step allows us to compute the entries $S[m, m+K+s]$ which are at most $K$ and identify those which have value larger than $K$. By capping the latter entries at $K$, the claim implies that we get 2-approximations of the capped distances $\mathrm{ED}^{\le K}(X, Y_s)$ for every $s$.

We now prove the claimed bounds. The lower bound $S[m, m+K+s] \le \mathrm{ED}(X, Y_s)$ holds by the definition of $S[m, m+K+s]$. For the upper bound, let $s'$ be the shift which minimizes the expression for $S[m, m+K+s]$ in Equation (23), i.e., let $s'$ be such that $S[m, m+K+s] = \mathrm{ED}(X, Y[K+s' \,..\, m+K+s))$. We will use that the edit distance of any two strings $A, B$ is at least their difference in lengths $||A|-|B||$. In particular, we have $S[m, m{+}K{+}s] \ge |s - s'|$. By the triangle inequality, we have that

$$\begin{aligned}\mathrm{ED}(X, Y_s) &\le \mathrm{ED}(X, Y[K+s' \,..\, m+K+s)) + \mathrm{ED}(Y[K+s' \,..\, m+K+s), Y_s) \\ &\le S[m, m+K+s] + |s - s'| \le 2 \cdot S[m, m+K+s],\end{aligned}$$

where the second inequality follows because we can transform $Y[K+s' \,..\, m+K+s)$ into $Y_s = Y[K+s \,..\, m+K+s)$ by deleting or inserting $|s-s'|$ characters. This proves the claim and thus finishes the proof of Theorem 7.20. ◀

# References


1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. “Tight hardness results for LCS and other sequence similarity measures”. In: *56th annual IEEE symposium on foundations of computer science (FOCS 2015)*. IEEE Computer Society, 2015, pages 59–78. 10.1109/FOCS.2015.14.

2 Amir Abboud, Karl Bringmann, and Nick Fischer. “Stronger 3-SUM lower bounds for approximate distance oracles via additive combinatorics”. In: *55th annual ACM symposium on theory of computing (STOC 2023)*. To appear. ACM, 2023. 10.48550/arXiv.2211.07058.

3 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. “SETH-based lower bounds for subset sum and bicriteria path”. In: *ACM trans. algorithms* 18.1 (2022), pages 6:1–6:22. 10.1145/3450524.

4 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. “Hardness of approximation in P via short cycle removal: Cycle detection, distance oracles, and beyond”. In: *54th annual ACM symposium on theory of computing (STOC 2022)*. ACM, 2022, pages 1487–1500. 10.1145/3519935.3520066.

5 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. “Subcubic equivalences between graph centrality problems, APSP and diameter”. In: *26th annual ACM-SIAM symposium on discrete algorithms (SODA 2015)*. SIAM, 2015, pages 1681–1697. 10.1137/1.9781611973730.112.

6 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. “Simulating branching programs with edit distance and friends, or: A polylog shaved is a lower bound made”. In: *48th annual ACM symposium on theory of computing (STOC 2016)*. ACM, 2016, pages 375–388. 10.1145/2897518.2897653.

7 Amir Abboud, Seri Khoury, Oree Leibowitz, and Ron Safier. “Listing 4-cycles”. In: *Corr* (2022). 10.48550/arXiv.2211.10022.

8 Amir Abboud, Kevin Lewi, and Ryan Williams. “Losing weight by gaining edges”. In: *22th annual european symposium on algorithms (ESA 2014)*. Vol. 8737. Lecture Notes in Computer Science. Springer, 2014, pages 1–12. 10.1007/978-3-662-44777-2_1.

9 Amir Abboud and Virginia Vassilevska Williams. “Popular conjectures imply strong lower bounds for dynamic problems”. In: *55th annual IEEE symposium on foundations of computer science (FOCS 2014)*. IEEE Computer Society, 2014, pages 434–443. 10.1109/FOCS.2014.53.

10 Karl R. Abrahamson. “Generalized string matching”. In: *SIAM j. comput.* 16.6 (1987), pages 1039–1051. 10.1137/0216067.

11 Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. “Lower bounds for multiplication via network coding”. In: *46th international colloquium on automata, languages, and programming (ICALP 2019)*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pages 10:1–10:12. 10.4230/LIPIcs.ICALP.2019.10.

12 Nir Ailon. “A lower bound for Fourier transform computation in a linear model over 2×2 unitary gates using matrix entropy”. In: *Chic. j. theor. comput. sci.* 2013 (2013). http://cjtcs.cs.uchicago.edu/articles/2013/12/contents.html.

13 Maor Akav and Liam Roditty. “An almost 2-approximation for all-pairs of shortest paths in subquadratic time”. In: *31st annual ACM-SIAM symposium on discrete algorithms (SODA 2020)*. SIAM, 2020, pages 1–11. 10.1137/1.9781611975994.1.

14 Josh Alman and Virginia Vassilevska Williams. “A refined laser method and faster matrix multiplication”. In: *32nd annual ACM-SIAM symposium on discrete algorithms (SODA 2021)*. SIAM, 2021, pages 522–539. 10.1137/1.9781611976465.32.

15 Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. “Witnesses for boolean matrix multiplication and for shortest paths”. In: *33rd annual IEEE symposium on foundations of computer science (FOCS 1992)*. IEEE Computer Society, 1992, pages 417–426. 10.1109/SFCS.1992.267748.

16 Noga Alon, Raphael Yuster, and Uri Zwick. “Finding and counting given length cycles”. In: *Algorithmica* 17.3 (1997), pages 209–223. 10.1007/BF02523189.

17 Amihood Amir, Ayelet Butman, and Ely Porat. “On the relationship between histogram indexing and block-mass indexing”. In: *Philosophical transactions of the royal society a: Mathematical, physical and engineering sciences* 372 (2014). 10.1098/rsta.2013.0132.

18 Amihood Amir, Oren Kapah, and Ely Porat. “Deterministic length reduction: fast convolution in sparse data and applications”. In: *18th annual symposium on combinatorial pattern matching (CPM 2007)*. Vol. 4580. Lecture Notes in Computer Science. Springer, 2007, pages 183–194. 10.1007/978-3-540-73437-6_20.

19 Alexandr Andoni. “High frequency moments via max-stability”. In: *IEEE international conference on acoustics, speech and signal processing (ICASSP 2017)*. IEEE, 2017, pages 6364–6368. 10.1109/ICASSP.2017.7953381.

