


EXACT AND PARAMETERIZED ALGORITHMS
FOR SUBSET SUM PROBLEMS

Tim Randolph

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2024

 2024

Timothy William Randolph

This work is licensed under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)

Abstract

Exact and Parameterized Algorithms for Subset Sum Problems

Tim Randolph

We present a variety of exact and parameterized algorithms for Subset Sum and other related problems. The major contributions of this thesis include:

1. Average-case algorithms for Generalized Subset Sum, the problem that generalizes both Subset Sum and Equal Subset Sum, as well as structural results describing the parameter regime in which solutions exist. These results extend the application of the “Representation Method” and are the fastest known in this setting.
2. A proof that Either-Or Subset Sum, the problem of solving *either* Subset Sum *or* Equal Subset Sum, can be solved exponentially faster than time $2^{0.5n}$ in the worst case. In our view, this result illustrates the potential of the “structure vs. randomness” approach for Subset Sum.
3. Algorithms that solve worst-case Subset Sum faster than time $2^{0.5n}$ by a polynomial factor. These improvements on the best known exact algorithms for Subset Sum represent the successful application of “log shaving” techniques to the problem.
4. Algorithms for Subset Sum and k -SUM with constant doubling. When considered in terms of a novel parameterization in the doubling constant, Subset Sum admits an XP-algorithm, while k -SUM is Fixed-Parameter Tractable. We also show that Subset Sum is FPT in the doubling constant if and only if an instance \mathcal{I} of Hyperplane-Constrained Integer Linear Programming with n variables, m constraints, and constraint matrix entries bounded by Δ can be solved in time $\Delta^{O(m)} \cdot \text{poly}(|\mathcal{I}|)$.

Table of Contents

- Acknowledgments** **ix**

- Source Materials** **x**

- 1 Introduction** **1**
 - 1.1 Core Concepts and Techniques 5
 - 1.1.1 The Meet-in-the-Middle Barrier 5
 - 1.1.2 The Representation Method 7
 - 1.1.3 Additive Structure 11
 - 1.1.4 Lower Bounds 13
 - 1.2 Prior State of the Art 14
 - 1.2.1 Generalized Subset Sum and Equal Subset Sum 15
 - 1.3 Contributions of This Thesis 16
 - 1.3.1 Chapter 3: Average-Case Algorithms for Subset Sum and Equal Subset Sum 16
 - 1.3.2 Chapter 4: Complementarity of Subset Sum and Equal Subset Sum 18
 - 1.3.3 Chapter 5: Log Shaving for Subset Sum 19

1.3.4	Chapter 6: Subset Sum Parameterized in the Doubling Constant . . .	20
2	Preliminaries	22
2.1	Notation	23
2.2	Problem Statements	31
2.2.1	Generalized Subset Sum	31
2.2.2	k -SUM	33
2.2.3	\mathcal{C} -Subset Sum and (\mathcal{C}, k) -SUM	34
2.3	Folklore and Utilities	35
2.3.1	One-Sided Error, Instance Splitting, and Guessing Solution Sizes . . .	35
2.3.2	Output-Linear Enumeration of $\vec{\mathfrak{S}}(Y)$	36
2.3.3	Bounds on Solution Size	39
2.3.4	Bounds on Input Size	41
2.3.5	Prime Hashing	43
3	Average-Case Algorithms for Subset Sum and Equal Subset Sum	47
3.1	Summary of Results	49
3.2	Structural Results	53
3.2.1	When Solutions Occur in the $C = [-d : d]$ Case	57
3.2.2	When Solutions Occur in the $C = [\pm d]$ Case	68
3.2.3	Expectation of \mathfrak{Z}	75

3.2.4	Upper Bound on the Second Moment of \mathcal{Z}	87
3.3	Algorithmic Results	109
3.3.1	Reduction to a Narrower Problem	111
3.3.2	Overview of the GSS Algorithm	114
3.3.3	Implementation Details and Signature Distribution Lemma	119
3.3.4	Proof of Correctness	127
3.3.5	Proof of Runtime	132
3.3.6	Average-Case GSS on Dense Instances: Proof of Theorem 2	138
3.4	Generalized Number Balancing	139
4	The Complementarity of Subset Sum and Equal Subset Sum: Solving an "Either-Or" Problem	142
4.1	Structure vs. Randomness and Subset Sum	143
4.2	Summary of Results	145
4.2.1	Solving EOSS Using Techniques From [AKKN16]	148
4.3	Easy ESS Instances	148
4.4	Subset Sum Instances Which Are Easy ESS Instances	150
4.5	Proof of Theorem 6: The Algorithm for EOSS	152
5	Beyond the Meet-in-the-Middle Barrier: Log Shaving for Subset Sum	159
5.1	Circuit RAM and Word RAM	161
5.2	Summary of Results	162

5.3	$\Omega(n^{0.5}/\log n)$ -Factor Speedup via Bit Packing	165
5.3.1	Adapting Algorithm 5.1 to Word RAM	171
5.4	$\Omega(n^{0.01})$ -Factor Speedup via Orthogonal Vectors and the Representation Method	172
5.4.1	Definitions and Notation	173
5.4.2	Proof of Theorem 11	174
5.4.3	Auxiliary Lemmas	183
5.4.4	Adapting Algorithm 5.2 to Word RAM	187
5.5	Subset Sum in Time $O(2^{n/2} \cdot n^{-0.5023})$	188
5.5.1	Auxiliary Lemmas	206
5.5.2	Adapting Algorithm 5.5 to Word RAM	215
6	Subset Sum Parameterized in the Doubling Constant	216
6.1	Summary of Results	219
6.2	Freiman's Theorem Made Constructive	222
6.2.1	Ruzsa's Modeling Lemma	224
6.2.2	Bogolyubov's Lemma in $\mathbb{Z}/m\mathbb{Z}$	226
6.2.3	Finding a GAP in a Bohr Set	227
6.2.4	Ruzsa's Covering Lemma	229
6.2.5	Proof of Theorem 15: The Constructive Freiman's Theorem	230
6.2.6	Bounding GAP Coefficients	231
6.3	Integer Programming with Constant Doubling	232

6.3.1	\mathcal{C} -Binary ILP Feasibility	233
6.3.2	\mathcal{C} -Bounded ILP Feasibility	235
6.4	Subset Sum with Constant Doubling	236
6.4.1	Reduction from \mathcal{C} -Subset Sum to Hyperplane-Constrained Binary ILP Feasibility	236
6.4.2	Equivalence Between HBILP Feasibility and Subset Sum	239
6.4.3	Non-negativity for HBILP Feasibility	244
6.5	Unbounded Subset Sum with Constant Doubling	247
6.6	k -SUM with Constant Doubling	251
7	Future Work	254
7.1	The Meet-in-the-Middle Barrier, Reconsidered	254
7.1.1	Unbalanced Subset Sum Instances	255
7.1.2	Random-Like Instances.	256
7.1.3	Structured Instances	257
7.2	Open Questions from This Thesis	260
	References	262
A	Appendix to Chapter 2	270
A.1	Reduction of Multilist k -SUM to Single-List k -SUM	270
A.2	Reduction of Multiset k -SUM to k -SUM	272

List of Figures

1.1	The Meet-in-the-Middle algorithm	6
1.2	Cartoon of partial candidate solutions	7
1.3	Cartoon of filtering partial candidate solutions	8
1.4	Cartoon of comparing partial candidate solutions	9
1.5	Cartoon of a set with additive structure	12
2.1	Efficiently enumerating $\vec{\S}(Y)$ given the (multi)set Y	37
3.1	Plot of Λ	51
3.2	Growth of $\vec{\S}(\vec{x}_{[\ell]})$ when sparse	55
3.3	Growth of $\vec{\S}(\vec{x}_{[\ell]})$ when dense	55
3.4	Example plots of f and g	72
3.5	Illustration of (3.78)	96
3.6	Cartoon of $ G(\theta_1, \theta_2) ^n$ in one dimension	99
3.7	Illustration of R_{c_1, c_2}	100
3.8	Visual aid for the proof of Claim 10	107

3.9	Outline of the <code>GSS</code> algorithm	120
4.1	Plot of $g(S /n, \beta)$ for $\beta \approx 0.139$	153
4.2	The <code>EitherOrSubsetSum</code> algorithm	154
5.1	The <code>Bit-Packing</code> algorithm	167
5.2	The <code>RepresentationOV</code> algorithm	176
5.3	The <code>Residue-Couple-List</code> subroutine	185
5.4	Plot of $\alpha_*(\rho)$, $\gamma_*(\rho)$, $\beta(\rho)$, $\varepsilon'_1(\rho)$ in the density range $\frac{1}{2^{-H(1/4)}} < \rho \leq 1 + \Theta(\frac{1}{\log n})$	205
5.5	The <code>Sample-Packing</code> subroutine	210
5.6	The <code>Sample-Searching</code> subroutine	212

List of Tables

1.1	Worst-case algorithms for Subset Sum.	14
1.2	Average-case algorithms for Subset Sum.	15
1.3	Worst-case Algorithms for Generalized Subset Sum.	16
1.4	Average-case Algorithms for Generalized Subset Sum.	18
1.5	Worst-Case Algorithms for \mathcal{C} -Subset Sum.	21
1.6	Worst-Case Algorithms for (\mathcal{C}, k) -SUM.	21
3.1	Runtime of the GSS algorithm	51

Acknowledgements

First and foremost, I owe an enormous thanks to my advisors, Xi Chen and Rocco Servedio, who have served as my primary academic mentors for the past six years. This thesis would not have been possible without their unwavering support and optimism. Yaonan Jin and Karol Węgrzycki contributed directly to several of the works on which this thesis is built, and have been excellent collaborators. Josh Alman, Karl Bringmann, and Jesper Nederlof generously volunteered their time to serve on my thesis committee. My other academic mentors, including Nick Arnosti, Adam Cannon, Pamela Harris, and Bill Lenhart, have all helped to make this work possible.

My family and my friends in academia have provided a different kind of invaluable support. Mom, dad, and John; Clayton and Shivam, and many others, I owe you my sincere thanks. Finally, I'm grateful to Katie and Kona, whose faith in my ability to finish this thesis has never wavered.

Source Materials

Material in this thesis has been adapted from the following previous works:

- **Chapter 3.** Xi Chen, Yaonan Jin, Tim Randolph, and Rocco Servedio. "Average-Case Subset Balancing Problems." *SODA* 2022.
- **Chapter 4.** Tim Randolph. "A Hybrid Algorithm for Subset Sum and Equal Subset Sum." Preprint, 2023.
- **Chapter 5.** Xi Chen, Yaonan Jin, Tim Randolph, and Rocco Servedio. "Subset Sum in time $2^{n/2}/poly(n)$." *RANDOM/APPROX*, 2023.
- **Chapter 6.** Tim Randolph and Karol Węgrzycki. "Parameterized Algorithms on Integer Sets with Small Doubling: Integer Programming, Subset Sum and k-SUM." Preprint, 2024.

Chapter 1

Introduction

Subset Sum is a difficult algorithmic problem with a distinguished history. As 0-1 Knapsack, it made Karp's list of 21 original NP-complete problems [Kar72]; it served as one of the earliest applications for exact exponential and pseudopolynomial algorithms based on dynamic programming [HS74]; and it continues to inspire interest among cryptographers, complexity theorists and mathematicians today. Moreover, even among the many classic algorithmic problems, Subset Sum occupies a special position. Along with its very close relations, such as Partition and Knapsack, Subset Sum asks one of the most fundamental questions about a set of numbers. This becomes apparent when one considers the many equivalent "interpretations" of the Subset Sum problem:

1. Consider an integer set and a target integer t . Does any subset of the set add up to t ?
2. Consider a set of items, each of which has a certain weight and a certain value. Does there exist any collection of items that exceeds a certain value threshold while remaining within a certain weight limit?
3. Consider an abelian group operation (for example, addition over the integers) that

maps subsets of a group back to group elements. Which elements appear in the image of this function?

4. Consider a Binary Integer Linear Program specified by a constraint matrix $A \in \mathbb{Z}^{1 \times n}$ and a target b . Does there exist a feasible solution $x \in \{0, 1\}^n$ satisfying $Ax = b$?

The many interpretations of Subset Sum motivate extensions, including settings in which conditions are imposed on the input (randomness, additive structure, or bounded size), modifications of the domain (vectors, reals, or group elements), and modifications of the coefficient set (allowing elements to be used two, three, or an arbitrary number of times; subtracted, etc.) Different communities of researchers study the algorithmic problem from different angles, producing parallel and sometimes complementary literatures on heuristic, cryptographic, exact and parameterized algorithms for the problem. This thesis focuses primarily on exact algorithms, as well as algorithms parameterized in the solution size and the amount of additive structure in the input.¹

Applications may already provide sufficient reason to study Subset Sum. However, the problem also provides connections to additive combinatorics, phase transitions in the behavior of random sets, information theory, and fine-grained and parameterized complexity, as well as opportunities for delicate algorithm design. This thesis presents a collection of original results and situates them in the larger context of the landscape of difficulties and opportunities surrounding Subset Sum.

¹One flourishing line of research that we do not consider is that of *pseudopolynomial* algorithms for Subset Sum; that is, algorithms parameterized in the size of the target or largest input. Recent progress on algorithms for this problem includes improvements by Bringmann and Koiliaris and Xu [Bri17, KX19] as well as results parameterized in the size of the largest input element [PRW21, BW21, CLMZ24]. A recent breakthrough by Abboud, Bringmann, Hermelin and Shabtay showed SETH-based lower bounds for this problem, and the introduction to their paper also serves as a good starting point for the exploration of related research [ABHS22].

The question of whether Subset Sum can be solved in time $O^*(2^{(0.5-\varepsilon)n})$ ² for some constant $\varepsilon > 0$ is one of the major questions in exact algorithms (see explicit mentions in [Woe08, CFJ⁺14, AKKN15, AKKN16, NW21], among others). Despite many attempts, Horowitz and Sahni’s classic $O^*(2^{0.5n})$ -time Meet-in-the-Middle algorithm remains the fastest worst-case algorithm³ even as the current year marks the 50th anniversary of its original publication [HS74].⁴ However, this apparent difficulty has not discouraged work on the Subset Sum problem. Instead, the stubbornness of the “Meet-in-the-Middle barrier” has fueled work on variants and related settings. An early improvement occurred in 1981, when Schroepel and Shamir improved the space requirement of $O^*(2^{n/2})$ to $O^*(2^{n/4})$ while retaining the $O^*(2^{n/2})$ runtime [SS81]. An exciting recent result by Nederlof and Węgrzycki further improved the time-space tradeoff for the problem, in particular presenting an $O(2^{n/2})$ -time algorithm that runs in space $O(2^{0.249999n})$ [NW21]. Breaking the Meet-in-the-Middle Barrier remains a tantalizing direction for future research. A complementary goal would be that of establishing lower bounds for exact algorithms: currently, no $2^{\Omega(n)}$ lower bound is known, even conditioning on strong complexity-theoretic assumptions such as the Strong Exponential Time Hypothesis (SETH).

