Top-k-Convolution and the Quest for Near-Linear Output-Sensitive Subset Sum*

Karl Bringmann† Vasileios Nakos‡

Abstract

In the classical SubsetSum problem we are given a set X and a target t, and the task is to decide whether there exists a subset of X which sums to t. A recent line of research has resulted in $\widetilde{O}(t)$-time algorithms, which are (near-)optimal under popular complexity-theoretic assumptions. On the other hand, the standard dynamic programming algorithm runs in time $O(n \cdot |\mathcal{S}(X,t)|)$, where $\mathcal{S}(X,t)$ is the set of all subset sums of X that are smaller than t. Furthermore, all known pseudopolynomial algorithms actually solve a stronger task, since they actually compute the whole set $\mathcal{S}(X,t)$.

As the aforementioned two running times are incomparable, in this paper we ask whether one can achieve the best of both worlds: running time $\widetilde{O}(|\mathcal{S}(X,t)|)$. In particular, we ask whether $\mathcal{S}(X,t)$ can be computed in near-linear time in the output-size. Using a diverse toolkit containing techniques such as color coding, sparse recovery, and sumset estimates, we make considerable progress towards this question and design an algorithm running in time $\widetilde{O}(|\mathcal{S}(X,t)|^{4/3})$.

Central to our approach is the study of *top-k-convolution*, a natural problem of independent interest: given sparse polynomials with non-negative coefficients, compute the lowest k non-zero monomials of their product. We design an algorithm running in time $\widetilde{O}(k^{4/3})$, by a combination of sparse convolution and sumset estimates considered in Additive Combinatorics. Moreover, we provide evidence that going beyond some of the barriers we have faced requires either an algorithmic breakthrough or possibly new techniques from Additive Combinatorics on how to pass from information on restricted sumsets to information on unrestricted sumsets.

*This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

†Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany. bringmann@cs.uni-saarland.de

‡Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany. vnakos@mpi-inf.mpg.de

Contents

1 Introduction

1.1 Subset Sum

SubsetSum is a fundamental problem at the intersection of computer science, mathematical optimization, and operations research. In this problem, given a set X of n integers and a target t, the task is to decide whether there exists a subset of X that sums to t. The problem belongs to Karp's initial list of NP-complete problems [29], and it has given rise to a plethora of algorithmic techniques, see, e.g., the monographs [30, 34]. Apart from being a cornerstone in algorithm design, SubsetSum draws its importance from being a special case of many other problems, like Knapsack or Integer Programming. It has also played a role in cryptography, as Merkle and Hellman [35] based their cryptosystem on this problem, see also [43, 13, 18, 40, 26].

Several classic algorithms for SubsetSum are typically taught in undergraduate courses, including the meet-in-the-middle algorithm running in time $O(2^{n/2})$ [25] and Bellman's dynamic programming algorithm running in pseudopolynomial time $O(n \cdot t)$ [12].

Surprisingly, after decades of research, major algorithmic advances were still discovered in the last 10 years, e.g., [39, 33, 21, 7, 24, 8, 9, 32, 11, 38, 14, 31, 27, 1, 10]. Among these developments, the most relevant for this paper are improvements over Bellman's $O(n{\cdot}t)$ algorithm: Koiliaris and Xu [31] designed a deterministic algorithm running in time[1] $\widetilde{O}(\min\{\sqrt{n}{\cdot}t, t^{4/3}\})$, and Bringmann [14] devised a randomized algorithm running in time $\widetilde{O}(t)$ (which was improved in terms of log factors in [27]). The running time $\widetilde{O}(t)$ of the randomized algorithms is optimal under the Strong Exponential Time Hypothesis [1] as well as under the SetCover Hypothesis [20].

Thus, research on pseudopolynomial algorithms for SubsetSum with respect to parameter t is more or less finished. However, it remains to study whether the recent improvements generalize to other parameters as well as to variants of SubsetSum. For instance, this has been done for the ModularSubsetSum problem in [10]. In this paper, we start from the observation that Bellman's classic dynamic programming algorithm can be implemented to run in time $O(n \cdot |\mathcal{S}(X,t)|)$, where $\mathcal{S}(X,t)$ is the set of all subset sums of X below t. Since $|\mathcal{S}(X,t)|$ can be much smaller than t, so far the running times $\widetilde{O}(t)$ and $O(n \cdot |\mathcal{S}(X,t)|)$ are incomparable. Thus, despite the running time $\widetilde{O}(t)$ being matched by a conditional lower bound, in situations where $|\mathcal{S}(X,t)|$ is small Bellman's algorithm can outperform the recent improved algorithms. To obtain the best of both worlds, it would thus be desirable to consider $|\mathcal{S}(X,t)|$, rather than t, as the parameter to measure the computational complexity of the problem, and to similarly shave off the factor n from the running time of Bellman's algorithm. In particular, since all previous pseudopolynomial algorithms for SubsetSum produce all attainable subset sums smaller than t, a natural question is whether one can design a *near-linear output-sensitive* algorithm.

Question 1.1. *Is there an algorithm that computes $S(X,t)$ in time $\widetilde{O}(|\mathcal{S}(X,t)|)$?*

Our work struggles to make progress towards understanding SubsetSum under this new computational perspective, and it lead us to study a new type of sparse convolution problem.

1.2 Top-k-Convolution

Convolution and Boolean convolution are fundamental computational primitives that frequently arise in algorithm design, e.g., when combining solutions of two subproblems. The *Boolean convolution* $f \circledast g$ of vectors $f, g \in \{0,1\}^u$ is the vector with entries $(f \circledast g)_k = \bigvee_{0 \le i \le k} f_i \wedge g_{k-i}$ for

[1] By $\widetilde{O}(T)$ we hide factors of the form $\mathrm{polylog}(T)$ as well as factors $\mathrm{polylog}(u)$, where u is the universe size, and $\mathrm{polylog}(t)$, where t is the target.

$0 \leq k < 2u$. This arises when we split a problem into two subproblems, so that the whole problem has a solution of size k if and only if for some i the left subproblem has a solution of size i and the right subproblem has a solution of size $k-i$. Moreover, Boolean convolution is equivalent to *sumset computation*, where we are given sets $A, B \subseteq \{0, 1, \ldots, u-1\}$ and the task is to compute $A + B$, the set of all sums $a + b$ with $a \in A$, $b \in B$.

The *convolution* $f \star g$ of vectors $f, g \in \mathbb{R}^u$ is the vector with entries $(f \star g)_k = \sum_{i=0}^{k} f_i \cdot g_{k-i}$ for $0 \leq k < 2u$. When we split a problem into two subproblems, and f_i and g_i count the number of size-i solutions of the left and right subproblem, then $(f \star g)_k$ counts the number of size-k solutions of the whole problem. Moreover, convolution is equivalent to polynomial multiplication, where we are given the coefficients of two polynomials and want to compute the coefficients of their product.

Boolean convolution can be solved via convolution, and convolution can be solved in time $O(u \log u)$ using Fast Fourier Transform (FFT). However, when the input vectors are sparse, we can ask for algorithms that compute convolutions much faster than performing FFT. Specifically, the ultimate goal is an algorithm running in near-linear output-sensitive time, i.e., in near-linear time in terms of the number of non-zero entries of $f \star g$. This practically and theoretically relevant problem is called sparse convolution (or sparse polynomial multiplication) and has been studied, e.g., in [19, 41, 36, 46, 6, 17, 42, 37, 15]. The ultimate goal of near-linear output-sensitive time has been achieved by Cole and Hariharan [19] for non-negative vectors by a Las Vegas algorithm, in [37] for general vectors by a Monte Carlo algorithm, and in [15] for non-negative vectors by an almost linear time deterministic algorithm.

In this paper we study a natural variant that we call *Top-k-Convolution*: Given vectors $f, g \in \mathbb{R}^u$, compute the k lowest non-zero entries of $f \star g$. Formally, denoting by $i_1 < i_2 < \ldots < i_\ell$ all indices of non-zero entries of $f \star g$, our goal is to compute the pairs $(i_1, (f \star g)_{i_1}), \ldots, (i_k, (f \star g)_{i_k})$. *Boolean Top-$k$-Convolution* is defined analogously, i.e., the task is to compute the lowest k indices $i_1 < \ldots < i_k$ of 1-entries of $f \circledast g$.

"Top-k" problems, that ask for the k best solutions, are well motivated from a practical perspective, e.g., for displaying search results. Note that in the setting where we split a problem into two subproblems, Boolean Top-k-Convolution asks for the k smallest solution sizes. In the polynomial multiplication setting, Top-k-Convolution asks for the k lowest-degree monomials in the product of two given sparse polynomials. The problem is equivalent if we replace "lowest" by "highest" (by reversing f and g), and thus Top-k-Convolution is also equivalent to computing the k highest-degree monomials in the product of two given sparse polynomials. As an additional major application, we present a connection to SubsetSum in this paper.

Therefore, sparse Top-k-Convolution is a well-motivated problem with several applications, and it is surprising that to the best of our knowledge it has not been explicitly studied before. Throughout the paper we will assume that f, g are non-negative vectors, since this is satisfied in many applications and the possible cancelations resulting from negative entries make the problem considerably harder (indeed, sparse convolution with negative entries has been solved much later [37] than on non-negative vectors [19]). Note that Top-k-Convolution on non-negative vectors can be solved naively in time $O(k^2)$: The lowest k non-zeros of $f \star g$ are among the combinations of the lowest k non-zeros of f and the lowest k non-zeros of g. This allows us to assume that f and g are k-sparse, and we can compute $f \star g$ naively in time $O(k^2)$ to obtain its lowest k non-zeros.

1.3 Our Contribution

We initiate the study of SubsetSum with respect to the parameter $|\mathcal{S}(X, t)|$, making considerable progress in this direction. We show an "output-sensitivity preserving" reduction from SubsetSum to Top-k-Convolution, such that if the latter can be solved in near-linear output-sensitive time so can

the former. Then, we investigate upper and lower bounds for Top-k-Convolution on non-negative vectors (as well as related restricted variants of sumset computation and sparse convolution). In particular, we present a randomized $\widetilde{O}(k^{4/3})$-time algorithm for Top-k-Convolution, resulting in time $\widetilde{O}(|\mathcal{S}(X,t)|^{4/3})$ for SubsetSum. Our algorithms fall into a natural class that we call "rectangle covering algorithms", for which we show that they cannot yield near-linear time. Our technical machinery draws from a wide range of techniques such as color coding, sparse convolutions, and sumset estimates from Additive Combinatorics.

2 Results and Techniques

2.1 Preliminaries

We write $\mathbb{N} = \{0, 1, \ldots\}$ and $[n] = \{0, 1, \ldots, n\}$ for any $n \in \mathbb{N}$. For sets $A, B \subseteq \mathbb{N}$ define $A + B = \{a + b \mid a \in A,\, b \in B\}$. We also define $\max(A) = \max_{x \in A} x$, and similarly $\min(A)$. For any set $X \subseteq \mathbb{N}$ we define $\Sigma(X) = \sum_{x \in X} x$. For $t \in \mathbb{N}$ we define the set of all subset sums of X below t as

$$\mathcal{S}(X,t) = \{\Sigma(Y) \mid Y \subseteq X,\, \Sigma(Y) \leq t\}.$$

For integers a, b we write $[a, b] = \{a, \ldots, b\}, (a, b) = \{a + 1, \ldots, b - 1\}$, and similarly for $(a, b]$ and $[a, b)$.

For a vector $f \in \mathbb{R}^d$ we let $\|f\|_0$ be its number of non-zero entries. For a set $S \subseteq [d]$ we denote by f_S the vector that is zeroed out outside of S, i.e., $(f_S)_i = f_i$ for $i \in S$ and $(f_S)_i = 0$ otherwise. We shall use 0-indexed vectors throughout the paper. We also use a non-standard notation of $\widetilde{O}(T)$ throughout the paper that hides factors of the form $\text{polylog}(T)$ as well as $\text{polylog}(u)$, where u is the universe size, or $\text{polylog}(t)$, where t is the target.

We shall need the following result by Cole and Hariharan and an immediate corollary.

Theorem 2.1 ([19])**.** *Given non-negative vectors f, g of length d, we can compute $f \star g$ in expected time $O\left(\|f \star g\|_0 \cdot \log^2 d\right)$.*

Theorem 2.2. *Given $A, B \subseteq [u]$, we can compute $A + B$ in expected time $O\left(|A + B| \cdot \log^2 u\right)$.*

Proof. Let f be the indicator vector of A, g be the indicator vector of $B, d = 2u + 1$ and use Theorem 2.1. □

2.1.1 Problem Definitions

Our work is concerned with the following problems.

Definition 2.3 (Subset Sum)**.** *Given a set $X \subseteq \mathbb{N}$ and a number t, compute $\mathcal{S}(X,t) = \{\Sigma(Y) \mid Y \subseteq X,\, \Sigma(Y) \leq t\}$. We measure the running time in terms of $|\mathcal{S}(X,t)|$. We write $n = |X|$.*

Definition 2.4 (Top-k-Convolution)**.** *Given two vectors $f, g \in \mathbb{R}^u$ and a parameter k, compute the first k non-zero entries of $f \star g$. In other words, find $(i_1, (f \star g)_{i_1}), (i_2, (f \star g)_{i_2}), \ldots (i_k, (f \star g)_{i_k})$ where $i_1 < i_2 < \ldots < i_k$ are the smallest k indices on which $f \star g$ is non-zero. We measure the running time in terms of k. We write $n = \|f\|_0$ and $m = \|g\|_0$.*

Top-k-convolution is equivalent to the following problem, as we will show in Lemma 3.1.

Definition 2.5 (Prefix-Restricted Convolution)**.** *Given a positive integer u and two vectors $f, g \in \mathbb{R}^u$, compute $(f \star g)_{[u]}$, i.e., compute (a sparse representation of) the first u entries of $f \star g$. We measure the running time in terms of the output-size* $\text{out} = \|(f \star g)_{[u]}\|_0$*. We write $n = \|f\|_0$ and $m = \|g\|_0$.*

The following problem is a Boolean version of prefix-restriced convolution.

Definition 2.6 (Prefix-restricted Sumset Computation)**.** *Given $u \in \mathbb{N}$ and $A, B \subseteq [u]$, compute $(A + B) \cap [u]$. We measure the running time in terms of the output-size* $\mathrm{out} = |(A + B) \cap [u]|$*. We write $n = |A|$ and $m = |B|$.*

2.1.2 Covering of Prefix-Restricted Sumsets

For a set $A \subseteq \mathbb{N}$ of size n, we implicitly assume that A is sorted and we denote its elements in sorted order as $A_1 < A_2 < \ldots < A_n$. Moreover, for $I \subseteq \{1, \ldots, n\}$ we write A_I for $\{A_i \mid i \in I\}$.

We define the notion of a covering of a restricted sumset $(A, B, [u])$. Intuitively, we want to cover $(A+B)\cap[u]$ by sumsets of the form A_I+B_J, that is, we want to have $(A+B)\cap[u] \subseteq \bigcup_{(I,J)\in\mathcal{C}} A_I+B_J$.

Definition 2.7. *Let $u \in \mathbb{N}$ and $A, B \subseteq [u]$ with $n = |A|$, $m = |B|$. A **covering** of $(A, B, [u])$ is a family $\mathcal{C}$ such that:*

1. *$\mathcal{C}$ consists of pairs (I, J) where $I \subseteq \{1, \ldots, n\}$ and $J \subseteq \{1, \ldots, m\}$.*
2. *For any $1 \le i \le n$, $1 \le j \le m$ with $A_i + B_j \in [u]$ there exists $(I, J) \in \mathcal{C}$ with $(i, j) \in I \times J$.*

*We call a covering $\mathcal{C}$ **unique** if the pair in property 2. is unique, i.e., for any $1 \le i \le n$, $1 \le j \le m$ with $A_i + B_j \in [u]$ there exists a* unique *pair $(I, J) \in \mathcal{C}$ such that $(i, j) \in I \times J$.*

*We call a covering $\mathcal{C}$ a **rectangle covering** if for any $(I, J) \in \mathcal{C}$ the sets I and J are intervals, meaning that they consists of contiguous elements in the sorted order of A and B, respectively.*

*The **cost** of a covering $\mathcal{C}$ is*

$$\sum_{(I,J)\in\mathcal{C}} |A_I + B_J| .$$

This notion is useful because of the following fact.

Observation 2.8. *Given a covering $\mathcal{C}$ of $(A, B, [u])$ of cost c, we can compute the prefix-restricted sumset $(A + B) \cap [u]$ in expected time $\widetilde{O}(c)$.*

Proof. Using output-sensitive sumset computation (Theorem 2.2), we can compute $A_I + B_J$ in expected time $O(|A_I+B_J| \log^2 u)$. Thus, we can compute $R := \bigcup_{(I,J)\in\mathcal{C}} A_I+B_J$ in time $O(c \log^2 u)$. By the covering property, we can simply return $R \cap [u] = (A + B) \cap [u]$. □

We refer to an algorithm making use of Observation 2.8 as a *covering algorithm*. We call it a *unique-rectangle-covering algorithm* if the used covering is a unique rectangle covering. This is a natural class of algorithms, as we also explain in Section 2.3.2. All algorithms presented in this paper are unique-rectangle-covering algorithms.

2.2 Formal Statement of Results

2.2.1 Main Results

As the technical core of our paper, we present an efficient construction of low-cost coverings.

