Parameterized Algorithms on Integer Sets with Small Doubling: Integer Programming, Subset Sum and k-SUM

Tim Randolph
Harvey Mudd College, Claremont, CA, USA

Karol Węgrzycki
Saarland University and Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

We study the parameterized complexity of algorithmic problems whose input is an integer set A in terms of the *doubling constant* $\mathcal{C} := |A+A|/|A|$, a fundamental measure of additive structure. We present evidence that this new parameterization is algorithmically useful in the form of new results for two difficult, well-studied problems: Integer Programming and Subset Sum.

First, we show that determining the feasibility of bounded Integer Programs is a tractable problem when parameterized in the doubling constant. Specifically, we prove that the feasibility of an integer program $\mathcal{I}$ with n polynomially-bounded variables and m constraints can be determined in time $n^{O_{\mathcal{C}}(1)} \cdot \mathsf{poly}(|\mathcal{I}|)$ when the column set of the constraint matrix has doubling constant $\mathcal{C}$.

Second, we show that the Subset Sum and Unbounded Subset Sum problems can be solved in time $n^{O_C(1)}$ and $n^{O_{\mathcal{C}}(\log\log\log n)}$, respectively, where the O_C notation hides functions that depend only on the doubling constant $\mathcal{C}$. We also show the equivalence of achieving an FPT algorithm for Subset Sum with bounded doubling and achieving a milestone result for the parameterized complexity of Box ILP. Finally, we design near-linear time algorithms for k-SUM as well as tight lower bounds for 4-SUM and nearly tight lower bounds for k-SUM, under the k-SUM conjecture.

Several of our results rely on a new proof that Freiman's Theorem, a central result in additive combinatorics, can be made efficiently constructive. This result may be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Parameterized algorithms, parameterized complexity, additive combinatorics, Subset Sum, integer programming, doubling constant

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.96

Related Version *Full Version*: `https://arxiv.org/abs/2407.18228`

Funding This work is a part of project TIPEA (Karol Węgrzycki) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. and 850979).

Acknowledgements The authors thank Lars Rohwedder for insightful discussions that helped to clarify the connections between $\mathcal{C}$-Subset Sum and Hyperplane-Constrained BILP, as well as several anonymous reviewers for helpful feedback on earlier drafts.

1 Introduction

Given a subset X of a group, the *doubling constant*

$$\mathcal{C} := \mathcal{C}(X) = \frac{|X+X|}{|X|}$$

is one measure used to capture the amount of "additive structure" in X. (Here, $X+X$ denotes the *sumset* $\{a+b \ : \ a,b \in X\}$.) This value ranges (on integer sets) from $2-o_n(1)$ for arithmetic progressions to $\frac{n}{2}+o_n(1)$ when all sums are distinct, and is central to the study of additive combinatorics. If the doubling constant $\mathcal{C}$ is truly constant (that is, independent of the set cardinality $|X|$), this indicates that X is a "highly structured" set with respect to addition: for example, the statements that

- X has constant doubling ($|X+X| \leq c_1|X|$), that
- the iterated sumset $sX := \underbrace{X+X+X\cdots+X}_{s\text{ times}}$ is at most $c_2 := c_2(s)$ times $|X|$, and that
- X (likewise $X+X$ and sX) can be contained in a *generalized arithmetic progression* of dimension c_3 and volume $c_4|X|$,

are all equivalent up to the choice of constants c_1, c_2, c_3, and c_4 (c.f. [34] Proposition 2.26). The many fruitful applications of the doubling constant illustrate its significance as a robust measurement of additive structure (for an overview, see [34], especially Chapter 2).

In this work, we consider the parameterized complexity of problems on integer sets with respect to the doubling constant. Specifically, we focus on two problems for which additive structure is particularly helpful: Integer Programming and Subset Sum.

1.1 Related Work

Integer Programming and Subset Sum are not only problems in which additive structure plays an important role: they are also both well-studied and stubbornly difficult, to the point where significant work has gone into analysing their parameterized complexity and demarcating classes of tractable instances.

Integer Linear Programming

Many problems in *combinatorial optimization* can be formulated as an *integer linear program* (ILP). An ILP is an optimization problem of the following form:

$$\max\left\{c^T x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0}\right\},$$

where $A \in \mathbb{Z}^{m\times n}$, $c \in \mathbb{Z}^n$ and $b \in \mathbb{Z}^m$. (ILPs of the form $Ax \geq b$ can be converted to this form using slack variables.) Unlike linear programming, integer programming is NP-complete. Due to its generality and both practical and theoretical importance, the complexity of ILP has been rigorously studied through the lens of *parameterized complexity.* Lenstra [27] has shown that an integer linear program with a fixed number of variables can be solved in polynomial time. His algorithm was subsequently improved, and the current record is $(\log n)^{O(n)}$ [32]. The question of whether this can be brought down to $2^{O(n)}$ is one of the most prominent open questions in the theory of algorithms.

ILP can also be parameterized in the number of constraints m and the maximum absolute value of any coefficient in the constraint matrix, $\Delta := \|A\|_\infty$. In 1981, Papadimitriou [30] presented an $(m\Delta)^{O(m^2)}$-time algorithm, and the best algorithms for ILP parameterized in m and Δ continue to improve: see [22, 17] for recent progress. Another class of tractable instances of ILP rely on structural properties of the constraint matrix (see [14, 24, 12, 13]).

Subset Sum

Along with the closely related Knapsack problem, the Subset Sum problem is the canonical NP-complete problem concerning addition in integer sets. In addition to NP-completeness, the problem appears difficult from the standpoint of exact algorithms: despite significant attention (see, e.g., [37, 4, 29]), solving Subset Sum in time $2^{(1/2-c)n}$ for some constant $c > 0$ remains a major open problem. Except for "log shaving" results that improve runtime by subexponential factors [11], the exact runtime has not been improved in 50 years [21]. The lack of progress in exact algorithms motivates parameterized approaches, including a long line of pseudopolynomial-time algorithms parameterized by the size of the target [6, 26, 2] and the largest input integer [17, 8, 31, 10].

However, these parameterized results do not take advantage of structural properties of the input when the input numbers are very large. Therefore, we complement the parameterization based on the input size by considering the parameterized complexity of Subset Sum with respect to the doubling constant. This choice is natural not only because the doubling constant is essential to the study of integer sets under addition, but also because existing results from additive combinatorics give strong bounds on the search space: Freiman's Theorem bounds the number of distinct subset sums of an n-element input set by $n^{f(\mathcal{C})}$, where f is a function that depends only on $\mathcal{C}$.

The parameterization of Subset Sum in the cardinality of the solution k, otherwise known as k-SUM, has an entire literature of its own. Simple "meet-in-the-middle" algorithms that run in time $O(n^{\lceil k/2 \rceil})$ are conjectured to be optimal up to polynomial factors. The results of Abboud, Bringmann, and Fischer, and of Jin and Xu, suggest that the hardest instances of k-SUM are those with very little additive structure, such as Sidon sets [1, 23]. Parameterizing k-SUM in the doubling constant allows us to make analogous conclusions for the more general case of k-SUM: we can now prove results of the form, "k-SUM instances with strong additive structure (i.e., small doubling constant) are easy".

Algorithms and Additive Combinatorics

This paper is also motivated by an emerging trend in fine-grained complexity and algorithms: "importing" results from additive combinatorics. In several recent works, researchers have achieved breakthroughs by taking existential results from the field of additive combinatorics and modifying their proofs to make them explicitly and efficiently constructive.

For example, in 2015 Chan and Lewenstein proved a version of the Balog-Szemeredi-Gowers (BSG) theorem that allows certain sets guaranteed by the theorem to be constructed algorithmically [9]. They then leveraged this result to solve the $(\min, +)$-convolution and 3-SUM problems on monotone sets of small integers. Recently, the constructive BSG theorem found new applications. In 2022, Abboud, Bringmann and Fischer used this result, as well as a constructive version of Ruzsa's covering lemma, as a key ingredient in their proofs of lower bounds for approximate distance oracles and listing 4-cycles [1]. In the same year, Jin and Xu independently proved similar lower bounds and used the constructive BSG theorem to reduce 3-SUM to 3-SUM on Sidon sets [23]. More broadly, these works reflect the increasing role of additive combinatorics in algorithms over the last few decades; for general references, see [35, 36, 5, 28].