Another tool relevant for this thesis emerged in 2010, when Howgrave-Graham and Joux made a significant breakthrough by showing that Subset Sum could be solved in time $O^*(2^{0.337n})$ in the average case, under a reasonable heuristic assumption [HGJ10].⁵ Sub-

²We use $O^*(\cdot)$ notation to suppress $\text{poly}(n)$ factors.

³At least, up to exponential factors: for subexponential improvements, see [Chapter 5](#) of this thesis.

⁴See [Section 1.1.1](#) for a modern presentation of Horowitz and Sahni’s algorithm, as well as an overview of the Meet-in-the-Middle barrier.

⁵Although the original paper claims a running time of $O^*(2^{0.311n})$, a correction of the original analysis gives this $O^*(2^{0.337n})$ runtime; see [BCJ11, Section 2.2] for details. Other difficulties with the original analysis have since been ironed out.

sequent works, including [BCJ11, BBSS20, CJRS22], refined what became known as the *Representation Method*,⁶ and whittled the average-case exponent down to $O^*(2^{0.283n})$. The following decade witnessed a flurry of results in related settings, including better time-space tradeoffs [DDKS12, AKKM13, NW21], a faster polynomial-space algorithm [BGNV18], and fast algorithms for large classes of sufficiently “random-like” instances [AKKN15, AKKN16]. Because this thesis adapts the Representation Method to several different purposes, we provide an introduction to the approach in Section 1.1.2. Speaking broadly, those instances of Subset Sum on which the Representation Method fails appear to have highly unusual (that is, “non-random”) additive structure. The extent to which “structure” and “randomness” can be exploited to design faster algorithms for Subset Sum problems remains to be fully determined.

In 2019, Mucha, Nederlof, Pawlewicz and Węgrzycki used the Representation Method to establish an $O^*(3^{0.488n})$ -time worst-case algorithm for *Equal Subset Sum*, the Subset Sum variant that asks for two different input subsets with the same sum [MNPW19]. This resolved an open question of Woeginger, who observed that the Meet-in-the-Middle approach to Equal Subset Sum runs in time $O^*(3^{0.5n})$ and asked whether this was a barrier for Equal Subset Sum analogous to the $O^*(2^{0.5n})$ runtime barrier for Subset Sum [Woe08]. This result also raises a natural follow-up question: if Subset Sum and Equal Subset Sum are viewed as the problems of achieving a target sum by assigning coefficients in $\{0, 1\}$ and $\{-1, 0, 1\}$, respectively, to the input set, what is the best possible runtime for other coefficient sets? Does the Meet-in-the-Middle barrier generalize?

⁶See Section 1.1.2 for a full introduction to the Representation Method.

1.1 Core Concepts and Techniques

Before proceeding to more precise summaries of the state of the art for Subset Sum problems and the contributions of this thesis, we introduce several “big concepts” that shape the thinking of researchers working on Subset Sum and related problems. These include long-standing barriers, such as the $O^*(2^{n/2})$ runtime of Meet-in-the-Middle, and useful high-level techniques, such as the Representation Method. Collectively, these concepts shape the common wisdom regarding which problems are likely to be tractable, informative, and otherwise deserving of further study.

1.1.1 The Meet-in-the-Middle Barrier

The classic Meet-in-the-Middle algorithm was introduced by Horowitz and Sahni in 1974 [HS74], and will serve as a baseline for comparison throughout this thesis. Simply put, the algorithm divides the input into two halves A and B , enumerates $\vec{\S}(A)$ and $\vec{\S}(B)$,⁷ and for each $a \in \vec{\S}(A)$ searches $\vec{\S}(B)$ for $t - a$ (Algorithm 1.1).

A careful implementation of the Meet-in-the-Middle algorithm runs in time $O(2^{n/2})$, without hidden polynomial factors: the sorted lists in Step 1 can be enumerated in output-linear time (Section 2.3.2), and Step 2 also runs in linear time.⁸

Correctness of Algorithm 1.1 follows from the observation that if for some $a \in \vec{\S}(A)$ there exists $b \in \vec{\S}(B)$ with $a + b = t$, we cannot possibly increment our increasing pointer past a : this would require reaching an element $b' \in \vec{\S}(B)$ such that $a + b' < t$, which would be

⁷ $\vec{\S}(A)$, using the double-S symbol \S , is our special notation for the set of subset sums $\{\sum_{y \in Y} y \mid Y \subseteq A\}$; $\vec{\S}(A)$ indicates the sorted list containing the subset sums of A . See Section 2.1.

⁸An even simpler variant of the algorithm replaces Step 2 with $2^{n/2}$ binary searches of $\vec{\S}(B)$, one for each element of $\vec{\S}(A)$. This increases the runtime by a factor of $O(n)$ for the binary search.

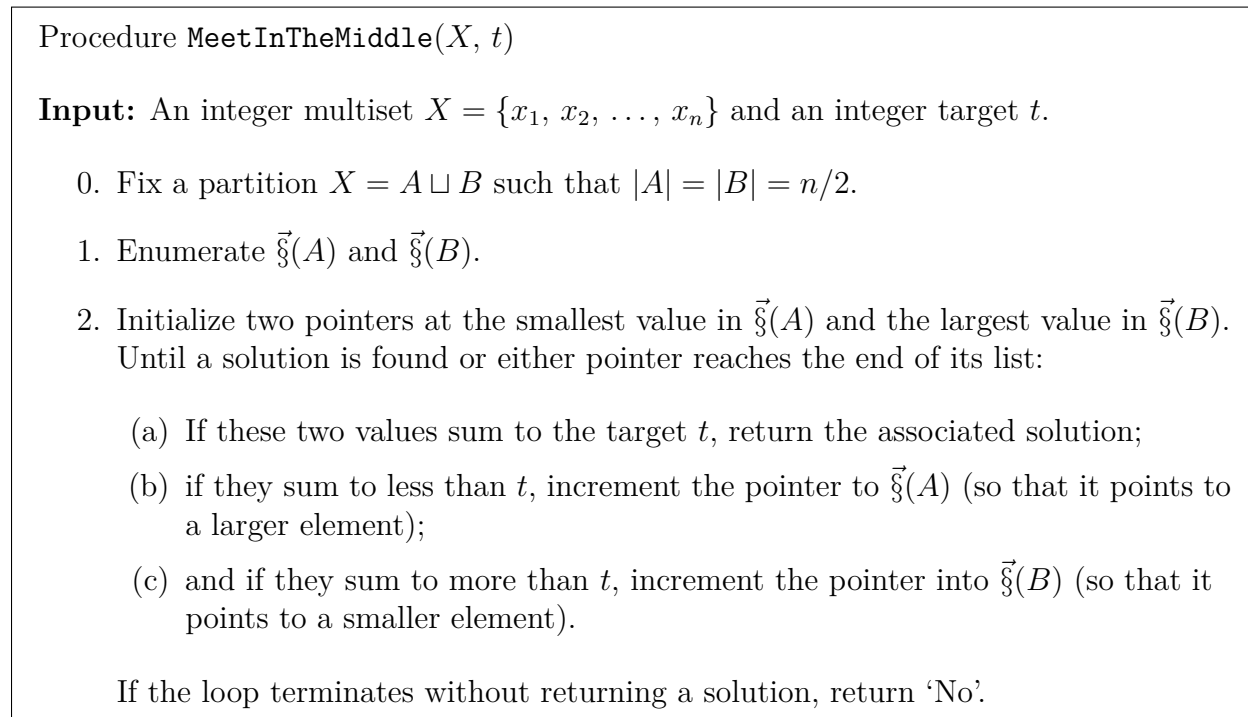


Figure 1.1: The classic Meet-in-the-Middle algorithm, adapted from [HS74].

impossible without first seeing b . By symmetry, we cannot increment past b without first seeing a . Thus the algorithm must discover the solution $a + b$ (unless another solution is discovered first).

Improving on Meet-in-the-Middle is a major open problem in exact algorithms. Among the results in this thesis are small $\text{poly}(n)$ -factor improvements on Meet-in-the-Middle, the first improvements in 50 years (Chapter 5). However, there remains no known worst-case algorithm that runs in time $O(2^{(0.5-\varepsilon)n})$ for any constant $\varepsilon > 0$.

The existence of the Meet-in-the-Middle barrier has spurred progress on variant settings and related problems. In [HGJ10], Howgrave-Graham and Joux gave an algorithm that solves *average-case* Subset Sum instances in time $O(2^{0.337n})$, which was later improved upon by Becker, Coron, and Joux and subsequently by Bonnetain, Bricout, Schrottenloher, and

Shen [BCJ11, BBSS20]. These works pioneered the *Representation Method*, which continues to prove fruitful and is explained in the next section.

1.1.2 The Representation Method

The Representation Method is a three-step process that makes it possible to break the Meet-in-the-Middle barrier for Subset Sum when certain conditions on the input are satisfied. Among the contributions of this thesis is the refinement and generalization of this approach (c.f. algorithms in Chapter 3, Chapter 4 and Chapter 5).

In a nutshell, the technique works as follows:

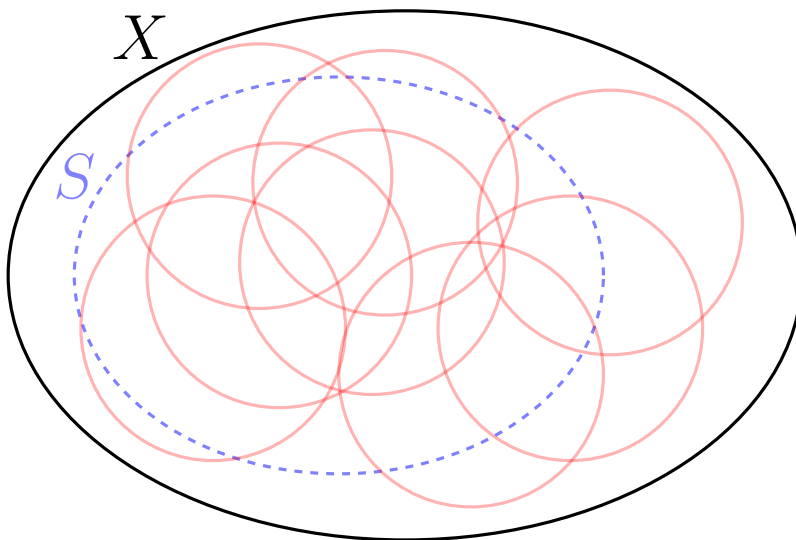


Figure 1.2: Partial candidate solutions (red) are sampled uniformly from the input set X and overlap with an unknown solution S (blue) to different extents.

1. Given a Subset Sum input, construct exponentially many *partial candidate solutions*, that is, create a search space of input subsets or other objects in which each element constitutes a guess at a partial solution. In [HGJ10], partial solutions are subsets of

the input of size approximately $n/4$, corresponding to an expected solution size of $n/2$. The algorithm's job is now to recover two partial candidate solutions that make a complete, correct solution.

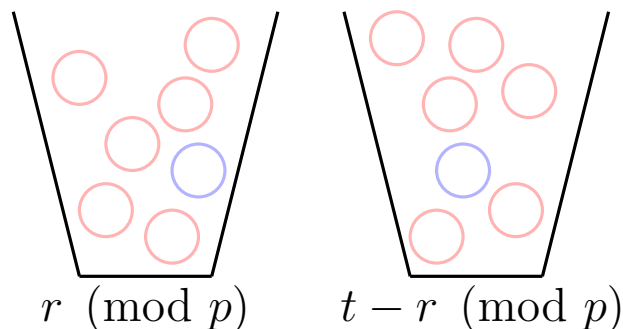


Figure 1.3: Partial candidate solutions (red circles) are sorted by residue class modulo a large random prime p . For any fixed solution S satisfying $\Sigma(S) = t$, if a partial solution $P \subseteq S$ (left blue circle) falls into $r \pmod{p}$, then $S \setminus P$ (right blue circle) falls into $(t - r) \pmod{p}$.

2. To speed up solution recovery, the algorithm employs a filtering process. There are many ways to divide a solution in half, and thus every solution will have many *representations* as a *solution pair* of partial candidate solutions.

The goal is now to throw out many partial candidate solutions while retaining at least one representation of the solution with high probability. Typically, this is done by hashing the partial candidate solutions into residue classes modulo a large prime. Conveniently, this preserves the two halves of corresponding solution pairs. For example, in [HGJ10], partial candidate solutions are hashed by their sum into residue classes modulo a large random prime p . The algorithm then discards most partial candidate solutions, keeping only those that fall into the residue classes r and $t - r \pmod{p}$ for a certain residue r . This ensures that if one half of a solution pair falls into residue

class r , we retain the other half, which falls into residue class $t - r$.

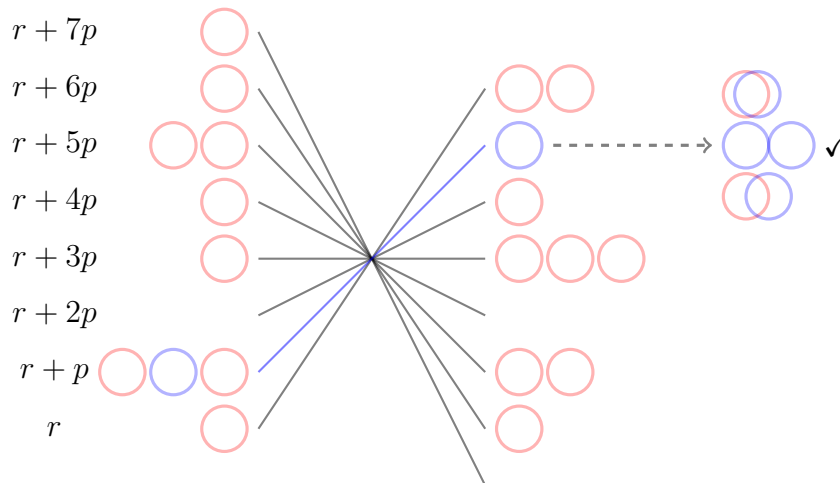


Figure 1.4: Comparing partial candidate solutions in the residue class $r \pmod{p}$ using Meet-in-the-Middle. Note that multiple partial candidate solutions (red circles) may correspond to the same value. To recover a solution, the algorithm must find a pair of partial solutions that sum to t and are disjoint (blue circles).