Theorem 2.9 (Covering Construction, Section 7)**.** *Given $A, B \subseteq [u]$, in expected time $\widetilde{O}(\mathrm{out}^{4/3})$ we can compute a unique rectangle covering of $(A, B, [u])$ of cost $\widetilde{O}(\mathrm{out}^{4/3})$, where* $\mathrm{out} = |(A+B)\cap[u]|$*.*

By Observation 2.8, this yields an $\widetilde{O}(\mathrm{out}^{4/3})$ Las Vegas algorithm for prefix-restricted sumset computation. Via simple reductions, we obtain similar algorithms for our convolution problems.

Corollary 2.10 (Top-k-Convolution and Related Problems, Section 3)**.** *Top-k-convolution on non-negative vectors can be solved in expected time $\widetilde{O}(k^{4/3})$. Prefix-restricted sumset computation and prefix-restricted convolution on non-negative vectors can be solved in expected time $\widetilde{O}(\text{out}^{4/3})$.*

By carefully adapting a recent pseudopolynomial $\widetilde{O}(n+t)$-time SubsetSum algorithm [14] to use prefix-restricted sumset computation as a subroutine, we obtain our main result for SubsetSum. This can be seen as a reduction from SubsetSum to prefix-restricted sumset computation.

Theorem 2.11 (SubsetSum, Section 9)**.** *Given $X \subseteq \mathbb{N}$ and $t \in \mathbb{N}$, we can compute the set $\mathcal{S}(X,t)$ in time $\widetilde{O}(|\mathcal{S}(X,t)|^{4/3})$ with high probability.*

Since all of these results depend on our technical core (Theorem 2.9), we also study limitations of rectangle-covering algorithms. The following result shows that our approach of using rectangle coverings to solve top-k-convolution and SubsetSum cannot achieve near-linear output-sensitive algorithms.

Theorem 2.12 (Lower Bound on Rectangle Coverings, Section 8)**.** *There exists an infinite sequence of tuples $(A,B,[u])$, with $u \in \mathbb{N}$ and $A,B \subseteq [u]$, such that any rectangle covering of $(A,B,[u])$ has cost $\Omega(\text{out}^{1.047})$, where $\text{out} = |(A+B) \cap [u]|$.*

We remark that this result crucially uses *rectangle* coverings; we do not rule out the existence of non-rectangle coverings of near-linear cost.

In the remainder of Section 2.2 we present additional related results.

2.2.2 Additional Results: Relaxing the Upper Bound

In our results so far we have relaxed the ultimate goal of algorithms running in time $\widetilde{O}(\text{out})$ to time $\widetilde{O}(\text{out}^{4/3})$. We can alternatively relax the goal by measuring the running time in terms of a slightly larger upper bound. Specifically, for prefix-restricted sumset computation so far we measured time in terms of the output-size $\text{out} = |(A+B) \cap [u]|$. Now we will measure the running time in terms of $|(A+B) \cap [(1+\zeta)u]|$ for some $\zeta > 0$, while the task still is to compute the set $(A+B) \cap [u]$. Since one could expect that for "realistic" instances (A,B,u) there are not many sums in the boundary region $(A+B) \cap [u,(1+\zeta)u]$, algorithms that perform well with respect to this new measure might perform well in practice. We show the following.

Theorem 2.13 (Relaxed Upper Bound, Section 6)**.** *Prefix-restricted sumset computation can be solved in expected time $\widetilde{O}(\zeta^{-1} \cdot |(A+B) \cap [(1+\zeta)u]| + \zeta^{-2})$. Prefix-restricted convolution on non-negative vectors can be solved in expected time $\widetilde{O}(\zeta^{-1} \cdot \|(f \star g)_{[(1+\zeta)u]}\|_0 + \zeta^{-2})$. SubsetSum can be solved in time $\widetilde{O}(\zeta^{-1} \cdot |\mathcal{S}(X,(1+\zeta)t)| + \zeta^{-2})$ with high probability.*

Note that $|\mathcal{S}(X,(1+\zeta)t)| \leq (1+\zeta)t$, and thus plugging in any constant $\zeta > 0$ yields an algorithm running in time $\widetilde{O}(t)$, which is as good as [14, 27].

2.2.3 Additional Results: Interval-Restricted Sumset and Convolution

So far we studied the problem of computing the first u output values of sumset computation and convolution. More generally, we could ask to compute all output values at given positions $P \subseteq \mathbb{N}$. This is a well-motivated generalization, as it corresponds to computing specific coefficients of the product of two polynomials, which comes up in counting problems.

However, here we show that even when the desired positions P form an *interval* then near-linear output-sensitive algorithms are unlikely to exist. Specifically, we study the following problems.

Definition 2.14 (Interval-Restricted Sumset Computation)**.** *Given positive integers $\ell \le u$ and $A, B \subseteq [u]$, compute $(A+B) \cap [\ell, u]$. We denote the output-size by* $\mathrm{out} = |(A+B) \cap [\ell, u]|$*. We write $n = |A|$ and $m = |B|$.*

Definition 2.15 (Interval-Restricted Convolution)**.** *Given positive integers $\ell \le u$ and vectors $f, g \in \mathbb{R}^u$, compute $(f \star g)_{[\ell,u]}$, i.e., compute (a sparse representation of) the entries of $(f \star g)_i$ for $\ell \le i \le u$. We denote the output-size by* $\mathrm{out} = \|(f \star g)_{[\ell,u]}\|_0$*. We write $n = \|f\|_0$ and $m = \|g\|_0$.*

We design algorithms for these problems with the following running time. This uses a natural generalization of coverings, where we simply replace the set $[u]$ by $[\ell, u]$.

Theorem 2.16 (Algorithm for Interval-Restricted)**.** *Interval-restricted sumset computation and interval-restricted non-negative convolution can be solved in expected time $\widetilde{O}(n + m + \sqrt{nm\,\mathrm{out}})$.*

We present conditional lower bounds based on two classical problems: Boolean matrix multiplication and sliding window Hamming distance. In Boolean matrix multiplication, we are asked to multiply two Boolean $n \times n$-matrices. By a reduction to multiplying real-valued matrices, Boolean matrix multiplication can be solved in time $O(n^\omega)$, where $\omega \le 2.373$ is the exponent of fast matrix multiplication [47, 22]. Limitations of fast matrix multiplication have been recently studied, and it was shown that known techniques cannot facilitate $\omega < 2 + \frac{1}{6}$ [5, 4, 3]. In particular, $\omega > 2$ is a barrier that known techniques cannot overcome, also for Boolean matrix multiplication.

In the sliding window Hamming distance problem, we are given a text of length $2n$ and a pattern of length n, both over a (possibly large) alphabet Σ, and want to find the Hamming distance between the pattern and every length-n substring of the text. This problem admits a $\widetilde{O}(n^{3/2})$-time algorithm, and it is a major open question in string algorithms whether a faster algorithm exists [2, 23].

Theorem 2.17 (Hardness for Interval-Restricted)**.** *Let $\delta > 0$. If interval-restricted sumset computation is in time $\widetilde{O}((n+m+\mathrm{out})^{\omega/2-\delta})$, then Boolean matrix multiplication is in time $\widetilde{O}(n^{\omega-2\delta})$.*

If interval-restricted convolution on non-negative vectors is in time $\widetilde{O}(n+m+(nm\,\mathrm{out})^{1/2-\delta})$, then sliding window Hamming distance is in time $\widetilde{O}(n^{3/2-3\delta})$.

2.3 Overview of Techniques

2.3.1 Reduction from Subset Sum to Prefix-Restricted Sumset Computation

To reduce SubsetSum to prefix-restricted sumset computation, our starting point is the pseudopolynomial $\widetilde{O}(t)$-time algorithm of [14], which uses a combination of color coding and FFT. We observe that in this algorithm all uses of FFT in fact compute prefix-restricted sumsets, so we can replace FFT by an algorithm for prefix-restricted sumset computation. This directly yields a reduction from SubsetSum to prefix-restricted sumset computation, but it does not directly give the desired guarantees. Our goal is to make this reduction "output-sensitivity preserving", that is, if we can compute prefix-restricted sumsets in near-linear output-sensitive time then the reduction yields an algorithm for SubsetSum that runs in near-linear output-sensitive time.

To this end, we need to take a closer look at the algorithm in [14]. Given (X, t), this algorithm splits X into the set $X^{(L)}$ of numbers that are larger than $t/\operatorname{polylog}(n)$ and the remaining small numbers $X^{(S)}$. Any subset summing to at most t uses at most $\operatorname{polylog}(n)$ large numbers, which enables a color-coding-based algorithm in [14]. We observe that, without any problems, replacing the usage of FFT in that part of the algorithm by near-linear output-sensitive prefix-restricted sumset computation yields a near-linear output-sensitive algorithm for handling the large numbers $X^{(L)}$. The small numbers $X^{(S)}$ are then further split at random into two subsets $X^{(1)}, X^{(2)}$. By concentration inequalities, this splits the subset Y into two subsets $Y^{(i)} = Y \cap X^{(i)}$ satisfying with high

probability $\Sigma(Y^{(i)}) \leq (1+\epsilon)t/2$. We can therefore find $\Sigma(Y^{(1)})$ and $\Sigma(Y^{(2)})$ by recursive calls on $(X^{(1)}, (1+\epsilon)t/2)$ and $(X^{(2)}, (1+\epsilon)t/2)$. Since we recurse on two subproblems and the target bound t is roughly split in half, one can argue that the total running time is $\widetilde{O}(t)$ [14].

In this paper, we instead measure the running time in terms of $|\mathcal{S}(X,t)|$. Hence, it does not suffice that the target bound t is roughly split in half; we also need that the measure $|\mathcal{S}(X,t)|$ is roughly split in half. Indeed, if we can show this, then we again obtain two subproblems and the measure $|\mathcal{S}(X,t)|$ is roughly split in half, so we can argue that the running time is $\widetilde{O}(|\mathcal{S}(X,t)|)$. This requires novel insights into the set of all subset sums, to obtain inequalities relating the number of subset sums of X to that of its subsets. Specifically, we show that $X^{(1)}$ and $X^{(2)}$ satisfy

$$|\mathcal{S}(X^{(1)}, (1+\epsilon)t/2)| + |\mathcal{S}(X^{(2)}, (1+\epsilon)t/2)| \leq (1+O(\epsilon))|\mathcal{S}(X,t)| + O(1).$$

This inequality only holds if every element of $X^{(1)}, X^{(2)}$ is sufficiently smaller than the target t. Indeed, in the proof of this inequality we use that all numbers in $X^{(1)}, X^{(2)}$ are small, which justifies why we removed the large numbers $X^{(L)}$.

2.3.2 Restricted Sumset Computation

We now give an overview of our techniques for restricted sumset computation, and discuss the relation to classical results from Additive Combinatorics.

Consider the prefix-restricted sumset computation of $(A+B) \cap [u]$. Note that we can assume $A, B \subseteq [u]$, since larger numbers cannot form sums of at most u. We also always implicitly assume that for every number $a \in A$ there is a number $b \in B$ with $a+b \leq u$, since otherwise we can remove a from A without changing $(A+B) \cap [u]$. In other words, we assume $\max(A) + \min(B) \leq u$, and symmetrically $\min(A) + \max(B) \leq u$, which can be ensured after a $O(|A|+|B|)$-time preprocessing.

Standard sumset computation does not suffice. Standard sparse convolution (Theorem 2.2) allows us to compute $A+B$ in near-linear output-sensitive time $\widetilde{O}(|A+B|)$. In particular, since $A+B \subseteq [2u]$, we can compute $(A+B) \cap [2u]$ in near-linear output-sensitive time. Although this might seem close at first glance, careful inspection reveals that it can be much larger than the goal $|(A+B) \cap [u]|$. Indeed, we can construct sets $A, B \subseteq [u]$ with $|A| = |B| = n$ satisfying $|(A+B) \cap [2u]| = \Omega(n^2)$ and $|(A+B) \cap [u]| = O(n)$. Hence, in general we cannot afford to compute $(A+B) \cap [2u]$ when our goal is to compute $(A+B) \cap [u]$ in near-linear output-sensitive time.

For this construction, assume that u is a sufficiently large even integer and let

$$X := [n] = \{0, 1, 2, \ldots, n\}, \qquad Y := n[n] = \{0, n, 2n, \ldots, n^2\},$$
$$A := \{0\} \cup (\{\tfrac{u}{2}\} + X), \qquad B := \{0\} \cup (\{\tfrac{u}{2}\} + Y).$$

One can check that $\max(A) + \min(B), \min(A) + \max(B) \leq \frac{u}{2} + n^2 \leq u$. Since for any $x \in X$, $y \in Y$ the sum $(\frac{u}{2} + x) + (\frac{u}{2} + y)$ is larger than u, we obtain $|(A+B) \cap [u]| \leq |\{0\}| + |X| + |Y| = 2n+1$. Moreover, one can check that $X + Y = [n^2 + n]$, and thus $|(A+B) \cap [2u]| = |A+B| \geq n^2$.

A natural class of algorithms. We investigate a natural and safe algorithmic approach: To compute $(A+B) \cap [u]$, we make sure that every sum $A_i + B_j \leq u$ is computed, where $1 \leq i \leq n$, $1 \leq j \leq m$. Since we can use as a subroutine that standard sumset computation is in near-linear output-sensitive time, it is natural to compute sumsets $A_I + B_J$ for some $I \subseteq \{1, \ldots, n\}$ and $J \subseteq \{1, \ldots, m\}$. Combining these two arguments, we arrive at our notion of *covering*. In particular, we want to compute a covering $\mathcal{C}$ of $(A, B, [u])$, and then use Observation 2.8 to compute

$\left(\bigcup_{(I,J)\in\mathcal{C}} A_I + B_J\right) \cap [u] = (A+B) \cap [u]$ in time proportional to the cost of $\mathcal{C}$. Hence, covering algorithms are a natural class of algorithms for prefix-restricted sumset computation.

We are particularly interested in *unique* coverings, because they allow us to compute prefix-restricted convolutions: A pair $A_i + B_j$ being covered exactly once corresponds to adding the product $f_i \cdot g_j$ exactly once to the output value $(f \star g)_{i+j}$.

Moreover, we are interested in *rectangle* coverings, i.e., the case of I, J being intervals, because (i) they have a more geometric flavour and are thus easier to argue about, (ii) all constructions of non-rectangular coverings that we came up with could be adapted to become rectangular, so we are not aware of any better non-rectangular coverings, and (iii) for rectangular coverings we can show (unconditional) lower bounds.

Designing covering algorithms. In what follows, for ease of exposition we shall assume that $|A| = |B| = n$. Our first algorithmic step is to reduce to a promise version of the problem, where we know $\text{out} = |(A+B) \cap [u]|$ up to a constant factor. (In fact, we could even assume to know a superset of $(A+B) \cap [u]$ whose size is at most a constant factor larger, but we do not know how to exploit such a set in this context.)

The easiest possible covering algorithm partitions $\{1, \dots, n\}$ into k consecutive intervals $I_1, \dots, I_k$ for A, and similarly into $J_1, \dots, J_k$ for B. We may ignore pairs (x, y) with $\min(A_{I_x}) + \min(B_{J_y}) > u$, since they do not contain any sum $A_i + B_j \le u$. For all remaining (x, y), we construct the pairs (I_x, J_y) to form a covering $\mathcal{C}$. In the following we discuss two ways of choosing $I_1, \dots, I_k$ and $J_1, \dots, J_k$.

Choosing the interval partitioning such that it splits the universe $[u]$ into parts of length u/k ensures that any sum $s \in A_I + B_J$ for $(I, J) \in \mathcal{C}$ satisfies $s \le (1 + \frac{2}{k})u$. Moreover, we show that any $s \in \mathbb{N}$ appears in at most k sumsets $A_I + B_J$, $(I, J) \in \mathcal{C}$ (this holds because every "diagonal" of subproblems contains distinct sums). In total, we can bound the cost of $\mathcal{C}$ by $k \cdot |(A+B) \cap [(1+\frac{2}{k})u]|$, which for $\zeta = 2/k$ yields one result of Theorem 2.13, see Section 6.

Alternatively, we can choose the interval partitioning such that it splits A and B into parts of size n/k. There are $O(k)$ pairs $(I, J) \in \mathcal{C}$ along the "boundary", where $\min(A_I + B_J) \le u < \max(A_I + B_J)$. For each such pair we bound the cost trivially by $(\frac{n}{k})^2$. The remaining pairs only cover sums in $(A + B) \cap [u]$, and each such sum is covered at most k times, as in the previous paragraph. In total, this yields cost at most $k \cdot \text{out} + \frac{n^2}{k}$, which is $O(n \cdot \text{out}^{1/2})$ by optimizing over k. Since $n \le \text{out}$, in particular the cost is at most $O(\text{out}^{3/2})$. Generalizing this idea to the interval-restricted case yields Theorem 2.16, see Section 5. For the interval-restricted case, this cost bound turns out to be (conditionally) optimal, see Theorem 2.17.