20 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. “Polylogarithmic approximation for edit distance and the asymmetric query complexity”. In: *51st annual IEEE symposium on foundations of computer science (FOCS 2010)*. IEEE Computer Society, 2010, pages 377–386. 10.1109/FOCS.2010.43.

21 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. “Streaming algorithms via precision sampling”. In: *52nd annual IEEE symposium on foundations of computer science (FOCS 2011)*. IEEE Computer Society, 2011, pages 363–372. 10.1109/FOCS.2011.82.

22 Alexandr Andoni and Negev Shekel Nosatzki. “Edit distance in near-linear time: it’s a constant factor”. In: *61st annual IEEE symposium on foundations of computer science (FOCS 2020)*. IEEE, 2020, pages 990–1001. 10.1109/FOCS46700.2020.00096.

23 Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. “Estimating the longest increasing subsequence in nearly optimal time”. In: *63rd annual IEEE symposium on foundations of computer science (FOCS 2022)*. IEEE, 2022, pages 708–719. 10.1109/FOCS54457.2022.00073.

24 Alexandr Andoni and Krzysztof Onak. “Approximating edit distance in near-linear time”. In: *SIAM j. comput.* 41.6 (2012), pages 1635–1648. 10.1137/090767182.

25 Andrew Arnold and Daniel S. Roche. “Output-sensitive algorithms for sumset and sparse polynomial multiplication”. In: *40th international symposium on symbolic and algebraic computation (ISSAC 2015)*. ACM, 2015, pages 29–36. 10.1145/2755996.2756653.

26 Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. “Near-linear time construction of sparse neighborhood covers”. In: *SIAM j. comput.* 28.1 (1998), pages 263–277. 10.1137/S0097539794271898.

27 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. “Fast and simple modular subset sum”. In: *4th symposium on simplicity in algorithms (SOSA 2021)*. SIAM, 2021, pages 57–67. 10.1137/1.9781611976496.6.

28 Arturs Backurs and Piotr Indyk. “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)”. In: *SIAM j. comput.* 47.3 (2018), pages 1087–1097. 10.1137/15M1053128.

29 Antal Balog. “Many additive quadruples”. In: *Additive combinatorics*. Vol. 43. CRM Proc. Lecture Notes. Amer. Math. Soc., 2007, pages 39–49. https://doi.org/10.1090/crmp/043.

30 Antal Balog and Endre Szemerédi. “A statistical theorem of set addition”. In: *Combinatorica* 14 (1994), pages 263–268. https://doi.org/10.1007/BF01212974.

[31] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. “Approximating edit distance efficiently”. In: *45th annual IEEE symposium on foundations of computer science (FOCS 2004)*. IEEE Computer Society, 2004, pages 550–559. 10.1109/FOCS.2004.14.

[32] Ilya Baran, Erik D. Demaine, and Mihai Patrascu. “Subquadratic algorithms for 3SUM”. In: *Algorithmica* 50.4 (2008), pages 584–596. 10.1007/s00453-007-9036-3.

[33] Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. “Distance oracles for unweighted graphs: breaking the quadratic barrier with constant additive error”. In: *35th international colloquium on automata, languages, and programming (ICALP 2008)*. Vol. 5125. Lecture Notes in Computer Science. Springer, 2008, pages 609–621. 10.1007/978-3-540-70575-8_50.

[34] Surender Baswana, Vishrut Goyal, and Sandeep Sen. “All-pairs nearly 2-approximate shortest-paths in $O(n^2 \operatorname{polylog} n)$ time”. In: *22nd annual symposium on theoretical aspects of computer science (STACS 2005)*. Vol. 3404. Lecture Notes in Computer Science. Springer, 2005, pages 666–679. 10.1007/978-3-540-31856-9_55.

[35] Surender Baswana and Telikepalli Kavitha. “Faster algorithms for all-pairs approximate shortest paths in undirected graphs”. In: *SIAM j. comput.* 39.7 (2010), pages 2865–2896. 10.1137/080737174.

[36] Surender Baswana and Sandeep Sen. “Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time”. In: *ACM trans. algorithms* 2.4 (2006), pages 557–577. 10.1145/1198513.1198518.

[37] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. “A sublinear algorithm for weakly approximating edit distance”. In: *35th annual ACM symposium on theory of computing (STOC 2003)*. ACM, 2003, pages 316–324. 10.1145/780542.780590.

[38] Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. “Oblivious string embeddings and edit distance approximations”. In: *17th annual ACM-SIAM symposium on discrete algorithms (SODA 2006)*. ACM Press, 2006, pages 792–801. http://dl.acm.org/citation.cfm?id=1109557.1109644.

[39] Ulrich Baum. “Existence and efficient construction of fast Fourier transforms on supersolvable groups”. In: *Comput. complex.* 1 (1991), pages 235–256. 10.1007/BF01200062.

[40] Ulrich Baum and Michael Clausen. “Some lower and upper complexity bounds for generalized Fourier transforms and their inverses”. In: *SIAM j. comput.* 20.3 (1991), pages 451–459. 10.1137/0220028.

[41] Ulrich Baum, Michael Clausen, and Benno Tietz. “Improved upper complexity bounds for the discrete Fourier transform”. In: *Appl. algebra eng. commun. comput.* 2 (1991), pages 35–43. 10.1007/BF01810853.

[42] Walter Baur and Volker Strassen. “The complexity of partial derivatives”. In: *Theor. comput. sci.* 22 (1983), pages 317–330. 10.1016/0304-3975(83)90110-X.

[43] Uri Ben-Levy and Merav Parter. “New $(\alpha, \beta)$ spanners and hopsets”. In: *31st annual ACM-SIAM symposium on discrete algorithms (SODA 2020)*. SIAM, 2020, pages 1695–1714. 10.1137/1.9781611975994.104.

[44] Michael Ben-Or and Prasoon Tiwari. “A deterministic algorithm for sparse multivariate polynominal interpolation”. In: *20th annual ACM symposium on theory of computing (STOC 1988)*. ACM, 1988, pages 301–309. 10.1145/62212.62241.

[45] Elwyn R. Berlekamp. “Nonbinary BCH decoding”. In: *IEEE trans. inf. theory* 14.2 (1968), pages 242. 10.1109/TIT.1968.1054109.

[46] Thomas Beth. *Verfahren der schnellen Fourier-Transformation: Die allgemeine diskrete Fourier-Transformation–ihre algebraische Beschreibung, Komplexität und Implementierung*. Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1984. ISBN: 9783519023630. https://books.google.de/books?id=hk7vAAAAMAAJ.