3. Finally, we search for a solution by comparing partial candidate solutions in the selected residue classes. This step typically uses a Meet-in-the-Middle approach.

The key observation of [HGJ10] is that any Subset Sum solution of size $n/2$ can be decomposed into $\Omega^*(2^{0.5n})$ solution pairs, each consisting of two disjoint sets of size $n/4$. Since there are

$$\binom{n}{n/4} = \Theta^*(2^{H(1/4)n}) = O(2^{0.8113n})$$

input subsets of size $n/4$ by Stirling's Approximation (2.3), hashing over a prime of size $O^*(2^{0.5n})$ produces residue classes containing $2^{H(1/4)n-0.5n} \approx 2^{0.3113n}$ partial candidate solutions in expectation.

However, several conditions must be met before the Representation Method works. The

following three issues are particularly salient:

1. The algorithm must obtain the subset of partial candidate solutions that fall into a certain residue class without enumerating all $O(2^{0.811n})$ partial candidate solutions.
2. Partial candidate solutions must distribute well over residue classes: for our algorithm to work, our chosen residue classes must contain at least one solution pair and not too many partial candidate solutions in total.
3. Finally, complications may arise during solution recovery. For example, in [HGJ10], the algorithm must ignore pairs of partial candidate solutions that add to the target but are not disjoint (*pseudosolutions*). If there are too many pseudosolutions, this causes a slowdown in the recovery step: if many pseudosolutions in the r and $t - r$ residue classes have exactly the same sum, they must all be compared to each other, eliminating the speed-up from Meet-in-the-Middle.

To resolve issue (1), enumerating a subset of partial candidate solutions can typically be accomplished via dynamic programming. With high probability over uniformly random input, solutions distribute well and few pseudosolutions appear, so issues (2) and (3) are not a problem in this case. Indeed, the technique can be adapted to Equal Subset Sum and Generalized Subset Sum by generalizing partial candidate solutions from subsets of size $n/4$ to coefficient-weighted partial solutions organized by “assignment profile” (see [Chapter 3](#)). Moreover, the approach can be adapted to worst-case settings, as long as the issues described above can be resolved by other means. [Chapter 4](#) and [Chapter 5](#) illustrate two adaptations of the Representation Method to worst-case input settings.

1.1.3 Additive Structure

Given an integer set A , what do its sumset

$$A + A = \{a_1 + a_2 \mid a_1, a_2 \in A\},$$

its iterated sumset

$$sA = \underbrace{A + A + \cdots + A}_s,$$

and its set of subset sums

$$\S(A) = \{\Sigma(T) \mid T \subseteq S\}$$

look like? Roughly, if the answer is “like a generalized arithmetic progression” (see below), we say that the set is additively structured, and if the answer is “like a random set”, we say that the set lacks additive structure. These two informal descriptions correspond to two ends of a continuum stretching from, for example:

- A has constant doubling ($|A + A| \leq C|A|$ for a constant C),
- sA is at most a constant times $|A|$,
- A (and $A + A$, sA , and $\S(A)$) can be contained in a generalized arithmetic progression of dimension $O_C(1)$ and volume $O_C(|A|)$, and
- other equivalent statements, for example, in terms of *Ruzsa distance* and *approximate groups* (see, e.g., [TV06] Proposition 2.26),

at the structured extreme all the way to

- A has linear doubling ($|A + A| \geq c\frac{|A|^2}{2}$ for some constant c),

- $sA \geq c \binom{|A|}{s}$,
- and $\xi(A) \geq c2^{|A|}$

at the random extreme.

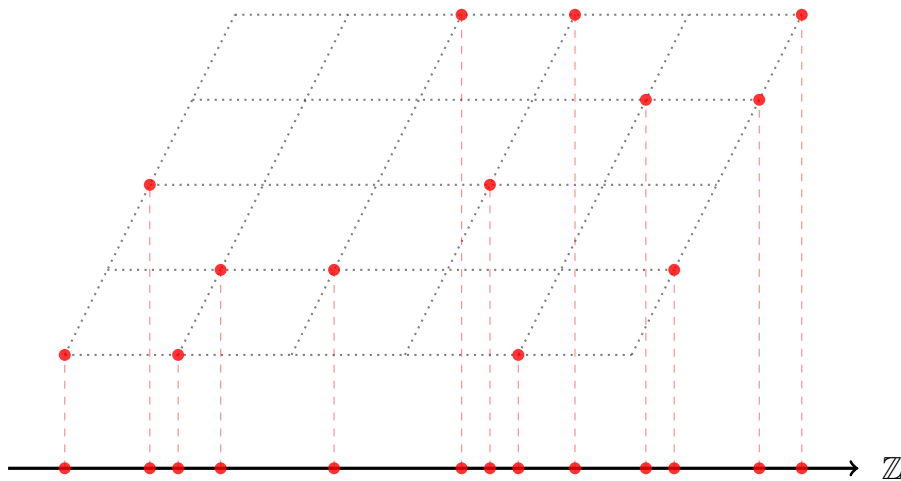


Figure 1.5: A set with additive structure (red set) is contained in a compact Generalized Arithmetic Progression (GAP). A GAP can be thought of as the projection of a multi-dimensional arithmetic progression (dotted grid) onto the line. Note that multiple sets of “GAP coordinates” may correspond to a single integer.

This way of thinking about sets is commonplace in additive combinatorics, but until recently has seen few applications to the Subset Sum problem.⁹ In [Chapter 6](#), we consider Subset Sum problems parameterized in the doubling constant. We use this parameterization to prove, roughly speaking, that if the input set X has strong additive structure, we can more efficiently solve Subset Sum problems.

⁹Early algorithms for dense subset sum due to Chaimovich, Freiman, and Galil, and Galil and Margalit, are an exception [[CFG89](#), [GM91](#)], and have influenced recent work on the problem [[BW21](#), [Bri23](#)].

1.1.4 Lower Bounds

Conditional lower bounds for Subset Sum problems are few. However, there are several which provide fundamental limitations on what we should expect to achieve.

ETH-hardness

The popular *Exponential Time Hypothesis* (ETH), due to Impagliazzo and Paturi, is the assumption that 3-SAT cannot be solved in time $2^{o(n)}$ [IP01]; under this hypothesis, Subset Sum cannot be solved in time $2^{o(n)}$. Because the doubling constant has size $O(n)$ for all sets, the ETH further implies the nonexistence of a $2^{o(C)} \cdot n^{O(C/\log(C))}$ -time algorithm for \mathcal{C} -Subset Sum.

SETH-hardness

The *Strong* Exponential Time Hypothesis extends the ETH to the statement that k -SAT cannot be solved in time $O(2^{(1-\varepsilon)n})$ for any $\varepsilon > 0$ independent of k [CIP09]. Although assuming SETH is not known to imply stronger lower bounds than $2^{o(n)}$, a recent breakthrough work by Abboud, Bringmann, Hermelin, and Shabtay produced lower bounds on pseudopolynomial algorithms for this problem. They prove that, under SETH, for any constant $\varepsilon > 0$ there exists $\delta > 0$ such that Subset Sum cannot be solved in time $O(t^{1-\varepsilon}2^{\delta n})$, and k -SUM cannot be solved in time $O(t^{1-\varepsilon}n^{\delta k})$ ([ABHS22] Theorem 1.1).

3-SUM and k -SUM Conjectures

3-SUM is easily solved in time $O(n^2)$, but conjecture once held that $o(n^2)$ -time algorithms for this problem were impossible. Increased interest in this conjecture was ignited by Gajentaan

and Overmars, who in 1993 showed that many problems in computational geometry require algorithms with asymptotic runtimes at least as large as 3-SUM [GO95]. In 2014, Grønlund and Pettie solved 3-SUM in time $O(n^2/(\frac{\log n}{\log \log n})^{2/3})$, disproving the original conjecture and leading to a series of improvements by $\text{polylog}(n)$ factors [GP18, GS15, Fre17, Cha19]. The current conjecture is that 3-SUM cannot be solved in time $n^{2-\varepsilon}$ for any constant $\varepsilon > 0$. The stronger k -SUM conjecture hypothesizes that k -SUM cannot be solved in time $n^{\lceil k/2 \rceil - \varepsilon}$ for any constant $\varepsilon > 0$.

1.2 Prior State of the Art

Work	Time	Space
[HS74]	$O^*(2^{0.5n})$	$O^*(2^{0.5n})$
[SS81]	$O^*(2^{0.5n})$	$O^*(2^{0.25n})$
[BGNV18]	$O(2^{0.860n})$	$\text{poly}(n)$
[NW21]	$O^*(2^{n/2})$	$O^*(2^{0.249999n})$

Table 1.1: Worst-case algorithms for Subset Sum.

Setting pseudopolynomial and parameterized algorithms aside, worst-case Subset Sum saw few improvements in the 40 years following Horowitz and Sahni’s development of the Meet-in-the-Middle algorithm. However, the last decade has seen a several exciting results for the problem. In 2018, Bansal, Garg, Nederlof, and Vyas resolved a longstanding open problem by designing the first $O^*(2^{(1-\varepsilon)n})$ -time, polynomial space algorithm for the problem. In 2021, Nederlof and Węgrzycki broke the “Schroppel-Shamir Space Barrier” by developing faster algorithms for the Orthogonal Vectors problem on d -dimensional binary vectors with support $d/4$. The looming challenge remains to break the Meet-in-the-Middle barrier,

but faster polynomial-space algorithms and $O(2^{n/2})$ -time algorithms with improved space complexity both have the potential to yield insight into the problem.

Work	Time	Space
[HS74]	$O^*(2^{0.5n})^\dagger$	$O^*(2^{0.5n})$
[HGJ10]	$O^*(2^{0.337n})$	$O^*(2^{0.337n})$
[BCJ11]	$O^*(2^{0.291n})$	$O^*(2^{0.291n})$
[BSS20]	$O^*(2^{0.283n})$	$O^*(2^{0.283n})$

[†] Worst-case runtime.

Table 1.2: Average-case algorithms for Subset Sum.

The timeline of average-case algorithms for Subset Sum is more straightforward. After Howgrave-Graham and Joux inaugurated the Representation Method in 2010, follow-up works by Becker, Coron and Joux and by Bonnetain, Br icout, Schrottenloher, and Shen improved the exponent by designing approaches that generalized the notion of a partial solution. All three algorithms consider the ‘‘cryptographic’’ or ‘‘hidden solution’’ variant of average-case Subset Sum, in which the solution is fixed by selecting and adding together a random subset of the input.

1.2.1 Generalized Subset Sum and Equal Subset Sum

The *Generalized Subset Sum* problem, defined formally below (Section 2.2.1), asks the following: given an input vector $\vec{x} = (x_1, x_2, \dots, x_n)$ and a target t , does there exist a coefficient vector $\vec{c} \in C^n$ such that $\vec{x} \cdot \vec{c} = t$? If $C = \{0, 1\}$, GSS reduces to Subset Sum, while if $C = \{-1, 0, 1\}$ and $t = 0$, GSS is equivalent to *Equal Subset Sum*, the problem of finding two subsets of the input with the same sum.

Extending Meet-in-the-Middle in the natural way results in algorithms for GSS that run

Work	$C = [-1 : 1]$ (ESS)	$C = [\pm 2]$	$C = [\pm d]$	$C = [-d : d]$
Meet-in-the-Middle	$O^*(3^{n/2}) = O^*(1.733^n)$	$O^*(4^{0.5n})$	$O^*(C ^{0.5n})$	$O^*(C ^{0.5n})$
[MNPW19]	$O^*(3^{0.488n}) = O^*(1.709^n)$			

Table 1.3: Worst-case Algorithms for Generalized Subset Sum.

in time $O^*(|C|^{0.5n})$ for any fixed coefficient set C . However, the Meet-in-the-Middle barrier does not appear to generalize robustly to larger coefficient sets. This was discovered by Mucha, Nederlof, Pawlewicz and Węgrzycki, who introduced a faster algorithm for worst-case Equal Subset Sum in 2019 [MNPW19].

1.3 Contributions of This Thesis

The following section contextualizes the results of this thesis, chapter by chapter.

1.3.1 Chapter 3: Average-Case Algorithms for Subset Sum and Equal Subset Sum

Chapter 3 sets the stage for the rest of the thesis by introducing the Generalized Subset Sum problem and building intuition about the properties of “average” Subset Sum instances. The main message of the section is that, at least in the average case, all subset balancing problems share a common structure.

We begin by addressing a natural question:

Question 1. *When is an instance of GSS with elements sampled uniformly at random likely to have a solution?*

The answer depends on the size of the range from which inputs are sampled (parame-

terized by the *range bound* \hat{x}), the target t , and the coefficient set C . The answer, in the natural case in which C is symmetric about 0 and t is not too large, turns out to be, very roughly, “when \hat{x} is less than $|C|^n$ ”. Although this answer is intuitive, it is surprisingly difficult to obtain: Borgs, Chayes, and Pittel provide good bounds on the phase transition in the $C = \{0, 1\}$ case [BCP01], but their argument is somewhat involved and becomes more so when generalized to larger C (see Section 3.2.2 and the following two subsections).

In Theorems 3 and 4, we answer Question 1 by bounding the ranges in which average-case GSS is likely and unlikely to yield solutions for coefficient sets of the form $[-d : d]$ and $[\pm d]$ (that is, $\{-d, \dots, -1, 0, 1, \dots, d\}$ and $\{-d, \dots, -1, 1, \dots, d\}$, respectively), as well as scale multiples and translations of these sets. For $C = [\pm d]$, we generalize the methods of [BCP01]. However when $C = [-d : d]$, we present a novel approach that uses more elementary tools, results in stronger probability guarantees, and aligns well with combinatorial intuitions about the problem (Section 3.2).

After establishing the parameter regimes in which average-case GSS is likely and unlikely to have solutions, we leverage our structural results to develop a suite of algorithms for average-case balancing problems. These generalize the Representation Method and break the Meet-in-the-Middle barrier for GSS in the average case. Our results are summarized in the table below, listed with previous worst-case results for comparison. In the general case in which $C = [\pm d]$ or $[-d : d]$, our algorithm runs in time $O^*(|C|^{\Lambda(|C|)^n}) = O^*(|C|^{(0.5 - \Omega(1/|C|))^n})$, where Λ denotes the function

$$\Lambda(z) := \max \begin{cases} 1 - \frac{z+1}{2z} \log_z(z+1) + \frac{1}{z} \log_z(2) \\ \frac{2}{3} - \frac{z+1}{3z} \log_z\left(\frac{z+1}{2}\right) \end{cases} \quad (1.1)$$

(plotted in [Figure 3.1](#)).