So far, we used simple charging arguments to obtain a covering of cost $O(\text{out}^{3/2})$. We show that for prefix-restricted sumset computation we can bypass the hardness results for the interval-restricted case and reduce the exponent further to 4/3. This requires insights from Additive Combinatorics. Specifically, to obtain an $\widetilde{O}(\text{out}^{4/3})$ algorithm, we make use of Ruzsa's triangle inequality. This inequality from Additive Combinatorics is a charging argument that implies

$$|A_I + B_J| \le \frac{|A_I + B_{J'}||A_{I'} + B_{J'}||A_{J'} + B_J|}{|I'||J'|},$$

for any sets $I, I', J, J' \subseteq \{1, \dots, n\}$. In particular, if we choose I' and J' such that the three sumsets on the right hand side are all subsets of $[u]$, then the size of every sumset on the right hand side is bounded from above by $\text{out} = |(A+B) \cap [u]|$. With some care, this inequality can be used to show

$$|A_I + B_J| \le \frac{\text{out}^3}{\min(I)\min(J)}.$$

This allows us to bound the number of "bad" sums in $A_I + B_J$, namely the sums in $(A_I + B_J) \setminus [u]$, in terms of the output-size and the position of the rectangle (I, J). However, it turns out that this bound is not good enough, essentially because it yields a worst-case bound that holds for all rectangles (I, J). We thus replace it with an improved bound that holds for "most" rectangles (I, J).

Combining this bound with several tricks from our previous covering constructions allows us to construct a covering of cost $\widetilde{O}(\text{out}^{4/3})$, see Section 7.

Connection to the Balog-Szemerédi-Gowers Theorem. Probably the most important result on restricted sumsets is the Balog-Szemerédi-Gowers (BSG) theorem [45, Theorem 2.13]. Given sets $A, B \subseteq \mathbb{Z}$ and a set of pairs $G \subseteq A \times B$, the BSG theorem is used to pass from information about the *restricted* sumset $A +_G B = \{a + b \mid (a, b) \in G\}$ to information about an *unrestricted* sumset $A_I + B_J$, for some $I, J \subseteq \{1, \ldots, n\}$ (where we continue to assume that $|A| = |B| = n$). Specifically, the BSG theorem states that if $|G| = \delta n^2$ and $|A +_G B| = cn$, then there exist sets $I, J \subseteq \{1, \ldots, n\}$ of size $\Omega(\delta^2 n)$ with $|A_I + B_J| \leq O((c/\delta)^5 n)$.

In a breakthrough paper, Chan and Lewenstein [17] algorithmically exploited the BSG theorem in order to solve several problems with additive structure. On a high level, their approach uses the BSG theorem repeatedly, constructing a sequence $(I^{(1)}, J^{(1)}), \ldots, (I^{(k)}, J^{(k)})$ of subsets $I^{(i)}, J^{(i)} \subseteq \{1, \ldots, n\}$ and a remainder $R \subseteq A \times B$ such that

$$A +_G B = \left(\bigcup_{i=1}^{k} (A_{I^{(i)}} + B_{J^{(i)}}) \right) \bigcup \{a + b : (a, b) \in R\} \tag{1}$$

$$\tag{2}$$

For $|A +_G B| = cn$, this decomposition satisfies that all sumsets are small, i.e., $|A_{I^{(i)}} + B_{J^{(i)}}| = O((kc)^5 n)$, and the remainder is not too large, i.e., $|R| = O(n^2/k)$. Using this decomposition, one can compute (a superset of) $A +_G B$ by computing the unrestricted sumsets $A_{I^{(i)}} + B_{J^{(i)}}$ and then iterating over all elements of R. Using near-linear output-sensitive sumset computation, this takes total time $\widetilde{O}(k^6 c^5 n + n^2/k)$. For several applications considered in [17], this approach yields non-trivial improvements. We note, however, that constructing the sequence and the remainder in their setting requires quadratic time in n.

A careful reader could notice that our notion of a *covering* is very similar to the decomposition computed by Chan and Lewenstein [17]. In our situation, we are interested in the set $G = \{(a, b) \in A \times B \mid a + b \leq u\}$, since our goal is to compute $A +_G B = (A + B) \cap [u]$. A covering $\mathcal{C}$ satisfies $A +_G B \subseteq \bigcup_{(I,J) \in \mathcal{C}} A_I + B_J =: T$, from which we can compute $A +_G B$ as $T \cap [u]$. Except for the remainder set R, this is analogous to Chan and Lewenstein's decomposition. Thus, both notions produce a certain form of covering of $A +_G B$.[2] Unfortunately, in our situation using their decomposition as a black box is worse than the simple covering of cost $O(\text{out}^{3/2})$ for two reasons. The first is that the time needed to construct their decomposition is $\Theta(|A||B|)$, which could be up to out^2. The second reason is that the bound they get on the cost of the covering they produce is strictly worse than $\text{out}^{3/2}$. Thus, even if their decomposition was given to us for free, it would be worse than the easy approach which uses elementary charging arguments.

Our approach in this paper can be seen as using the arguments of the proof of the BSG theorem in an ad-hoc manner for our specific set G. Indeed, at the heart of the BSG theorem lies a charging argument that is analogous to Ruzsa's triangle inequality (see [17, Section 7.3]), and careful inspection of the algorithmic version of the BSG theorem (see [17, Sections 2 and 7]) shows that it can

[2] We remark that Chan and Lewenstein [17] construct their decomposition for a somewhat different reason, since they are not (immediately) concerned with output-sensitivity.

be modified to produce a rectangle covering for our set G with no loss in parameters. We present a highly optimized variant of this ad-hoc construction in this paper.

In summary, our algorithms can be viewed as ad-hoc BSG-type theorems for a special choice of the restriction set G, obtaining bounds that seem unreachable in the more general setting.

Connection to Freiman-type Theorems. Another celebrated result among combinatorialists is Freiman's theorem [45, Theorem 5.33]: If $A \subseteq \mathbb{N}$ satisfies $|A + A| \leq c|A|$, then A is contained in an $f(c)$-dimensional arithmetic progression of length $g(c) \cdot |A|$. Such a result could potentially be useful in our situation, since for a rectangle with $A_I + B_J \subseteq [u]$ we have $|A_I + B_J| \leq \text{out}$, and thus a small output-size could imply that A_I and B_J are essentially arithmetic progressions, which is easy to exploit. However, the state-of-the-art bounds on $f(\cdot), g(\cdot)$ are exponential and thus not useful in applications. This is why much of Additive Combinatorics has focused on bypassing the need for Freiman's theorem, to prove robust theorems with polynomial parameter dependence.

Recently, [44] announced such a robust extension of another theorem by Freiman, called the $3k-4$ theorem, which roughly states that if $|A +_G B|$ is "small" and $|G|$ is "large" ($|G| \geq (1-\epsilon)|A||B|$), then one can extract highly non-trivial information about A and B. However, the "small" and "large" conditions are too restrictive for our algorithmic applications, and it is unclear to us at the moment to what extent this result can be exploited algorithmically. We expect that further interaction with the field of Additive Combinatorics yields more insights that lead to new algorithmic machinery.

Selection from $X + Y$. A problem that seems related at first glance is selecting the k-th element from $X + Y$ and row-sorted matrices, see [28] and references therein. The crucial difference is that in that line of research $X + Y$ is treated as a *multiset*, instead of a set. For example, they consider $\{1, 2\} + \{1, 2, 3\} = \{2, 3, 3, 4, 4, 5\}$, while we define $\{1, 2\} + \{1, 2, 3\} = \{2, 3, 4, 5\}$. Note that in our situation the presence of multiplicities is exactly what we want to avoid. It is vital for our algorithm to process every element in the sumset the minimum number of times possible, and not proportional to the number of times it appears, since we measure running time with respect to the size of the sumset (without multiplicities). For this reason, the ideas of [28] do not seem applicable for top-k and prefix-restricted convolution.

Lower Bound on Rectangle Coverings. Let us also briefly discuss the construction of instances $(A, B, [u])$ on which any rectangle covering has superlinear cost (see Theorem 2.12). We build such instances in two steps.

First, we construct "many" sets $X^{(1)}, \ldots, X^{(g)}$ and $Y^{(1)}, \ldots, Y^{(g)}$ such that $|X^{(\ell)} + Y^{(\ell)}|$ is "large", while $|X^{(\ell)} + Y^{(\ell')}|$ is "small" for any $\ell \neq \ell'$. To this end, we pick appropriate parameters m, t, and greedily choose an error-correcting code over $\{0, 1\}^t$ where every codeword has Hamming weight exactly $t/2$. We define $X^{(\ell)}$ to be the set of all t-digit numbers in the m-ary number system with the constraint that their r-th digit is 0 if the ℓ-th codeword has a 1 at position r. Similarly, we define $Y^{(\ell)}$ to be the set of all t-digit numbers in the m-ary system with the constraint that their r-th digit is 0 if the ℓ-th codeword has a 0 at position r.

In the second step, we contruct the set A as a union over several shifted copies of each $X^{(\ell)}$. The set B is constructed similarly from the sets $Y^{(\ell)}$. In this construction, we ensure that $(A + B) \cap [u]$ contains no sumset $X^{(\ell)} + Y^{(\ell)}$, so the output-size is "small". We also ensure that any rectangle (I, J), that covers many parts of $(A + B) \cap [u]$ at once, needs to cover a sumset $X^{(\ell)} + Y^{(\ell)}$, which results in a "large" sumset $|A_I + B_J|$. Therefore, any covering either contains a rectangle of "large" cost or consists of "many" rectangles; in both cases we obtain a superlinear lower bound on the cost in terms of the output-size $|(A + B) \cap [u]|$.

2.4 Organization

In Section 3 we reduce top-k-convolution and related problems to finding a covering, proving Corollary 2.10. In Section 4 we show how to learn the output-size up to a constant factor for restricted convolution problems. In Section 5 we study interval-restricted convolution, proving Theorems 2.16 and 2.17. In Section 6, we relax the upper bound, proving part of Theorem 2.13. In Section 7 we describe and analyze our $\widetilde{O}(\text{out}^{4/3})$-cost covering for prefix-restricted sumset computation (Theorem 2.9) and we prove a lower bound for rectangle coverings (Theorem 2.12). Section 9 then contains the reduction from SubsetSum to prefix-restricted sumset computation, proving Theorem 2.11 and finishing the proof of Theorem 2.13.

3 Top-k-Convolution

In this section, we present some easy reductions among Top-k-Convolution and related problems, proving Corollary 2.10. We start by proving that Top-k-Convolution is equivalent to Prefix-Restricted convolution.

Lemma 3.1. *If Top-k-Convolution on non-negative vectors can be solved in time $\widetilde{O}(k^\alpha)$, then Prefix-Restricted convolution on non-negative vectors can be solved in time $\widetilde{O}(\text{out}^\alpha)$, and vice versa.*

Proof. To solve Prefix-Restricted sparse convolution, we run Top-k-Convolution for $k = 2^0, 2^1, 2^2, \dots$ until we reach a value of k so that the k-th non-zero entry lies above u. Then we have found all elements of the Prefix-Restricted convolution, and we only computed at most twice as many non-zero entries as necessary.

To solve Top-k-Convolution on vectors $f, g \in \mathbb{R}^d_{\geq 0}$, we perform binary search over $\{0, 1, \dots, 2d\}$, to find the smallest u such that Prefix-Restricted convolution on $[u]$ returns a k-sparse vector. To obtain the desired running time we add a small twist: if the execution time of Prefix-Restricted sumset computation on $[u]$ is more than $k^\alpha \cdot \operatorname{polylog} d$, i.e., more than what it would be for output size k, then we abort and look for a smaller u.

Both reductions only add a log-factor to the running time. □

We next show that coverings not only allow us to compute Prefix-Restricted sumsets, but they even enable the computation of Prefix-Restricted convolutions, if the covering is *unique*.

Lemma 3.2. *Suppose that, given (A, B, u), we can compute a* unique covering *of $(A, B, [u])$ with cost $\widetilde{O}(\text{out}^\alpha)$ in expected time $\widetilde{O}(\text{out}^\alpha)$, where* $\text{out} := |(A + B) \cap [u]|$. *Then Prefix-Restricted convolution on non-negative vectors can be solved in expected time $\widetilde{O}(\text{out}^\alpha)$.*

Proof. For a vector f we denote by f_S the same vector where every entry outside of S is zeroed out. Given non-negative vectors f, g, let $A := \operatorname{supp}(f)$ and $B := \operatorname{supp}(g)$. The output-size of Prefix-Restricted convolution on f, g is the number of non-zero entries of $(f \star g)_{[u]}$. This is the same as the output-size of Prefix-Restricted sumset computation on A, B, namely $|(A + B) \cap [u]|$. Thus, the two output-sizes coincide and we denote both by "out". In particular, we can afford to compute a unique covering $\mathcal{C}$ of $(A, B, [u])$.

Using output-sensitive convolution of non-negative vectors (Theorem 2.1, [19]), we can compute $f_S \star g_T$ in time $\widetilde{O}(\|f_S \star g_T\|_0)$. Therefore, we can compute

$$h := \sum_{(I,J)\in\mathcal{C}} f_{A_I} \star g_{B_J},$$

in time proportional to the cost of the covering, up to log factors. By the properties of a unique covering, any non-zero product $f_j \cdot g_{i-j}$ appears exactly once in the definition of h, for any $0 \le j \le i \le u$. We thus have $h_{[u]} = (f \star g)_{[u]}$. □

Corollary 2.10 now follows.

Proof of Corollary 2.10. To solve Prefix-Restricted sumset computation in time $\widetilde{O}(\text{out}^{4/3})$, we combine our covering construction (Theorem 2.9) with Observation 2.8. To solve Prefix-Restricted convolution on non-negative vectors in time $\widetilde{O}(\text{out}^{4/3})$, we combine our covering construction (Theorem 2.9) with Lemma 3.2. To solve Top-k-Convolution on non-negative vectors in time $\widetilde{O}(k^{4/3})$, we combine the result for Prefix-Restricted convolution with the equivalence shown in Lemma 3.1. □

4 Restricted Sumset Computation: Learning the Output-Size

In this section, we show that we can assume to know the output-size up to a constant factor. We will later use this for Prefix-Restricted and for Interval-Restricted sumset computation. Here we will discuss the more general setting of Interval-Restricted sumset computation; the same construction works for the Prefix-Restricted case.

Suppose that we are given sets A, B and integers ℓ, u and we want to compute their Interval-Restricted sumset $(A+B) \cap [\ell, u]$. We show that we can assume to know $|(A+B) \cap [\ell, u]|$ up to a constant factor. In fact, we reduce Interval-Restricted sumset computation to a seemingly much easier variant, where additionally we are given a set T of size $\Theta(|(A+B) \cap [\ell, u]|)$ which contains $(A+B) \cap [\ell, u]$.

Definition 4.1 (Interval-Restricted Sumset Computation with a Promise (IR-SMP))**.** *Given sets $A, B \subseteq \mathbb{Z}$, numbers $\ell \le u$, and a set T of size $|T| \le 6|(A+B) \cap [\ell, u]| + 9$, such that*

$$(A+B) \cap [\ell, u] \subseteq T,$$

compute the set

$$(A+B) \cap [\ell, u].$$

We present a reduction from the problem variant without promise to the variant with promise.

Lemma 4.2. *We can reduce a given instance (A, B, ℓ, u) of Interval-Restricted sumset computation to $O(\log(u-\ell))$ instances of IR-SMP, each of the form (A', B', ℓ', u', T') with $|A'| \le |A|$, $|B'| \le |B|$, $\ell' \le \ell$, $u' \le u$, $u' - \ell' \le u - \ell$, and $|T'| = O(|(A+B) \cap [\ell, u]|)$. The reduction runs in linear time in its output-size.*

Proof. In what follows, for integers x, y we define $x \div y = \lfloor \frac{x}{y} \rfloor$. For a set Z we write $Z \div x = \{z \div x \mid z \in Z\}$, and we write $xZ = \{x \cdot z \mid z \in Z\}$.

Let (A, B, ℓ, u) be a given instance of Interval-Restricted sumset computation. We compute better and better approximations of the desired output $(A+B) \cap [\ell, u]$ by computing the sets

$$S^{(i)} := \big((A \div 2^i) + (B \div 2^i)\big) \cap \big[\ell \div 2^i, u \div 2^i\big],$$

for $i = r, \ldots, 0$, where $r := \lceil \log(u - \ell) \rceil$. See Algorithm 1 for pseudocode. Note that $S^{(0)}$ is the desired output $(A+B) \cap [\ell, u]$. We compute each set $S^{(i)}$ by calling an IR-SMP oracle on a suitable promise $T^{(i)}$, determined as follows.

For $i = r$, the interval $[\ell \div 2^i, u \div 2^i]$ has length at most 2. Thus, the set $T^{(r)} := \{\ell \div 2^i, \ldots, u \div 2^i\}$ is a valid promise from which our assumed IR-SMP oracle can compute the set $S^{(r)}$.

For $i < r$, we claim that a valid promise for the instance $(A \div 2^i, B \div 2^i, \ell \div 2^i, u \div 2^i)$ is given by

$$T^{(i)} := \big(2S^{(i+1)} + \{0,1,2\}\big) \cup \{(\ell \div 2^i), (\ell \div 2^i) + 1, (u \div 2^i)\}.$$

If this claim holds, then from $S^{(r)}$ we can compute $S^{(r-1)}, S^{(r-2)}, \ldots, S^{(0)}$, finishing our reduction.