1.2 Our Results

Contribution 1: *A Constructive Freiman's Theorem in Near-Linear FPT Time*

We begin by unlocking a new tool to help us manipulate sets with significant additive structure. *Freiman's Theorem*, a cornerstone result in additive combinatorics, states that every integer set with constant doubling is contained inside a small (generalized) arithmetic progression. Naively constructing this generalized arithmetic progression takes XP-time. We make the construction efficient by showing how an algorithm can obtain such an arithmetic progression in time $\widetilde{O}_{\mathcal{C}}(n)$[1] (Theorem 5). Later in the paper, we use this theorem to reduce Subset Sum with constant doubling to a constrained integer programming problem (Contribution 3) and to design efficient algorithms for Unbounded Subset Sum (Contribution 4). We hope that, like the constructive BSG theorem discussed above, the constructive statement of Freiman's Theorem may find other independent applications.

Contribution 2: *Integer Programming with Constant Doubling*

An integer program specified by a constraint matrix $A \in \mathbb{Z}^{m \times n}$ and solution vector $b \in \mathbb{Z}^m$ is *feasible* if there exists a solution $x \in \mathbb{Z}^n_{\geq 0}$ such that $Ax = b$. The ILP is *binary* if the variables are further restricted to $x \in \{0, 1\}^n$.

In our setting, we consider integer programs in which the set of column vectors $\mathcal{A} := \{A[\cdot, j] \mid j \in [n]\}$ has constant doubling: $|\mathcal{A} + \mathcal{A}| \leq \mathcal{C}|\mathcal{A}|$, for a constant $\mathcal{C}$. We prove:

▶ **Theorem 1.** *An instance $\mathcal{I}$ of $\mathcal{C}$-Binary ILP Feasibility on n variables can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \mathsf{poly}(|\mathcal{I}|)$.*[2]

This follows from Freiman's Theorem (without construction) and a dynamic programming algorithm. The theorem also holds when the variables $x_1, x_2, \ldots, x_n$ have upper and lower bounds of magnitude $\mathsf{poly}(n)$.

Contribution 3: *Subset Sum with Constant Doubling*

Our result for integer programming with constant doubling implies an $n^{O_{\mathcal{C}}(1)}$-algorithm for Subset Sum (Corollary 9).

Assuming the Exponential Time Hypothesis (ETH), there is no $2^{o(n)}$ time algorithm for Subset Sum. Because $\mathcal{C} = O(n)$, this means that we cannot hope for a $2^{o(\mathcal{C})} n^{o(\mathcal{C}/\log(\mathcal{C}))}$ algorithm for $\mathcal{C}$-Subset Sum under the ETH. However, this lower bound does not exclude an $2^{O(\mathcal{C})} \cdot n^{O(1)}$ algorithm. A natural question is thus whether our upper bound can be improved to an Fixed-Parameter Tractable (FPT) result: can $\mathcal{C}$-Subset Sum be solved in time $O_{\mathcal{C}}(\mathsf{poly}(n))$? We show that this result appears unlikely by way of an interesting connection to the feasibility of integer programs with binary variables.

▶ **Theorem 2.** *It is possible to solve $\mathcal{C}$-Subset Sum in time $O_{\mathcal{C}}(\mathsf{poly}(n))$ if and only if* Hyperplane-Constrained Binary ILP *(HBILP) can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(|\mathcal{I}|))$, where $|\mathcal{I}|$ is the size of the instance.*

HBILP considers a constraint matrix $A \in \mathbb{Z}^{m \times n}$ with entries bounded by $\Delta := \|A\|_\infty$, and asks whether there exists a solution $x \in \{0, 1\}^n$ such that $\langle Ax, s \rangle = t$ for a certain target t and "step vector" s orthogonal to a hyperplane. The best existing algorithm solves HBILP Feasibility in time $O_m(|\mathcal{I}|) + \Delta^{O(m^2)}$ ([15], Corollary 1[3]).

We also prove a reduction from ILP Feasibility with bounded variables to HBILP feasibility (Lemma 17). Thus Theorem 2 implies that an FPT algorithm for Subset Sum with constant doubling would imply a $\Delta^{O(m)} \cdot \mathsf{poly}(n)$ algorithm for ILP Feasibility with bounded variables

[1] We use $O_{\mathcal{C}}$ notation to indicate the suppression of terms that depend only on $\mathcal{C}$. For example, $O_{\mathcal{C}}(n^2) = f(\mathcal{C}) \cdot O(n^2)$ for some computable function f. $\widetilde{O}$ hides factors polylogarithmic in the argument, in this case $\log(n)$.
[2] We write $|\mathcal{I}|$ to denote the size of the ILP instance $\mathcal{I}$. In the word RAM model (see Section 2), this is $\mathsf{poly}(m, n)$.
[3] Corollary 2 in the arXiv preprint, 2303.02474.

(Corollary 16). As previously noted in [15], reducing the exponent of Δ from $O(m^2)$ to $O(m)$ would be analogous to the recent improvement achieved by Eisenbrand and Weismantel for integer programs with *unbounded* variables [17].

Such an algorithm for ILP Feasibility would resolve the feasibility portion of one of the most significant open questions in the parameterized complexity of integer programming: whether the $(\Delta^{O(m)} \cdot O_m(|\mathcal{I}|))$-time algorithm for ILPs with unbounded variables can be extended to ILPs with bounded variables [17, 22, 25]. This would be a breakthrough in the area [22, 25]; accordingly, finding an FPT algorithm for $\mathcal{C}$-Subset Sum is at least as difficult.

Contribution 4: *Unbounded Subset Sum with Constant Doubling*

We can reduce an instance of Unbounded Subset Sum with constant doubling to an ILP with m constraints, n binary variables, and entries of A bounded by $\Delta = n^{O(1/d(\mathcal{C}))}$ using our constructive Freiman's theorem. Because solvable ILPs with bounded Δ admit solutions with small support, this allows us to solve Unbounded Subset Sum in time $n^{O_\mathcal{C}(\log\log\log n)}$, or $n^{O_\mathcal{C}(1)}$ under the hypothesis that a v-variable ILP $\mathcal{I}$ can be solved in time $2^{O(v)}\mathsf{poly}(\mathcal{I})$ (Theorem 21).

Contribution 5: *k-SUM with Constant Doubling*

The application of recent algorithms for sparse nonnegative convolution [7] allow us to efficiently solve k-SUM with constant doubling in time $\widetilde{O}(\mathcal{C}^{\lceil k/2 \rceil} \cdot 2^{O(k)} \cdot n)$ (see Theorem 23).

Because the k-SUM conjecture implies a lower bound of $\Omega(\mathcal{C}^{\lceil k/2 \rceil - 1} n)$, this leaves a $\mathcal{C}$-factor gap. Part of the gap can be explained by the fact that the Plünnecke-Ruzsa inequality, which we use to derive the upper bound, does not give the optimal exponent for $\mathcal{C}$; applying recent improvements to the inequality narrows the gap slightly. In the specific case of $(\mathcal{C}, 4)$-SUM, our algorithm achieves a runtime of $\widetilde{O}(\mathcal{C}n)$, which is optimal up to polylogarithmic factors under the k-SUM conjecture.

1.3 Organization

We begin with mathematical preliminaries in Section 2 In Section 4, we present our algorithms for ILP feasibility with bounded doubling. Finally, we present our bounds for Subset Sum in Section 5, Unbounded Subset Sum in Section 6, and k-SUM in Section 7.

2 Preliminaries

RAM Model. Throughout the paper, we use the standard *word RAM* model, in which input integers fit into a single machine word and logical and arithmetic operations on machine words take time $O(1)$. If we make the weaker assumption that operations on b-bit words take $\mathsf{polylog}(b)$ time, this adds a $\mathsf{polylog}(b)$ factor to Theorem 5 and the results that rely on it.

Big-O Notation. We use $O_\mathcal{C}$ notation to indicate we have suppressed terms that depend only on $\mathcal{C}$. For example, $O_\mathcal{C}(n^2) = f(\mathcal{C}) \cdot O(n^2)$ for some computable function f. $\widetilde{O}$ notation suppresses polylogarithmic factors of n and Δ: for instance, $n \log^2(n) = \widetilde{O}(n)$.