Work	$C = [-1 : 1]$ (ESS)	$C = [\pm 2]$	$C = [\pm d], [-d : d]$
Meet-in-the-Middle [†]	$O^*(3^{n/2}) = O^*(1.733^n)$	$O^*(4^{0.5n})$	$O^*(C ^{0.5n})$
[MNPW19] [†]	$O^*(3^{0.488n}) = O^*(1.709^n)$		
[CJRS22]	$O^*(3^{0.387n}) = O^*(1.530^n)$	$O^*(4^{0.400n})$	$O^*(C ^{\Lambda(C)})$

[†]Worst-case runtime.

Table 1.4: Average-case Algorithms for Generalized Subset Sum.

1.3.2 Chapter 4: Complementarity of Subset Sum and Equal Subset Sum

[Chapter 4](#) presents a standalone result that highlights a complementarity between the hard instances of Subset Sum and the hard instances of Equal Subset Sum. Suppose that we relax the requirements of Generalized Subset Sum by allowing ourselves to solve a GSS instance (X, t) on *at least one* of the coefficient sets $\{0, 1\}$ (Subset Sum) or $\{-1, 0, 1\}$ (Equal Subset Sum). (We allow our algorithm to return negative results, for example “No, there is no Subset Sum solution.”)

Perhaps surprisingly, for this “Either-Or Subset Sum” problem, the Representation Method can be adapted to the worst-case. The result is a worst-case algorithm that solves Either-Or Subset Sum in time $O(2^{(0.5-\varepsilon)n})$ for $\varepsilon > 0.03$, exponentially faster than the best worst-case algorithm for Subset Sum or ESS.

We can view this result as proof that the instances on which the Representation Method fails to solve Subset Sum in time faster than Meet-in-the-Middle all have certain properties that allow us to quickly solve Equal Subset Sum. (Specifically, we show that these instances either have small Equal Subset Sum solutions or generate an Equal Subset Sum solution

naturally in the course of running a Representation Method algorithm.) If we persist in thinking of the Representation Method as an algorithm that works well on “random-like” instances of Subset Sum, then EOSS algorithms that run in time $O^*(2^{(0.5-\varepsilon)n})$ serve as proof that the “structure” that defeats the Representation Method is concretely useful in solving a closely related Subset Sum problem.

1.3.3 Chapter 5: Log Shaving for Subset Sum

The original 3-SUM conjecture speculated that 3-SUM could not be solved in time $o(n^2)$. Algorithms running in time $O(n^2/\text{polylog}(n))$ eventually disproved this first statement of the conjecture, reducing it to the supposition that 3-SUM cannot be solved in time $O(n^{2-\varepsilon})$ for any constant $\varepsilon > 0$. This is perhaps the most famous example of the strategy of chipping away at algorithmic barriers by reducing runtimes by polylog factors, otherwise known as “log shaving.”

Chapter 5 considers the following question:

Question 2. *Can Subset Sum be solved in time $o(2^{n/2})$?*

Note that an $o(2^{n/2})$ -time algorithm for Subset Sum is not necessarily a $O(2^{(1/2-\varepsilon)n})$ algorithm for any constant $\varepsilon > 0$, in the same way that an $o(n^2)$ algorithm for 3-SUM is not necessarily an $O(n^{2-\varepsilon})$ -time algorithm. We further note that the analogy between the two cases is imperfect: while there is evidence that an $O(n^{2-\varepsilon})$ -time algorithm for 3-SUM is unlikely, both in the form of lower bounds in restricted models of computation [KLM19] and because it would imply surprising improvements for many other “3-SUM equivalent” problems, there is no strong evidence that an $O(2^{(1/2-\varepsilon)n})$ -time algorithm for Subset Sum is impossible.

However, although we might expect eventually to see exponential improvements on Meet-in-the-Middle, the current year (2024) marks the 50th anniversary of Horowitz and Sahni’s original publication with no such progress [HS74]. As such, in [Chapter 5](#) we treat the Meet-in-the-Middle barrier with the dignity of a fine-grained complexity hypothesis and answer [Question 2](#) in the affirmative. Specifically, we design several algorithms which solve the Subset Sum problem in time $2^{n/2}/\text{poly}(n)$ in the worst case. The first of the three algorithms we present solves Subset Sum in time $\tilde{O}(2^{n/2}/\sqrt{n})$ by adapting existing bit-packing strategies, while our final algorithm incorporates problem-specific improvements to improve the runtime beyond this point.

1.3.4 Chapter 6: Subset Sum Parameterized in the Doubling Constant

The doubling constant, defined for an integer set A as the smallest value \mathcal{C} such that $|A+A| \leq \mathcal{C}|A|$, is a parameter commonly used in additive combinatorics to capture the amount of additive structure of an integer set. A small doubling constant is evidence of a set with significant additive structure, and sets with small constant doubling approximate generalized arithmetic progressions (see [Section 1.1.3](#)). This prompts a natural question:

Question 3. *Can Subset Sum instances with small doubling constant be solved quickly (specifically, faster than trivial bounds on the search space would suggest?) Is Subset Sum fixed-parameter tractable (FPT) in the doubling constant?*

We address this question by proving that Freiman’s theorem is fixed-parameter tractable (FPT) in the doubling constant, which implies that we can efficiently construct a small generalized arithmetic progression that contains any Subset Sum instance with constant doubling.

We then use our constructive Freiman’s theorem to demonstrate that the complexity of Subset Sum, parameterized in the doubling constant, is closely related to the complexity of integer programming: specifically, Subset Sum is FPT in the doubling constant if and only if the feasibility of a Hyperplane-Constrained Binary Integer Program can be tested in time $\Delta^{O(m)} \cdot \text{poly}(n)$, where Δ denotes a bound on the size of the largest entry of the constraint matrix, m is the number of constraints, and n is the number of variables.

Work	Time
[RW23]	$O_{\mathcal{C}}(n^{f(\mathcal{C})})$

Table 1.5: Worst-Case Algorithms for \mathcal{C} -Subset Sum.

Work	Time
Meet-in-the-Middle	$O_k(n^{\lceil k/2 \rceil})$
[RW23]	$O_{\mathcal{C},k}(n \log n)$

Table 1.6: Worst-Case Algorithms for (\mathcal{C}, k) -SUM.

We also establish positive results for Subset Sum and k -SUM parameterized in the doubling constant, summarized in Tables 1.5 and 1.6. Both algorithms are themselves straightforward, but the results reinforce the conclusion that Subset Sum is a much more tractable problem when the doubling constant is used as a parameter. Compare the $O_{\mathcal{C}}(n^{f(\mathcal{C})})$ -time algorithm for \mathcal{C} -Subset Sum and the $O_{\mathcal{C},k}(n \log n)$ -time algorithm for (\mathcal{C}, k) -Subset Sum to the fastest known algorithms for (unparameterized) Subset Sum and for k -SUM. The first is an XP-algorithm with respect to \mathcal{C} , as compared to an XP-algorithm with respect to k (k -SUM) or an exponential-time algorithm when unparameterized; the second is a FPT algorithm (in fact, almost Fixed-Parameter Linear) with respect to k and \mathcal{C} , as opposed to an XP-algorithm with respect to k .

Chapter 2

Preliminaries

This chapter organizes the background information required for this thesis. Experienced readers may wish to skim the section on notation and use the remaining sections for reference as they become necessary.

The chapter contains the following subsections:

- **Notation.** Mathematical shorthand and notation. Most is standard but the double-S (\S) notation used specifically for subset sums is specific to this work.
- **Problem Statements.** Formal statements of the algorithmic problems we consider most frequently. Problems less frequently referred to are introduced in the text.
- **Folklore and Utility Lemmas.** Useful identities, reductions, and algorithmic building blocks.
- **Core Concepts and Techniques.** An introduction to the high-level obstacles, tools and opportunities surrounding subset sum problems.

2.1 Notation

Variables and Constants. With occasional exceptions, we adopt the following conventions throughout this thesis: lowercase Roman letters (ℓ, n , etc.) denote variables; lowercase Greek letters (ε, α , etc.) denote numerical constants; capital Roman letters (L, W , etc.) denote sets, multisets, or multidimensional arrays such as matrices; and calligraphic capital letters (\mathcal{W}, \mathcal{Q} , etc.) denote collections of sets.

An arrow ($\vec{\cdot}$) over a character indicates a vector ($\vec{x} = (x_1, x_2, \dots, x_n)$, etc.).

Logarithms. When written without a specified base, $\log(\cdot)$ denotes the base-2 logarithm.

Big-O. We augment standard big- O notation as follows:

We use O -tilde notation ($\tilde{O}, \tilde{\Omega}, \tilde{\Theta}$) to suppress $\text{polylog}(n)$ factors, regardless of the argument. For example, we have $\tilde{O}(2^n) = O(2^n \cdot \text{polylog}(n))$ and $\tilde{\Omega}(n) = \Omega(\frac{n}{\text{polylog}(n)})$. Likewise, the O -star notation (O^*, Ω^*, Θ^*) suppresses $\text{poly}(n)$ factors, regardless of the argument. For example, $O^*(2^n) = O(2^n \cdot \text{poly}(n))$ and $\Omega^*(1) = n^{-O(1)}$.

Variables in the subscript of big- O notation are treated as constants; that is, terms which depend solely on these variables are suppressed. For example, $O_k(n) = f(k) \cdot O(n)$ for some function $f(k)$ that does not depend on n .

Probability. Random variables appear in boldface. We write “ $\mathbf{x} \sim S$ ” to indicate that the element \mathbf{x} is sampled uniformly at random from the finite (multi)set S .

Sets and Lists. We write $[a : b]$ for the set of integers $\{a, a + 1, \dots, b\}$ and $[a]$ for $\{1, 2, \dots, a\}$. Especially when discussing the Generalized Subset Sum problem, it will be helpful to discuss sets of small integers symmetric about zero. Consistent with our prior

notation, we write $[-a : a]$ for the set $\{-a, -a+1, \dots, a+1, a\}$. We add the shorthand $[\pm a]$ for the set $\{\pm 1, \pm 2, \dots, \pm a\} = [-a : a] \setminus \{0\}$.

The square cup (\sqcup) denotes a disjoint union. For example, $X = A \sqcup B \sqcup C$ indicates a tripartition of X into the disjoint sets A , B , and C .

Given a (multi)set or list Y of integers, we adopt several shorthands.

$$\Sigma(Y) := \sum_{y \in Y} y$$

denotes the sum of the elements of Y , while 2^Y denotes the power set (set of all subsets) of Y . We write

$$Y + \alpha := \{y + \alpha \mid y \in Y\},$$

$$Y \pmod{p} := \{y \pmod{p} \mid y \in Y\}, \text{ and}$$

$$f(Y) := \{f(y) \mid y \in Y\}$$

when f takes elements of Y as arguments. We make one major exception to the general rule: if s is an integer, sA does *not* denote the set $\{sy \mid y \in Y\}$, instead, it denotes the iterated sumset (see notation for additive combinatorics, below).

We write

$$\text{diam}(Y) := \max_{y_1, y_2 \in Y} |y_1 - y_2|$$

for the *diameter* of the set Y .

We index into lists using square brackets: given a list L , $L[i]$ denotes its i th element, and $L[a : b]$ denotes the sublist $L[a], L[a+1], \dots, L[b]$.

Vectors. Given an n -element vector \vec{x} and a set $T \subseteq [n]$, we write \vec{x}_T to denote the projection of \vec{x} onto the indices given by T . We define the support $\text{supp}(\vec{x}) \subseteq [n]$ to be the set of nonzero coordinates of \vec{x} .

We use the \circ symbol to represent the operation of vector concatenation; for example $(a_1, a_2, a_3) \circ (b_1, b_2) = (a_1, a_2, a_3, b_1, b_2)$.

Given two m -dimensional vectors \vec{x} and \vec{y} , we use the center dot to denote the dot product

$$\vec{x} \cdot \vec{y} := x_1 y_1 + \cdots + x_m y_m.$$

For $a, b \in \mathbb{Z}_{\geq 0}^n$ 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_{lex} is a total order and that every set of vectors $S \subseteq \mathbb{Z}_{\geq 0}^n$ 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 i th row and j th column of A , respectively.

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

Error terms. Given a function or constant $\rho > 0$ we write $a \pm \rho$ as shorthand for the range $[a - \rho, a + \rho]$, as in the expression $x \in (1 \pm \rho)f(n)$. Likewise, for $\varepsilon \in (0, 1)$ and $b \geq 1$, we write $a \in b^{1 \pm \varepsilon}$ to indicate $a \in [b^{1-\varepsilon}, b^{1+\varepsilon}]$.

We use similar notation for asymptotic error terms: for example, $x \in f(n) \pm o(n)$ indicates that for any positive constant ε there exists n_0 such that $x \in f(n) \pm \varepsilon n$ for all $n > n_0$.

Norms. Given an n -dimensional vector \vec{x} , we write the p -norm of \vec{x} as

$$\|\vec{x}\|_p := \left(\sum_{i \in [n]} |x_i|^p \right)^{1/p}.$$

We make use of the Manhattan norm $\|\vec{x}\|_1$, the Euclidean norm $\|\vec{x}\|_2$, and the maximum norm $\|\vec{x}\|_\infty := \max_i |x_i|$.

When we apply a p -Norm to a matrix, we consider the matrix as a 1-dimensional array: for example, given $A \in \mathbb{Z}^{m \times n}$, $\|A\|_\infty$ denotes $\max_{i,j} A[i,j]$.

Given a real number r , we write $\|r\|_{\mathbb{R} \setminus \mathbb{Z}}$ to denote the distance from the nearest integer.

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

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

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

of all integer linear combinations of v_1, v_2, \dots, v_d , and each point in Λ is referred to as 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 $\text{vol}(T)$ to denote the volume of T .

Entropy. For $x, y \in [0, 1]$, we let

$$H(x) := -x \log_2(x) - (1 - x) \log_2(1 - x) \quad (2.1)$$

denote the binary entropy function, and write $H^{-1}(y)$ to denote the (smaller) pre-image of this function. Overloading notation, we also write

$$H(\alpha_1, \alpha_2, \dots, \alpha_m) := - \sum_{i \in [m]} \alpha_i \log_2(\alpha_i) \quad (2.2)$$

for the the entropy function on m arguments, which is well-defined for nonnegative inputs $\alpha_1, \alpha_2, \dots, \alpha_m$ satisfying $\sum_{i \in [m]} \alpha_i = 1$.