Let us also claim that any set $S^{(i)}$ has size $|S^{(i)}| = O(|(A+B) \cap [\ell, u]|)$. Then in particular the set $T^{(i)}$ has size $|T^{(i)}| = O(|S^{(i)}|) = O(|(A+B) \cap [\ell, u]|)$. Hence, all constructed instances satisfy the claimed properties, and we reduced the given instance (A, B, ℓ, u) to $O(\log(u - \ell))$ calls to an IS-SMP oracle. This finishes the proof, except that it remains to prove the two claims.

Claim 4.3. *The set $S^{(i)}$ has size $|S^{(i)}| \le 2|(A+B) \cap [\ell, u]| + 2$.*

Proof. It is easy to check that for any integers a, b we have

$$2^i\big((a \div 2^i) + (b \div 2^i)\big) \;\le\; a + b \;\le\; 2^i\big((a \div 2^i) + (b \div 2^i)\big) + 2^i, \tag{3}$$

or, equivalently,

$$\frac{a+b}{2^i} - 1 \;\le\; (a \div 2^i) + (b \div 2^i) \;\le\; \frac{a+b}{2^i}. \tag{4}$$

Any number in $s \in S^{(i)}$ can be expressed as $s = (a \div 2^i) + (b \div 2^i)$ with

$$(\ell \div 2^i) \le s \le (u \div 2^i).$$

Suppose that we have $s \in [(\ell \div 2^i) + 1, (u \div 2^i) - 1]$. Then by inequalities (3), it follows that

$$\ell \;\le\; 2^i((\ell \div 2^i) + 1) \;\le\; 2^i s \;\le\; a + b \;\le\; 2^i s + 2^i \;\le\; 2^i(u \div 2^i) \;\le\; u,$$

Thus, we obtain that the number $s' := a + b$ lies in $(A+B) \cap [\ell, u]$. We say that s *charges* s'. From inequalities (4), we infer that any number $s' \in (A+B) \cap [\ell, u]$ can only be charged by numbers in $[s'/2^i - 1, s'/2^i]$, so it is charged by at most two numbers $s \in S^{(i)}$. This charging scheme proves

$$\big|S^{(i)} \cap [(\ell \div 2^i) + 1, (u \div 2^i) - 1]\big| \;\le\; 2 \cdot \big|(A+B) \cap [\ell, u]\big|.$$

We add 2 to account for the numbers $\ell \div 2^i$ and $u \div 2^i$. This yields the claim. □

Algorithm 1 Reduction of Interval-Restricted Sumset Computation to its Promise Version

procedure ComputeIntervalRestrictedSumset(A, B, ℓ, u)
▷ $A, B \subseteq [u]$, $\ell \le u$
▷ We assume oracle access to Interval-Restricted sumset computation with a promise (IR-SMP)
  $r \leftarrow \lceil \log(u - \ell) \rceil$
  $T^{(r)} \leftarrow \{\ell \div 2^r, \ldots, u \div 2^r\}$
  $S^{(r)} \leftarrow$ IR-SMP$(A \div 2^r, B \div 2^r, \ell \div 2^r, u \div 2^r, T^{(r)})$
  ▷ Call to the promise problem
  for $i = r - 1$ down to 0 **do**
    $T^{(i)} \leftarrow \big(2S^{(i+1)} + \{0,1,2\}\big)$
    $T^{(i)} \leftarrow T^{(i)} \cup \{(\ell \div 2^i), (\ell \div 2^i) + 1, (u \div 2^i)\}$
    $S^{(i)} \leftarrow$ IR-SMP$(A \div 2^i, B \div 2^i, \ell \div 2^i, u \div 2^i, T^{(i)})$
    ▷ Call to the promise problem
  Return $S^{(0)}$

Claim 4.4. *The set $T^{(i)}$ is a valid promise for the instance $(A \div 2^i, B \div 2^i, \ell \div 2^i, u \div 2^i)$. That is, we have $T^{(i)} \supseteq S^{(i)}$ and $|T^{(i)}| \le 6|S^{(i)}| + 9$.*

Proof. Since $x \div 2^{i+1} = (x \div 2^i) \div 2$, it suffices to prove the claim for $i = 0$. The same proof then also works for larger i after replacing A by $A \div 2^i$, B by $B \div 2^i$, ℓ by $\ell \div 2^i$, and u by $u \div 2^i$.

We first show that $|T^{(0)}| \le 6|S^{(0)}| + 9$. Recall that in Claim 4.3 we proved that

$$|S^{(1)}| \;\le\; 2 \cdot |(A+B) \cap [\ell, u]| + 2 \;=\; 2|S^{(0)}| + 2.$$

We combine this with the inequality $|T^{(0)}| \le 3|S^{(1)}| + 3$ that we obtain from the construction of $T^{(0)}$. This yields the claimed inequality $|T^{(0)}| \le 6|S^{(0)}| + 9$.

Next we prove $T^{(0)} \supseteq S^{(0)}$. Consider any $a \in A$, $b \in B$ with $a + b \in \big[\ell + 2, u - 1\big]$. Using inequalities (4) we obtain

$$\begin{aligned} \ell \div 2 \;\le\; \frac{\ell}{2} \;\le\; \frac{a+b}{2} - 1 \;&\le\; (a \div 2) + (b \div 2) \\ &\le\; \frac{a+b}{2} \;\le\; \frac{u-1}{2} \;\le\; u \div 2. \end{aligned}$$

Therefore, the sum $(a \div 2) + (b \div 2)$ is in $S^{(1)}$. Now, since $a + b$ can be found among the three numbers

$$2\big((a \div 2) + (b \div 2)\big), \;\; 2\big((a \div 2) + (b \div 2)\big) + 1, \;\; 2\big((a \div 2) + (b \div 2)\big) + 2,$$

it follows that $a + b \in (2S^{(1)} + \{0, 1, 2\}) \subseteq T^{(0)}$. Recall that here we assumed $a + b \in [\ell + 2, u - 1]$. The boundary numbers $\ell, \ell + 1$, and u are handled by explicitly adding them to the set $T^{(0)}$ in its construction. We thus obtain $S^{(0)} \subseteq T^{(0)}$. □

These claims finish the proof of Lemma 4.2. □

We obtain the following easy corollary where we essentially ignore the superset T and only keep its size $|T|$, which is a constant-factor approximation of the output-size.

Lemma 4.5 (Interval-Restricted Sumset Computation with Approximate Output-size)**.** *Suppose that given (A, B, ℓ, u) and an additional input $\widetilde{\text{out}}$ satisfying $\text{out} \le \widetilde{\text{out}} \le 6\text{out} + 9$, we can compute a covering of $(A, B, [\ell, u])$ of cost $O(c)$ in time $O(T)$, where c and T are monotone functions of $|A|, |B|, u, \text{out}$. Then Interval-Restricted sumset computation can be solved in time $\widetilde{O}(c + T)$.*

An analogous statement holds for Prefix-Restricted sumset computation.

Proof. Given an instance (A, B, ℓ, u) of Interval-Restricted sumset computation, we run Lemma 4.2 to reduce to $O(\log(u - \ell))$ promise instances of the form (A', B', ℓ', u', T'). By the properties of T', for $\widetilde{\text{out}}' := |T'|$ we have $\text{out}' \le \widetilde{\text{out}}' \le 6\text{out}' + 9$, where $\text{out}' := |(A' + B') \cap [\ell', u']|$. Hence, we can use our assumed algorithm to compute a covering of $(A', B', [\ell', u'])$ of cost $O(c)$ in time $O(T)$. By Observation 2.8, we can thus solve the instance $(A', B', [\ell', u'])$ in time $\widetilde{O}(c + T)$. Over $O(\log(u - \ell))$ many constructed instances, we obtain the same time bound up to log-factors. □

5 Interval-Restricted Sumset Computation

In this subsection we prove Theorems 2.16 and 2.17.

5.1 An $\widetilde{O}(\sqrt{mn \cdot \text{out}})$-time Algorithm

In this subsection we prove Theorem 2.16.

Proof. Given A, B, ℓ, u, we will compute the Interval-Restricted sumset $(A+B)\cap[\ell,u]$ in time $\widetilde{O}(n+m+\sqrt{mn\cdot\text{out}})$. More precisely, we will compute a covering of $(A,B,[\ell,u])$ of cost $O(\sqrt{mn\cdot\text{out}})$ in time $O(n+m)$, assuming that we know an approximation $\widetilde{\text{out}}$ of the output-size satisfying $\text{out} \le \widetilde{\text{out}} \le O(\text{out})$ (we can assume this by Lemma 4.5).

Set $q := \lceil (nm/\widetilde{\text{out}})^{1/2} \rceil$. Note that since $n, m \le \text{out} \le n \cdot m$ we have $1 \le q = O(\min\{n,m\})$. We assume that n and m are divisible by q, which we can ensure by duplicating at most q elements in A and B (with the slight abuse of transitioning to multi-sets).

We split A and B into q subsets by defining for any $1 \le i, j \le q$:

$$\begin{aligned} I_i &:= \{(i-1)\cdot n/q + 1, \dots, i \cdot n/q\}, \\ J_j &:= \{(j-1)\cdot m/q + 1, \dots, j \cdot m/q\}, \\ A^{(i)} &:= A_{I_i}, \\ B^{(j)} &:= B_{J_j}. \end{aligned}$$

We let $\mathcal{C}$ be the set of all pairs (I_i, J_j) with

$$\min(A^{(i)}) + \min(B^{(j)}) \le u \quad \text{and} \quad \max(A^{(i)}) + \max(B^{(j)}) \ge \ell.$$

Observe that $\mathcal{C}$ is a unique-rectangle-covering of $(A,B,[\ell,u])$. We can easily compute $\mathcal{C}$ by brute force, by iterating over all $1 \le i, j \le q$ and testing whether to put (I_i, J_j) into $\mathcal{C}$ in time $O(1)$, see Algorithm 2. This takes total time $O(q^2) = O(nm/\text{out}) = O(\min\{n,m\})$, assuming that we have random access to A and B.

Algorithm 2 Covering for Interval-Restricted Sumset Computation

```
1:  procedure FindCovering(A, B, ℓ, u, out~)
2:      q ← ⌈(nm/out~)^{1/2}⌉
3:      C ← ∅
4:      for i = 1 to q do
5:          for j = 1 to q do
6:              I ← {(i−1)·n/q + 1, ..., i·n/q}
7:              J ← {(j−1)·m/q + 1, ..., j·m/q}
8:              if min(A_I) + min(B_J) ≤ u then
9:                  if max(A_I) + max(B_J) ≥ ℓ then
10:                     C ← C ∪ {(I, J)}
11:     Return C
```

Recall that the cost of a covering $\mathcal{C}$ is

$$\sum_{(I,J)\in\mathcal{C}} |A_I + B_J|.$$

We split $\mathcal{C}$ into two parts, the rectangles in the interior and the rectangles at the boundary:

$$\begin{aligned} \mathcal{C}_{\text{int}} &:= \{(I,J) \in \mathcal{C} \mid A_I + B_J \subseteq [\ell,u]\}, \\ \mathcal{C}_{\text{bd}} &:= \mathcal{C} \setminus \mathcal{C}_{\text{int}}. \end{aligned}$$

For the interior rectangles, we split their cost into diagonal sums of the form

$$\sum_{(I_i, J_{i+\Delta}) \in \mathcal{C}_{\text{int}}} |A^{(i)} + B^{(i+\Delta)}|,$$

for $-q < \Delta < q$. We claim that each such diagonal sum is bounded from above by out. Indeed, for consecutive terms along a diagonal we have

$$\max(A^{(i)}) + \max(B^{(i+\Delta)}) < \min(A^{(i+1)}) + \min(B^{(i+\Delta+1)}).$$

Therefore, the output-sizes $|A^{(i)} + B^{(i+\Delta)}|$ are disjoint contributions to out, for fixed Δ and ranging over all i. It follows that a diagonal sum is bounded by out, and since there are $2q-1$ diagonals, we obtain

$$\sum_{(I,J) \in \mathcal{C}_{\text{int}}} |A_I + B_J| \leq (2q-1)\text{out} = O(\sqrt{nm \cdot \text{out}}).$$

We argue geometrically about the boundary. Observe that $\mathcal{C}_{\text{bd}}$ contains at most two rectangles per diagonal, which yields $|\mathcal{C}_{\text{bd}}| \leq 4q$. For each $(I,J) \in \mathcal{C}_{\text{bd}}$ we use the trivial upper bound $|A_I + B_J| \leq |A_I| \cdot |B_J| = nm/q^2$. In total, this yields cost

$$\sum_{(I,J) \in \mathcal{C}_{\text{bd}}} |A_I + B_J| \leq 4q \cdot \frac{nm}{q^2} = O(\sqrt{nm \cdot \text{out}}).$$

The contribution from both parts is the same, so in total we bounded the cost of $\mathcal{C}$ by $O(\sqrt{nm \cdot \text{out}})$. This finishes the proof. □

5.2 Hardness Results

In this subsection we prove Theorem 2.17.

Proof. We want to prove hardness of Interval-Restricted sumset computation. Note that here we analyze running time in terms of $n = |A|$, $m = |B|$, and out, which are all invariant under shifting A by adding a number q to each $a \in A$; similarly for B. Therefore, we may drop the assumption that A, B are sets of positive integers and let them be subsets of $\mathbb{Z}$ instead. This is the case because we can shift A, B and the interval appropriately, so that every number in the input becomes non-negative.

Reduction from Boolean Matrix Multiplicaion to Interval-Restricted Sumset Computation: In Boolean matrix multiplication we are given $n \times n$ matrices $\overline{A}, \overline{B}$ with entries in $\{0,1\}$ and want to compute their product C with $C_{ij} = \bigvee_r \overline{A}_{ir} \wedge \overline{B}_{rj}$.

Given matrics $\overline{A}, \overline{B}$, we construct sets A, B as

$$A := \{rM^2 + \overline{A}_{ir} \cdot M + i \mid i, r \in [n]\},$$
$$B := \{-rM^2 + \overline{B}_{rj} \cdot M + nj \mid r, j \in [n]\},$$

where M is any integer greater than $10(n^2+n)$. We also set

$$\ell := 2M + n + 1, \quad u := 2M + n^2 + n.$$

We observe the following.

1. Every integer of the form $(\overline{A}_{ir} + \overline{B}_{rj}) \cdot M + (i + nj)$ with $i, r, j \in [n]$ is contained in $A + B$.

2. For $r \neq r'$ any sum $(rM^2 + \overline{A}_{ir} \cdot M + i) + (-r'M^2 + \overline{B}_{r'j} \cdot M + nj)$ is either less than $-M^2 + 2M + n^2 + n < 0$ or at least $M^2 + n + 1$, and hence outside of $[\ell, u]$.

3. If $\overline{A}_{ir} \wedge \overline{B}_{rj} = 1$, then $(\overline{A}_{ir} + \overline{B}_{rj})M + (i + nj) = 2M + (i + nj)$.

4. If $\overline{A}_{ir} \wedge \overline{B}_{rj} = 0$, then $(\overline{A}_{ir} + \overline{B}_{rj})M + (i + nj) \leq M + (i + nj) < 2M$.

It follows that from $(A + B) \cap [2M + n + 1, 2M + n^2 + n]$ we can infer all entries of the product matrix C.

Note that the output-size is out $\leq u - \ell + 1 = n^2$. Hence, for any $\delta > 0$, an $O((|A| + |B| + \text{out})^{\omega/2-\delta})$-time algorithm for Interval-Restricted sumset computation would yield an $O(n^{2\omega-2\delta})$-time algorithm for Boolean matrix multiplication.

Reduction from Sliding Window Hamming Distance to Interval-Restricted Convolution: Note that Interval-Restricted convolution allows us to not only to compute $(A + B) \cap [\ell, u]$, but also the number of ways an element $x \in (A + B) \cap [\ell, u]$ can be written as $x = a + b$ with $a \in A$, $b \in B$; let us call this number the multiplicity of x. Indeed, for the indicator vectors of A and B, the x-th entry of their convolution is the multiplicity of x in $A + B$.

In the sliding window Hamming distance problem we are given a text t of length $2n$ and a pattern p of length n and want to compute the Hamming distance between the pattern and every length-n substring of the text. Given such an instance t, p, we construct sets A, B as

$$A := \{M \cdot t_i + i \mid 1 \leq i \leq 2n\}, \quad B := \{-M \cdot p_j - j \mid 1 \leq j \leq n\},$$

where $M := 100n$. We also set $\ell := 1$, $u := n$ and compute, as mentioned in the previous paragraph, the Interval-Restricted sumset $(A + B) \cap [\ell, u]$ as well as the multiplicity of every x in this set.

Fix a $1 \leq i \leq n$ and observe that

1. If $t_{i+j} = p_j$ then the the pair $(i + j, j)$ will contribute 1 to the multiplicity of i.

2. If $t_{i+j} \neq p_j$ then the pair $(i + j, j)$ will contribute 1 to the multiplicity of a coefficient outside of the interval $[1, n]$, by the choice of M.

3. Every pair (i', j) with $i' \leq j$ contributes 1 to a coefficient outside of the interval $[1, n]$.

It follows that we can read off the Hamming distance between the pattern p and the i-th length-n substring of t from the multiplicity of i in the output. This completes the reduction from sliding window Hamming distance to prefix-restricted convolution. Since out $\leq n$, any $\widetilde{O}(|A| + |B| + (|A| \cdot |B| \cdot \text{out})^{1/2-\delta})$-time algorithm for Interval-Restricted comvolution would solve sliding window Hamming distance in time $\widetilde{O}(n^{3/2-3\delta})$. □

6 Relaxed Version of Prefix-Restricted Convolution

We show how to solve Prefix-Restricted convolution on any instance (A, B, u) in time $\widetilde{O}(\zeta^{-1}|(A + B) \cap [u(1+\zeta u)]| + \zeta^{-2})$, for any $\zeta \leq 1$, proving Theorem 2.13. More precisely, we prove the following theorem, from which we conclude Theorem 2.13 using Observation 2.8.