Sets. We write $[n]$ for the integer set $\{1, 2, \dots, n\}$ and $[a : b]$ (with $a \leq b$) for the integer set $[a, a+1, a+2, \dots, b]$. The *diameter* of an integer set A, denoted $\operatorname{diam}(A)$, is $\max_{a,b \in A} |a - b|$. We write $\Sigma(X)$ as shorthand for the sum of elements $\sum_{x \in X} x$, and $\Sigma(2^X)$ as shorthand for the set of subset sums $\{\Sigma(X') \ : \ X' \subseteq X\}$.

Vectors. Given a vector $x \in \mathbb{Z}^n$, $\text{supp}(x) \subseteq [n]$ denotes the set of non-zero coordinates of x.

For $a, b \in \mathbb{Z}^n_{\geq 0}$ we say that a is *lexicographically prior* to b, denoted $a \prec_{\text{lex}} b$, if and only if there exists $k \in [n]$ such that $a[k] < b[k]$ and for every $1 \leq i < k$ it holds that $a_i = b_i$. Observe that $\prec_{\text{lex}}$ is a total order and that every set of vectors $S \subseteq \mathbb{Z}^n_{\geq 0}$ contains a unique element that is lexicographically minimal.

Matrices. Given a $m \times n$ matrix A, we write $A[i, j]$ to denote the component of A at row i, column j. We write $A[i, \cdot]$ and $A[\cdot, j]$ to denote the ith row and jth column of A, respectively.

We write $J_{m \times n}$ to denote the $m \times n$ matrix in which each entry is 1.

Group Theory and Linear Algebra. Given an integer m, we write $\mathbb{Z}_m$ to denote the cyclic group of order m (under addition). When p is prime, every element of $\mathbb{Z}_p$ is a generator except for 0.

A *lattice* in $\mathbb{R}^d$ is defined by d linearly independent vectors $v_1, v_2, \ldots, v_d \in \mathbb{R}^d$, collectively referred to as the *basis* of the lattice. The lattice itself is the set

$$\Lambda = \left\{ \sum_{i \in [d]} a_i v_i \;\middle|\; a_i \in \mathbb{Z} \right\}$$

of all integer linear combinations of $v_1, v_2, \ldots, v_d$. Each point in Λ is a *lattice vector.*

The *determinant* of a lattice, denoted $\det(\Lambda)$, is the determinant of the matrix whose columns are the lattice basis. Geometrically, $\det(\Lambda)$ is the volume of the *fundamental parallelepiped* spanned by the lattice basis. In general, if T is a convex body, we write $vol(T)$ to denote the volume of T.

Given two m-dimensional vectors x and y, $\langle x, y \rangle$ denotes the dot product $x_1 y_1 + \cdots + x_m y_m$.

Norms. Given a real number r, we write $\|r\|_{\mathbb{R}/\mathbb{Z}}$ to denote distance from the nearest integer. Given a finite-dimensional vector v, the norm $\|v\|_\infty$ denotes the largest absolute value of any coordinate.

Additive Combinatorics. Given an integer set X, $X + X$ denotes the *sumset* $\{a + b : a, b \in X\}$. We write sX, where s is a positive integer, as shorthand for the iterated sumset $\underbrace{X + \ldots + X}_{s \text{ times}}$.

A *generalized arithmetic progression* (GAP) P is an integer set

$$P = \{\ell_1 y_1 + \ell_2 y_2 + \cdots + \ell_d y_d \;:\; 0 \leq \ell_i < L_i, \forall i \in [d]\},$$

defined by the integer vector $y = \{y_1, y_2, \ldots, y_d\}$ and the dimension bounds $L_1, L_2, \ldots, L_d$. We say that P has *dimension* d and *volume* $\prod_{i \in [d]} L_i$. When we write that an algorithm "explicitly constructs" or "returns" P, we mean specifically that the algorithm computes y_i and L_i for all $i \in [d]$. We can think of P as a projection of a d-dimensional parallelepiped onto the line. P is *proper* if $|P| = L_1 L_2 \ldots L_d$, that is, if each point in the parallelepiped projects to a unique point on the line.

The Plünnecke-Ruzsa Inequality bounds the size of sumsets using the doubling constant.

▶ **Lemma 3** (Plünnecke-Ruzsa Inequality). *If X is a finite subset of an abelian group and $|X + X| \leq \mathcal{C}|X|$ for a constant $\mathcal{C}$, then for all nonnegative integers s and t, $|sX - tX| \leq \mathcal{C}^{s+t}|X|$.*

3 Freiman's Theorem Made Constructive in FPT Time

Freiman's Theorem states that any integer set X with constant doubling is contained inside a GAP of constant dimension and volume at most $|X|$ times a constant.

▶ **Theorem 4** (Freiman's Theorem, [19], see [38] for a modern presentation). *Any finite integer set X with $|X+X| \le \mathcal{C}|X|$ is contained in a GAP P of dimension $d(\mathcal{C})$ and volume $v(\mathcal{C})|X|$, where d and v are computable functions that depend only on $\mathcal{C}$.*

We make this statement constructive by showing an algorithm that, given X, can explicitly construct the progression P in FPT time. In fact, the construction is near-linear, losing only a polylog(n) factor and a (large) function of $\mathcal{C}$.

▶ **Theorem 5** (FPT Freiman's Theorem). *Let A be a set of n integers satisfying $|A+A| \le \mathcal{C}|A|$. There exists an $\widetilde{O}_{\mathcal{C}}(n)$ algorithm that, with probability $1-n^{-\gamma}$ for an arbitrarily large constant $\gamma > 0$, returns[4] an arithmetic progression*

$$P = \{x_1\ell_1 + x_2\ell_2 + \cdots + x_{d(\mathcal{C})}\ell_{d(\mathcal{C})} \ : \ \forall i, \ell_i \in [L_i]\} \supseteq A$$

with dimension $d(\mathcal{C})$ and volume $v(\mathcal{C}) \cdot |A|$, where d and v are computable functions that depend only on $\mathcal{C}$.[5]

The following observation further simplifies Theorem 5.

▶ **Observation 6.** *In the GAP P guaranteed by Theorem 5, without loss of generality we can assume $L_i \le n^{2/d(\mathcal{C})}$ for all $i \in [d(\mathcal{C})]$, where $d(\mathcal{C})$ denotes the dimension of P.*

We defer the proofs of Theorem 5 and Observation 6 to the full version of the paper.

4 Integer Programming with Constant Doubling

For an integer program, we consider the doubling constant of the *column set* of the constraint matrix A as our parameter. This is because the column is the smallest unit affected by each variable x_i when we compute the product Ax; as a result, duplicate columns in A play a similar role to duplicate elements in a Subset Sum instance, and indeed can often be eliminated without loss of generality. This formulation allows A to contain duplicate entries (for example, multiple 0's and 1's) as long as all columns are distinct.

Given a matrix A, we use the shorthand $\mathcal{A} := \mathcal{A}(A) = \{A[\cdot, j] \mid j \in [n]\}$ to denote the set of column vectors of A. Vector set addition (that is, $\mathcal{A}+\mathcal{A}$) is defined in the natural way, using vector instead of integer addition.

Problem 1: $\mathcal{C}$-Integer Linear Programming (ILP) Feasibility

In: An integer linear program specified by an integer matrix $A \in \mathbb{Z}^{m\times n}$ with n distinct columns and an integer target $b \in \mathbb{Z}^m$, such that the column set $\mathcal{A} := \mathcal{A}(A)$ satisfies $|\mathcal{A}+\mathcal{A}| \le \mathcal{C}|\mathcal{A}|$ for a constant $\mathcal{C}$ independent of m and n.
Out: Vector $x \in \mathbb{Z}^n_{\ge 0}$ such that $Ax = b$, or "NO" if no solution exists.

[4] Specifically, we compute the values $x_1, x_2, \ldots, x_{d(\mathcal{C})}$ and $L_1, L_2, \ldots, L_{d(\mathcal{C})}$.
[5] We make the standard assumption that arithmetic operations on integers require $O(1)$ time.