Stirling's Approximation. We use the following well-known consequences of Stirling's approximation, which states

$$n! = \Theta^* \left(\frac{n^n}{e^n} \right).$$

For each integer $j \in [0 : n/2]$, we have

$$\binom{n}{j} = \Theta(\sqrt{n} 2^{H(j/n)n}); \text{ moreover} \quad (2.3)$$

$$\sum_{i \leq j} \binom{n}{i} \leq 2^{H(j/n)n}. \quad (2.4)$$

Similar bounds hold for the multinomial coefficients, which are given by

$$\binom{n}{\alpha_1 n, \alpha_2 n, \dots, \alpha_m n} := \frac{n!}{(\alpha_1 n)! (\alpha_2 n)! \dots (\alpha_m n)!}$$

for nonnegative $\alpha_1, \alpha_2, \dots, \alpha_m$ such that $\sum_{i \in [m]} \alpha_i = 1$, assuming $\alpha_i n$ is an integer for every

$i \in [m]$. Substituting using Stirling's approximation yields

$$\binom{n}{\alpha_1 n, \alpha_2 n, \dots, \alpha_m n} = \Theta^* \left(\prod_{i \in [m]} \alpha_i^{-\alpha_i n} \right) \quad (2.5)$$

$$= \Theta^* \left(2^{H(\alpha_1, \alpha_2, \dots, \alpha_m)n} \right). \quad (2.6)$$

Birthday Paradox. The Birthday Paradox (named after the surprising fact that 23 people sampled at random are more likely than not to contain a pair sharing a birthday) formalizes the intuition that we are highly likely to see at least one collision after sampling $O(\sqrt{n})$ times from a set of cardinality n .

Lemma 1 (The Birthday Bound). *Let Y be a set of cardinality $|Y| = n$ and fix $a := a(n) > 0$. If we sample $a\sqrt{n}$ elements uniformly at random with replacement from Y , the probability we see some element twice is*

$$1 - e^{-\Omega(a^2)}.$$

Proof. For simplicity, condition on the event that we do not see a collision in our first $a\sqrt{n}/2$ samples from Y . In this event the probability we see no collisions in the second $a\sqrt{n}/2$ samples from Y is at most

$$\left(1 - \frac{a}{2\sqrt{n}} \right)^{\frac{a\sqrt{n}}{2}} \leq e^{-\frac{a^2}{4}}$$

by Bernoulli's inequality. □

Additive Combinatorics. Given an integer set A (or, more generally, any subset of a

group), the *sumset* is denoted

$$A + A := \{a_1 + a_2 \mid a_1, a_2 \in A\}.$$

The *doubling constant* of the (finite) set A is

$$\mathcal{C} = \mathcal{C}_A := \frac{|A + A|}{|A|},$$

with the subscript omitted when the set is clear from context. We write sA as shorthand for the iterated sumset $\underbrace{A + \dots + A}_s$.

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

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

defined by the integer *step vector* $\vec{y} = \{y_1, y_2, \dots, y_d\}$ and the integer dimension bounds L_1, L_2, \dots, L_d . We say that P has *dimension* d and *size* (or *volume*) $|P|$.

We can think of a GAP P as a d -dimensional parallelepiped projected onto the integer line using \vec{y} . We say that P is *proper* if $|P| = L_1 L_2 \dots L_d$, that is, if each point in the parallelepiped projects to a unique point on the line.

Subset Sums. Because we refer to sets of subset sums constantly throughout the thesis, we introduce some new notation. The choice of the *section sign* or *double-s* symbol (§) is intended to suggest the set of subset sums.

- $\S(Y)$ denotes the set of subset sums of Y ; that is, the set

$$\S(Y) = \{\Sigma(T) \mid T \subseteq Y\}.$$

Likewise, $\vec{\S}(Y)$ denotes the list containing the elements of $\S(Y)$ in sorted order from small to large.

- If we treat the input as a vector $\vec{x} = (x_1, x_2, \dots, x_n)$, we likewise write $\S(\vec{x})$ for the set of subset sums $\vec{x} \cdot \{0, 1\}^n$.
- In the Generalized Subset Sum problem, we may seek a solution $\vec{c} \in C^n$ satisfying $\vec{x} \cdot \vec{c} = t$ for some coefficient set $C \neq \{0, 1\}$. In that case, we overload notation and write

$$\S(\vec{x}) := \S_C(\vec{x}) = \vec{x} \cdot C^n$$

to denote the set of all sums that result from taking the dot product of the input vector \vec{x} and a coefficient vector from C^n .

- Finally, we extend our notation by writing

$$\S_\alpha(Y) := \{\Sigma(T) \mid T \subseteq Y, |T| \leq \alpha|Y|\}$$

to denote the set of subset sums of Y that can be made by adding together at most $\alpha|Y|$ elements of Y . For example, the Representation Method ([Section 1.1.2](#)) makes use of the *quartersums* $\S_{1/4}(Y)$ of Y , while [Chapter 4](#) makes use of the *halfsums* $\S_{1/2}(Y)$ of Y .

We will sometimes assume that, given an element $y \in \S(Y)$ or $\vec{\S}(Y)$, our algorithms can

recover a subset $Y' \subseteq Y$ such that $\Sigma(Y') = y$. This is without loss of generality when $\S(Y)$ is generated algorithmically, in which case we can create a data structure that stores one or more sets Y' with $\Sigma(Y') = y$ for each $y \in \S(Y)$ without changing the asymptotic runtime of the procedure.

2.2 Problem Statements

We use the term “Subset Sum” informally to refer to the family of algorithmic problems in which the input consists of a set X of numbers and a target t and the goal is to recover a subset of X that adds up to t . However, this description leaves certain parameters unspecified, including bounds (if any) on the size of the numbers, the group of which X is a subset (\mathbb{Z} , \mathbb{R} , or something else), whether the input may contain duplicate numbers, and the memory model of the computing machine used to solve the problem. This section specifies several Subset Sum problems as well as other closely related problems.

2.2.1 Generalized Subset Sum

The following problem generalizes many Subset Sum problems.

Problem 1: Generalized Subset Sum (GSS)**Input:**

- An integer range bound \hat{x} .
- An input vector $\vec{x} = (x_1, x_2, \dots, x_n)$ or multiset $X = \{x_1, x_2, \dots, x_n\}$ satisfying $x_i \in [0 : \hat{x} - 1]$ for all $i \in [n]$.
- A set $C \subset \mathbb{Z}$ of allowed coefficients.
- A target integer t , sometimes fixed at $t = 0$.

Output: A coefficient vector $\vec{c} \in C^n$ such that $\vec{c} \cdot \vec{x} = t$, if one exists, or ‘No’ if there is no solution. Equivalently, when the input is considered as the multiset X , we accept the submultiset $S \subseteq X$ with $\Sigma(S) = t$.

Writing the solution as an n -dimensional coefficient vector highlights the close connection between Subset Sum and Integer Programming, to which we will return in [Chapter 6](#).

GSS can be reduced to several other Subset Sum problems by choosing certain fixed values for the coefficient set C :

- If we set $C = \{0, 1\}$, we recover the standard **Subset Sum** problem. Hereafter, we refer to this problem as “Vanilla” Subset Sum to distinguish it from other variants.
- If we set $C = \{-1, 1\}$, we get the **Partition** problem (equivalent, via a simple translation, to Vanilla Subset Sum).
- If we set $C = \{-1, 0, 1\}$ and $t = 0$, we get the **Equal Subset Sum** problem (ESS).

Here, the $t = 0$ requirement can be relaxed; if $t = 0$, we typically disallow the trivial solution $\vec{c} = \vec{0}$.

- If we set $C = \mathbb{N}$, we get the **Unbounded Subset Sum** problem.

2.2.2 k -SUM

Perhaps the most familiar parameterization of Subset Sum is that in terms of the number of solution elements. This problem is known as k -SUM:

Problem 2: k -SUM

Input:

- An integer parameter k .
- An integer set $X = \{x_1, x_2, \dots, x_n\}$.
- A target integer t , sometimes fixed at $t = 0$.

Output: $S \subseteq X$ with $|S| = k$ such that $\Sigma(S) = t$, or ‘No’ if there is no solution.

There are at least two other common formulations of k -SUM.

- In **Multilist k -SUM**, k distinct lists of cardinality n are specified, and solutions must select one element from each list.
- In **Multiset k -SUM**, X may contain duplicate elements.

Multilist k -SUM can be reduced to k -SUM in time $O_k(n)$ and with an $O_k(1)$ -factor runtime blow-up, while Multiset k -SUM can be reduced to k -SUM in time $O_k(n \cdot \log n)$, at the cost of

an $O_k(\log n)$ -factor runtime blow-up. Both reductions use standard arguments. We include a proof of each reduction for completeness in [Appendices A.1](#) and [A.2](#).

2.2.3 \mathcal{C} -Subset Sum and (\mathcal{C}, k) -SUM

[Chapter 6](#) of this thesis considers the parameterization of Subset Sum and k -SUM in the doubling constant, a choice which quantifies the amount of “additive structure” in the input. Specifically, we consider the following variants of Subset Sum and k -SUM.

Problem 3: \mathcal{C} -Subset Sum

Input:

- An integer parameter \mathcal{C} .
- An integer set $X = \{x_1, x_2, \dots, x_n\}$ such that $|X + X| \leq \mathcal{C}|X|$.
- A target integer t , sometimes fixed at $t = 0$.

Output: $S \subseteq X$ such that $\Sigma(S) = t$, or ‘No’ if there is no solution.

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

- Integer parameters \mathcal{C} and k .
- An integer set $X = \{x_1, x_2, \dots, x_n\}$ such that $|X + X| \leq \mathcal{C}|X|$.
- A target integer t , sometimes fixed at $t = 0$.

Output: $S \subseteq X$ with $|S| = k$ such that $\Sigma(S) = t$, or ‘No’ if there is no solution.

2.3 Folklore and Utilities

A variety of folklore results, lemmas, reductions and transformations facilitate easier handling of the Subset Sum problem and allow us to make useful assumptions without loss of generality.

2.3.1 One-Sided Error, Instance Splitting, and Guessing Solution Sizes

A candidate solution to Subset Sum can be checked in linear time by simply adding up the numbers. This allows us to assume, at the cost (at worst) of an additional $O(n)$ factor in the runtime, that none of our algorithms return false positives. In other words, whenever we design a randomized algorithm to recover a Subset Sum solution, we may assume that it returns a correct solution or “Failure”, never an incorrect solution.

This in turn allows us to more easily substitute a set of sub-instances for the original

problem, a trick we will refer to as “Instance Splitting”. For example, suppose we know that a Subset Sum instance Y has a solution if and only if at least one of a family of subinstances $\mathcal{Y} = (Y_1, Y_2, \dots, Y_{f(n)})$ admits a solution. Because of the one-sided error property, if we have a randomized algorithm for Subset Sum, we can solve each subinstance in parallel and recover a solution to the original problem at the cost of an additional $f(n)$ factor in the runtime, without worrying about false positives.

As an example of this technique, consider the application of guessing solution size. Trivially, the Subset Sum instance (Y, t) admits a solution if and only if (Y, t) admits a solution of size s for at least one value $s \in [n]$. Suppose we have an algorithm that takes the solution size s as an input parameter and solves the Subset Sum problem, returning a correct solution only if there exists a solution of size s . We can then run this algorithm n times, once for every $s \in [n]$, effectively “guessing” the solution size while increasing runtime by an $O(n)$ factor. In this situation, we can assume that our algorithm correctly guesses the size of a solution without loss of generality.

2.3.2 Output-Linear Enumeration of $\vec{\xi}(Y)$

A basic primitive for our algorithms is the sorted list $\vec{\xi}(Y)$. As it happens, with a careful implementation, we can enumerate the elements of $\vec{\xi}(Y)$ in (almost) output-linear time. To begin, [Algorithm 2.1](#) displays an $O(2^{|Y|})$ -time folklore algorithm that enumerates $\vec{\xi}(Y)$:

Lemma 2 (Sorted Sum Enumeration; Folklore). *There exists an algorithm ([Algorithm 2.1](#)) that, given an integer (multi)set Y , enumerates the sorted list $\vec{\xi}(Y)$ in time $O(2^{|Y|})$.*

Proof. Refer to [Algorithm 2.1](#), which is a variation on a merge sort algorithm. Since

$$|\vec{\xi}(Y_{[i-1]})| = |\vec{\xi}'(Y_{[i-1]})|$$

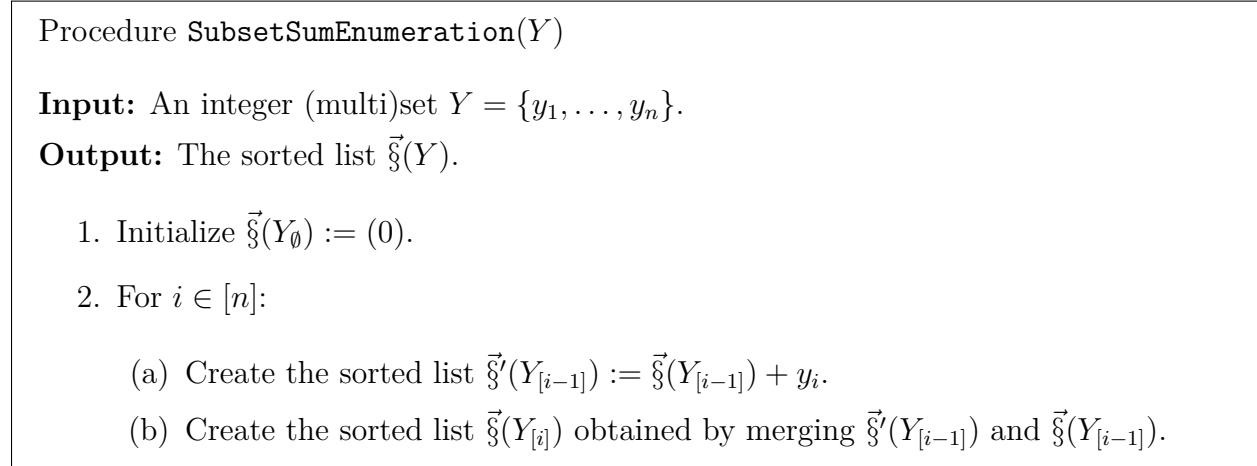


Figure 2.1: Efficiently enumerating $\vec{\S}(Y)$ given the (multi)set Y .

$$\begin{aligned} &\leq |\vec{\S}(Y_{[i]})| \\ &\leq 2|\vec{\S}(Y_{[i-1]})| \end{aligned}$$

for each $i \in [n]$, the runtime of [Algorithm 2.1](#) is

$$\begin{aligned} \sum_{i \in [n]} O(|\vec{\S}(Y_{[i-1]})| + |\vec{\S}(Y_{[i]})|) &= \sum_{i \in [n]} O(|\vec{\S}(Y_{[i]})|) \\ &= \sum_{i \in [n]} O(2^i) \\ &= O(2^n). \end{aligned} \quad \square$$

Moreover, using the same analysis it is easily verified that [Algorithm 2.1](#) runs in time $O^*(|\vec{\S}(Y)|)$ for any Y . This is useful when $\vec{\S}(Y)$ is very small.