Theorem 6.1. *Given sets A, B and a target u we can compute a unique-rectangle-covering of $(A, B, [u])$ with cost*

$$O(\zeta^{-1}|(A + B) \cap [(1 + \zeta)u]|)$$

in time $O(|A| + |B| + \zeta^{-2})$.

Proof. Find the smallest ℓ such that $2^{-\ell} \le \zeta$. It suffices to solve the problem for $\zeta = 2^{-\ell}$, since $(A+B) \cap [u + 2^{-\ell}u] \subseteq (A+B) \cap [u + \zeta u]$, and $2^{\ell} \le 2\zeta^{-1}$. Thus, we can assume that $1/\zeta$ is a power of 2. Moreover, by shifting u as well as A by a suitable number we can assume that u is a power of 2. We split the set $[u]$ into the intervals

$$U_r := [(r-1) \cdot (\zeta u)/2 + 1, r \cdot (\zeta u)/2] .$$

Moreover, we define the following sets for $1 \le i, j \le 2/\zeta$:

$$\begin{aligned} A^{(i)} &:= A \cap U_i, \\ B^{(j)} &:= B \cap U_j, \\ I_i &:= \{i' \in [n] : A_{i'} \in U_i\} \\ J_j &:= \{j' \in [m] : B_{j'} \in U_j\}. \end{aligned}$$

Let $\mathcal{C}$ be the set of all pairs (I_i, J_j) for which $(A^{(i)} + B^{(j)}) \cap [u]$ is not empty; this condition can be easily checked by testing whether

$$\min(A^{(i)}) + \min(B^{(j)}) \le u.$$

Note that $\mathcal{C}$ can be computed in time $O(\zeta^{-2})$, assuming that we have random access to A and B. Observe that $\mathcal{C}$ is indeed a unique-rectangle-covering of $(A+B) \cap [u]$. Recall that the cost of $\mathcal{C}$ is

$$\sum_{(I,J) \in \mathcal{C}} |A_I + B_J|.$$

Similarly to the argument in the proof of Theorem 2.16, this sum can be decomposed into $4/\zeta$ diagonal sums of the form

$$\sum_i |A^{(i)} + B^{(i+\Delta)}|,$$

where the sum is over all i such that $(I_i, J_{i+\Delta}) \in \mathcal{C}$. We again use

$$\max(A^{(i)}) + \max(B^{(j)}) < \min(A^{(i+1)}) + \min(B^{(j+1)}).$$

Moreover, for any $(i,j) \in P$ we now have

$$\begin{aligned} \max(A^{(i)}) + \max(B^{(j)}) &\le \left(\min(A^{(i)}) + \frac{\zeta u}{2}\right) + \left(\min(B^{(j)}) + \frac{\zeta u}{2}\right) \\ &= \left(\min(A^{(i)}) + \min(B^{(j)})\right) + \zeta u \\ &\le u + \zeta u. \end{aligned}$$

It follows that every diagonal sum contributes at most

$$|(A+B) \cap [u + \zeta u]| .$$

Summing over all diagonals, in total we can bound the cost of $\mathcal{C}$ by $O(\zeta^{-1} |(A+B) \cap [u + \zeta u]|)$. ☐

7 Construction of the $\widetilde{\mathbf{O}}(\mathrm{out}^{4/3})$-cost Covering

This section is devoted to proving the technical core of our Prefix-Restricted sumset algorithm, specifically we prove Theorem 2.9.

7.1 An Additive Combinatorics Ingredient: Ruzsa's Triangle Inequality

The following is a classical result from Additive Combinatorics. We present a self-contained proof.

Lemma 7.1 (Ruzsa's Triangle Inequality, see also [16, Theorem 2])**.** *For any $A, B, C \subseteq \mathbb{Z}$ we have*

$$|A - B| \leq \frac{|A - C| \cdot |C - B|}{|C|}.$$

Proof. We associate every $s \in A - B$ with the lexicographically smallest pair $(a, b) \in A \times B$ such that $s = a - b$, and we denote this pair by $(a(s), b(s))$. Consider the mapping

$$\begin{aligned} (A - B) \times C &\to (A - C) \times (C - B) \\ (s, c) &\mapsto (a(s) - c, c - b(s)) \end{aligned}$$

We claim that this mapping is injective. Indeed, from an image $(x, y) = (a(s) - c, c - b(s))$ we can infer $s = a(s) - b(s) = x + y$. The value s then determines $a(s)$ and $b(s)$, so we can infer $c = y + b(s)$. We thus recovered the corresponding preimage (s, c).

Since this mapping is injective, we obtain $|A - B| \cdot |C| \leq |A - C| \cdot |C - B|$. □

We will use the following simple corollary.

Lemma 7.2 (Corollary of Ruzsa's Triangle Inequality)**.** *For any $X, Y, Z, W \subseteq \mathbb{Z}$ we have*

$$|X + Y| \leq \frac{|X + Z| \cdot |Z + W| \cdot |W + Y|}{|Z| \cdot |W|}.$$

Proof. First use Ruzsa's triangle inequality on $A = X$, $B = -Y$, $C = -Z$ to obtain

$$|X + Y| \leq \frac{|X + Z| \cdot |Z - Y|}{|Z|}.$$

Then use Ruzsa's triangle inequality on $A = Z$, $B = Y$, $C = -W$ to obtain

$$|Z - Y| \leq \frac{|Z + W| \cdot |W + Y|}{|W|}.$$

Plugging the latter into the former proves the claim. □

7.2 Description of the Algorithm

Given $A, B \subseteq [u]$, we write $n = |A|$, $m = |B|$, and $\text{out} = |(A + B) \cap [u]|$. We describe an algorithm that computes a unique rectangle covering $\mathcal{C}$ of $(A, B, [u])$ of cost $\widetilde{O}(\text{out}^{4/3})$ in time $\widetilde{O}(\text{out}^{4/3})$. However, as a subroutine we will use output-sensitive sumset computation (Theorem 2.2), which shows that it is hard to completely separate the tasks of computing a covering and computing the Prefix-Restricted sumset itself.

Invoking Lemma 4.5, it suffices to solve the promise problem where we are given a value $\widetilde{\text{out}}$ guaranteed to satisfy $\text{out} \leq \widetilde{\text{out}} \leq O(\text{out})$. We can assume that $\widetilde{\text{out}}$ is larger than some absolute constant, since otherwise we have $|A|, |B| \leq \text{out} \leq \widetilde{\text{out}} = O(1)$, so we can compute a trivial covering of cardinality 1 and cost $|A| \cdot |B| = O(1)$.

We maintain families $\mathcal{C}, \mathcal{D}$, initialized to $\mathcal{C} = \emptyset$ and $\mathcal{D} = \{([n], [m])\}$, with the invariant that $\mathcal{C} \cup \mathcal{D}$ is a unique rectangle covering of $(A, B, [u])$. We refer to the rectangles in $\mathcal{D}$ as the *unprocessed*

subproblems, or simply *subproblems*. The algorithm is finished when there are no more unprocessed subproblems, i.e., $\mathcal{D} = \emptyset$, and then we return the unique rectangle covering $\mathcal{C}$ as output.

We associate to every subproblem $(I, J) \in \mathcal{D}$ the *type* (x, y) for $x = \lceil \log |I| \rceil$ and $y = \lceil \log |J| \rceil$. Initially, there is exactly one subproblem $([n], [m])$ of type $(\lceil \log n \rceil, \lceil \log m \rceil)$.

We define a total order on types: $(x, y) \prec (x', y')$ iff $x + y < x' + y'$, or $x + y = x' + y'$ and $x < x'$. Our algorithm processes types in descending order according to this total order. For any type (x, y), we process all subproblems of type (x, y) in one batch. Upon processing a subproblem, our algorithm may generate further subproblems of strictly smaller type. As we will see below, all subproblems of the same type that we generate are disjoint, i.e., for any subproblems $(I, J), (I', J')$ of the same type we have $I \cap I' = \emptyset$ and $J \cap J' = \emptyset$.

Since we want to process all subproblems of a particular type (x, y) in one batch, we need to store the family $\mathcal{D}$ in such a way that we can efficiently enumerate all subproblems of a type (x, y). To this end, we store $\mathcal{D}$ in a standard data structure such as a self-adjusting binary search tree, where subproblems are first compared according to their type and then according to the endpoints of I and J. Note that we can store any subproblem (I, J) using $O(1)$ integers, since I and J are intervals.

We set the parameter $q := \lceil \widetilde{\text{out}}^{1/3} \rceil$.

To finish the description of the algorithm, it remains to describe how we process all subproblems of a particular type (x, y) in one batch. We consider two cases.

Case 1: $2^{x+y} \leq \widetilde{\text{out}}$. Then for each subproblem (I, J) of type (x, y) we move (I, J) from $\mathcal{D}$ to $\mathcal{C}$.

Case 2: $2^{x+y} > \widetilde{\text{out}}$. If $\mathcal{D}$ contains more than q subproblems of type (x, y), then we do the following. We start computing $A_I + B_J$ for each of these subproblems in parallel (using Theorem 2.2), and we stop once all but q of these calls have finished. For each finished call $A_I + B_J$, we move (I, J) from $\mathcal{D}$ to $\mathcal{C}$.

At this point, we have at most q subproblems of type (x, y) left. We split each such subproblem $(I, J) = ([i_1, i_2], [j_1, j_2])$ as follows. We set $i := \lfloor (i_1 + i_2)/2 \rfloor$ and determine the maximum index $j \in J$ with $A_i + B_j \leq u$. This splits I into $I_1 = [i_1, i]$ and $I_2 = [i + 1, i_2]$ and J into $J_1 = [j_1, j]$ and $J_2 = [j + 1, j_2]$. We add (I_1, J_1) to the output $\mathcal{C}$. Moreover, we add the subproblems (I_1, J_2) and (I_2, J_1) to $\mathcal{D}$ and we remove (I, J) from $\mathcal{D}$. Since $u < A_i + B_{j+1} \leq A_{i+1} + B_{j+1}$, we can ignore the subproblem (I_2, J_2).

This finishes the description of our algorithm, for pseudocode see Algorithm 3.

7.3 Analysis

The correctness of the algorithm, meaning that the output is a unique rectangle covering, follows from the next claim and the fact that the algorithm stops when $\mathcal{D} = \emptyset$.

Claim 7.3 (Invariant that $\mathcal{C} \cup \mathcal{D}$ is a covering)**.** *At any point during the algorithm, the family $\mathcal{C} \cup \mathcal{D}$ is a unique rectangle covering of $(A, B, [u])$.*

Proof. Moving (I, J) from $\mathcal{D}$ to $\mathcal{C}$ does not change this property. Therefore, the only crucial step is the splitting of I into I_1, I_2 and of J into J_1, J_2, where we add (I_1, J_1) to $\mathcal{C}$ and add $(I_1, J_2), (I_2, J_1)$ to $\mathcal{D}$. Observe that at this point we have $u < A_i + B_{j+1} \leq A_{i+1} + B_{j+1}$, and thus the rectangle (I_2, J_2) does not contain any sum below u, so it is unnecessary for a covering. It follows that $\mathcal{C} \cup \mathcal{D}$ remains a covering. Moreover, noting that I_1, I_2, J_1, J_2 are again intervals, it remains rectangular. Finally, since no pair (i, j) is contained in two subproblems $I_b \times J_{b'}$, it remains unique. □

We will need the following claims to analyze the running time and the cost of the covering.

Algorithm 3

procedure CoveringConstruction($A, B, u, \widetilde{\text{out}}$)
  $n \leftarrow |A|$, $m \leftarrow |B|$, $q \leftarrow \widetilde{\text{out}}^{1/3}$
  Initialize $\mathcal{C} \leftarrow \emptyset$, $\mathcal{D} \leftarrow \{([n], [m])\}$
  while $\mathcal{D} \neq \emptyset$ **do**
    Let (x, y) be the largest type such that $\mathcal{D}$ contains subproblems of type (x, y)
    // process all subproblems of type (x, y) in one batch:
    if $2^{x+y} \leq \widetilde{\text{out}}$ **then**
      for each $(I, J) \in \mathcal{D}$ of type (x, y): Move (I, J) from $\mathcal{D}$ to $\mathcal{C}$
    else
      if $\mathcal{D}$ contains more than q subproblems of type (x, y) **then**
        Compute $A_I + B_J$ in parallel for all $(I, J) \in \mathcal{D}$ of type (x, y)
        Stop once all but q of these calls finished
        for each finished call $A_I + B_J$: Move (I, J) from $\mathcal{D}$ to $\mathcal{C}$
      for each remaining $(I, J) \in \mathcal{D}$ of type (x, y) **do**
        Split $I = [i_1, i_2]$ at $i = \lfloor (i_1 + i_2)/2 \rfloor$ into I_1 and I_2
        Determine the maximum $j \in J$ with $A_i + B_j \leq u$
        Split J at j into J_1 and J_2
        Add (I_1, J_1) to $\mathcal{C}$
        Add $(I_1, J_2), (I_2, J_1)$ to $\mathcal{D}$
        Remove (I, J) from $\mathcal{D}$
  return $\mathcal{C}$

Claim 7.4 (Subproblems of type (x, y) form a staircase)**.** *For any distinct subproblems $(I, J), (I', J')$ of the same type (x, y), we have $\max(I') < \min(I)$ and $\max(J) < \min(J')$ (or vice versa). Moreover, if this holds then $A_{I'} + B_J \subseteq [u]$.*

Proof. This is true in the beginning since $|\mathcal{D}| = 1$. Moving (I, J) from $\mathcal{D}$ to $\mathcal{C}$ cannot violate this property. Therefore, the only crucial step is the splitting of I into I_1, I_2 and of J into J_1, J_2, where we remove (I, J) from $\mathcal{D}$ and add $(I_1, J_2), (I_2, J_1)$ to $\mathcal{D}$. In this situation, clearly $(I_1, J_2), (I_2, J_1)$ form a staircase, and $A_{I_1} + B_{J_1} \subseteq [u]$. Thus, the property is maintained "locally". It is not hard to see that the property is also maintained "globally", when we have distinct subproblems $(I, J), (I', J')$ that satisfy the property and we split both of them into subproblems. □

Claim 7.5 (Any subproblem creates at most two subproblems of strictly smaller type)**.** *Processing a subproblem $(I, J) \in \mathcal{D}$ can cause the insertion of at most two new subproblems into $\mathcal{D}$, both of strictly smaller type.*

Proof. The only point at which we add new subproblems to $\mathcal{D}$ is the splitting phase. So let (I, J) be a subproblem of type (x, y) and consider the splitting of I into I_1, I_2 and of J into J_1, J_2, where we remove (I, J) from $\mathcal{D}$ and add $(I_1, J_2), (I_2, J_1)$ to $\mathcal{D}$. Since we split I at the midpoint, we have $|I_r| \leq \lceil |I|/2 \rceil$, for any $r \in \{1, 2\}$. We clearly also have $|J_r| \leq |J|$. Hence, the new type $(x_r, y_r) = (\lceil \log |I_r| \rceil, \lceil \log |J_r| \rceil)$ satisfies[3] $x_r < x$ and $y_r \leq y$. As this implies $x_r + y_r < x + y$, the newly added subproblems have a strictly smaller type. □

[3]To be precise, here we use that for any integers z, w the inequalities $2^{w-1} < z \leq 2^w$ imply $2^{w-2} < \lceil z/2 \rceil \leq 2^{w-1}$, and thus $\lceil \log(\lceil z/2 \rceil) \rceil \leq \lceil \log z \rceil - 1$.

Claim 7.6 (Invariant on the number of subproblems)**.** *At any point during the algorithm, there are at most* $2q\log(n)\log(m)$ *subproblems of any fixed type* (x, y)*.*

Proof. The claim is immediate for $(x, y) = (\lceil \log n \rceil, \lceil \log m \rceil)$. Fix a type (x, y) and assume that the claim is true for every type (x', y') with $(x, y) \prec (x', y')$. Note that for any type (x', y') at most q subproblems of type (x', y') reached the splitting phase, and each such subproblem gave rise to at most two newly added subproblems. Since the number of different types is at most $\log(n)\log(m)$, we obtain the claimed bound on the number of subproblems of type (x, y). □

The following claim lies at the core of the analysis of our covering construction, as it bounds the running time and added cost of Case 2.