If each variable x_i is constrained to satisfy $x_i \in [\ell_i : u_i]$, where ℓ_i and u_i indicate the lower and upper bounds of a range of valid variable assignments, we refer to the problem as **$\mathcal{C}$-Bounded ILP Feasibility**. Further restricting the variables to $x \in \{0,1\}^n$ yields **$\mathcal{C}$-Binary ILP Feasibility**.

▶ Remark 7. Bounded ILPs with n variables and $|\ell_i|, |u_i| = O(\mathsf{poly}(n))$ for $i \in [n]$ can be converted into equivalent binary ILPs with $\mathsf{poly}(n)$ variables by duplicating columns of A.

4.1 $\mathcal{C}$-Binary ILP Feasibility

Given a constraint matrix with constant doubling, Freiman's Theorem bounds the number of possible values for Ax corresponding to any variable assignment if the variables are binary or bounded. This allows us to solve the problem efficiently via dynamic programming, and does not actually require constructing the GAP guaranteed by Freiman's Theorem.[6]

▶ **Theorem 1.** *An instance $\mathcal{I}$ of $\mathcal{C}$-Binary ILP Feasibility on n variables can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \mathsf{poly}(|\mathcal{I}|)$.*[7]

Proof. Fix an instance of $\mathcal{C}$-Binary ILP feasibility specified by $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$, with the column set $\mathcal{A}$ satisfying $|\mathcal{A}+\mathcal{A}| \le \mathcal{C}|\mathcal{A}|$.

Let $L := \Sigma(2^{\mathcal{A}})$ denote the list of all (vector) sums that can be attained by adding together any subset of the columns of A. Equivalently, this is the set of possible outputs Ax for any $x \in \{0,1\}^n$. Our first goal is to bound $|L|$.

First, we observe that there exists a GAP P of dimension $d(\mathcal{C})$ and volume $v(\mathcal{C})n$ with

$$\mathcal{A} \subseteq P = \{x_1k_1 + x_2k_2 + \cdots + x_{d(\mathcal{C})}k_{d(\mathcal{C})} \ : \ \forall i, k_i \in [K_i]\},$$

where $x_i \in \mathbb{Z}^m$ for all $i \in [d(\mathcal{C})]$. This is true even though $\mathcal{A}$ is a set of integer vectors, as Freiman's Theorem holds for torsion-free[8] commutative groups ([33], Theorem 8.1).

Thus L is contained in the GAP

$$P' = \{x_1k_1 + x_2k_2 + \cdots + x_{d(\mathcal{C})}k_{d(\mathcal{C})} \ : \ \forall i, k_i \in [n \cdot K_i]\},$$

which implies

$$|L| \le |P'| \le n^{d(\mathcal{C})}|P| = n^{d(\mathcal{C})}v(\mathcal{C})n = n^{O_{\mathcal{C}}(1)}. \tag{1}$$

To complete the proof, we claim that we can enumerate L efficiently via dynamic programming, using the following procedure: Initially, we set $L_1 = A[1,\cdot]$. Then, we iterate $i = 2, 3, \ldots, n$. In the ith iteration, we construct the sorted list L_i, defined as:

$$L_i := L_{i-1} \cup \{a + A[i,\cdot] \mid a \in L_{i-1}\}.$$

Finally, we return list $L = L_n$. Correctness of the above algorithm follows immediately by a construction. For the running time, observe that L_i can be constructed in $O(|L_i|)$ time. Because each of the n iterations of the subprocedure takes time $O(|L_i|) = O(|L|)$, the total runtime is, by (1), at most $n \cdot O(|L|) = n^{O_{\mathcal{C}}(1)}$. ◀

[6] The constructive Freiman's theorem will be required later, specifically in Lemma 11 and Theorem 21. The current result emphasizes the usefulness of parameterization in the doubling constant.

[7] We write $|\mathcal{I}|$ to denote the size of the ILP instance $\mathcal{I}$. In the word RAM model (see Section 2), this is $\mathsf{poly}(m,n)$.

[8] That is, groups in which only the identity element has finite order.

4.2 $\mathcal{C}$-Bounded ILP Feasibility

In general, ILPs with polynomially bounded variables can be converted to ILPs with binary variables (see Remark 7); however, the straightforward reduction can create many duplicate columns in the resulting Binary ILP. Although it is possible to get rid of the duplicate columns, it is easier to extend the previous result to $\mathcal{C}$-Bounded ILP Feasibility directly:

▶ **Corollary 8.** *An instance $\mathcal{I}$ of $\mathcal{C}$-Bounded ILP Feasibility such that $\ell_i \leq x_i \leq u_i$ and $|\ell_i|, |u_i| = \mathsf{poly}(n)$ for $i \in [n]$ can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \mathsf{poly}(|\mathcal{I}|)$.*

Proof. Modify the proof of Theorem 1 by considering the list L' of all possible outputs Ax for each valid assignment of variables x, using the variable bounds $x_i \in [\ell_i : u_i]$ for $i \in [n]$ instead of $x \in \{0, 1\}^n$. As before, we bound $|L'|$.

Observe that L' is contained in the GAP P'' obtained by scaling each range bound L_i of P by a factor of $n^{O(1)}$, where the hidden constant is determined by the bounds on the variables. It follows that $|L'| = n^{O_{\mathcal{C}}(1)}$. We can enumerate L' by modifying the procedure given above so that Step 2 merges a polynomial number of lists, one for each variable assignment. ◀

5 Subset Sum with Constant Doubling

We now consider the useful applications of parameterization in the doubling constant to Subset Sum. Formally, we consider the following problem:

Problem 2: $\mathcal{C}$-Subset Sum

In: An integer set $Z = \{z_1, z_2, \ldots, z_n\}$ such that $|Z + Z| \leq \mathcal{C}|Z|$ and an integer target t.
Out: $S \subseteq Z$ such that $\Sigma(S) = t$, or "NO" if no solution exists.

$\mathcal{C}$-Subset Sum is equivalent to $\mathcal{C}$-Binary ILP with a single constraint. As a result, Theorem 1 yields the following corollary for Subset Sum with n variables:

▶ **Corollary 9** ($\mathcal{C}$-Subset Sum is in XP)**.** *$\mathcal{C}$-Subset Sum can be solved in time $n^{O_{\mathcal{C}}(1)}$.*

At this point, it is natural to wonder whether $\mathcal{C}$-Subset Sum can be solved in time $O_{\mathcal{C}}(1) \cdot n^{O(1)}$: that is, whether Subset Sum is in FPTwith respect to the doubling constant. While we cannot yet prove or disprove this statement, we can show that it is equivalent to an open problem in the parameterized complexity of integer programming. The remainder of this section proves this reduction in both directions.

5.1 Reduction from $\mathcal{C}$-Subset Sum to Hyperplane-Constrained Binary ILP Feasibility

Recent generalizations of Integer Programming consider the problem of optimizing the value $g(Ax)$ in place of Ax, where $g : \mathbb{R}^m \to \mathbb{R}$ is a low-dimensional objective function [15]. The mapping given by Freiman's Theorem provides a natural reduction from Subset Sum with constant doubling to a problem of this form. Specifically, $\mathcal{C}$-Subset Sum reduces to a Binary ILP feasibility problem in which the constraint matrix A has bounded entries and a feasible solution is any x satisfying $\langle Ax, s\rangle = t$ for a specific "step vector" s.

Problem 3: Hyperplane-Constrained Binary ILP (HBILP) Feasibility

In: An integer matrix $A \in \mathbb{Z}^{m \times n}$, a step vector $s \in \mathbb{Z}^m$, and a target integer t. We let $\Delta := \|A\|_\infty$, the magnitude of A's largest entry.
Out: A vector $x \in \{0,1\}^n$ such that $\langle Ax, s\rangle = t$, or "NO" if no solution exists.

The reduction from $\mathcal{C}$-Subset Sum to HBILP Feasibility (Lemma 11) is straightforward but relies crucially on our constructive Freiman's Theorem.

▶ **Lemma 10.** *For any fixed instance (Z, t) of $\mathcal{C}$-Subset Sum, there exists a HBILP Feasibility instance given by $A \in \mathbb{Z}^{d(\mathcal{C}) \times n}$, $s \in \mathbb{Z}^{d(\mathcal{C})}$, and t for some function $d(\mathcal{C})$ such that a vector $x \in \{0,1\}^n$ satisfies*

$$Ax = t \quad \textit{if and only if} \quad \sum_{i\,:\,x_i=1} z_i = t.$$

Moreover, $\Delta := \|A\|_\infty \le n^{2/d(\mathcal{C})}$, and the reduction can be computed in time $\widetilde{O}_{\mathcal{C}}(n)$ with success probability $1 - n^{-\gamma}$ for an arbitrarily small constant γ.