Corollary 1 (Sorted Sum Enumeration for Very Small $|\vec{\S}(Y)|$). *[Algorithm 2.1](#) runs in time $O^*(|\vec{\S}(Y)|)$.*

For the later purposes of log shaving, we observe that we can shave polynomial factors from the runtime of [Algorithm 2.1](#) if $|\S(Y)|$ is smaller than 2^n by a $\text{poly}(n)$ factor:

Lemma 3 (Sorted Sum Enumeration for Small $|\S(Y)|$). *If Y is a multiset of $|Y| = n$ integers with $|\S(Y)| \leq 2^n \cdot n^{-\varepsilon}$ for some constant $\varepsilon > 0$, [Algorithm 2.1](#) runs in time $O(2^n \cdot n^{-\varepsilon} \cdot \log n)$.*

Proof. If $|\S(Y)| \leq 2^n \cdot n^{-(1+\varepsilon)}$, then because

$$|\vec{\S}(Y_{[i]})| \leq |\vec{\S}(Y_{[n]})| = |\S(Y)|$$

for each $i \in [n]$, it is easy to verify that the algorithm runs in time $O(n \cdot |\S(Y)|) = O(2^n \cdot n^{-\varepsilon})$.

Now suppose that $|\S(Y)| \geq 2^n \cdot n^{-(1+\varepsilon)}$. Using the bound $|\vec{\S}(Y_{[i]})| \leq 2^i$ for $i \leq \log |\S(Y)|$ and the bound $|\vec{\S}(Y_{[i]})| \leq |\S(Y)|$ for $\log |\S(Y)| < i \leq n$, the runtime of [Algorithm 2.1](#) is

$$\begin{aligned} \sum_{i \in [n]} O(|\vec{\S}(Y_{[i]})|) &= \sum_{i \leq \log |\S(Y)|} O(2^i) + \sum_{\log |\S(Y)| < i \leq n} O(|\S(Y)|) \\ &= O(|\S(Y)|) + (1 + \varepsilon) \cdot \log n \cdot O(|\S(Y)|) \\ &= O(|\S(Y)| \cdot \log n) \\ &= O(2^n \cdot n^{-\varepsilon} \cdot \log n). \quad \square \end{aligned}$$

Remark 1. [Lemma 3](#) gives a tight analysis of the algorithm when $|\S(Y)| = 2^n \cdot n^{-\varepsilon}$, as can be seen by considering the case of the n -element multiset

$$Y = (2^0, 2^1, \dots, 2^{n-\varepsilon \log n - 1}, 1, 1, \dots, 1).$$

2.3.3 Bounds on Solution Size

The Subset Sum problem (Y, t) admits a solution if and only if the “complement” problem $(Y, \Sigma(Y) - t)$ admits a solution. (This is because $\Sigma(Y') = t$ for some $Y' \subseteq Y$ if and only if $\Sigma(Y \setminus Y') = \Sigma(Y) - t$.) Because our algorithms may attempt to solve both the given problem and its complement, we can thus assume without loss of generality that some solution has size at most $n/2$ (if a solution exists). Note the casual use of Instance Splitting here.

Define a *balanced* Subset Sum solution to be $Y' \subseteq Y$ with $\Sigma(Y') = t$ and $|Y'| \in n/2 \pm o(n)$. Subset Sum instances with unbalanced solutions can be solved faster than Meet-in-the-Middle, and it is generally true that balanced instances are the hardest class of instances for Subset Sum problems, at least using current approaches.

A neat deterministic technique known as the *sliding window approach* (see, e.g., [DGIM02]), when combined with Instance Splitting, allows us to sample exactly half of an arbitrary input subset without loss of generality.

Lemma 4 (Sliding Window Sampling). *Fix a list X of length n and an arbitrary subset $A \subseteq X$. For $i \in [n]$, consider the partition family $(Y_i, Z_i)_{i \in [n]}$ given by*

$$Y_i := \{x_i, \dots, x_{i+n/2-1 \pmod n}\}, Z_i = X \setminus Y_i.$$

There exists $i \in [n]$ such that

$$|Y_i \cap A| \in \frac{|A|}{2} \pm 1/2.$$

Proof. Observe that

$$\mathbf{E}_{i \in [n]} [|Y_i \cap A|] = \sum_{i \in [n]} \frac{|Y_i \cap A|}{n} = \frac{|A|}{2}$$

and that $|Y_i \cap A|$ increases or decreases by at most 1 every time we increment i . Thus, to achieve the expectation, there must exist $i \in [n]$ satisfying the lemma condition. \square

Lemma 5 (Solving Unbalanced Subset Sum). *Fix a Subset Sum instance (X, t) . For any $a := a(n) \in (0, n/2]$, we can deterministically recover a Subset Sum solution of size at most a in time $O^*\left(\binom{n/2}{a/2}\right) = O(2^{H(a/n)n/2})$ if one exists.*

Proof. Using [Lemma 4](#) and instance splitting; select a partition (Y_i, Z_i) of X such that

$$|Y_i \cap S| \in \frac{|S|}{2} \pm 1/2.$$

Enumerate all at most $\frac{a+1}{2}$ -sized subsets of Y_i and Z_i , respectively, and check the resulting sorted lists of sums for a pair that sums to t . This takes time

$$\sum_{i \leq \frac{a+1}{2}} \binom{n/2}{i} \leq 2^{H(2j/n)n/2} = O(2^{H(a/n)n/2}).$$

by Stirling's Inequality ([2.4](#)). \square

We can also speed up the Meet-in-the-Middle approach if we can identify any set of size at most $n/2$ with very few subset sums. This idea goes back to [[AKKN15](#), [AKKN16](#)].

Lemma 6 (Solving Subset Sum with a Structured Subset (C.f. [[AKKN16](#)] Lemma 3.2)). *Fix a Subset Sum instance (X, t) , and suppose there exists a subset $Y \subseteq X$, known to the algorithm, of size $|Y| \leq n/2$ satisfying*

$$|\S(Y)| \leq 2^{|Y|} \cdot n^{-\varepsilon}$$

for some constant $\varepsilon > 0$. We can solve the instance in time $\tilde{O}(2^{n/2} \cdot n^{-\varepsilon/2})$.

Proof. Let A be any set of cardinality $\binom{n+\varepsilon \log n}{2}$ satisfying

$$Y \subseteq A \subseteq X.$$

We have

$$|\xi(A)| \leq 2^{|A \setminus Y|} \cdot |\xi(Y)| \leq 2^{n/2} \cdot n^{-\varepsilon/2}.$$

We also have

$$|\xi(X \setminus A)| \leq 2^{|X \setminus A|} = 2^{n/2} \cdot n^{-\varepsilon/2}.$$

By [Lemma 3](#), it takes time $O(2^{n/2} \cdot n^{-\varepsilon/2} \cdot \log n)$ to enumerate the sorted lists $\vec{\xi}(A)$ and $\vec{\xi}(X \setminus A)$ and run [Algorithm 1.1](#). □

2.3.4 Bounds on Input Size

If $t = 2^{o(n)}$, standard dynamic programming algorithms solve Subset Sum in time $O(nt) = 2^{o(n)}$ [[Bel66](#)], and these methods are easily adapted to GSS. Thus, when considering exponential-time algorithms for GSS, we can assume without loss of generality that $t = |C|^{\Omega(n)}$.

On the other hand, (Generalized) Subset Sum instances with extremely large input integers can be transformed, via standard reductions, into instances of size $t = |C|^{O(n)}$. We prove the following lemma:

Lemma 7 (Input Range Reduction (c.f. [[AKKN16](#)], Lemma 2.2)). *Fix a GSS instance given by (\hat{x}, \vec{x}, C, t) , for some*

$$\hat{x} = \underbrace{2^{2^{2^{\dots 2^n}}}}_{2^{o(n)}},$$

i.e., a power tower of height $2^{o(n)}$. For any $\varepsilon > 0$, there exists a poly(n)-time algorithm that

transforms (\hat{x}, \vec{x}, C, t) into a new GSS instance $(\hat{x}', \vec{x}', C, t')$ such that $\hat{x}' < 2^{(1+\varepsilon)n}$ and

$$\text{For all } c \in C^n, \vec{c} \cdot \vec{x}' = t' \text{ if and only if } \vec{c} \cdot \vec{x} = t$$

with probability $1 - e^{-\Omega(n)}$.

Proof. Define

$$s_{max} := n \cdot (\max_{c \in C} |c|) \cdot \hat{x},$$

which serves as an upper bound on $|\vec{x} \cdot \vec{c}|$ for any $\vec{c} \in C^n$. Sample a random prime

$$\mathbf{p} \sim [\log_2(s_{max}) \cdot 2^{(1+\varepsilon)n} : \log_2(s_{max}) \cdot 2^{(1+\varepsilon)n+1}].$$

Using \mathbf{p} , we define a new, random Subset Sum instance

$$(\vec{x}', \mathbf{t}') := (\vec{x}, t) \pmod{\mathbf{p}}.$$

Taking a union bound over the probability given in [Lemma 8](#), it follows that the chance that any sum $s \in \mathfrak{S}(\vec{x}) \setminus \{t\}$ satisfies $s = t \pmod{\mathbf{p}}$ is at most

$$|\mathfrak{S}(X)| \cdot \frac{\log_2(s_{max})}{\log_2(s_{max})2^{(1+\varepsilon)n}} \leq \frac{1}{2^{\varepsilon n}},$$

where we use the fact that $|\mathfrak{S}(X)| \leq 2^n$ for every set of cardinality n .

Because the new instance is created by taking remainders modulo \mathbf{p} , if $\vec{c} \cdot \vec{x} = t$ for some $\vec{c} \in C^n$, this implies $\vec{c} \cdot \vec{x}' = \mathbf{t}'$ deterministically, and thus we have $\vec{c} \cdot \vec{x}' = \mathbf{t}'$ if and only if $\vec{c} \cdot \vec{x} = t$ with probability $1 - e^{-\Omega(n)}$.

The new range bound satisfies

$$\widehat{\mathbf{x}}' = \mathbf{p} \leq \log_2(s_{max})2^{(1+\varepsilon)n},$$

which is $O^*(2^{(1+\varepsilon)n})$ for $s_{max} = 2^{n^{O(1)}}$. If s_{max} is still too large, we repeat the reduction $2^{o(n)}$ times and take a union bound over the probability that any iteration fails to complete the proof. \square

The particular statement of [Lemma 7](#) is designed to emphasize the fact that extremely large input sizes are no obstacle; for any remotely plausible input, we can reduce each integer to size $2^{O(n)}$ with very high probability. Moreover, the specific parameters of the lemma can be easily tweaked: we achieve an exponentially small error probability by reducing the input to size at most $2^{(1+\varepsilon)n}$ for an arbitrarily small epsilon, but input bounds of size $O^*(2^n)$ can be achieved with error probability $o(1)$.

This reduction does not affect our assumption of one-sided error: upon recovering \vec{c} such that $\vec{c} \cdot \vec{x}' = t'$, our algorithm can confirm that $\vec{c} \cdot \vec{x} = t$ before returning any answer (at least, as long as we can efficiently multiply the input integers).

2.3.5 Prime Hashing

A central step in the Representation Method involves sorting a set family according to the residue class of each set's sum, modulo a large prime number p . That is, we are given a set family $\mathcal{X} = (X_1, X_2, \dots, X_m)$ and would like to recover every $X_i \in \mathcal{X}$ such that $\Sigma(X_i) = r \pmod{p}$. This “prime hashing” operation serves to “filter” the set family while preserving some information about the sum of each set (its residue class modulo p).

In order for this step to work, it is important that the set family distributes “nicely”

over the residue classes: we want the elements of $|Y|$ to fall into many different classes. Fortunately, this event obtains with high probability as long as $\text{diam}(|Y|)$ is not too large. The following folklore result is closely related to the Prime Number Theorem:

Lemma 8 (Folklore). *For any sufficiently large integer r and any positive integer x , a prime p chosen uniformly at random from $[r : 2r]$ divides x with probability at most $\log_2(x)/r$.*

With this tool, we can derive a useful lemma regarding the distribution of sets over random primes:

Lemma 9 (Prime Distribution Lemma). *Given an integer set Y of n elements, fix an integer bound p_{\max} satisfying*

$$|Y| \log_2(\text{diam}(Y)) < n^k \cdot p_{\max} \tag{2.7}$$

for some positive constant k , and sample $\mathbf{p} \sim [p_{\max}, 2p_{\max}]$. With constant probability, the set of residue classes

$$\mathbf{R} := \left\{ r \in [\mathbf{p}] \mid |\{y \in Y : y = r \pmod{\mathbf{p}}\}| \geq \frac{|Y|}{2\mathbf{p}} \right\}$$

has cardinality $|\mathbf{R}| = \Omega(|Y|n^{-k})$.

Proof. Define the random set

$$\mathbf{P} := \{y_1 \neq y_2 \in Y \times Y \mid \mathbf{p} \text{ divides } |y_1 - y_2|\}.$$

We will refer to $(y_1, y_2) \in \mathbf{P}$ as a *colliding pair* with respect to \mathbf{p} .

It follows from [Lemma 8](#) that for any pair of distinct elements $y_1, y_2 \in Y$,

$$\Pr_{\mathbf{p}}[\mathbf{p} \text{ divides } |y_1 - y_2|] \leq \frac{\log_2(\text{diam}(Y))}{p_{\max}}.$$

By linearity of expectation over the elements of Y , the expected cardinality of \mathbf{P} is

$$\mathbf{E}_{\mathbf{p}}[|\mathbf{P}|] < |Y|^2 \frac{\log_2(\text{diam}(Y))}{p_{max}} \quad (2.8)$$

$$< |Y|n^k, \quad (2.9)$$

where the second line follows from (2.7). Thus

$$|\mathbf{P}| < 2 \cdot |Y|n^k \quad (2.10)$$

with probability at least $1/2$ by Markov's inequality.