[47] Etienne Birmelé, Rui A. Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. “Optimal listing of cycles and st-paths in undirected graphs”. In: *24th annual ACM-SIAM symposium on discrete algorithms (SODA 2013)*. SIAM, 2013, pages 1884–1896. 10.1137/1.9781611973105.134.

[48] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. “Listing triangles”. In: *41st international colloquium on automata, languages, and programming (ICALP 2014)*. Vol. 8572. Lecture Notes in Computer Science. Springer, 2014, pages 223–234. 10.1007/978-3-662-43948-7_19.

[49] Leo I. Bluestein. “A linear filtering approach to the computation of discrete Fourier transform”. In: *Ieee transactions on audio and electroacoustics* 18.4 (1970), pages 451–455.

[50] Jean Bourgain. “On Lipschitz embedding of finite metric spaces in Hilbert space”. In: *Israel journal of mathematics* 52.1 (1985), pages 46–52. ISSN: 1565-8511. 10.1007/BF02776078.

[51] Joshua Brakensiek and Aviad Rubinstein. “Constant-factor approximation of near-linear edit distance in near-linear time”. In: *52nd annual ACM symposium on theory of computing (STOC 2020)*. ACM, 2020, pages 685–698. 10.1145/3357713.3384282.

[52] Karl Bringmann. “A near-linear pseudopolynomial time algorithm for subset sum”. In: *28th annual ACM-SIAM symposium on discrete algorithms (SODA 2017)*. SIAM, 2017, pages 1073–1084. 10.1137/1.9781611974782.69.

[53] Karl Bringmann. “Why walking the dog takes time: frechet distance has no strongly subquadratic algorithms unless SETH fails”. In: *55th annual IEEE symposium on foundations of computer science (FOCS 2014)*. IEEE Computer Society, 2014, pages 661–670. 10.1109/FOCS.2014.76.

[54] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. “A structural investigation of the approximability of polynomial-time problems”. In: *49th international colloquium on automata, languages, and programming (ICALP 2022)*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pages 30:1–30:20. 10.4230/LIPIcs.ICALP.2022.30.

[55] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. “Fine-grained completeness for optimization in P”. In: *24th international conference on approximation, randomization, and combinatorial optimization (APPROX/RANDOM 2021)*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pages 9:1–9:22. 10.4230/LIPIcs.APPROX/RANDOM.2021.9.

[56] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. “Almost-optimal sublinear-time edit distance in the low distance regime”. In: *54th annual ACM symposium on theory of computing (STOC 2022)*. ACM, 2022, pages 1102–1115. 10.1145/3519935.3519990.

[57] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. “Improved sublinear-time edit distance for preprocessed strings”. In: *49th international colloquium on automata, languages, and programming (ICALP 2022)*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pages 32:1–32:20. 10.4230/LIPIcs.ICALP.2022.32.

[58] Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. “Faster minimization of tardy processing time on a single machine”. In: *47th international colloquium on automata, languages, and programming (ICALP 2020)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pages 19:1–19:12. 10.4230/LIPIcs.ICALP.2020.19.

[59] Karl Bringmann, Nick Fischer, and Marvin Künnemann. “A fine-grained analogue of schaefer’s theorem in P: dichotomy of $\exists^k\forall$-quantified first-order graph properties”. In: *34th computational complexity conference (CCC 2019)*. Vol. 137. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pages 31:1–31:27. 10.4230/LIPIcs.CCC.2019.31.

[60] Karl Bringmann, Nick Fischer, and Vasileios Nakos. “Deterministic and Las Vegas algorithms for sparse nonnegative convolution”. In: *33rd annual ACM-*

*SIAM symposium on discrete algorithms (SODA 2022)*. SIAM, 2022, pages 3069–3090. 10.1137/1.9781611977073.119.

61 Karl Bringmann, Nick Fischer, and Vasileios Nakos. “Sparse nonnegative convolution is equivalent to dense nonnegative convolution”. In: *53rd annual ACM symposium on theory of computing (STOC 2021)*. ACM, 2021, pages 1711–1724. 10.1145/3406325.3451090.

62 Karl Bringmann and Tobias Friedrich. “Exact and efficient generation of geometric random variates and random graphs”. In: *40th international colloquium on automata, languages, and programming (ICALP 2013)*. Vol. 7965. Lecture Notes in Computer Science. Springer, 2013, pages 267–278. 10.1007/978-3-642-39206-1_23.

63 Karl Bringmann and Marvin Künnemann. “Quadratic conditional lower bounds for string problems and dynamic time warping”. In: *56th annual IEEE symposium on foundations of computer science (FOCS 2015)*. IEEE Computer Society, 2015, pages 79–97. 10.1109/FOCS.2015.15.

64 Karl Bringmann and Vasileios Nakos. “A fine-grained perspective on approximating subset sum and partition”. In: *32nd annual ACM-SIAM symposium on discrete algorithms (SODA 2021)*. SIAM, 2021, pages 1797–1815. 10.1137/1.9781611976465.108.

65 Karl Bringmann and Vasileios Nakos. “Fast $n$-fold boolean convolution via additive combinatorics”. In: *48th international colloquium on automata, languages, and programming (ICALP 2021)*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pages 41:1–41:17. 10.4230/LIPIcs.ICALP.2021.41.

66 Karl Bringmann and Vasileios Nakos. “Top-$k$-convolution and the quest for near-linear output-sensitive subset sum”. In: *52nd annual ACM symposium on theory of computing (STOC 2020)*. ACM, 2020, pages 982–995. 10.1145/3357713.3384308.

67 David E. Cardoze and Leonard J. Schulman. “Pattern matching for spatial point sets”. In: *39th annual IEEE symposium on foundations of computer science (FOCS 1998)*. IEEE Computer Society, 1998, pages 156–165. 10.1109/SFCS.1998.743439.

68 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. “Approximating edit distance within constant factor in truly sub-quadratic time”. In: *J. ACM* 67.6 (2020), pages 36:1–36:22. 10.1145/3422823.

69 Timothy M. Chan. “The art of shaving logs”. In: *13th algorithms and data structures symposium (WADS 2013)*. Vol. 8037. Lecture Notes in Computer Science. Springer, 2013, pages 231. 10.1007/978-3-642-40104-6_20.

70 Timothy M. Chan and Qizheng He. “Reducing 3SUM to convolution-3SUM”. In: *3rd symposium on simplicity in algorithms (SOSA 2020)*. SIAM, 2020, pages 1–7. 10.1137/1.9781611976014.1.

71 Timothy M. Chan and Moshe Lewenstein. “Clustered integer 3SUM via additive combinatorics”. In: *47th annual ACM symposium on theory of computing (STOC 2015)*. ACM, 2015, pages 31–40. 10.1145/2746539.2746568.