Proof. Fix an instance of $\mathcal{C}$-Subset Sum given by an integer set Z satisfying $|Z + Z| \le \mathcal{C}|Z|$ and an integer target t. Apply Theorem 5, which fails with probability $n^{-\gamma}$ and otherwise produces a GAP $P = \{y_1\ell_1 + y_2\ell_2 + \cdots + y_d\ell_d \ : \ \forall i, \ell_i \in [L_i]\}$ of dimension $d := d(\mathcal{C})$ and volume $v(\mathcal{C})n$ containing Z.

For each $z_i \in Z$, let $v(z_i) = (v_1, v_2, \ldots, v_d)$ be an arbitrary $d(\mathcal{C})$-dimensional integer vector satisfying

$$y_1v_1 + y_2v_2 + \ldots y_dv_d = z_i \text{ and } \forall i \in [d], v_i \in [L_i].$$

We can think of $v(z_i)$ as the d-dimensional "GAP coordinates" of the input element z_i. $v(z_i)$ is guaranteed to exist by Freiman's theorem, and we can recover it in time $O(|P|) = O_{\mathcal{C}}(n)$ via exhaustive search of P. (However, $v(z_i)$ is not guaranteed to be unique.)

To complete the reduction, set

$$A \in \mathbb{Z}^{d(\mathcal{C}) \times n} \text{ with } \forall j \in [n], A[\cdot, j] = v(z_j),$$

set $s := (y_1, y_2, \ldots, y_d)$ and preserve the same target t. Note that $\|A\|_\infty \le n^{2/d(\mathcal{C})}$ without loss of generality by Observation 6.

We claim that for any binary vector $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$,

$$\langle Ax, s\rangle = \sum_{i\,:\,x_i=1} z_i. \tag{2}$$

To see this, observe that

$$\begin{aligned}\langle Ax, s\rangle &= \sum_{i\in[d]} y_i \sum_{x_j=1} A[i,j] = \sum_{x_j=1} y_1A[1,j] + y_2A[2,j] + \ldots y_dA[d,j] \\ &= \sum_{x_j=1} \langle y, v(z_j)\rangle = \sum_{j\,:\,x_j=1} z_j.\end{aligned}$$

Thus $\langle Ax, s\rangle = t$ if and only if $\sum_{i\,:\,x_i=1} z_i = t$, and there is a one-to-one correspondence between solutions to our $\mathcal{C}$-Subset Sum instance and our HBILP feasibility instance. ◀

5.2 Equivalence Between HBILP Feasibility and Subset Sum

▶ **Theorem 2.** *It is possible to solve $\mathcal{C}$-Subset Sum in time $O_{\mathcal{C}}(\mathsf{poly}(n))$ if and only if* Hyperplane-Constrained Binary ILP *(HBILP) can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(|\mathcal{I}|))$, where $|\mathcal{I}|$ is the size of the instance.*

Theorem 2 follows immediately from the next two lemmas, which show reductions in both directions. The first is a consequence of the reduction in Section 5.1:

▶ **Lemma 11.** *If HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$, then $\mathcal{C}$-Subset Sum can be solved in time $O_{\mathcal{C}}(\mathsf{poly}(n))$ with success probability $1 - n^{-\gamma}$ for an arbitrarily large constant $\gamma > 0$.*

Proof. In polynomial time (in the size of the input), we can preprocess an instance of Subset Sum and produce an equivalent one such that all integers are bounded by $2^{\mathsf{poly}(n)}$ (see, e.g., [18, 20]).[9] Next, we use the reduction given in Lemma 10, which takes time $\widetilde{O}_{\mathcal{C}}(n)$ and succeeds with probability $1 - n^{-\gamma}$, and solve the resulting HBILP instance in time

$$\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n)) = (n^{2/d(\mathcal{C})})^{O(d(\mathcal{C}))} \cdot O_{\mathcal{C}}(\mathsf{poly}(n)) = O_{\mathcal{C}}(\mathsf{poly}(n)). \qquad \blacktriangleleft$$

Moreover, we can assume that each entry of A is non-negative and that any solution vector x has fixed support exactly q for some $q = \Theta(n)$:

▶ **Observation 12.** *Let $\mathcal{I}$ be an instance of HBILP feasibility given by the constraint matrix $A \in \mathbb{Z}^{m \times n}$, the step vector $s \in \mathbb{Z}^m$, and the target $t \in \mathbb{Z}$. Without loss of generality, we can assume that entries of A are non-negative and that every solution x has fixed support size q for some $q = \Theta(n)$.*

The proof of this observation is in the full version of the paper.

▶ **Lemma 13.** *If $\mathcal{C}$-Subset Sum admits an $O_{\mathcal{C}}(\mathsf{poly}(n))$-time algorithm, HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$.*

Proof. Fix an instance of HBILP Feasibility given by the matrix $A \in \mathbb{Z}^{m \times n}$, the vector $s \in \mathbb{Z}^m$, and the integer target t. Let $\Delta := \|A\|_\infty$.

We perform the reduction in two steps. First, we self-reduce our HBILP instance to another HBILP instance A', s', t' with the property that every column $A'[\cdot, j]$ of A' has a unique dot product $\langle A'[\cdot, j], s' \rangle$. We then reduce A', s', t' to $\mathcal{C}$-Subset Sum.

If A contains any column with only zeroes, then the value of the corresponding entry of x does not matter, and we can safely delete it. Thus we can assume without loss of generality that each column of A has at least one nonzero entry.

By Observation 12, we can assume that each entry of A is non-negative and that any solution vector x has fixed support exactly q for some $q = \Theta(n)$.

Step 1: Self-reduction. In order to construct the instance A', s', t', define $M := nm\Delta\|s\|_\infty + 1$, which satisfies $M > \langle Ax, s \rangle$ for any $x \in \{0, 1\}^n$ by construction. Moreover, let $k := \lceil \log_\Delta(n) \rceil$.

Let $R \in [\Delta]^{k \times n}$ be the matrix whose columns are vectors in $[0 : \Delta - 1]^k$ in lexicographically increasing order. Because the number of such vectors is at least n, every column of R is different. Recall that $J_{k \times n}$ denotes the $k \times n$ matrix containing only 1's and let $\overline{R}$ be $\Delta \cdot J_{k \times n} - R$.

[9] See discussion about the computational model in Section 2.

Create the block matrix $A' \in \mathbb{Z}_{\geq 0}^{(m+k)\times 2n}$ as follows. The top-left block is A, the bottom-left block is R, the bottom-right block is $\overline{R}$ and each entry in the top-right block is 0. Observe that every column in $\widetilde{A}$ is distinct because each column in R is distinct and no column in A is all 0's.

Create $s' \in \mathbb{Z}_{\geq 0}^{m+k}$ as follows. The first m entries of s' are s, and the remaining k entries are the vector $v = (M\Delta^0, M\Delta^1, \ldots, M\Delta^{k-1})$. Finally, set $t' := t + q\Delta\|v\|_1$ to complete the reduction.

$$A' := \left(\begin{array}{c|c} A & \mathbf{0} \\ \hline R & \overline{R} \end{array}\right) \qquad s' := \left(\begin{array}{c} s \\ \hline v \end{array}\right)$$

▷ Claim 14. For every distinct pair of indices $i, j \in [2n]$, $\langle A'[\cdot, i], s'\rangle \neq \langle A'[\cdot, j], s'\rangle$.

Proof. Begin with the first n columns. For all $i \in [n]$, we can break down the relevant dot product into two pieces corresponding to the top and bottom portions of A':

$$\langle A'[\cdot, i], s'\rangle = \langle A[\cdot, i], s\rangle + \langle R[\cdot, i] \cdot v\rangle.$$

First, observe that $\langle R[\cdot, i], v\rangle$ is distinct for every $i \in [n]$ by construction and the components of v increase by factors of Δ.