Claim 7.7. *Fix a type* (x, y) *with* $2^{x+y} > \widetilde{\text{out}}$*. Lines 11-13 of Algorithm 3 take time* $\widetilde{O}(\text{out}^{4/3})$ *and add rectangles of total cost* $\widetilde{O}(\text{out}^{4/3})$ *to* $\mathcal{C}$*.*

Proof. We denote by $(I_1, J_1), (I_2, J_2), \ldots, (I_R, J_R)$ the subproblems of type (x, y). We can assume $R \geq q$, since otherwise lines 11-13 are not called. Note that $R \leq 2q\log(n)\log(m)$ by Claim 7.6. By Claim 7.4 these subproblems form a staircase, so we can assume that

$$\max(I_1) < \min(I_2), \max(I_2) < \min(I_3), \ldots, \max(I_{R-1}) < \min(I_R)$$
$$\min(J_1) > \max(J_2), \min(J_2) > \max(J_3), \ldots, \min(J_{R-1}) > \max(J_R).$$

Claim 7.4 also implies that for any $r < \ell$ we have

$$A_{I_r} + B_{J_\ell} \subseteq [u]. \tag{5}$$

Set $\delta := 1/(6\log(n)\log(m))$, so that $3\delta R \leq q$. For any $r \in [R]$ set

$$S(r) := \sum_{i=1}^{r-1} \sum_{j=r+1}^{R} |A_{I_r} + B_{J_j}| + |A_{I_i} + B_{J_j}| + |A_{I_i} + B_{J_r}|.$$

Claim 7.8. *There are at least* $(1-\delta)R$ *indices* r *with* $S(r) \leq 6R\,\text{out}/\delta$*.*

Proof. By simple counting how often a summand $|A_{I_r} + B_{J_\ell}|$ can appear, we observe that

$$\sum_{r=1}^{R} S(r) \leq 3R \sum_{r<\ell} |A_{I_r} + B_{J_\ell}|.$$

Note that we have $|A_{I_r} + B_{J_\ell}| \leq \text{out}$ by equation (5). However, we need a stronger property. We decompose the sum $\sum_{r<\ell} |A_{I_r} + B_{J_\ell}|$ into $2R-1$ diagonal sums of the form $\sum_r |A_{I_r} + B_{J_{r+\Delta}}|$. Since $\max(A_{I_r} + B_{J_\ell}) < \min(A_{I_{r+1}} + B_{J_{\ell+1}})$, all summands in a diagonal sum are disjoint, and thus each diagonal sum is bounded from above by out. We thus obtain

$$\sum_{r<\ell} |A_{I_r} + B_{J_\ell}| \leq (2R-1)\text{out},$$

which yields

$$\sum_{r=1}^{R} S(r) \leq 6R^2\text{out}.$$

By Markov's inequality, it follows that all but δR indices r satisfy $S(r) \leq 6R\,\text{out}/\delta$. □

In the remainder we consider only indices $\delta R \le r \le (1-\delta)R$ with $S(r) \le 6R\,\mathrm{out}/\delta$; note that there are at least $(1-3\delta)R$ such indices r. For any such r, there are at least $\frac{\delta R^2}{2}$ pairs (i,j) with $1 \le i < r$ and $r < j \le R$. Hence, there exist i, j with $i < r < j$ and

$$|A_{I_r} + B_{J_j}| + |A_{I_i} + B_{J_j}| + |A_{I_i} + B_{J_r}| \le \left(\frac{6R\,\mathrm{out}}{\delta}\right) \Big/ \left(\frac{\delta R^2}{2}\right) = \frac{12\,\mathrm{out}}{\delta^2 R}.$$

We continue by invoking the corollary of Ruzsa's triangle inequality (Lemma 7.2), see Figure 1 for an illustration:

$$|A_{I_r} + B_{J_r}| \le \frac{|A_{I_r} + B_{J_j}| \cdot |B_{J_j} + A_{I_i}| \cdot |A_{I_i} + B_{J_r}|}{|A_{I_i}| \cdot |B_{J_j}|} \le \frac{12^3 \mathrm{out}^3/(\delta^6 R^3)}{2^{x-1} \cdot 2^{y-1}}.$$

By the case assumption $2^{x+y} > \widetilde{\mathrm{out}} \ge \mathrm{out}$ and $R \ge q = \lceil \widetilde{\mathrm{out}}^{1/3} \rceil \ge \mathrm{out}^{1/3}$, and by our choice of $\delta = 1/(6\log(n)\log(m))$, we obtain

$$|A_{I_r} + B_{J_r}| \le O(\mathrm{out} \cdot \log^{12}(nm)).$$

Recall that this holds for at least $(1-3\delta)R \ge R - q$ many indices r.

Since we compute all sumsets $A_{I_r} + B_{J_r}$ in parallel and stop once all but q calls are finished, it follows that we stop after spending time $\widetilde{O}(\mathrm{out})$ for each call. Over $R \le 2q\log(n)\log(m) = \widetilde{O}(\mathrm{out}^{1/3})$ calls, this takes total time $\widetilde{O}(\mathrm{out}^{4/3})$. Moreover, any finished call ran in time $\widetilde{O}(\mathrm{out})$, so it has output-size $\widetilde{O}(\mathrm{out})$, so it contributes cost $\widetilde{O}(\mathrm{out})$. Thus, we add rectangles of total cost $\widetilde{O}(\mathrm{out}^{4/3})$ in every invocation of lines 11-13 of Algorithm 3. □

Claim 7.9 (Cost Bound)**.** *For any type (x,y) we add rectangles of total cost $\widetilde{O}(\mathrm{out}^{4/3})$ to $\mathcal{C}$.*

Proof. If $2^{x+y} > \widetilde{\mathrm{out}}$, then the cost contributed by lines 11-13 of Algorithm 3 is bounded in Claim 7.7. Additionally, in the splitting phase (line 18) we add a rectangle (I_1, J_1) satisfying $A_{I_1} + B_{J_1} \subseteq [u]$, and thus $|A_{I_1} + B_{J_1}| \le \mathrm{out}$. Since we split at most q subproblems, we add a total cost of $\widetilde{O}(q \cdot \mathrm{out}) = \widetilde{O}(\mathrm{out}^{4/3})$.

If $2^{x+y} \le \widetilde{\mathrm{out}}$, then by the trivial bound each added rectangle (I,J) has cost at most $|I| \cdot |J| \le 2^{x+y} \le \widetilde{\mathrm{out}} = O(\mathrm{out})$. By Claim 7.6 we have $\widetilde{O}(q)$ subproblems, and thus we add a total cost of $\widetilde{O}(q \cdot \mathrm{out}) = \widetilde{O}(\mathrm{out}^{4/3})$. □

Over all types (x,y), we obtain a total cost of $\widetilde{O}(\mathrm{out}^{4/3})$. The running time bound of $\widetilde{O}(\mathrm{out}^{4/3})$ follows along the same lines. This finishes the proof of Theorem 2.9.

8 Lower Bound on Coverings

This section is devoted to proving Theorem 2.12. We will use the following notation. For a vector x we write $w(x)$ for its Hamming weight, i.e., its number of non-zero coordinates. For vectors x, y we write $d_H(x,y)$ for their Hamming distance, i.e., the number of coordinates in which they differ.

For sets X, Y we denote their symmetric difference by $X \triangle Y := (X \setminus Y) \cup (Y \setminus X)$. We identify sets with their indicator vectors. In particular, note that if x, y are the indicator vectors of sets X, Y, respectively, then $d_H(x,y) = |X \triangle Y|$.

Recall the definition of the binary entropy function $h(x) = -x\log x - (1-x)\log(1-x)$ and note that $h(1/2) = 1$. Finally, we use notation $O_\delta(.)$, $\Omega_\delta(.)$, and $\Theta_\delta(.)$ to hide constants that only depend on the parameter δ.

The following is a standard construction of an error correcting code.

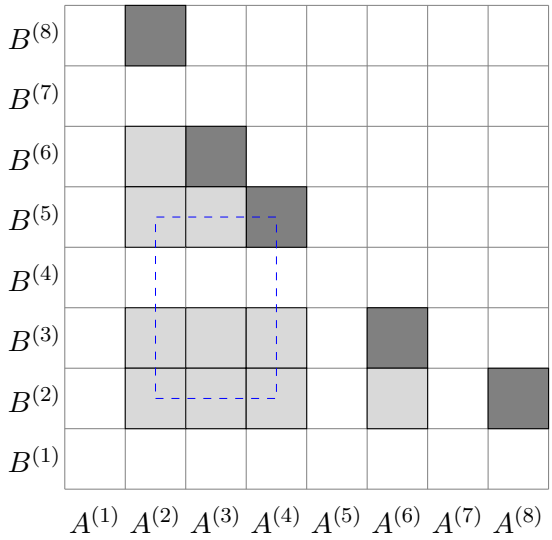

Figure 1: Illustration of the proof of Claim 7.8. The thick gray squares depict the unprocessed subproblems $(I_1, J_1), (I_2, J_2), \ldots$ of a specific type (x, y). The light gray squares depict pairs (I_r, J_ℓ) for $r < \ell$. The vertices of the blue rectangle correspond to (starting from the top-right corner and going clockwise) $(I_r, J_r), (I_r, J_j), (I_i, J_j), (I_i, J_r)$. From such a 4-tuple, where all sumsets $|A_{I_r} + B_{J_j}|, |A_{I_i} + B_{J_j}|, |A_{I_r} + B_{J_j}|$ are "small", Ruzsa's triangle inequality (Lemma 7.2) allows us to bound $|A_{I_r} + B_{J_r}|$ from above.

Lemma 8.1 (Constant-weight Binary Code)**.** *Fix $0 \le \delta < 1/2$. For any even integer t there exists a code $\mathcal{E} \subseteq \{0,1\}^t$ such that:*

- *Each codeword $x \in \mathcal{E}$ has weight $w(x) = t/2$,*
- *Any two codewords $x, y \in \mathcal{E}$ with $x \neq y$ satisfy $d_H(x, y) \ge \delta t$, and*
- *The number of codewords is $|\mathcal{E}| = \Omega_\delta(2^{(1-h(\delta))t})$.*

Proof. We pick $\mathcal{E}$ greedily among all $\binom{t}{t/2}$ many vectors $x \in \{0,1\}^t$ with Hamming weight $t/2$. Whenever we pick a vector x, we mark all vectors y within distance δt of x as unpickable. Note that we mark at most $\sum_{i=0}^{\delta t} \binom{t}{i}$ many vectors y. In total, this process picks $|\mathcal{E}| \ge \binom{t}{t/2} / \left(\sum_{i=0}^{\delta t} \binom{t}{i}\right)$ vectors. The claim now follows from the following facts about binomial coefficients:

- $\sum_{i=0}^{\alpha n} \binom{n}{i} = \Theta_\alpha\left(\binom{n}{\alpha n}\right)$ for any $0 \le \alpha < 1/2$,
- $\binom{n}{\alpha n} = \Theta_\alpha\left(2^{h(\alpha)n}/\sqrt{n}\right)$.

Indeed, with these facts we obtain $|\mathcal{E}| \ge \binom{t}{t/2} / \left(\sum_{i=0}^{\delta t} \binom{t}{i}\right) = \Omega_\delta\left(\binom{t}{t/2} / \binom{t}{\delta t}\right) = \Omega_\delta\left(2^{(1-h(\delta))t}\right)$. ☐

We next lift the above code to a family of sets where the sumset $X^{(i)} + Y^{(j)}$ has large cardinality if $i = j$, and small cardinality if $i \neq j$.

Lemma 8.2. *Fix $0 \le \delta < 1/2$. For any integers $m, t \ge 2$, where t is even, for*

$$\sigma := m^t, \quad \alpha := 2^{\delta t/2} m^{(1-\delta/2)t}, \quad \text{and some} \quad g = \Omega_\delta\left(2^{(1-h(\delta))t}\right),$$

there exist sets $X^{(1)}, .., X^{(g)}, Y^{(1)}, .., Y^{(g)} \subseteq [\sigma]$ satisfying

1. $|X^{(i)}| = |Y^{(i)}| = \sigma^{1/2}$ *for any* $1 \le i \le g$,
2. $|X^{(i)} + Y^{(i)}| = \sigma$ *for any* $1 \le i \le g$, *and*
3. $|X^{(i)} + Y^{(j)}| \le \alpha$ *for any* $i \neq j$.

Proof. Let $\mathcal{E} \subseteq \{0,1\}^t$ be the code given by Lemma 8.1 and set $g := |\mathcal{E}|$. Let $I^{(1)}, \dots, I^{(g)} \subseteq \{0, 1, \dots, t-1\}$ be sets such that the codewords in $\mathcal{E}$ are the indicator vectors of $I^{(1)}, \dots, I^{(g)}$. For any $1 \le \ell \le g$, we write

$$\overline{I}^{(\ell)} := \{0, \dots, t-1\} \setminus I^{(\ell)}.$$

Moreover, for any $I \subseteq \{0, .., t-1\}$ we set

$$S(I) := \Big\{ \sum_{i \in I} \eta_i \cdot m^i \;\Big|\; 0 \le \eta_i < m \Big\},$$

where η_i ranges over all integers between 0 and $m-1$. Note that $S(I)$ is the set of all numbers with t digits in the m-ary system for which every digit outside of I is 0. Alternatively, it can be viewed as the sumset of $|I|$ many arithmetic progressions of length m and step sizes $\{m^i\}_{i \in I}$. Finally, we set

$$X^{(\ell)} := S(I^{(\ell)}) \qquad \text{and} \qquad Y^{(\ell)} = S(\overline{I}^{(\ell)}).$$

Note that any number in $S(I)$ is bounded from above by $\sum_{i=0}^{t-1}(m-1) \cdot m^i = m^t - 1$, and thus we have $X^{(\ell)}, Y^{(\ell)} \subseteq [m^t]$ for any ℓ. It remains to verify the three claims.

For (1.), note that for any ℓ we have $|X^{(\ell)}| = |S(I^{(\ell)})| = m^{|I^{(\ell)}|} = m^{t/2}$, since the code $\mathcal{E}$ has constant weight $t/2$. The same holds for $Y^{(\ell)}$.

For the remaining claims, for any ℓ, h we write

$$\big|X^{(\ell)} + Y^{(h)}\big| \;=\; \big|S(I^{(\ell)}) + S(\overline{I}^{(h)})\big| \;=\; \Big|\Big\{ \sum_{i \in I^{(\ell)}} \eta_i \cdot m^i + \sum_{j \in \overline{I}^{(h)}} \eta'_j \cdot m^j \;\Big|\; 0 \le \eta_i, \eta'_j < m \Big\}\Big|. \tag{6}$$

For (2.), we use that $I^{(\ell)} \cup \overline{I}^{(\ell)}$ is a partitioning of $\{0, \dots, t-1\}$ to obtain for any ℓ

$$\big|X^{(\ell)} + Y^{(\ell)}\big| \;=\; \Big|\Big\{ \sum_{i=0}^{t-1} \eta_i \cdot m^i \;\Big|\; 0 \le \eta_i < m \Big\}\Big| \;=\; m^t.$$

For (3.), for any $\ell \neq h$ we write $I := I^{(\ell)}$ and $J := \overline{I}^{(h)}$ and express the right hand side of (6) in terms of the intersection $I \cap J$ and the symmetric difference $I \triangle J$ as

$$\big|X^{(\ell)} + Y^{(h)}\big| \;=\; \Big|\Big\{ \sum_{i \in I \cap J} \eta_i \cdot m^i + \sum_{j \in I \triangle J} \eta'_j \cdot m^j \;\Big|\; 0 \le \eta_i \le 2m-2,\, 0 \le \eta'_j < m \Big\}\Big|.$$

This allows us to bound

$$\big|X^{(\ell)} + Y^{(h)}\big| \;\le\; (2m-1)^{|I \cap J|} \cdot m^{|I \triangle J|} \;\le\; 2^{|I \cap J|} \cdot m^{|I \cap J| + |I \triangle J|} \;=\; 2^{|I \cap J|} \cdot m^{|I \cup J|} \;=\; 2^t \Big(\frac{m}{2}\Big)^{|I \cup J|},$$

where we have used inclusion-exclusion $|I \cup J| = |I| + |J| - |I \cap J|$ combined with $|I| = |J| = t/2$ in the last step. We now use the fact that for any $S, T \subseteq \{0, \dots, t-1\}$ we have

$$\big|S \cup \big(\{0, \dots, t-1\} \setminus T\big)\big| = \frac{1}{2}\big(2t + |S| - |T| - |S \triangle T|\big).$$

Plugging in the bounds of $|I^{(\ell)}| = t/2$ and $|I^{(\ell)} \triangle I^{(h)}| \geq \delta t$ for any $\ell \neq h$ by the properties of the code $\mathcal{E}$, we obtain the bound

$$|I \cup J| \;=\; \big|I^{(\ell)} \cup \overline{I}^{(h)}\big| \;\leq\; \Big(1 - \frac{\delta}{2}\Big)t.$$

Together, this yields

$$\big|X^{(\ell)} + Y^{(h)}\big| \;\leq\; 2^t \Big(\frac{m}{2}\Big)^{(1-\delta/2)t} = 2^{\delta t/2} m^{(1-\delta/2)t},$$

finishing the proof.