72 Timothy M. Chan and R. Ryan Williams. “Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky”. In: *ACM trans. algorithms* 17.1 (2021), pages 2:1–2:14. 10.1145/3402926.

73 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. “Almost optimal distance oracles for planar graphs”. In: *51st annual ACM symposium on theory of computing (STOC 2019)*. ACM, 2019, pages 138–151. 10.1145/3313276.3316316.

74 Shiri Chechik. “Approximate distance oracles with constant query time”. In: *46th annual ACM symposium on theory of computing (STOC 2014)*. ACM, 2014, pages 654–663. 10.1145/2591796.2591801.

[75] Shiri Chechik. “Approximate distance oracles with improved bounds”. In: *47th annual ACM symposium on theory of computing (STOC 2015)*. ACM, 2015, pages 1–10. 10.1145/2746539.2746562.

[76] Shiri Chechik. “Near-optimal approximate decremental all pairs shortest paths”. In: *59th annual IEEE symposium on foundations of computer science (FOCS 2018)*. IEEE Computer Society, 2018, pages 170–181. 10.1109/FOCS.2018.00025.

[77] Shiri Chechik and Tianyi Zhang. “Nearly 2-approximate distance oracles in subquadratic time”. In: *33rd annual ACM-SIAM symposium on discrete algorithms (SODA 2022)*. SIAM, 2022, pages 551–580. 10.1137/1.9781611977073.26.

[78] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. “Maximum flow and minimum-cost flow in almost-linear time”. In: *63rd annual IEEE symposium on foundations of computer science (FOCS 2022)*. IEEE, 2022, pages 612–623. 10.1109/FOCS54457.2022.00064.

[79] Qi Cheng. “Constructing finite field extensions with large order elements”. In: *SIAM j. discret. math.* 21.3 (2007), pages 726–730. 10.1137/S0895480104445514.

[80] Qi Cheng. “On the construction of finite field elements of large order”. In: *Finite fields and their applications* 11.3 (2005). Ten Year Anniversary Edition!, pages 358–366. ISSN: 1071-5797. https://doi.org/10.1016/j.ffa.2005.06.001.

[81] Michael Clausen. “Fast generalized Fourier transforms”. In: *Theor. comput. sci.* 67.1 (1989), pages 55–63. 10.1016/0304-3975(89)90021-2.

[82] Edith Cohen. “Fast algorithms for constructing $t$-spanners and paths with stretch $t$”. In: *SIAM j. comput.* 28.1 (1998), pages 210–236. 10.1137/S0097539794261295.

[83] Edith Cohen and Uri Zwick. “All-pairs small-stretch paths”. In: *J. algorithms* 38.2 (2001), pages 335–353. 10.1006/jagm.2000.1117.

[84] Richard Cole and Ramesh Hariharan. “Verifying candidate matches in sparse and wildcard matching”. In: *34th annual ACM symposium on theory of computing (STOC 2002)*. ACM, 2002, pages 592–601. 10.1145/509907.509992.

[85] James W. Cooley and John W. Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of computation* 19 (1965), pages 297–301.

[86] Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. “On problems equivalent to $(\min,+)$-convolution”. In: *ACM trans. algorithms* 15.1 (2019), pages 14:1–14:25. 10.1145/3293465.

[87] Martin Dietzfelbinger. “Universal hashing via integer arithmetic without primes, revisited”. In: *Adventures between lower bounds and higher altitudes – essays dedicated to juraj hromkovič on the occasion of his 60th birthday*. Vol. 11011. Lecture Notes in Computer Science. Springer, 2018, pages 257–279. 10.1007/978-3-319-98355-4_15.

[88] Jack J. Dongarra and Francis Sullivan. “Guest editors introduction to the top 10 algorithms”. In: *Comput. sci. eng.* 2.1 (2000), pages 22–23. 10.1109/MCISE.2000.814652.

[89] Dorit Dor, Shay Halperin, and Uri Zwick. “All-pairs almost shortest paths”. In: *SIAM j. comput.* 29.5 (2000), pages 1740–1759. 10.1137/S0097539797327908.

[90] Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. “New tradeoffs for decremental approximate all-pairs shortest paths”. In: *Corr* (2022). 10.48550/arXiv.2211.01152.

[91] Ran Duan, Hongxun Wu, and Renfei Zhou. “Faster matrix multiplication via asymmetric hashing”. In: *Corr* (2022). 10.48550/arXiv.2210.10173.

[92] Alok Dutt and Vladimir Rokhlin. “Fast Fourier transforms for nonequispaced data”. In: *SIAM j. sci. comput.* 14.6 (1993), pages 1368–1393. 10.1137/0914081.

93 Michael Elkin and David Peleg. “$(1+\epsilon,\beta)$-spanner constructions for general graphs”. In: *SIAM j. comput.* 33.3 (2004), pages 608–631. 10.1137/S0097539701393384.

94 Martin Farach. “Optimal suffix tree construction with large alphabets”. In: *38th annual IEEE symposium on foundations of computer science (FOCS 1997)*. IEEE Computer Society, 1997, pages 137–143. 10.1109/SFCS.1997.646102.

95 Michael J. Fischer and Albert R. Meyer. “Boolean matrix multiplication and transitive closure”. In: *12th annual symposium on switching and automata theory (swat 1971)*. IEEE Computer Society, 1971, pages 129–131. 10.1109/SWAT.1971.4.

96 Michael J. Fischer and Michael S. Paterson. *String-matching and other products*. Tech. rep. USA: Massachusetts Institute of Technology, 1974.

97 Nick Fischer and Rob van Glabbeek. “Axiomatising infinitary probabilistic weak bisimilarity of finite-state behaviours”. In: *J. log. algebraic methods program.* 102 (2019), pages 64–102. 10.1016/j.jlamp.2018.09.006.

98 Nick Fischer and Christian Ikenmeyer. “The computational complexity of plethysm coefficients”. In: *Comput. complex.* 29.2 (2020), pages 8. 10.1007/s00037-020-00198-4.

99 Sebastian Forster, Gramoz Goranci, and Monika Henzinger. “Dynamic maintenance of low-stretch probabilistic tree embeddings with applications”. In: *32nd annual ACM-SIAM symposium on discrete algorithms (SODA 2021)*. SIAM, 2021, pages 1226–1245. 10.1137/1.9781611976465.75.

100 Simon Foucart and Holger Rauhut. *A mathematical introduction to compressive sensing*. Birkhäuser Basel, 2013. ISBN: 9780817649487 0817649484. 10.1007/978-0-8176-4948-7.