Condition on the choice of a prime \mathbf{p} satisfying (2.10) and suppose for contradiction that \mathbf{R} has cardinality

$$|\mathbf{R}| < \frac{|Y|}{18 \cdot n^k}. \quad (2.11)$$

Together, the residue classes in $[\mathbf{p}] \setminus \mathbf{R}$ contain at most

$$\frac{|Y|}{2\mathbf{p}} \cdot \mathbf{p} = \frac{|Y|}{2}$$

elements of Y by definition, which means that residue classes in \mathbf{R} contain at least $|Y|/2$ elements of Y . Suppose these elements are distributed evenly among the residue classes in \mathbf{R} , thus minimizing collisions. It follows that the number of colliding pairs is at least

$$\begin{aligned} |\mathbf{P}| &\geq |\mathbf{R}| \cdot \frac{\binom{|Y|}{2|\mathbf{R}|}}{2} \cdot \left(\frac{|Y|}{2|\mathbf{R}|} - 1\right) \\ &= \frac{|Y|}{4} \cdot \left(\frac{|Y|}{2|\mathbf{R}|} - O(1)\right) \\ &\geq |Y| \cdot \left(\frac{18n^k}{8} - O(1)\right) \end{aligned}$$

$$= (2 + O(1))|Y| \cdot n^k. \tag{2.12}$$

Here, the first line counts the number of collisions (with the simplifying assumption that $|Y|/2$ divides evenly into $|R|$; this changes the calculation by a factor of at most $1 \pm o(1)$). The second line is elementary algebra and the third substitutes using (2.11). This contradicts (2.10), and thus (2.11) is false with constant probability, completing the proof of the lemma. \square

Taking the limit as $k \rightarrow 0$ gives the following corollary:

Lemma 10 (Second Prime Distribution Lemma). *Given an integer set Y of n elements, fix an integer bound p_{max} satisfying $|Y| \log_2(\text{diam}(Y)) < \gamma \cdot p_{max}$ for any positive constant γ , and sample $\mathbf{p} \sim [p_{max}, 2p_{max}]$. With constant probability, we have*

$$|Y \pmod{p}| = \Omega(|Y|).$$

Proof. Follow the proof of Lemma 10 with γ in place of n^k , and choose constants sufficiently large in (2.10) and (2.11) that (2.12) holds. \square

Chapter 3

Average-Case Algorithms for Subset Sum and Equal Subset Sum

This chapter uses material from [\[CJRS22\]](#). Although what follows has been refurbished for inclusion in this thesis, the arguments presented represent the collaborative efforts of the four original authors: Xi Chen, Yaonan Jin, Rocco A. Servedio, and I.

This chapter contains the following subsections:

- **Summary of Results.** Randomized algorithms for GSS on coefficient sets $C = [\pm d]$ and $[-d : d]$ and average-case input sampled from $[0 : \hat{x} - 1]^n$.
- **Structural Results.** Theorems characterizing when we should and should not expect GSS solutions to occur in the average case, parameterized in terms of the input range bound \hat{x} and the coefficient set C .
- **Algorithmic Results.** A description of our algorithmic approach, including analyses of correctness and runtime.

- **Generalized Number Balancing.** Corollary results for Generalized Number Balancing, effectively an optimization variant of GSS, in the average case.

In light of the broad body of work that has been done on the average-case version of the original Subset Sum problem, it is natural to consider average-case Generalized Subset Sum (GSS), in which each component of the input vector \vec{x} is selected uniformly and independently at random. The most natural candidate for the input range is the set $[0 : \hat{x} - 1]^n$, where $\hat{x} := \hat{x}(n) = 2^{\Theta(n)}$ is the bound on the input size.¹

There are two natural average-case variants of the GSS problem: in the first variant, the target value is obtained by sampling a subset of the input at random, thus creating a “hidden” solution. Thus every instance of this average-case problem variant admits a solution. This variant of the problem is motivated by cryptographic applications, and corresponds to the average-case variant of Subset Sum studied by [HGJ10, BCJ11] and others. We refer to this variant as the “cryptographic version” of the average-case GSS problem. In the second version, which we consider in this section, the target is a fixed value independent of the random input vector \vec{x} . This “balancing version” of the problem is the more natural generalization of Equal Subset Sum (in which we typically consider the fixed target 0). Since both Yes-instances and No-instances are possible for this variant, a natural goal that arises in its study is to understand the probability (as a function of the various parameter settings) that a solution exists. Structural questions of this type have in fact been the subject of considerable study for the special case of $C = \{\pm 1\}$; see for example [BCP01, Lue98].

¹Subset Sum on smaller inputs can be solved in subexponential time via dynamic programming, while Subset Sum on superexponential inputs can be effectively reduced to Subset Sum on singly-exponential inputs (Lemma 7).

In the context of the current thesis, this chapter serves as a detailed introduction to the use of the Representation Method (Section 1.1.2); in what follows, we generalize the method to average-case GSS. In doing so, we hope to further the reader’s intuitions about the technique.

Our structural results provide some facts about the “typical” form of the set of subset sums $\mathfrak{S}(\vec{x})$: when the inputs are sampled uniformly at random over a range of size $\hat{x} := \hat{x}(n)$, we expect to see solutions and a rapid increase in the number of collisions (pairs of input subsets with the same sum) when $\hat{x} \ll |C|^n$. This contrasts sharply with the set of worst-case instances that appear to be the most difficult, in which we see collisions early and often, and motivates the strategy of attempting to exploit this structure to develop better worst-case algorithms. In subsequent chapters we will depart from such “typical” input sets and consider those in which $\mathfrak{S}(\vec{x})$ has various kinds of significant structure.

3.1 Summary of Results

We consider Generalized Subset Sum² (GSS) supported on two families of coefficient sets symmetric about 0: $[\pm d]$ and $[-d : d]$, where d is an integer constant. As noted in Section 2.2.1, this choice of coefficient sets generalizes the problems Partition and Equal Subset Sum. Via a simple transformation, $[\pm d]$ also generalizes Vanilla Subset Sum: while the coefficient set for this problem is $C = \{0, 1\}$, a Vanilla Subset Sum instance \vec{x} with target τ can be easily converted to GSS on $[\pm 1]$ by setting a new target $\tau' := 2\tau - \sum_{i \in [n]} x_i$. Using similar transformations, the results in this chapter can be extended to scale multiples and translations of $[\pm d]$ and $[-d : d]$.

²As formally defined in Section 2.2.1.

Our main algorithmic result for average-case GSS is as follows.

Theorem 1 (Algorithm for Average-Case GSS). *Fix any $d \in \mathbb{N}$ and let $C = [\pm d]$ or $[-d : d]$. For any constant $\varepsilon > 0$, there is a randomized algorithm for average-case GSS with running time $O^*(|C|^{\Lambda(|C|)^{n+\varepsilon n}})$ ³ where*

$$\Lambda(z) := \max \begin{cases} 1 - \frac{z+1}{2z} \log_z(z+1) + \frac{1}{z} \log_z(2) \\ \frac{2}{3} - \frac{z+1}{3z} \log_z\left(\frac{z+1}{2}\right) \end{cases}. \quad (3.1)$$

For any range bound $\hat{x} := \hat{x}(n)$ and any target τ with $|\tau| = o(n\hat{x})$, the algorithm succeeds on the GSS instance $(\hat{x}, \tau, \vec{\mathbf{x}})$ with probability at least

$$\begin{aligned} 1 - e^{-\Omega(n)} & \text{ when } C = [-d : d] \text{ and} \\ 1 - o(1) & \text{ when } C = [\pm d], \end{aligned}$$

over the draw of $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$ and the randomness of the algorithm.

A runtime of $|C|^{\alpha n}$ for GSS should be thought of as analogous to a runtime of $2^{\alpha n}$ for Subset Sum. We note that $\Lambda(z) = 0.5 - \Omega(1/z)$, and thus our algorithm beats the Meet-in-the-Middle runtime of $O^*(|C|^{0.5n})$ by an exponential margin for every constant d . [Figure 3.1](#) plots the function Λ and [Table 1.3](#) lists our algorithm's runtime on various coefficient sets.

As a special case of [Theorem 1](#), we obtain an average-case algorithm for Equal Subset Sum that significantly improves on the $O^*(3^{0.488n})$ worst-case runtime of [\[MNPW19\]](#):

Corollary 2 (Average-Case Equal Subset Sum). *There exists an algorithm that solves*

³Note that the runtime of our algorithm is independent of the input bound M . This occurs because the probability of a Yes-instance is exponentially small when $M = 2^{\omega(n)}$.

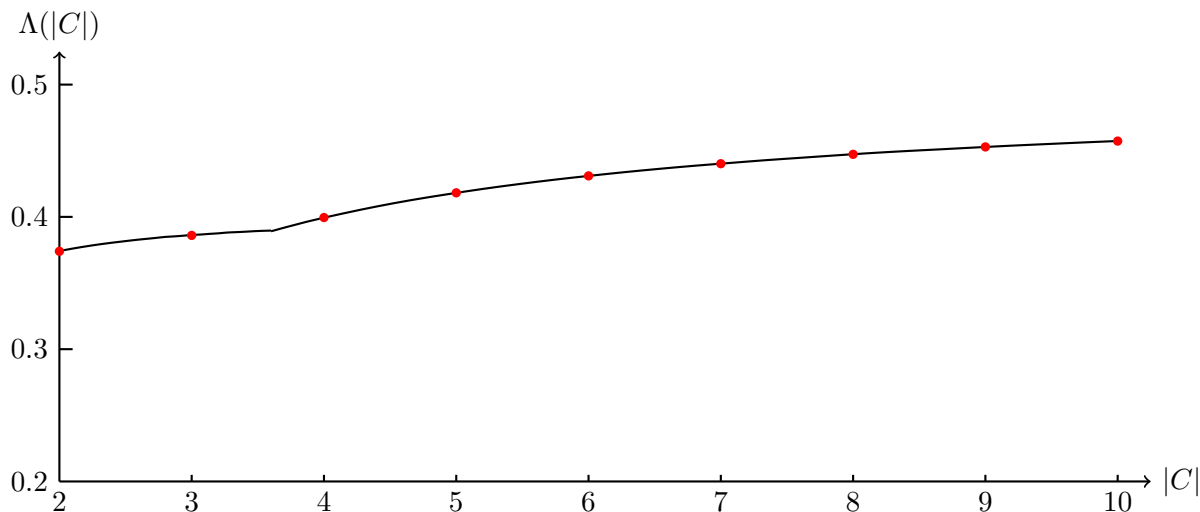


Figure 3.1: Plot of Λ . The red points plot $\Lambda(z)$ for $z \in [2 : 10]$.

$d \in \mathbb{N}$		1	2	5	10
Runtime	$[\pm d]$	$O^*(C ^{0.375n})^\dagger$	$O^*(C ^{0.400n})$	$O^*(C ^{0.458n})$	$O^*(C ^{0.479n})$
	$[-d : d]$	$O^*(C ^{0.387n})$	$O^*(C ^{0.419n})$	$O^*(C ^{0.462n})$	$O^*(C ^{0.480n})$

[†]Our $[\pm 1]$ case differs from the “cryptographic” average-case Subset Sum problem considered in [HGJ10, BCJ11, Böh11, BBSS20], because we consider the “balancing” problem (for which a solution may or may not exist).

Table 3.1: Runtime of our algorithm for average-case GSS on various coefficient sets.

Average-Case Equal Subset Sum in time $O(3^{0.387n})$ with success probability $1 - e^{-\Omega(n)}$.

Our algorithm has the additional property that it runs faster on *dense* instances, i.e., ones for which \hat{x} is substantially less than $|C|^n$. Intuitively, this is possible because in this regime there are likely to be many solutions.

Theorem 2 (Average-Case GSS on Dense Instances). *Fix $d \in \mathbb{N}$, $C = [\pm d]$ or $[-d : d]$, $\hat{x} = |C|^{\alpha n + o(n)}$ for some $\alpha \in (0, 1)$ and a target τ with $|\tau| = o(\hat{x}n)$. For any constant $\varepsilon > 0$,*

there exists an algorithm that solves average-case GSS in time

$$O^*(|C|^{\alpha\Lambda(|C|)n+\varepsilon n}),$$

where Λ is as defined as in (3.1) and plotted in Figure 3.1. The algorithm succeeds with probability at least

$$\begin{aligned} 1 - e^{-\Omega(n)} & \text{ when } C = [-d : d] \text{ and} \\ 1 - o(1) & \text{ when } C = [\pm d], \end{aligned}$$

over the draw of $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$ and the randomness of the algorithm.

That is, we achieve the same result as Theorem 1, but with the exponent in the runtime scaled down by a factor of α .

The proof of Theorem 2 is similar to the proof of Theorem 1, with the following observation enabling a speedup: if $\hat{x} \leq |C|^{(1-\varepsilon)n}$ for any $\varepsilon > 0$, then we can reduce the instance size to n' so that \hat{x} falls inside the window $|C|^{(1\pm\varepsilon)n'}$ and Algorithm 3.9 applies. Moreover, shrinking the instance size results in a faster running time. Theorems 1 and 2 are proved in Section 3.3 below.

Crucial ingredients underlying our algorithms are new structural results on the probability that random GSS instances have solutions. The following result identifies the regimes in which GSS instances on $C = [-d : d]$ are very likely, and very unlikely, to have solutions in the average case.

Theorem 3 (GSS Solution Probability for $C = [-d : d]$). *Let $C = [-d : d]$ for some fixed $d \in \mathbb{N}$, and fix any constant $\varepsilon > 0$. For $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$ and any integer τ satisfying*

$|\tau| = o(\hat{x}n)$, we have

$$\Pr_{\vec{x}} \left[\exists \vec{c} \in C^n \setminus \{\vec{0}\}^4 : \vec{x} \cdot \vec{c} = \tau \right] \begin{cases} \geq 1 - e^{-\Omega(n)} & \text{if } \hat{x} \leq |C|^{(1-\varepsilon)n} \\ \leq |C|^n / \hat{x} & \text{if } \hat{x} \geq |C|^n. \end{cases}$$

Our proof of the previous theorem uses elementary methods and a novel combinatorial argument. We also prove an analogous result for the $C = [\pm d]$ case by a different method: in this case, we adapt the analysis of [BCP01], which results in slightly worse probability guarantees. (See [Corollary 3](#) and [Theorem 4](#), below.)