Second, because $0 < \langle A[\cdot, i], s\rangle \leq M$, the $\langle A[\cdot, i], s\rangle$ term of the dot product $\langle A'[\cdot, i], s'\rangle$ is not large enough to interfere with the $\langle R[\cdot, i], v\rangle$ term, and thus the first n columns of A' have distinct dot products with s'.

Because $\overline{R} = \Delta \cdot J_{k\times n} - R$, and because no column of A consists of all 0's by assumption, similar arguments show that the value $\langle A'[\cdot, i], s'\rangle$ is distinct for every column $i \in [2n]$. ◁

▷ Claim 15. The ILP instance (A', s', t') has a solution if and only if the instance (A, s, t) has a solution (and the solution to A, s, t can be recovered efficiently from the solution of A', s', t').

Proof. Suppose x satisfies $\langle Ax, s\rangle = t$. Recall that x has support exactly q by Observation 12 without loss of generality. Thus the vector x' created by concatenating two copies of x satisfies

$$\langle A'x', s'\rangle = t + q\Delta\|v\|_1 = t'.$$

Moreover, any vector $y' \in \{0, 1\}^{2n}$ that satisfies $\langle A'y', s'\rangle = t'$ must satisfy

$$\langle A(y'_1, y'_2, \ldots, y'_n), s\rangle = t$$

by construction. This is because the $[R \mid \overline{R}]$ submatrix of A' can contribute to t' only in multiples of M. Because $M > \langle Ax, s\rangle$, we know that $\langle A(y'_1, y'_2, \ldots, y'_n), s\rangle < M$, and thus this product evaluates to t. ◁

Step 2: Reduction to $\mathcal{C}$-Subset Sum. Consider the integer vector

$$z := (\langle s', A'[\cdot, 1]\rangle, \langle s', A'[\cdot, 2]\rangle, \ldots, \langle s', A'[\cdot, 2n]\rangle) \tag{3}$$

and let $Z = \{z_1, z_2, \ldots, z_{2n}\}$ denote the set containing the components of z. (Note that Z is a proper set and contains no duplicates, by Claim 14.) We proceed to consider Z, t as an instance of Subset Sum.

Because $\langle A'x, s'\rangle = t'$ if and only if $\langle x, z\rangle = t'$ by construction (3), we have a one-to-one correspondence between solutions to our Subset Sum and HBILP Feasibility instances: any subset of Z that adds to t' corresponds to a binary vector $x \in \{0,1\}^{2n}$ such that $\langle A'x, s'\rangle = t'$, which can be used to recover a solution for the original instance A, s, t by Claim 15. It remains to show that an $O_{\mathcal{C}}(\mathsf{poly}(n))$ algorithm for $\mathcal{C}$-Subset Sum will allow us to solve the problem in the claimed time.

We begin by bounding the doubling constant $\mathcal{C}$ of Z. By the definition of z, we have that $z_j = \langle s, A'[\cdot, j]\rangle$ for all $j \in [n]$, and thus Z is a subset of the GAP

$$Y := \{y_1 s_1 + y_2 s_2 + \cdots + y_m s_m + y_{m+1} M \ : \ -\Delta \le y_i \le \Delta, \forall i \in [m]; 0 < y_{m+1} < \Delta^{k-1}\}.$$

Note here that the dimension of Y is $m+1$ instead of $m+k$, as we have chosen to represent the component of each $z_j \in Z$ divisible by M into a single large dimension.

We claim that we can assume $|Z| = \Omega(|Y|)$ without loss of generality. To see this, observe that we can inflate $|Z|$ by adding up to $|Y|$ dummy elements from the translated GAP $t+Y$. Because every such element is greater than t, and each is contained in a translation of Y, we create no additional solutions and increase $|Y+Y|$ by at most a factor of 2.

We have that

$$|Z+Z| \le |Y+Y| \le 2^{m+1} \cdot |Y| = O_m(1) \cdot |Z|,$$

where the first inequality follows from the fact that $Z \subseteq Y$, the second follows from the fact that $|Y|$ has dimension $m+1$, and the third follows from the fact that $|Z| = \Omega(|Y|)$.

Thus Z, t is an instance of $O_m(1)$-Subset Sum whose solutions correspond directly to solutions of our original HBILP feasibility instance. Also, $|Z| = O(|Y|) = \Delta^{O(m)+k}$. Because $\Delta^k = O(n)$ by the definition of k, an algorithm for Subset Sum that runs in time $O_{\mathcal{C}}(\mathsf{poly}(n))$ solves Z, t in time $O_m(\mathsf{poly}(\Delta^m \cdot n)) = \Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$ as claimed. ◀

5.2.1 Reduction from BILP Feasibility to HBILP Feasibility

An FPT algorithm for $\mathcal{C}$-Subset Sum further implies a $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$ algorithm for Bounded ILP feasibility, i.e., an extension of Eisenbrand and Weismantel's improvement for Unbounded ILPs to determining feasibility for Bounded ILPs.

▶ **Corollary 16.** *If $\mathcal{C}$-Subset Sum can be solved in $O_{\mathcal{C}}(\mathsf{poly}(n))$, then Bounded ILPs defined by $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$ with $\Delta := \|A\|_\infty$ and each variable x_i bounded by $\mathsf{poly}(n)$ can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$.*

Corollary 16 is a straightforward corollary of Lemma 17, which reduces ILP Feasibility with binary variables to HBILP feasibility, and the fact that ILPs with bounded variables can be reduced to binary ILPs (Remark 7). We defer the proof of Lemma 17 to the full paper.

▶ **Lemma 17.** *If HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$, Binary ILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(\mathsf{poly}(n))$.*

6 Unbounded Subset Sum with Constant Doubling

$\mathcal{C}$-Unbounded Subset Sum is equivalent to an unbounded integer program with a single constraint. In this section, we prove a near-XP algorithm for $\mathcal{C}$-Unbounded Subset Sum by first using the constructive Freiman's theorem to map instances to integer programs with small coefficients, then using existing methods to find small-support solutions to the integer programs. The proof of the lemma uses techniques that are standard in the literature (see, e.g., [16]); nevertheless, we are not aware of a prior proof of the following statement.

▶ **Lemma 18** (ILP Solutions with small support). *Let $A \in \mathbb{Z}^{m\times n}$ with $\Delta := \|A\|_\infty$. In $(n\Delta)^{O(m)}$ time we can find a set $\mathcal{X} \subseteq \{0,1\}^n$ with the following property: For any target vector $b \in \mathbb{Z}^m$ corresponding to at least one solution $x \in \mathbb{Z}^n_{\geq 0}$ with $Ax = b$, there exists a small-support solution $y \in \mathbb{Z}^n_{\geq 0}$ satisfying*

$$Ay = b, \ \ \operatorname{supp}(y) \in \mathcal{X} \quad \textit{and} \quad |\operatorname{supp}(y)| \leq m \log_2(2n\Delta + 1).$$

Proof. We begin with a bound on the support of lexicographically minimal solutions that follows standard arguments.

▷ Claim 19. Let $A \in \mathbb{Z}^{m\times n}$ with $\Delta = \|A\|_\infty$, and let $y \in \mathbb{Z}^n_{\geq 0}$ be the lexicographically minimal vector such that $Ay = b$ for some $b \in \mathbb{Z}^m$. Then $|\operatorname{supp}(y)| \leq m \log_2(2n\Delta + 1)$.

Proof. Assume for contradiction that $2^{|\operatorname{supp}(y)|} > (2n\Delta + 1)^m$. Because $Ax \leq (2n\Delta + 1)^m$ for any $x \in \{0,1\}^n$, there must exist two different vectors $v, w \in \{0,1\}^n$ such that (i) $\operatorname{supp}(v), \operatorname{supp}(w) \subseteq \operatorname{supp}(y)$, and (ii) $Av = Aw$, by the pigeonhole principle.