101 Anka Gajentaan and Mark H. Overmars. “On a class of $O(n^2)$ problems in computational geometry”. In: *Comput. geom.* 5 (1995), pages 165–185. 10.1016/0925-7721(95)00022-2.

102 Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. 3rd. Cambridge University Press, 2013. 10.1017/CBO9781139856065.

103 Anna C. Gilbert, Sudipto Guha, Piotr Indyk, S. Muthukrishnan, and Martin Strauss. “Near-optimal sparse Fourier representations via sampling”. In: *34th annual ACM symposium on theory of computing (STOC 2002)*. ACM, 2002, pages 152–161. 10.1145/509907.509933.

104 Anna C. Gilbert and Piotr Indyk. “Sparse recovery using sparse matrices”. In: *Proc. IEEE* 98.6 (2010), pages 937–947. 10.1109/JPROC.2010.2045092.

105 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. “Approximate sparse recovery: optimizing time and measurements”. In: *42nd annual ACM symposium on theory of computing (STOC 2010)*. ACM, 2010, pages 475–484. 10.1145/1806689.1806755.

106 Anna C. Gilbert, Senthilmurugan Muthukrishnan, and Martin J. Strauss. “Improved time bounds for near-optimal sparse Fourier representations”. In: *Wavelets xi*. Vol. 5914. International Society for Optics and Photonics. SPIE, 2005, pages 59141A. 10.1117/12.615931.

107 Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. “$\ell_2/\ell_2$-foreach sparse recovery with low risk”. In: *40th international colloquium on automata, languages, and programming (ICALP 2013)*. Vol. 7965. Lecture Notes in Computer Science. Springer, 2013, pages 461–472. 10.1007/978-3-642-39206-1_39.

108 Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. “Essentially optimal sparse polynomial multiplication”. In: *45th international symposium on symbolic and algebraic computation (ISSAC 2020)*. ACM, 2020, pages 202–209. 10.1145/3373207.3404026.

109 Bernard Gold and Charles M. Rader. *Digital processing of signals*. McGraw-Hill, 1969.

110 Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. “Gap edit distance via non-adaptive queries: simple and optimal”. In: *63rd an-*

*nual IEEE symposium on foundations of computer science (FOCS 2022)*. IEEE, 2022, pages 674–685. 10.1109/FOCS54457.2022.00070.

[111] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. “Sublinear algorithms for gap edit distance”. In: *60th annual IEEE symposium on foundations of computer science (FOCS 2019)*. IEEE Computer Society, 2019, pages 1101–1120. 10.1109/FOCS.2019.00070.

[112] Oded Goldreich. “Combinatorial property testing (a survey)”. In: *Randomization methods in algorithm design*. Vol. 43. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1997, pages 45–59. 10.1090/dimacs/043/04.

[113] Oded Goldreich, Shafi Goldwasser, and Dana Ron. “Property testing and its connection to learning and approximation”. In: *J. ACM* 45.4 (1998), pages 653–750. 10.1145/285055.285060.

[114] Timothy W. Gowers. “A new proof of Szemerédi’s theorem”. In: *Gafa geometric and functional analysis* 11 (Aug. 2001), pages 465–588. 10.1007/s00039-001-0332-9.

[115] Ronald L. Graham. “Bounds on multiprocessing timing anomalies”. In: *SIAM journal of applied mathematics* 17.2 (1969), pages 416–429. 10.1137/0117039.

[116] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997. ISBN: 0-521-58519-8. 10.1017/cbo9780511574931.

[117] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. “Deterministic algorithms for decremental approximate shortest paths: faster and simpler”. In: *31st annual ACM-SIAM symposium on discrete algorithms (SODA 2020)*. SIAM, 2020, pages 2522–2541. 10.1137/1.9781611975994.154.

[118] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. “Nearly optimal sparse Fourier transform”. In: *44th annual ACM symposium on theory of computing (STOC 2012)*. ACM, 2012, pages 563–578. 10.1145/2213977.2214029.

[119] Ishay Haviv and Oded Regev. “The restricted isometry property of subsampled Fourier matrices”. In: *27th annual ACM-SIAM symposium on discrete algorithms (SODA 2016)*. SIAM, 2016, pages 288–297. 10.1137/1.9781611974331.ch22.

[120] Danny Hermelin, Hendrik Molter, and Dvir Shabtay. “Single machine weighted number of tardy jobs minimization with small weights”. In: *Corr* (2022). https://arxiv.org/abs/2202.06841.

[121] Joris van der Hoeven and Grégoire Lecerf. “On the complexity of multivariate blockwise polynomial multiplication”. In: *37th international symposium on symbolic and algebraic computation (ISSAC 2012)*. ACM, 2012, pages 211–218. 10.1145/2442829.2442861.

[122] Qiao-Long Huang. “Sparse polynomial interpolation based on derivatives”. In: *J. symb. comput.* 114 (2023), pages 359–375. 10.1016/j.jsc.2022.06.002.

[123] Qiao-Long Huang. “Sparse polynomial interpolation over fields with large or zero characteristic”. In: *44th international symposium on symbolic and algebraic computation (ISSAC 2019)*. ACM, 2019, pages 219–226. 10.1145/3326229.3326250.

[124] Piotr Indyk. “Faster algorithms for string matching problems: Matching the convolution bound”. In: *39th annual IEEE symposium on foundations of computer science (FOCS 1998)*. IEEE Computer Society, 1998, pages 166–173. 10.1109/SFCS.1998.743440.

[125] Piotr Indyk and Michael Kapralov. “Sample-optimal Fourier sampling in any constant dimension”. In: *55th annual IEEE symposium on foundations of computer science (FOCS 2014)*. IEEE Computer Society, 2014, pages 514–523. 10.1109/FOCS.2014.61.

[126] Piotr Indyk, Michael Kapralov, and Eric Price. “(Nearly) sample-optimal sparse Fourier transform”. In: *25th annual ACM-SIAM symposium on discrete algorithms (SODA 2014)*. SIAM, 2014, pages 480–499. 10.1137/1.9781611973402.36.

[127] Piotr Indyk, Eric Price, and David P. Woodruff. “On the power of adaptivity in sparse recovery”. In: *52nd annual IEEE symposium on foundations of computer science (FOCS 2011)*. IEEE Computer Society, 2011, pages 285–294. 10.1109/FOCS.2011.83.

[128] Zahra Jafargholi and Emanuele Viola. “3SUM, 3XOR, triangles”. In: *Algorithmica* 74.1 (2016), pages 326–343. 10.1007/s00453-014-9946-9.