□

Lemma 8.3. *With parameters σ, α, g as in Lemma 8.2, there exist sets $A, B \subseteq \mathbb{N}$ and an integer u such that*

1. $|A|, |B| = g \cdot \sigma^{1/2}$,
2. $\text{out} := |(A + B) \cap [u]| = O(g^2 \cdot \alpha + \sigma)$, *and*
3. *Any rectangle covering of $(A, B, [u])$ has cost $\Omega(g \cdot \sigma)$.*

Proof. Let $X^{(1)}, .., X^{(g)}, Y^{(1)}, .., Y^{(g)}$ be the sets constructed in Lemma 8.2. For $i \in \mathbb{N}$ we let

$$E(i) := \begin{cases} i, & \text{if } i \text{ is even} \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, we let M be a sufficiently large integer; setting $M := 100(\sigma + g)$ suffices. With this setup, for any $1 \leq i, j \leq g$ we define $A^{(i)}$ and $B^{(j)}$ as appropriate shifts of $X^{(i)}$ and $Y^{(j)}$, respectively, more precisely we set

$$A^{(i)} := X^{(i)} + \big\{i \cdot M^2 + E(i) \cdot M\big\} \qquad \text{and} \qquad A := \bigcup_{i=1}^{g} A^{(i)},$$

$$B^{(j)} := Y^{(j)} + \big\{(g - j) \cdot M^2\big\} \qquad \text{and} \qquad B := \bigcup_{j=1}^{g} B^{(j)}.$$

Finally, we set

$$u := g \cdot M^2 + 2\sigma.$$

We now verify the three claims. The size bound $|A|, |B| = g \cdot \sigma^{1/2}$ is immediate from the property $|X^{(i)}|, |Y^{(j)}| = \sigma^{1/2}$.

For the output-size, we use the following claim.

Claim 8.4. *The following properties hold.*

1. *For any $i \neq j$ we have $|A^{(i)} + B^{(j)}| \leq \alpha$,*
2. *For any even i we have $(A^{(i)} + B^{(i)}) \cap [u] = \emptyset$, and*
3. *For any odd i we have $A^{(i)} + B^{(i)} \subseteq [g \cdot M^2, g \cdot M^2 + 2\sigma] \subseteq [u]$.*

Proof. Claim 1. is immediate from Lemma 8.2.3, since $A^{(i)}$ and $B^{(j)}$ are just shifts of $X^{(i)}$ and $Y^{(j)}$.

For even i we have $\min(A^{(i)}) \geq i \cdot M^2 + M$ and $\min(B^{(i)}) \geq (g-i) \cdot M^2$ and thus $\min(A^{(i)} + B^{(i)}) \geq g \cdot M^2 + M > u$, which shows claim 2.

For odd i we have $\min(A^{(i)}) \geq i \cdot M^2$ and thus $\min(A^{(i)} + B^{(i)}) \geq g \cdot M^2$. Similarly, we have $\max(A^{(i)}) \leq i \cdot M^2 + \sigma$ and $\max(B^{(j)}) \leq (g-i) \cdot M^2 + \sigma$ and thus $\max(A^{(i)} + B^{(i)}) \leq g \cdot M^2 + 2\sigma$. □

The above claim allows us to bound

$$\text{out} \;=\; \big(A \cap B\big) \cap [u] \;\leq\; \left(\sum_{i \neq j} |A^{(i)} + B^{(j)}|\right) + \left|\bigcup_{\text{odd } i} \big(A^{(i)} + B^{(i)}\big) \cap [u]\right| \;\leq\; g^2 \cdot \alpha + 2\sigma + 1.$$

It remains to analyze coverings. So let $\mathcal{C}$ be a rectangle covering of $(A, B, [u])$. See Figure 2 for an illustration. We construct a graph G with vertex set $V(G)$ consisting of all odd integers $1 \leq i \leq g$. We put an edge $(i, i+2)$ into $E(G)$ if there exists a rectangle in $\mathcal{C}$ that contains a pair in $A^{(i)} \times B^{(i)}$ as well as a pair in $A^{(i+2)} \times B^{(i+2)}$. Note that such a rectangle contains *all* pairs in $A^{(i+1)} \times B^{(i+1)}$ and thus has cost at least

$$\big|A^{(i+1)} + B^{(i+1)}\big| = \sigma.$$

We now consider two cases, depending on the number of edges in G.

If G has more than $g/4$ edges, then for at least $g/4$ even integers i all pairs in $A^{(i)} \times B^{(i)}$ are covered by $\mathcal{C}$. Since $A^{(i)} + B^{(i)} \subseteq [g \cdot M^2 + i \cdot M, g \cdot M^2 + i \cdot M + 2\sigma]$ has size σ, and M is large, all sumsets $A^{(i)} + B^{(i)}$ for even i are disjoint, and therefore any covering of $g/4$ such sumsets has cost at least $g\sigma/4$.

Otherwise, if G has at most $g/4$ edges, then it has at least $g/4$ components. Note that each component must be covered by distinct rectangles in $\mathcal{C}$. Since each component requires cost at least $|A^{(i)} + B^{(i)}| = \sigma$, the covering has a total cost of at least $g\sigma/4$. This finishes the proof. □

We are now ready to prove Theorem 2.12.

Proof. Lemma 8.3 yields for any $\delta > 0$ and integers m, t, where t is even, a tuple $(A, B, [u])$ with

$$\text{out} \;:=\; \big|(A + B) \cap [u]\big| \;=\; O_\delta\big(2^{(2-2h(\delta)+\delta/2)t} m^{(1-\delta/2)t} + m^t\big),$$

and any rectangle covering of $(A, B, [u])$ has cost $\Omega_\delta(2^{(1-h(\delta))t} m^t)$. Observe that we can write this cost bound in the form $\Omega_\delta(\text{out}^c)$ for

$$c := \frac{(1-h(\delta))t + t \log m}{\max\{(2 - 2h(\delta) + \delta/2)t + (1-\delta/2)t \log m,\; t \log m\}}.$$

Note that t cancels in this expression. We numerically optimize $c = c(\delta, m)$ by setting $m := 10$ and $\delta := 0.2709$, obtaining $c \geq 1.047$. In particular, we constructed an infinite sequence of tuples $(A, B, [u])$ for which any rectangle covering has cost $\Omega(\text{out}^{1.047})$, which proves Theorem 2.12. □

9 Reducing SubsetSum to Prefix-Restricted Sumset Computation

This section is devoted to proving Theorems 2.11 and 2.13. In particular, we give an output-sensitivity-preserving reduction from SubsetSum to top-k-convolution. For that, we need the following definition.

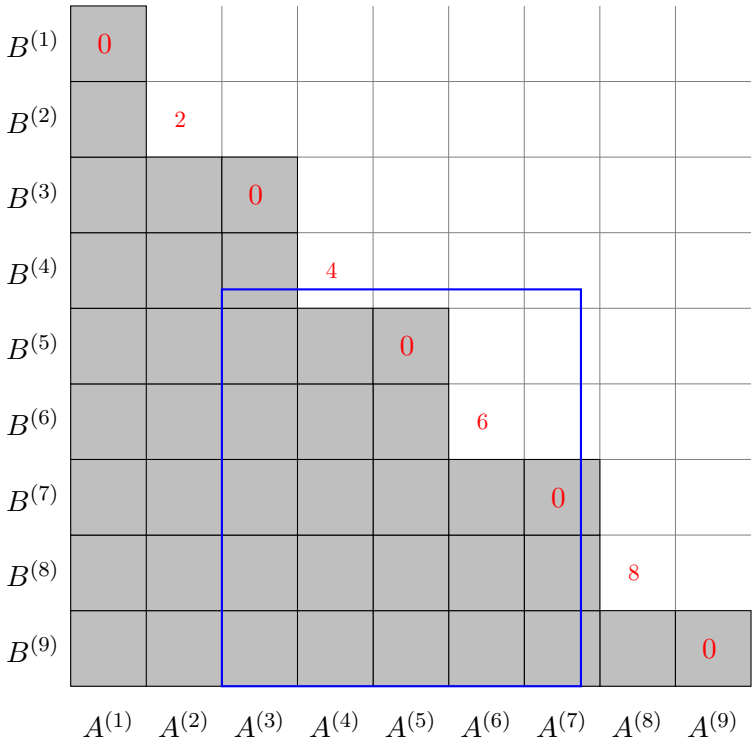

Figure 2: A tuple $(A, B, [u])$ constructed in the proof of Theorem 2.12. In gray we mark all the pairs (i, j) such that $A^{(i)} + B^{(j)}$ is contained in $[u]$. On the diagonal only every second pair belongs to the output. The numbers along the diagonal indicate the shift $E(i)$. The five boxes marked as "0" contribute to $(A + B) \cap [u]$ exactly the same numbers. The rectangles marked with $2, 4, 6$, and 8 have disjoint sumsets. By definition, a covering algorithm should cover all the boxes marked with 0. The indicated blue rectangle covers numbers from two boxes marked by 0, and thus also fully contains the box corresponding to pair $(A^{(6)}, B^{(6)})$ in between, which does not belong to the output.

Definition 9.1 ((α,ζ)-effective algorithm)**.** *Let sets $A, B \subseteq [u]$ with $|A| = n, |B| = m$. An (α,ζ)-effective algorithm is an algorithm for prefix-restricted sumset computation algorithm on instance $(A, B, [u])$ which runs in time*

$$\widetilde{O}(m + n + |(A + B) \cap [(1 + \zeta)u]|^{1+\alpha}).$$

The reduction is based on randomly dividing the input and conquering with a prefix-restricted sumset computation. In order to prove correctness, apart from the argumentation in [14], we additionally we need number-theoretic condition on how the set of attainable subset sums decreases under partition; this is captured in Lemma 9.3. The base cases of the algorithm are those instances where either there is only one element or every element is sufficiently large with respect to the target.

9.1 Handling Large Elements

In this subsection we treat one of the base instances of the more general algorithm, which are the instances where all elements are large with respect to the target. For a small technical reason in later subsections (in particular, in order to afford to take a union bound over all recursive calls), we need to define small numbers with respect to two parameters u, t. We shall analyze Algorithm 4. We classify an element as "heavy" if it is larger than $\frac{u}{\beta \log^3 t}$.

Algorithm 4

procedure SUBSETSUMFORLARGEELEMENTS(X, u, t, δ) ▷ $x \in \left[\frac{u}{\beta \log^3 t}, u\right], \forall x \in X; \beta = \Theta(1)$
  $O \leftarrow \emptyset$
  $R \leftarrow \Theta(\log(t/\delta))$
  $B \leftarrow 2\beta^2 \log^6 t$
  for $r \in [R]$ **do**
    Color X randomly with B different colors.
    for $b \in [B]$ **do**
      $X^{(r,b)} \leftarrow$ (elements of X which have received color b)$\bigcup\{0\}$.
    $O^{(r)} \leftarrow \emptyset$
    for $b \in [B]$ **do**
      $O^{(r)} \leftarrow \left(O^{(r)} + X^{(r,b)}\right) \bigcap [u]$ ▷ Oracle call with failure probability $\frac{\delta}{2BR}$
    $O \leftarrow O \cup O^{(r)}$
  Return O

The instance consisting solely of large items is much easier to solve, since only a polylogarithmic number of elements can participate in a subset sum which is at most t. Our algorithm in the next subsection will be recursive, and the next lemma will serve as one of the the bases of the recursion.

Lemma 9.2 (Guarantee for large elements)**.** *Suppose there exists an (α,ζ)-effective algorithm for solving (A, B, u) prefix-resticted sumset computation with probability $9/10$. Let parameters $u \leq t$. Let $X \subseteq \left[\frac{u}{\beta \log^3 t}, u\right]$ a set of positive integers, where β is a sufficiently large absolute constant. We can find $\mathcal{S}(X, u)$ using Algorithm 4 in time*

$$\widetilde{O}\left(|\mathcal{S}(X, u + \zeta \cdot u)|^{1+\alpha}\right),$$

with probability $1 - \delta$. We remind the reader that $\widetilde{O}(\cdot)$ hides factors also in $\log t$.

Algorithm 5

1: **procedure** SUBSETSUMREDUCTION(X, u, t) ▷ $u \leq t$
2: $X^{(S)} \leftarrow X \cap \left[\frac{u}{\beta \log^3 t}\right]$ ▷ Partition X to small and large element
3: $O \leftarrow$ SUBSETSUMFORLARGEELEMENTS$(X \setminus X^{(S)}, u, t, 1/\operatorname{poly}(u))$
4: $X^{(1)} \leftarrow$ Sample $X^{(S)}$ at rate $1/2$
5: $X^{(2)} \leftarrow X^{(S)} \setminus X^{(1)}$
6: $\epsilon \leftarrow \frac{1}{\log t}$
7: $O_1 \leftarrow$ SUBSETSUMREDUCTION $\left(X^{(1)}, (1+\epsilon)\frac{u}{2}, t\right)$
8: $O_2 \leftarrow$ SUBSETSUMREDUCTION $\left(X^{(2)}, (1+\epsilon)\frac{u}{2}, t\right)$
9: Return $(O + (O_1 + O_2) \cap [u]) \cap [u]$ ▷ Oracle call with failure probability $1/\operatorname{poly}(t)$

Proof. First of all, to accommodate the call in line 11 we note that any (a, ζ)-effective algorithm with success probability $9/10$ can be turned to an (a, ζ)-effective algorithm with success probability $1 - \delta$ by running in parallel $\Theta(\log(1/\delta))$ independent copies and stopping when a $7/10$ fraction of them has halted. Then at least half of them will have returned the same answer, and we can return this. Fix $x \in \mathcal{S}(X, t)$ and let $I \subseteq X$ such that $\Sigma(I) = x$. Since every element of X is at least $u/(\beta \log^3 t)$, it should be the case that $|I| \leq \beta \log^3 t$. For a fixed $r \in [R]$, the probability that all elements in I are colored with a different color is at least $\frac{1}{2}$ by standard arguments. Conditioning on the latter event happening, the computation in Lines 10, 11 will contain x with probability, by the same argument as the one appearing in [14, Lemma 3.1]. In particular, $x \in X^{(r,1)} + \ldots + X^{(r,B)}$, by the fact that every element of I belongs to a distinct set $X^{(r,b)}$, and those sets contain $0 \in X^{(r,b)}$ for all b.

All calls to prefix-restricted sumset computation will succeed with probability $\delta/2$ by a union bound. Repeating $R = \Theta(\log(u/\delta))$ times ensures that the coloring in line 6 will color every element of I with a different color, i.e. for some r it hold that $x \in X^{(r,1)} + \ldots + X^{(r,B)}$ except with probability $\left(\frac{1}{2}\right)^R = \delta/(2u)$. In turn, this implies that x will be found with probability $1 - \frac{\delta}{2u}$. Taking a union-bound over the at most u possible values of x we obtain that O shall contain every $x \in S(X, u)$ with probability $1 - \left(\frac{\delta}{2} + \frac{\delta}{2}\right) = 1 - \delta$. The running time of the algorithm follows by the fact that each computation involved is of the form $(A + B) \cap [u]$ with $A, B \in \mathcal{S}(X, u)$, and there are $R \cdot B = O(\log^6 t \cdot \log(u/\delta))$ such computations. This finishes the proof. □

9.2 General Algorithm

The reduction is presented in Algorithm 5. As mentioned in the overview, the algorithm partitions the set X to small elements (set $X^{(S)}$) and large elements (set $X \setminus X^{(S)}$). The large elements are handled by the algorithm presented in the previous subsection, Line 3. Next, $X^{(S)}$ is split to $X^{(1)}, X^{(2)}$ randomly in Line 4, and recursion takes places in each part with the appropriate change of the target (Lines 7,8).

9.3 A Number-Theoretic Lemma for the decrease of Subset Sums

The crux of the analysis is the following technical lemma, which postulates that the number of subsets sums of a set decreases in a suitable way when halving the set. This will allows us to control the running time of Algorithm 5. Note that this holds *for any* partitioning, not only for a random one.

Lemma 9.3 (Number of subset sums decreases appropriately)**.** *Let $Z \subseteq [u]$ be partitioned into $Z = Z^{(1)} \cup Z^{(2)}$. Set $\mu := \max(Z)/u$ and assume $\mu \leq \frac{1}{16}$. Then for any $0 \leq \epsilon \leq \frac{1}{4}$ we have*

$$\left|\mathcal{S}\left(Z^{(1)}, (1+\epsilon)\frac{u}{2}\right)\right| + \left|\mathcal{S}\left(Z^{(2)}, (1+\epsilon)\frac{u}{2}\right)\right| \leq \frac{|\mathcal{S}(Z,u)|+1}{1-2\epsilon-4\mu}$$

Proof. We denote

$$\begin{aligned}
A &:= \mathcal{S}\left(Z^{(1)}, (1+\epsilon)\frac{u}{2}\right), \\
B &:= \mathcal{S}\left(Z^{(2)}, (1+\epsilon)\frac{u}{2}\right), \\
C &:= \mathcal{S}(Z,u), \\
A' &:= A \cap \left((1-\epsilon-2\mu)\frac{u}{2}, (1+\epsilon)\frac{u}{2}\right].
\end{aligned}$$

With this notation, our goal is to show

$$|C| \geq (|A|+|B|)(1-2\epsilon-4\mu) - 1. \tag{7}$$

By symmetry, without loss of generality we can assume $\max(A) \leq \max(B)$.

It remains to consider the case $\max(A), \max(B) > (1-\epsilon)\frac{u}{2}$.

The statement is trivial if $\max(A) \leq (1-\epsilon)\frac{u}{2}$ (or $\max(B) \leq (1-\epsilon)\frac{u}{2}$), since then for $x \in B \setminus \{0\}$ we can map $I_Y(x)$ to $I_Y(x) \cup W$, yielding the following: $\Sigma(I_Y(x) \cup W) = x + \Sigma(W) > \max(A)$, obtaining $|B|-1$ distinct subset sums above $\max(A)$, from which we conclude that $|C| \geq |A|+|B|-1$.