Let $y_1 = y - w + v$ and $y_2 = y + w - v$. Observe that $Ay_1 = Ay_2$ and $y_1, y_2 \in \mathbb{Z}^n_{\geq 0}$ because $\operatorname{supp}(v), \operatorname{supp}(w) \subseteq \operatorname{supp}(y)$. Moreover, because $v \neq w$ we have that y_1 or y_2 is lexicographically smaller than y, contradicting the assumption that y is lexicographically minimal. ◁

Let $\mathcal{X} \subseteq \{0,1\}^n$ be the set of lexicographically minimal solutions to $Ax = b$ for every $b \in \mathbb{Z}^n_{\geq 0}$ with $\|b\|_\infty < n\Delta$. Clearly, $|\mathcal{X}| \leq (2n\Delta + 1)^m$ as this is the number of suitable b's. To construct $\mathcal{X}$ it remains to iterate over every $b \in \mathbb{Z}^n_{\geq 0}$ with $\|b\|_\infty < n\Delta$ and solve the following Integer Linear Program:

$$\max \left\{ \sum_{i=1}^{n} x_i \cdot M^i \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0} \right\},$$

where $M = 4n\Delta$. Note that this can be solved in $(n\Delta)^{O(m)}$ time by [17, Theorem 2.3] for each b. Hence, the set $\mathcal{X}$ can be constructed in the claimed time. Finally, it remains to show that for any feasible b, there exists a solution y with small support in $\mathcal{X}$.

▷ Claim 20. Let $b \in \mathbb{Z}^m$ be any vector for which there exists $x \in \mathbb{Z}^n_{\geq 0}$ with $Ax = b$. Then there also exists $y \in \mathbb{Z}^n_{\geq 0}$ such that $Ay = b$ and $\operatorname{supp}(y) \in \mathcal{X}$.

Proof. Let $z \in \mathbb{Z}^n_{\geq 0}$ be the lexicographically minimum vector such that $Az = b$. Let $\widehat{z} \in \{0,1\}^n$ be such that $\widehat{z}_i = 1$ iff $z_i \neq 0$ and $\widehat{z}_i = 0$ otherwise. Let $\widehat{b}$ be such that $A\widehat{z} = \widehat{b}$. Observe that $\|\widehat{b}\|_\infty < n\Delta$.

Hence it remains to show that $\widehat{z}$ is the lexicographically minimal vector for which $A\widehat{z} = \widehat{b}$. Assume for contradiction that there exists $\widehat{y} \in \mathbb{Z}^n_{\geq 0}$ such that $A\widehat{y} = \widehat{b}$ and $\widehat{y}$ is lexicographically smaller than $\widehat{z}$. Consider a vector $y = z - \widehat{z} + \widehat{y}$. Note, that $\operatorname{supp}(\widehat{z}) \subseteq \operatorname{supp}(z)$ so $y \in \mathbb{Z}^n_{\geq 0}$. Clearly $Ay = Az = b$. Moreover, because $\widehat{y}$ is lexicographically smaller than $\widehat{z}$, y is lexicographically smaller than z, contradicting our assumption that z is lexicographically minimal. ◁

Thus the set $\mathcal{X}$ satisfies the property stated in Lemma 18, concluding the proof. ◀

With Lemma 18 in hand, let us present our algorithm for $\mathcal{C}$-Unbounded Subset Sum.

▶ **Theorem 21** (Near-XP algorithm for $\mathcal{C}$-Unbounded Subset Sum). *$\mathcal{C}$-Unbounded Subset Sum can be solved in time $n^{O_{\mathcal{C}}(1)}$ if an ILP instance $\mathcal{I}$ on v variables can be solved in time $2^{O(v)}\mathsf{poly}(|\mathcal{I}|)$.*

As a result, the current best algorithm for ILP [32] solves $\mathcal{C}$-Unbounded Subset Sum in time $n^{O_{\mathcal{C}}(1) \log\log\log(n)}$.

Proof. Following the steps of our reduction from $\mathcal{C}$-Subset Sum to HBILP feasibility (Lemma 10), we can use the constructive Freiman's theorem[10] (Theorem 4) to encode the $\mathcal{C}$-Unbounded Subset Sum instance as an *Unbounded* Hyperplane-Constrained ILP Feasibility instance given by $A \in \mathbb{Z}_{\geq 0}^{d(\mathcal{C}) \times n}$ with $\Delta = \|A\|_\infty = n^{O(1/d(\mathcal{C}))}$, step vector ℓ, and target t.

We then use Lemma 18 to construct a set $\mathcal{X}$ of candidate supports in $(n\Delta)^{O(m)} = n^{O_\mathcal{C}(1)}$ time. For each support vector $x^* \in \mathcal{X}$, we reduce the ILP to variables in x^*. This gives us a program with $|x^*| = O(m \log_2(n\Delta)) = O_\mathcal{C}(\log(n))$ variables. Now, we encode this problem as the ILP

$$\left\{ x \in \mathbb{Z}_{\geq 0}^n \;\middle|\; \sum_{j=1}^{d} \ell_j \sum_{i \in x^*} a_{i,j} x_i = t \right\}.$$

Observe that this is equivalent to an instance of Unbounded Subset Sum with $O_\mathcal{C}(\log(n))$ items. Thus any algorithm for Unbounded Subset Sum (or, more generally, any algorithm for unbounded ILP) that runs in time $2^{O(v)}$ on instances with v variables would automatically yield an $n^{O_\mathcal{C}(1)}$ time algorithm for $\mathcal{C}$-Unbounded Subset Sum. Using the best-known algorithm for unbounded ILPs, which runs in time $(\log v)^{O(v)}$ [32], we get an $n^{O_\mathcal{C}(1) \log\log\log(n)}$-time algorithm. ◀

7 k-SUM with Constant Doubling

Our final contribution concerns the analogous problem of k-SUM with bounded doubling constant, which we refer to as $(\mathcal{C}, k)$-SUM. We prove Theorem 23 and observe that the same approach gives an algorithm for 4-SUM that is tight up to subpolynomial factors, assuming the k-SUM conjecture.

Problem 4: $(\mathcal{C}, k)$**-SUM**

In: An integer set $X = \{x_1, x_2, \dots, x_n\}$ such that $|X + X| \leq \mathcal{C}|X|$ and a target t.
Out: $S \subseteq X$ with $|S| = k$ such that $\Sigma(S) = t$, or "NO" if no solution exists.

We note that [1] and [23] also present algorithms for 3-SUM in cases where additive structure in the input is controlled by the doubling constant, and also make use of fast algorithms for sparse convolution. In both cases, these authors focus on the setting of tripartite 3-SUM under the condition that at least one of the three input sets A, B, and C is guaranteed to have a small doubling.

Before we continue, let us recall the standard "color-coding" technique that allows us to ensure that each integer in the solution is taken at most once.

▶ **Lemma 22.** *Let A be a set of n integers. There exists a set family $\mathcal{P} \subseteq \{(A_1, \dots, A_k)$ such that $A_1 \uplus \dots \uplus A_k$ is partition of $A\}$ with the following properties:*
1. *For any $S \subseteq A$ of cardinality $|S| = k$, there exists $(A_1, \dots, A_k) \in \mathcal{P}$ with $|S \cap A_i| = 1$ for all $i \in [k]$.*
2. $|\mathcal{P}| = 2^{O(k)} \cdot \log(n)$.

$\mathcal{P}$ can be constructed deterministically in $2^{O(k)} n \log(n)$ time.

[10] We remark that because the construction of the GAP guaranteed by Freiman's theorem is not the runtime bottleneck, a slower constructive algorithm might suffice for this step.

The proof of Lemma 22 is a reformulation of a standard construction of an (n, k)-perfect hash family (see [3], Section 4). For completeness, we include a standalone proof in the full version of the paper. We now commence with the proof of Theorem 23.

▶ **Theorem 23.** *Given an integer set X such that $|X + X| \leq \mathcal{C} \cdot |X|$ and an integer t, we can decide if there exists a set $\{x_1, \ldots, x_k\} \subseteq X$ such that $x_1 + \ldots + x_k = t$ in deterministic time $\widetilde{O}(\mathcal{C}^{\lceil k/2 \rceil} \cdot 2^{O(k)} \cdot n)$.*

Proof. Let X be an integer set of size n, and let $\{x_1, \ldots, x_k\} \subseteq X$ denote a set of k integers that sum to t. We commence by constructing the family $\mathcal{P}$ from Lemma 22 and guessing a partition $(X_1, \ldots, X_k) \in \mathcal{P}$ of X such that $x_i \in X_i$ for all $i \in [k]$. Observe that by Lemma 22 this incurs only an additional $2^{O(k)} \log(n)$ factor in the running time.