[129] Ce Jin and Hongxun Wu. “A simple near-linear pseudopolynomial time randomized algorithm for subset sum”. In: *2nd symposium on simplicity in algorithms (SOSA 2019)*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pages 17:1–17:6. 10.4230/OASIcs.SOSA.2019.17.

[130] Ce Jin and Yinzhan Xu. “Removing additive structure in 3SUM-based reductions”. In: *55th annual ACM symposium on theory of computing (STOC 2023)*. To appear. ACM, 2023. 10.48550/arXiv.2211.07058.

[131] Erich Kaltofen. “Fifteen years after DSC and WLSS2 what parallel computations I do today: Invited lecture at PASCO 2010”. In: *4th international workshop on parallel symbolic computation (PASCO 2010)*. ACM, 2010, pages 10–17. 10.1145/1837210.1837213.

[132] Erich Kaltofen and Yagati N. Lakshman. “Improved sparse multivariate polynomial interpolation algorithms”. In: *13th international symposium on symbolic and algebraic computation (ISSAC 1988)*. Vol. 358. Lecture Notes in Computer Science. Springer, 1988, pages 467–474. 10.1007/3-540-51084-2_44.

[133] Michael Kapralov. “Sample efficient estimation and recovery in sparse FFT via isolation on average”. In: *58th annual IEEE symposium on foundations of computer science (FOCS 2017)*. IEEE Computer Society, 2017, pages 651–662. 10.1109/FOCS.2017.66.

[134] Michael Kapralov. “Sparse Fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time”. In: *48th annual ACM symposium on theory of computing (STOC 2016)*. ACM, 2016, pages 264–277. 10.1145/2897518.2897650.

[135] Michael Kapralov, Ameya Velingker, and Amir Zandieh. “Dimension-independent sparse Fourier transform”. In: *30th annual ACM-SIAM symposium on discrete algorithms (SODA 2019)*. SIAM, 2019, pages 2709–2728. 10.1137/1.9781611975482.168.

[136] Richard M. Karp. “Reducibility among combinatorial problems”. In: *Symposium on the complexity of computer computations at the IBM thomas j. watson research center*. The IBM Research Symposia Series. Plenum Press, New York, 1972, pages 85–103. 10.1007/978-1-4684-2001-2_9.

[137] Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. “On minimizing tardy processing time, max-min skewed convolution, and triangular structured ilps”. In: *34th annual ACM-SIAM symposium on discrete algorithms (SODA 2023)*. SIAM, 2023, pages 2947–2960. 10.1137/1.9781611977554.ch112.

[138] Mathias Bæk Tejs Knudsen. “Additive spanners and distance oracles in quadratic time”. In: *44th international colloquium on automata, languages, and programming (ICALP 2017)*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pages 64:1–64:12. 10.4230/LIPIcs.ICALP.2017.64.

[139] Mathias Bæk Tejs Knudsen. “Linear hashing is awesome”. In: *57th annual IEEE symposium on foundations of computer science (FOCS 2016)*. IEEE Computer Society, 2016, pages 345–352. 10.1109/FOCS.2016.45.

[140] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. “Fast pattern matching in strings”. In: *SIAM j. comput.* 6.2 (1977), pages 323–350. 10.1137/0206024.

[141] Tomasz Kociumaka and Barna Saha. “Sublinear-time algorithms for computing & embedding gap edit distance”. In: *61st annual IEEE symposium on foundations of computer science (FOCS 2020)*. IEEE, 2020, pages 1168–1179. 10.1109/FOCS46700.2020.00112.

142 Konstantinos Koiliaris and Chao Xu. “Faster pseudopolynomial time algorithms for subset sum”. In: *ACM trans. algorithms* 15.3 (2019), pages 40:1–40:20. 10.1145/3329863.

143 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. “Higher lower bounds from the 3SUM conjecture”. In: *27th annual ACM-SIAM symposium on discrete algorithms (SODA 2016)*. SIAM, 2016, pages 1272–1287. 10.1137/1.9781611974331.ch89.

144 S. Rao Kosaraju. “Efficient tree pattern matching”. In: *30th annual IEEE symposium on foundations of computer science (FOCS 1989)*. IEEE Computer Society, 1989, pages 178–183. 10.1109/SFCS.1989.63475.

145 Michal Koucký and Michael E. Saks. “Constant factor approximations to edit distance on far input pairs in nearly linear time”. In: *52nd annual ACM symposium on theory of computing (STOC 2020)*. ACM, 2020, pages 699–712. 10.1145/3357713.3384307.

146 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. “On the fine-grained complexity of one-dimensional dynamic programming”. In: *44th international colloquium on automata, languages, and programming (ICALP 2017)*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pages 21:1–21:15. 10.4230/LIPIcs.ICALP.2017.21.

147 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. “Incremental string comparison”. In: *SIAM j. comput.* 27.2 (1998), pages 557–582. 10.1137/S0097539794264810.

148 Gad M. Landau and Uzi Vishkin. “Fast string matching with $k$ differences”. In: *J. comput. syst. sci.* 37.1 (1988), pages 63–78. 10.1016/0022-0000(88)90045-1.

149 Eugene L. Lawler and J. Michael Moore. “A functional equation and its application to resource allocation and sequencing problems”. In: *Management science* 16.1 (1969), pages 77–84. 10.1287/mnsc.16.1.77.

150 Hung Le and Christian Wulff-Nilsen. “Optimal approximate distance oracle for planar graphs”. In: *62nd annual IEEE symposium on foundations of computer science (FOCS 2021)*. IEEE, 2021, pages 363–374. 10.1109/FOCS52979.2021.00044.

151 Vladimir I. Levenshtein. “Binary codes capable of correcting deletions, insertions and reversals”. In: *Soviet physics doklady* 10.8 (1966), pages 707–710.

152 Lei Li. “On the arithmetic operational complexity for solving Vandermonde linear equations”. In: *Japan journal of industrial and applied mathematics* 17.15 (2000). 10.1007/BF03167332.

153 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. “Monochromatic triangles, intermediate matrix products, and convolutions”. In: *11th innovations in theoretical computer science conference (ITCS 2020)*. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pages 53:1–53:18. 10.4230/LIPIcs.ITCS.2020.53.

154 Bruce G. Lindsay. “On the determinants of moment matrices”. In: *The annals of statistics* 17.2 (1989), pages 711–721. 10.1214/aos/1176347137.