Let us now assume that this is not the case, and thus $\max(A) \geq (1-\epsilon)\frac{u}{2}$ and $\max(B) \geq (1-\epsilon)\frac{u}{2}$. We shall need the following two claims.

Claim 9.4. $|C| \geq |A| + |B| - |A'| - 1$

Proof. Note that

$$\left|C \cap \left[0, (1-\epsilon-2\mu)\frac{u}{2}\right]\right| \geq \left|A \cap \left[0, (1-\epsilon-2\mu)\frac{u}{2}\right]\right| = |A| - |A'|$$

Moreover, since all items are bounded by μu, we can choose $P \subseteq W$ such that

$$\Sigma(P) \in \left[(1-\epsilon-2\mu)\frac{u}{2}, (1-\epsilon)\frac{u}{2}\right].$$

To see that, let any ordering of elements of W, initialize P to the empty set, and start adding elements to it one by one; clearly at some point $\Sigma(P)$ will fall inside the aforementioned interval. Now, for every $x \in B \setminus \{0\}$, map $I_Y(x)$ to $I_Y(x) \cup P$ to obtain number $\Sigma(I_Y(x) \cup P) = x + \Sigma(P)$; this yields $|B|-1$ different sums, all in the interval $\left[(1-\epsilon-2\mu)\frac{u}{2}+1, u\right]$, and thus disjoint from the numbers in A counted above. Thus, we obtain at least $(|A| - |A'|) + (|B|-1)$ different numbers in C, yielding the proof of the claim.

□

Claim 9.5. $|C| \geq |A| + |A'|\left(\frac{1}{2\epsilon+4\mu} - 1\right)$.

Proof. In order to prove the desired lower bound, we shall look at two disjoint intervals $\left[0,(1+\epsilon)\frac{u}{2}\right]$, and $\left[(1+\epsilon)\frac{u}{2}+1,t\right]$.

In the interval $\left[0,(1+\epsilon)\frac{u}{2}\right]$ we shall simply count the numbers in A, $\left|A\cap\left[0,(1+\epsilon)\frac{u}{2}\right]\right|\leq|A|$.

For the other interval we argue as follows. There exists a sequence of sets

$$P_{i_0}\subseteq P_{i_0+1}\subseteq P_{i_0+2}\subseteq\ldots\subseteq Y$$

satisfying

$$\Sigma(P_i)\in\left[i(\epsilon+2\mu)u+1,i(\epsilon+2\mu)u+\mu u)\right],$$

for all $i\geq 1$ such that

$$i(\epsilon+2\mu)u+\mu u\leq\max(B). \tag{8}$$

Note that i_0 is the smallest non-zero i such that the above inequality holds. Call such i good.

To see the existence of such a sequence, intialize P to the empty set and starting adding elements of Y one by one. Since every element is at most μu and the $[i(\epsilon+2\mu)u+1,i(\epsilon+2\mu)u+\mu u)]$ is of length μ we obtain that existence of such a sequence.

Since $\max(B)\geq(1-\epsilon)\frac{u}{2}$ all i smaller than

$$\begin{aligned}\frac{(1-\epsilon)\frac{u}{2}-\mu u}{(\epsilon+2\mu)u}&=\frac{1-\epsilon-2\mu}{2(\epsilon+2\mu)}\\&=\frac{1}{2\epsilon+4\mu}-\frac{1}{2}.\end{aligned}$$

are good.

For any $x\in A'\subseteq\mathcal{S}(X,(1+\epsilon)\frac{u}{2})$ map $I_W(x)$ to $I_W(x)\cup P_i$, to obtain $|A'|$ different numbers in the interval

$$[i(\epsilon+2\mu)u+1,i(\epsilon+2\mu)u+\mu u)]+\left[(1-\epsilon-2\mu)\frac{u}{2},(1+\epsilon)\frac{u}{2}\right]=$$

$$\left[i(\epsilon+2\mu)u+(1-\epsilon-2\mu)\frac{u}{2}+1,(i+1)(\epsilon+2\mu)u+(1-\epsilon-2\mu)\frac{u}{2}\right].$$

The collection of those intervals across all i are pairwise disjoint as well as disjoint from the initial interval $\left[0,(1+\epsilon)\frac{u}{2}\right]$.

In order for all generated sums to be at most u, we also need $(i+1)(\epsilon+2\mu)u+(1-\epsilon-2\mu)\frac{u}{2}\leq u$, which boils down to $i+1\leq\frac{1+\epsilon+2\mu}{2\epsilon+4\mu}=\frac{1}{2}+\frac{1}{2\epsilon+4\mu}$. Since i is an integer, we have at least $\frac{1}{2\epsilon+4\mu}-1$ valid i's. Hence, we obtain

$$|C|\geq|A|+|A'|\left(\frac{1}{2\epsilon+4\mu}-1\right).$$

□

To finish the proof of the lemma we combine the two claims, by considering two cases:

- Case 1: $|A'|\leq(2\epsilon+4\mu)|B|$.

  Then Claim 9.3 yields

$$|C|\geq|A|+|B|-|A'|-1\geq|A|+|B|(1-2\epsilon-4\mu)-1\geq(|A|+|B|)(1-2\epsilon-4\mu)-1.$$

- Case 2: $|A'| \geq (2\epsilon + 4\mu) \cdot |B|$

  Then Claim 9.3 yields

$$|C| \geq |A| + |A|' \left(\frac{1}{2\epsilon + 4\mu} \right) \geq$$
$$|A| + |B|(2\epsilon + 4\mu) \left(\frac{1}{2\epsilon + 4\mu} \right) \geq$$
$$(|A| + |B|)(1 - 2\epsilon - 4\mu).$$

□

9.4 Putting Everything Together

We finish the reduction and the proofs of Theorems 2.13 and 2.11. The algorithms is one call to SubsetSumReduction(S, t, t).

Proof of correctness. We show that SubsetSumReduction(X, t, t) returns $\mathcal{S}(X, t)$ with constant probability. This will require that all recursive calls succeed. In particular, for every call in the

Claim 9.6. *Consider the execution of* SubsetSumReduction(X, u, t). *Fix $x \in \mathcal{S}(X, u)$ and set $I = X^{(S)} \cap I_X(x)$, i.e. the "part" of the representation x which is formed by small elements. It holds that*

$$\mathbb{P}\left\{ \Sigma(I \cap X^{(1)}) \notin \left[(1+\epsilon)\frac{u}{2} \right] \right\} \leq \frac{1}{\text{poly}(t)}.$$

In words, the sum of elements of I which belong to $X^{(1)}$ will be at most $(1+\epsilon)\frac{u}{2}$ with high probability. The analogous statement holds with $X^{(1)}$ replaced with $X^{(2)}$.

Proof. This claim easily follows by concentration bounds for bounded random variables. We shall use Bernstein's inequality which postulates that for a collection $\mathcal{C}$ of random variables $\{Z_e\}_{e \in \mathcal{C}}$ such that all $Z_e \in [0, K]$, it holds that

$$\mathbb{P}\left\{ \left| \sum_e Z_e - \mathbb{E} \sum_e Z_e \right| \geq \lambda \right\} \leq e^{-\frac{c\lambda}{K}} + e^{-\frac{c\lambda^2}{\sigma^2}},$$

where $\sigma^2 = \sum_e \mathbb{E}\left\{ (Z_e - \mathbb{E}Z_e)^2 \right\}, \lambda \geq 0$ and c is some absolute constant.

We apply the inequality the collection $C := I$ of independent random variables $\{Z_e\}_{e \in I} = \{e \cdot \mathbb{1}_{X^{(1)}}(e)\}_{e \in I}$ and $\lambda = \epsilon\frac{u}{2}$. In words, Z_e is e with probability $1/2$, and 0 otherwise. We have

1. $\mathbb{E}\left[\sum_e Z_e\right] \leq \frac{x}{2} \leq \frac{u}{2}$.
2. $K = \frac{u}{\beta \cdot \log^3 t}$ by definition of $X^{(S)}$, and
3. $\sigma^2 \leq \frac{Kx}{2}$ since

$$\sigma^2 \leq \sum_{e \in I} e^2 \cdot \mathbb{E}\left\{ \mathbb{1}_{X^{(1)}}(e) \right\} \leq K \sum_{e \in I} e \cdot \mathbb{E}\left\{ \mathbb{1}_{X^{(1)}}(e) \right\} = \frac{1}{2} K \sum_{e \in I} \frac{e}{2} \leq \frac{Kx}{2}.$$

We thus obtain

$$\mathbb{P}\left\{\Sigma(I\cap X^{(1)})\geq\epsilon\frac{u}{2}\right\}\leq e^{-\frac{c\beta\cdot\epsilon\log^3 t}{4}}+e^{-\frac{c\beta\cdot\epsilon^2u^2\cdot\log^3 t}{2ux}}\leq 1/\operatorname{poly}(t),$$

as long as β is sufficiently large compared to c. This finishes the proof of the claim.

□

Equipped with the above claim, we can now prove correctness of the reduction. Fix $x\in\mathcal{S}(X,t)$ and let $I=I_X(x)$. Let also $I_0=I\cap(X\setminus X^{(S)}), I_1=I\cap X^{(1)}, I_2=I\cap X^{(2)}$. Set also $x=y+z+w$, where $y=\Sigma(I_0), z=\Sigma(I_1), w=\Sigma(I_2)$. It holds that y will be returned by Algorithm 5 with probability $1-1/\operatorname{poly}(t)$ due to the call in Line 3. If the conclusion of Claim 9.6 holds for I, then this means that both $\Sigma(I\cap X^{(1)})$ and $\Sigma(I\cap X^{(2)})$ are at most $(1+\epsilon)\frac{u}{2}$ and hence $z\in\mathcal{S}(X^{(1)},(1+\epsilon)\frac{u}{2})$, $w\in\mathcal{S}(X^{(2)},(1+\epsilon)\frac{u}{2})$. Thus, if the recursive calls in Lines 7 and 8 as well as the call 3 succeed, then $x=y+z+w$ will be inserted to the output. This means that the correctness of the algorithm is guaranteed on the conclusion of Claim 9.6 holding in all recursive calls and on every call to SubsetSumforLargeElements being correct. Since in each recursive call t does not change and remains the same, each recursive splitting succeeds with probability $1-1/\operatorname{poly}(t)$, while every call to SubsetSumforLargeElements succeeds also with probability $1-1/\operatorname{poly}(t)$. This allows for a union-bound over all splitting and all calls to SubsetSumforLargeElements.

Proof of Desired Running Time. Due to the splitting lemma 9.3, the fact that $\epsilon=\frac{1}{\log t}\leq\frac{1}{\log u}\leq\frac{1}{4}$ at all times, and a straightforward induction, we have that the total output size for all problems in the ℓth level of the recursion tree during the execution of SubsetSumReduction(S,u,t) is upper bounded by

$$\frac{|\mathcal{S}(X,t)|}{(1-2\epsilon-2\mu)^\ell}=\widetilde{O}(|\mathcal{S}(X,t)|),$$

since $\ell\leq\log n\leq\log t$ and $\epsilon=\frac{1}{\log t},\mu=\frac{1}{\beta\log^3 t}$. Thus, over all recursion levels the total output size is still $\widetilde{O}(|\mathcal{S}(X,t)|)$. This shows that if we had the ideal $(0,0)$-effective algorithm, we would obtain a SubsetSum running in time $\widetilde{O}(\mathcal{S}(X,t))$. A similar analysis yields the desired reduction when we plug in a (α,ζ)-effective algorithm.

Obtaining Theorem 2.11. We plug in the $\widetilde{O}(\text{out}^{4/3})$-time algorithm for prefix-restricted sumset computation (Theorem 2.9).

Obtaining Theorem 2.13. We plug in the algorithm guaranteed by the first part of Theorem 2.13 and Observation 2.8.

10 Acknowledgements

We are grateful to Shachar Lovett for the resolution of an Additive Combinatorics question in an early stage of this work, which gave the core idea for Theorem 2.12, and for allowing us to include his construction in this paper.

11 Conclusion and Future Work

We initiated a line of research which strives for a SubsetSum algorithm that computes the set $\mathcal{S}(X,t)$, consisting of all subsets sums of X below t, in near-linear output-sensitive time $\widetilde{O}(|\mathcal{S}(X,t)|)$. Our approach lead us to studying a new type of convolution problem: In top-k-convolution the task is to compute the lowest k monomials in the product of two sparse polynomials. Many open problems are spawned by our work; here we present questions that are of particular interest to us.

Question 11.1. *Understand our notion of* covering *for prefix-restricted sumset computation, either by constructing a better covering or by proving a higher lower bound. Specifically, for non-rectangular coverings so far we have no superlinear lower bounds.*

Question 11.2. *Design any non-trivial algorithm that is not based on coverings, and thus exploits the additive structure in a different way.*

Question 11.3. *Are covering algorithms universal? More precisely, can we transform any algorithm into a covering algorithm, with a reasonable blow-up in the running time?*

Question 11.4. *So far we have no algorithm that always beats Bellman's algorithm with running time $O(n \cdot |\mathcal{S}(X,t)|)$. Specifically, can we solve* SubsetSum *in time $O(n^{1-\epsilon} \cdot |\mathcal{S}(X,t)|)$ for any $\epsilon > 0$? A possible approach to this question is to design faster algorithms for prefix-restricted sumset computation in the special situation where $A = \mathcal{S}(X^{(1)},t), B = \mathcal{S}(X^{(2)},t)$. Do these sets offer exploitable structure, e.g., giving rise to more sophisticated sumset estimates?*

References

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019.

[2] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

[3] Josh Alman. Limits on the universal method for matrix multiplication. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA.*, pages 12:1–12:24, 2019.

[4] Josh Alman and Virginia Vassilevska Williams. Further limitations of the known approaches for matrix multiplication. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 25:1–25:15, 2018.

[5] Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 580–591, 2018.

[6] Andrew Arnold and Daniel S Roche. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 29–36. ACM, 2015.

[7] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space–time tradeoffs for Subset Sum: An improved worst case algorithm. In *Proc. of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 45–56, 2013.

[8] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset Sum in the absence of concentration. In *Proc. of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 48–61, 2015.

[9] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense Subset Sum may be the hardest. In *Proc. of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 13:1–13:14, 2016.

[10] Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In *Proc. of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 58–69. SIAM, 2019.

[11] Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for Subset Sum, k-Sum and related problems. In *Proc. of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 198–209, 2017.

[12] Richard E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[13] Ernest F. Brickell and Andrew M. Odlyzko. Cryptanalysis: A survey of recent results. *Proceedings of the IEEE*, 76(5):578–593, 1988.

[14] Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *Proc. of of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.

[15] Karl Bringmann and Vasileios Nakos. Fast n-fold boolean convolution via additive combinatorics. *Under submission*, 2019.

[16] Boris Bukh. Walk through combinatorics:sumset inequalities. 2018.

[17] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proc. of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015.

[18] Benny Chor and Ronald R. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

[19] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proc. of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601. ACM, 2002.

[20] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41, 2016.

[21] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *Proc. of the 32nd Annual Conference on Advances in Cryptology (CRYPTO)*, pages 719–740, 2012.

[22] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

[23] Pawel Gawrychowski and Przemyslaw Uznanski. Optimal trade-offs for pattern matching with k mismatches. *arXiv preprint abs/1704.01311*, 2017.

[24] Omer Gold and Micha Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *Proc. of the 25th Annual European Symposium on Algorithms (ESA)*, pages 42:1–42:13, 2017.

[25] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974.

[26] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996.

[27] Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, volume 69 of *OASICS*, pages 17:1–17:6, 2019.

[28] Haim Kaplan, László Kozma, Or Zamir, and Uri Zwick. Selection from heaps, row-sorted matrices, and X+Y using soft heaps. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, volume 69 of *OASICS*, pages 5:1–5:21, 2019.

[29] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[30] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[31] Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for Subset Sum. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017.

[32] Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic time-space trade-offs for k-SUM. In *Proc. of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 58:1–58:14, 2016.

[33] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proc. of the 27th 2nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 777–789, 2011.

[34] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[35] Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[36] Michael Monagan and Roman Pearce. Parallel sparse polynomial multiplication using heaps. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pages 263–270. ACM, 2009.

[37] Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *arXiv preprint arXiv:1901.09355*, 2019.

[38] Jesper Nederlof. A short note on Merlin-Arthur protocols for subset sum. *Information Processing Letters*, 118:15–16, 2017.

[39] Kevin O'Bryant. Sets of integers that do not contain long arithmetic progressions. *Electronic Journal of Combinatorics*, 18(1):P59, 2011.

[40] Andrew M. Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and Computational Number Theory*, 42:75–88, 1990.

[41] Daniel S. Roche. Adaptive polynomial multiplication. *Proc. Milestones in Computer Algebra (MICA'08)*, pages 65–72, 2008.

[42] Daniel S. Roche. What can (and can't) we do with sparse polynomials? *arXiv preprint arXiv:1807.08289*, 2018.

[43] Adi Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Transactions on Information Theory*, 30(5):699–704, 1984.

[44] Xuancheng Shao and Wenqiang Xu. A robust version of Freiman's 3k–4 theorem and applications. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 166, pages 567–581. Cambridge University Press, 2019.

[45] Terence Tao and Van H. Vu. *Additive combinatorics*, volume 105. Cambridge University Press, 2006.

[46] Joris Van Der Hoeven and Grégoire Lecerf. On the complexity of multivariate blockwise polynomial multiplication. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 211–218. ACM, 2012.

[47] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. of the 44th Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.