Now, we use the sparse convolution algorithm of Bringmann et al. [7].

▶ **Lemma 24** (Theorem 1 in [7]). *Given two sets $A, B \subseteq [\Delta]$, the set $A + B \coloneqq \{a + b \mid a \in A, b \in B\}$ can be constructed deterministically in $\widetilde{O}(|A + B| \cdot \mathsf{polylog}(\Delta))$ time.*

We use Lemma 24 to enumerate two sets:

$$\mathcal{L} \coloneqq X_1 + \ldots + X_{\lfloor k/2 \rfloor} \qquad \text{and} \qquad \mathcal{R} \coloneqq X_{\lfloor k/2 \rfloor + 1} + \ldots + X_k.$$

Both $\mathcal{L}$ and $\mathcal{R}$ can be computed deterministically in $\widetilde{O}(k \cdot (|\mathcal{L}| + |\mathcal{R}|))$ time by repeatedly applying Lemma 24. Next, with both $\mathcal{L}$ and $\mathcal{R}$ in hand, we apply the meet-in-the-middle approach to recover a solution if one exists. This can be implemented in $\widetilde{O}(|\mathcal{L}| + |\mathcal{R}|)$ time by first sorting $\mathcal{L}$ and $\mathcal{R}$, and then for every element $a \in \mathcal{L}$ using binary search to decide if $t - a \in \mathcal{R}$. Finally, if for at least one $a \in \mathcal{L}$ we find an accompanying element in $\mathcal{R}$, we know that the instance has a solution.

As stated, the algorithm decides k-SUM without recovering a solution. However, given that a solution exists we can recover a solution via binary search at the cost of an additional $O_k(\log(n))$ factor. This concludes the description of the algorithm.

Correctness of the algorithm follows from the fact that Lemma 22 returns a valid partition, and from the definition of the sets $\mathcal{L}$ and $\mathcal{R}$. It remains to bound the runtime. Since the other steps of the algorithm take time $\widetilde{O}_k(n)$, the bottleneck occurs in the meet-in-the-middle step, which takes time $\widetilde{O}_k(|\mathcal{L}| + |\mathcal{R}|)$. Therefore it remains to bound the sizes of $\mathcal{L}$ and $\mathcal{R}$. Without loss of generality, consider $|\mathcal{R}|$, and observe

$$|\mathcal{R}| = |X_1 + \ldots + X_{\lceil k/2 \rceil}| \leq |\lceil k/2 \rceil X| = \mathcal{C}^{\lceil k/2 \rceil} |X|,$$

where the final step applies Plünnecke-Ruzsa (Lemma 3). ◀

In the specific case of $k = 4$, using the doubling constant directly gives a slightly better bound. The resulting algorithm for $(\mathcal{C}, 4)$-SUM is optimal up to subpolynomial factors under the 4-SUM conjecture.

▶ **Corollary 25.** *$(\mathcal{C}, 4)$-SUM can be solved in expected time $\widetilde{O}(\mathcal{C}n)$. Moreover, for any constant $\varepsilon > 0$, $(\mathcal{C}, 4)$-SUM cannot be solved in $O(\mathcal{C}^{1-\varepsilon}n)$ time unless 4-SUM can be solved in time $O(n^{2-\varepsilon})$ for $\varepsilon > 0$.*

Proof. The upper bound follows by analysis of the proof of Theorem 23. Recall that the bottleneck is $|\mathcal{R}|$, which in the case when $k = 4$ is $|X + X| \leq \mathcal{C} \cdot |X|$.

For the lower bound, observe that $|X + X| \leq |X|^2$ and therefore $\mathcal{C} \leq |X|$. Thus, any algorithm for $(\mathcal{C}, 4)$-SUM with runtime $O(\mathcal{C}^{1-\varepsilon}n)$ would yield an algorithm for 4-SUM that runs in $O(n^{2-\varepsilon})$ time. ◀

References

1 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-SUM Lower Bounds for Approximate Distance Oracles via Additive Combinatorics. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023. doi:10.1145/3564246.3585240.

2 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based Lower Bounds for Subset Sum and Bicriteria Path. *ACM Transactions on Algorithms (TALG)*, 18(1):1–22, 2022.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

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

5 Khodakhast Bibak. Additive Combinatorics: With a View Towards Computer Science and Cryptography—An Exposition. In *Number Theory and Related Fields*, pages 99–128, New York, NY, 2013. Springer New York.

6 Karl Bringmann. A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017.

7 Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and Las Vegas Algorithms for Sparse Nonnegative Convolution. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3069–3090. SIAM, 2022.

8 Karl Bringmann and Philip Wellnitz. On Near-Linear-Time Algorithms for Dense Subset Sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

9 Timothy M Chan and Moshe Lewenstein. Clustered Integer 3SUM via Additive Combinatorics. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 31–40, 2015.

10 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster Algorithms for Bounded Knapsack and Bounded Subset Sum Via Fine-Grained Proximity Results. *CoRR*, abs/2307.12582, 2023. doi:10.48550/arXiv.2307.12582.

11 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset Sum in Time $2^{n/2}/poly(n)$, 2023. doi:10.48550/arXiv.2301.07134.

12 Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-Structured Integer and Linear Programming in Strongly Polynomial and Near Linear Time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1666–1681. SIAM, 2021. doi:10.1137/1.9781611976465.101.

13 Jana Cslovjecsek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient Sequential and Parallel Algorithms for Multistage Stochastic Integer Programming Using Proximity. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.33.

14 Jana Cslovjecsek, Martin Koutecký, Alexandra Lassota, Michał Pilipczuk, and Adam Polak. Parameterized algorithms for block-structured integer programs with large entries. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 740–751. SIAM, 2024.

15 Daniel Dadush, Arthur Léonard, Lars Rohwedder, and José Verschae. Optimizing Low Dimensional Functions over the Integers. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 115–126. Springer, 2023.

16 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

17 Friedrich Eisenbrand and Robert Weismantel. Proximity Results and Faster Algorithms for Integer Programming Using the Steinitz Lemma. *ACM Transactions on Algorithms (TALG)*, 16(1):1–14, 2019.

18 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

19 Gregory A Freiman. On the addition of finite sets. In *Doklady Akademii Nauk*, volume 158, pages 1038–1041. Russian Academy of Sciences, 1964.

20 Danny Harnik and Moni Naor. On the Compressibility of NP Instances and Cryptographic Applications. *SIAM Journal on Computing*, 39(5):1667–1713, 2010. `doi:10.1137/060668092`.

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

22 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

23 Ce Jin and Yinzhan Xu. Removing Additive Structure in 3SUM-Based Reductions. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 405–418, 2023.

24 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Math. Program.*, 184(1):1–34, 2020. `doi:10.1007/S10107-019-01402-2`.

25 Dušan Knop, Michał Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *ACM Transactions on Computation Theory (TOCT)*, 12(3):1–19, 2020.

26 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms (TALG)*, 15(3):1–20, 2019.

27 Henrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. `doi:10.1287/MOOR.8.4.538`.

28 Shachar Lovett. *Additive Combinatorics and its Applications in Theoretical Computer Science*. Number 8 in Graduate Surveys. Theory of Computing Library, 2017. `doi:10.4086/toc.gs.2017.008`.

29 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1670–1683. ACM, 2021.

30 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. `doi:10.1145/322276.322287`.

31 Adam Polak, Lars Rohwedder, and Karol Węgrzycki. Knapsack and Subset Sum with Small Items. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ICALP.2021.106`.

32 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 974–988. IEEE, 2023. `doi:10.1109/FOCS57990.2023.00060`.

33 Imre Z Ruzsa. Sumsets and structure. *Combinatorial number theory and additive group theory*, pages 87–210, 2009.

34 Terence Tao and Van H Vu. *Additive Combinatorics*, volume 105. Cambridge University Press, 2006.

35 Luca Trevisan. Additive Combinatorics and Theoretical Computer Science. *ACM SIGACT News*, 40:50–66, 2009.

36 Emanuele Viola. *Selected Results in Additive Combinatorics: An Exposition.* Number 3 in Graduate Surveys. Theory of Computing Library, 2011. `doi:10.4086/toc.gs.2011.003`.

37 Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.
38 Yufei Zhao. Graph theory and additive combinatorics. *Notes for MIT*, 18:49–58, 2022.