155 Yaowei Long and Seth Pettie. “Planar distance oracles with better time-space tradeoffs”. In: *32nd annual ACM-SIAM symposium on discrete algorithms (SODA 2021)*. SIAM, 2021, pages 2517–2537. 10.1137/1.9781611976465.149.

156 Colin L. Mallows. “Patience sorting”. In: *SIAM review* 4.2 (1962), pages 148–149. 10.1137/1004036.

157 James L. Massey. “Shift-register synthesis and BCH decoding”. In: *IEEE trans. inf. theory* 15.1 (1969), pages 122–127. 10.1109/TIT.1969.1054260.

158 Jiří Matoušek. “On the distortion required for embedding finite metric spaces into normed spaces”. In: *Israel journal of mathematics* 93.1 (1996), pages 333–344. ISSN: 1565-8511. 10.1007/BF02761110.

159 Manor Mendel and Assaf Naor. “Ramsey partitions and proximity data structures”. In: *47th annual IEEE symposium on foundations of computer science (FOCS 2006)*. IEEE Computer Society, 2006, pages 109–118. 10.1109/FOCS.2006.65.

[160] Michael Mitzenmacher and Saeed Seddighin. “Improved sublinear time algorithm for longest increasing subsequence”. In: *32nd annual ACM-SIAM symposium on discrete algorithms (SODA 2021)*. SIAM, 2021, pages 1934–1947. 10.1137/1.9781611976465.115.

[161] Michael B. Monagan and Roman Pearce. “Parallel sparse polynomial multiplication using heaps”. In: *34th international symposium on symbolic and algebraic computation (ISSAC 2009)*. ACM, 2009, pages 263–270. 10.1145/1576702.1576739.

[162] Jacques Morgenstern. “How to compute fast a function and all its derivatives: a variation on the theorem of baur-strassen”. In: *SIGACT news* 16.4 (1985), pages 60–62. 10.1145/382242.382836.

[163] S. Muthukrishnan. “New results and open problems related to non-standard stringology”. In: *6th annual symposium on combinatorial pattern matching (CPM 1995)*. Vol. 937. Lecture Notes in Computer Science. Springer, 1995, pages 298–317. 10.1007/3-540-60044-2_50.

[164] Vasileios Nakos. “Nearly optimal sparse polynomial multiplication”. In: *IEEE trans. inf. theory* 66.11 (2020), pages 7231–7236. 10.1109/TIT.2020.2989385.

[165] Vasileios Nakos, Zhao Song, and Zhengyu Wang. “(Nearly) sample-optimal sparse Fourier transform in any dimension; RIPless and filterless”. In: *60th annual IEEE symposium on foundations of computer science (FOCS 2019)*. IEEE Computer Society, 2019, pages 1568–1577. 10.1109/FOCS.2019.00092.

[166] Ilan Newman and Nithin Varma. “New sublinear algorithms and lower bounds for LIS estimation”. In: *48th international colloquium on automata, languages, and programming (ICALP 2021)*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pages 100:1–100:20. 10.4230/LIPIcs.ICALP.2021.100.

[167] Rafail Ostrovsky and Yuval Rabani. “Low distortion embeddings for edit distance”. In: *J. ACM* 54.5 (2007), pages 23. 10.1145/1284320.1284322.

[168] Victor Y. Pan. *Structured matrices and polynomials: Unified superfast algorithms*. Berlin, Heidelberg: Springer-Verlag, 2001. ISBN: 0817642404.

[169] Merav Parter. “Bypassing Erdős’ girth conjecture: hybrid stretch and source-wise spanners”. In: *41st international colloquium on automata, languages, and programming (ICALP 2014)*. Vol. 8573. Lecture Notes in Computer Science. Springer, 2014, pages 608–619. 10.1007/978-3-662-43951-7_49.

[170] Mihai Patrascu and Liam Roditty. “Distance oracles beyond the thorup-zwick bound”. In: *SIAM j. comput.* 43.1 (2014), pages 300–311. 10.1137/11084128X.

[171] Mihai Patrascu, Liam Roditty, and Mikkel Thorup. “A new infinity of distance oracles for sparse graphs”. In: *53rd annual IEEE symposium on foundations of computer science (FOCS 2012)*. IEEE Computer Society, 2012, pages 738–747. 10.1109/FOCS.2012.44.

[172] Mihai Pătraşcu. “Towards polynomial lower bounds for dynamic problems”. In: *42nd annual ACM symposium on theory of computing (STOC 2010)*. ACM, 2010, pages 603–610. 10.1145/1806689.1806772.

[173] Nicholas Pippenger. “On the evaluation of powers and monomials”. In: *SIAM j. comput.* 9.2 (1980), pages 230–250. 10.1137/0209022.

[174] Helmut Plünnecke. “Eine zahlentheoretische anwendung der graphentheorie”. In: *Journal für die reine und angewandte mathematik* 1970.243 (1970), pages 171–183. doi:10.1515/crll.1970.243.171.

[175] Eric Price and Zhao Song. “A robust sparse Fourier transform in the continuous setting”. In: *56th annual IEEE symposium on foundations of computer science (FOCS 2015)*. IEEE Computer Society, 2015, pages 583–600. 10.1109/FOCS.2015.42.

[176] Eric Price and David P. Woodruff. “Applications of the shannon-hartley theorem to data streams and sparse recovery”. In: *45th IEEE international symposium on information theory (ISIT 2012)*. IEEE, 2012, pages 2446–2450. 10.1109/ISIT.2012.6283954.

177 Gaspard R. de Prony. “Essai éxperimental et analytique: Sur les lois de la dilatabilité de fluides élastique et sur celles de la force expansive de la vapeur de l’alkool, à différentes températures”. In: *Journal de l’école polytechnique floréal et plairial* 1 (1795), pages 24–76.

178 Zichao Qi, Yanghua Xiao, Bin Shao, and Haixun Wang. “Toward a distance oracle for billion-node graphs”. In: *Proc. VLDB endow.* 7.1 (2013), pages 61–72. 10.14778/2732219.2732225.

179 Daniel S. Roche. “Adaptive polynomial multiplication”. In: *Milestones in computer algebra (MICA 2008): A conference in honour of Keith Geddes’ 60th birthday* (2008), pages 65–72.

180 Daniel S. Roche. “What can (and can’t) we do with sparse polynomials?” In: *43th international symposium on symbolic and algebraic computation (ISSAC 2018)*. ACM, 2018, pages 25–30. 10.1145/3208976.3209027.

181 Daniel N. Rockmore. “Fast Fourier transforms for wreath products”. In: *Applied and computational harmonic analysis* 2.3 (1995), pages 279–292. ISSN: 1063-5203. https://doi.org/10.1006/acha.1995.1020.

182 Liam Roditty and Roei Tov. “Approximate distance oracles with improved stretch for sparse graphs”. In: *27th international conference on computing and combinatorics (COCOON 2021)*. Vol. 13025. Lecture Notes in Computer Science. Springer, 2021, pages 89–100. 10.1007/978-3-030-89543-3_8.

183 Liam Roditty and Virginia Vassilevska Williams. “Fast approximation algorithms for the diameter and radius of sparse graphs”. In: *45th annual ACM symposium on theory of computing (STOC 2013)*. ACM, 2013, pages 515–524. 10.1145/2488608.2488673.

184 J. Barkley Rosser and Lowell Schoenfeld. “Approximate formulas for some functions of prime numbers”. In: *Illinois journal of mathematics* 6.1 (1962), pages 64–94. 10.1215/ijm/1255631807.

185 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. “Approximation algorithms for LCS and LIS with truly improved running times”. In: *60th annual IEEE symposium on foundations of computer science (FOCS 2019)*. IEEE Computer Society, 2019, pages 1121–1145. 10.1109/FOCS.2019.00071.

186 Imre Z. Ruzsa. “An analog of Freiman’s theorem in groups”. In: *Structure theory of set addition*. Astérisque 258. Société mathématique de France, 1999. http://www.numdam.org/item/AST_1999__258__323_0/.

187 Michael E. Saks and C. Seshadhri. “Estimating the longest increasing sequence in polylogarithmic time”. In: *SIAM j. comput.* 46.2 (2017), pages 774–823. 10.1137/130942152.

188 Baruch Schieber and Pranav Sitaraman. “Quick minimization of tardy processing time on a single machine”. In: *Corr* (2023). 10.48550/arXiv.2301.05460.

189 Victor Shoup. “New algorithms for finding irreducible polynomials over finite fields”. In: *29th annual IEEE symposium on foundations of computer science (FOCS 1988)*. IEEE Computer Society, 1988, pages 283–290. 10.1109/SFCS.1988.21944.

190 Victor Shoup. “Searching for primitive roots in finite fields”. In: *22nd annual ACM symposium on theory of computing (STOC 1990)*. ACM, 1990, pages 546–554. 10.1145/100216.100293.

191 Igor E. Shparlinski. “On primitive elements in finite fields and on elliptic curves”. In: *Mathematics of the USSR-sbornik* 71.1 (1992), pages 41–50. 10.1070/SM1992v071n01ABEH001389.

192 Christian Sommer. “All-pairs approximate shortest paths and distance oracle preprocessing”. In: *43rd international colloquium on automata, languages, and programming (ICALP 2016)*. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pages 55:1–55:13. 10.4230/LIPIcs.ICALP.2016.55.

193 Christian Sommer, Elad Verbin, and Wei Yu. “Distance oracles for sparse graphs”. In: *50th annual IEEE symposium on foundations of computer science*

*(FOCS 2009)*. IEEE Computer Society, 2009, pages 703–712. 10.1109/FOCS.2009.27.

194 Kilian Stampfer and Gerlind Plonka. “The generalized operator based prony method”. In: *Constructive approximation* 52.2 (2020), pages 247–282. https://doi.org/10.1007/s00365-020-09501-6.

195 Benny Sudakov, Endre Szemerédi, and Van H. Vu. “On a question of Erdős and Moser”. In: *Duke mathematical journal* 129.1 (2005), pages 129–155. 10.1215/S0012-7094-04-12915-X.

196 Terence Tao and Van H. Vu. *Additive combinatorics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2006. 10.1017/CBO9780511755149.

197 Mikkel Thorup and Uri Zwick. “Approximate distance oracles”. In: *J. ACM* 52.1 (2005), pages 1–24. 10.1145/1044731.1044732.

198 Chris Umans. “Fast generalized DFTs for all finite groups”. In: *60th annual IEEE symposium on foundations of computer science (FOCS 2019)*. IEEE Computer Society, 2019, pages 793–805. 10.1109/FOCS.2019.00052.

199 Taras K. Vintsyuk. “Speech discrimination by dynamic programming”. In: *Cybernetics* 4.1 (1968), pages 52–57. ISSN: 1573-8337. 10.1007/BF01074755.

200 Robert A. Wagner and Michael J. Fischer. “The string-to-string correction problem”. In: *J. ACM* 21.1 (1974), pages 168–173. 10.1145/321796.321811.

201 R. Ryan Williams. “Faster all-pairs shortest paths via circuit complexity”. In: *SIAM j. comput.* 47.5 (2018), pages 1965–1985. 10.1137/15M1024524.

202 Virginia Vassilevska Williams. “On some fine-grained questions in algorithms and complexity”. In: *Proceedings of the international congress of mathematicians (ICM 2018)*. 2018, pages 3447–3487. 10.1142/9789813272880_0188.

203 Virginia Vassilevska Williams and R. Ryan Williams. “Subcubic equivalences between path, matrix, and triangle problems”. In: *J. ACM* 65.5 (2018), pages 27:1–27:38. 10.1145/3186893.

204 Virginia Vassilevska Williams and Yinzhan Xu. “Monochromatic triangles, triangle listing and APSP”. In: *61st annual IEEE symposium on foundations of computer science (FOCS 2020)*. IEEE, 2020, pages 786–797. 10.1109/FOCS46700.2020.00078.

205 Jack K. Wolf. “Decoding of Bose-Chaudhuri-Hocquenghem codes and Prony’s method for curve fitting”. In: *IEEE trans. inf. theory* 13.4 (1967), pages 608. 10.1109/TIT.1967.1054056.

206 Christian Wulff-Nilsen. “Approximate distance oracles with improved preprocessing time”. In: *23rd annual ACM-SIAM symposium on discrete algorithms (SODA 2012)*. SIAM, 2012, pages 202–208. 10.1137/1.9781611973099.18.

207 Christian Wulff-Nilsen. “Approximate distance oracles with improved query time”. In: *24th annual ACM-SIAM symposium on discrete algorithms (SODA 2013)*. SIAM, 2013, pages 539–549. 10.1137/1.9781611973105.39.

208 Andrew Chi-Chih Yao. “On the evaluation of powers”. In: *SIAM j. comput.* 5.1 (1976), pages 100–103. 10.1137/0205008.

209 Raphael Yuster and Uri Zwick. “Finding even cycles even faster”. In: *SIAM j. discret. math.* 10.2 (1997), pages 209–222. 10.1137/S0895480194274133.