Finally, our algorithms can be used to find exponentially precise solutions to *Generalized Number Balancing*, essentially an optimization variant of GSS in which input integers are sampled from the real range $(0, 1)$ and the objective is to find a sum as close as possible to the target. This extension is presented in [Section 3.4](#).

3.2 Structural Results

This section presents our structural results, which characterize when average-case GSS instances are likely to have a solution. Before proceeding to the full proof, we present a high-level sketch. We begin with some intuition for the distribution of sums that are achievable using coefficient set C .

Recall that $\S(Y) = \{\Sigma(T) \mid T \subseteq Y\}$ denotes the set of subset sums of Y . Adapting this

⁴The $\vec{c} = \vec{0}$ case is special: if $\tau \neq 0$, $\vec{0}$ is never a solution, and if $\tau = 0$, $\vec{0}$ is always a solution (and typically ignored).

notion to GSS, with respect to an input vector \vec{x} and a coefficient set C we define the set

$$\S(\vec{x}) := \S_C(\vec{x}) = \{\vec{c} \cdot \vec{x} : \vec{c} \in C^n \setminus \{\vec{0}\}\},$$

the set of all sums achievable given the input \vec{x} and the coefficient set C . (For convenience, we exclude the trivial solution vector $\vec{c} = \vec{0}$ to distinguish between the case in which the target 0 can be achieved by a nontrivial solution and when it cannot.)

Recall that \vec{x}_T , for any $T \subseteq [n]$, denotes the projection of \vec{x} onto the indices given by T ; this will come in handy later, when we argue by induction over $T = [1], [2], \dots, [n]$.

Since the coefficient set C is symmetric about 0, all elements of the random set $\S(\vec{x})$ have magnitude $O(\hat{x}n)$. Standard concentration arguments suggest the intuition that $\S(\vec{x})$ should be “tightly concentrated around the origin” in the sense that most of its elements should have magnitude roughly $O(\hat{x}\sqrt{n|C|})$. We will eventually make this intuition precise.

The $C = [-d : d]$ case. Perhaps surprisingly, our structural results for the coefficient sets $[-d : d]$ and $[\pm d]$ are established using very different techniques. Consider the case of $C = [-d : d]$ first. We are able to show that for any positive constant $\varepsilon > 0$, given $\hat{x} \leq |C|^{(1-\varepsilon)n}$ and any fixed integer offset τ with $|\tau| = o(\hat{x}n)$, the probability (over a uniform random $\vec{x} \sim [0 : \hat{x} - 1]^n$) that there exists a solution $\vec{c} \cdot \vec{x} = \tau$ with $\vec{c} \in C^n$ is *exponentially* close to 1. This extremely high probability that there exists a solution translates directly into the success probability of our algorithms in [Theorem 1](#) and [Theorem 2](#).

To establish this structural result, we employ a novel inductive proof. We analyze how the size of the set $|\S(\vec{x}_{[\ell]})|$ increases as a function of ℓ by considering the experiment in which $\mathbf{x}_1, \mathbf{x}_2, \dots$ are drawn in succession. We first argue that with very high probability $|\S(\vec{x}_{[\ell]})|$

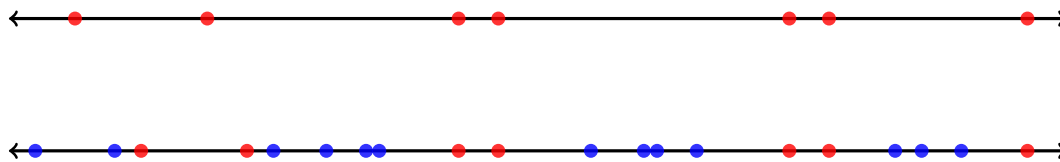


Figure 3.2: While $\mathfrak{S}(\vec{\mathbf{x}}_{[\ell]})$ (red set) is sparse, it grows quickly: $|\mathfrak{S}(\vec{\mathbf{x}}_{[\ell+1]})|$ (blue set) has cardinality nearly $|C| \cdot |\mathfrak{S}(\vec{\mathbf{x}}_{[\ell]})|$ with high probability.

increases rapidly with ℓ until, at some value $m < n$, it reaches a point at which it is dense on at least one “large” interval that is “close to” the origin. We then argue that at this point it suffices to draw a few more elements to ensure that some *partial* solution

$$\sum_{i \in [m']} c_i \mathbf{x}_i$$

will hit the offset τ with very high probability, where $m \leq m' \leq n$. The remaining elements $\mathbf{x}_{m'+1}, \dots, \mathbf{x}_n$ are simply assigned the 0 coefficient to complete the overall solution.

Figures 3.2 and 3.3 illustrate the two principles at work.

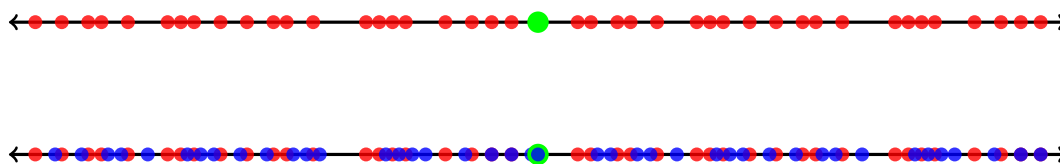


Figure 3.3: When $\mathfrak{S}(\vec{\mathbf{x}}_{[\ell]})$ (red set) is dense, $|\mathfrak{S}(\vec{\mathbf{x}}_{[\ell+1]})|$ (blue set) hits any fixed target point (green) with nontrivial probability.

The $C = [\pm d]$ case. When $C = [\pm d]$, the absence of the 0 coefficient is a fundamental obstacle to the previous approach, and indeed we do not know how to achieve an

exponentially high success probability for $C = [\pm d]$. Instead, to handle this case we use a very different proof strategy that extends the approach of [BCP01], who analyzed the $[\pm d]$ case for $d = 1$. Their analysis (see [BCP01, Theorem 2.1]) shows that for any $\hat{x} \leq 2^{(1-\varepsilon)n}$, the probability that a uniform random input $\vec{x} \sim [0 : \hat{x} - 1]^n$ admits a “perfect” solution $\vec{c} \in \{\pm 1\}^n$ is $1 - o(1)$, where a “perfect” solution is one satisfying $\vec{c} \cdot \vec{x} = 1$ if $\sum_{i \in [n]} x_i$ is odd and satisfying $\vec{c} \cdot \vec{x} = 0$ if $\sum_{i \in [n]} x_i$ is even.

We establish a similar result for $d > 1$. Specifically, we show that if $\hat{x} \leq |C|^{(1-\varepsilon)n}$ then given any fixed integer offset τ with $|\tau| = o(\hat{x}n)$, the probability (over a uniform draw of \vec{x} from $[0 : \hat{x} - 1]^n$) that there exists a solution $\vec{c} \cdot \vec{x} = \tau$ with $\vec{c} \in C^n$ is $1 - o(1)$. At a high level, our analysis establishing this follows the arguments of [BCP01]; we write the number of solutions as a random integral over all coefficient vectors, then bound suitable integrals to characterize the first moment and upper bound the second moment of the relevant random variable. Generalizing the proof strategy of [BCP01] to $[\pm d]$ significantly complicates the analysis but does not substantially change the underlying intuition.

Notably, [BCP01] confronted a parity issue in the case $C = [\pm 1]$: no solution summing to a target τ that has parity different from the sum of inputs is possible. Our analysis explains the parity issue as a result of constructive interference in the integrand of the solution-counting function and demonstrates that no such issues occur for $[\pm d]$ when $d > 1$ (compare the $\vec{x} \cdot \vec{c} \in \{\tau, \tau + 1\}$ in [Corollary 3](#) and $\vec{x} \cdot \vec{c} = \tau$ in [Theorem 4](#)).

This proof approach has the advantage that it neatly bounds the first and second moments of the number of solutions, not just the probability that a solution exists. However, the probability that a solution exists in the $\hat{x} \leq |C|^{(1-\varepsilon)n}$ regime will correspond to the failure probability of our algorithm later, so for our purposes the exponentially small failure probability given by [Theorem 3](#) is more valuable.

3.2.1 When Solutions Occur in the $C = [-d : d]$ Case

When the coefficient set contains 0, we analyze the probability that a solution exists by considering the growth rate of the sequence $|\mathfrak{S}(\vec{x}_{[1]})|$, $|\mathfrak{S}(\vec{x}_{[2]})|$, $|\mathfrak{S}(\vec{x}_{[3]})|$, etc. Once we come upon a set $\mathfrak{S}(\vec{x}_{[q]})$ that contains τ , the existence of a solution is ensured since any remaining input elements can be assigned the 0 coefficient. The structural result that we prove in this way is analogous to [Theorem 4](#) (the $C = [\pm d]$ case), but with exponentially small failure probability for small \hat{x} .

Upper Bound. We first prove the upper bound in the case that $\hat{x} \geq |C|^n$, which is the simpler half of the proof. Fix a coefficient vector $\vec{c} \in C^n \setminus \{\vec{0}\}$, and let $i^* \in [n]$ be an index such that $c_{i^*} \neq 0$. Conditioning on any outcome of $\vec{x}_{[n] \setminus \{i^*\}}$ from $[0 : \hat{x} - 1]^{n-1}$, $\vec{x} \cdot \vec{c} = \tau$ occurs with probability at most $1/\hat{x}$ over the selection of x_{i^*} . Union-bounding over all $\vec{c} \in C^n \setminus \{\vec{0}\}$ gives the claimed upper bound of $|C|^n/\hat{x}$.

Lower Bound. In the rest of the section, we focus on the case in which

$$\hat{x} \leq |C|^{(1-\varepsilon)n} \tag{3.2}$$

for some positive constant $\varepsilon > 0$.

Recall the definition of the set

$$\mathfrak{S}(\vec{x}) := \{\vec{c} \cdot \vec{x} : \vec{c} \in C^\ell \setminus \{\vec{0}\}\},$$

and observe that

$$\mathfrak{S}(\vec{x}_{[1]}) \subseteq |\mathfrak{S}(\vec{x}_{[2]})| \subseteq \cdots \subseteq |\mathfrak{S}(\vec{x}_{[n]})|.$$

That is, because $0 \in C$ by assumption, the set of subset sums achievable using the elements of $\vec{x}_{[\ell]}$ and the coefficient set C strictly grows as ℓ increases. Furthermore, \vec{x} admits a solution if and only if $\mathfrak{S}(\vec{x}_{[n]}) = \mathfrak{S}(\vec{x})$ contains τ .

Define the integer quantities

$$m := \lceil \rho n + \log_{|C|} \hat{x} \rceil \quad \text{and} \quad m' := \left\lceil \left(1 - \frac{\varepsilon}{3}\right) n \right\rceil, \quad (3.3)$$

$$\text{where the small constant } \rho := \frac{\varepsilon^2}{256d^2 \ln |C|}. \quad (3.4)$$

$\ell = m$ and $\ell = m'$ will mark important thresholds for $\mathfrak{S}(\vec{x}_{[\ell]})$, and their precise values are set for our later convenience in the proof of [Theorem 3](#). The first-time reader can ignore the exact values for now and simply keep in mind the rough size of each quantity:

$$m < \left(1 - \frac{5\varepsilon}{6}\right) n < \left(1 - \frac{\varepsilon}{3}\right) n \leq m' < n. \quad (3.5)$$

(Here, the first inequality follows from substituting for \hat{x} using [\(3.2\)](#).)

Our proof proceeds in three steps:

1. Show that with high probability over the draw of the first m elements, $\mathfrak{S}(\vec{x}_{[m]})$ occupies a constant fraction of some length- \hat{x} interval “close to” from the origin.

To formalize the notion of an interval “far” from the origin, given $\vec{x} \in [0 : \hat{x} - 1]^\ell$, we define the set of *large* values achievable with \vec{x} as

$$L(\vec{x}) := \left\{ s \in \mathfrak{S}(\vec{x}) : |s| \geq \frac{\varepsilon}{8} n \hat{x} \right\}. \quad (3.6)$$

This set will contain a small fraction of $\mathfrak{S}(\vec{x})$ with very high probability.

Lemma 11 (Dense Interval Lemma). *With probability $1 - e^{-\Omega(n)}$ over*

$\vec{x} \sim [0 : \hat{x} - 1]^m$, $\S(\vec{x}) \setminus L(\vec{x})$ occupies at least γ -fraction of some length- \hat{x} interval, where

$$\gamma := \frac{\rho^2}{48|C|^2}. \quad (3.7)$$

In other words, there exists a length- \hat{x} interval I such that

$$|I \cap (\S(\vec{x}) \setminus L(\vec{x}))| \geq \gamma \hat{x}.$$

The proof follows below.

2. Show that, if $\S(\vec{x}_{[m]})$ occupies a constant fraction of some length- \hat{x} interval close to the origin, with very high probability $\S(\vec{x}_{[m']})$ occupies a constant fraction of a length- \hat{x} interval containing τ .

Lemma 12 (Interval Shifting Lemma). *Let $\vec{x} \in [0 : \hat{x} - 1]^m$ be such that $\S(\vec{x}) \setminus L(\vec{x})$ occupies a γ -fraction of a length- \hat{x} interval, and let $\vec{y} \sim [0 : \hat{x} - 1]^{m'-m}$.*

With probability at least $1 - e^{-\Omega(n)}$, $\S(\vec{x} \circ \vec{y})$ occupies a γ -fraction of some length- \hat{x} interval containing τ .

The proof follows below.

3. Show that, if $\S(\vec{x}_{[m']})$ occupies a constant fraction of a length- \hat{x} interval containing τ , there exists a solution with very high probability: $\tau \in \S(\vec{x}_{[n]})$.

We begin with Step 3: completing the proof of [Theorem 3](#) assuming [Lemmas 11](#) and [12](#). We then prove the lemmas in the subsequent two sections.

Proof of Theorem 3. Combining Lemma 11 and Lemma 12, we have that with probability at least $1 - e^{-\Omega(n)}$ over $\vec{x} \sim [0 : \hat{x} - 1]^{m'}$, $\mathfrak{S}(\vec{x})$ contains at least a γ -fraction of some length- \hat{x} interval containing the target τ .

Fix an $\vec{x} \in [0 : \hat{x} - 1]^{m'}$ satisfying this property, and consider the draw of the remaining $n - m' = \Omega(n)$ random elements: $\mathbf{x}_{m'+1}, \dots, \mathbf{x}_n \sim [0 : \hat{x} - 1]$. If, for any $i \in [m' + 1 : n]$, it occurs that $\tau + \mathbf{x}_i$ or $\tau - \mathbf{x}_i = s \in \mathfrak{S}(\vec{x})$, we are done: we can recover a solution by setting the coefficients $c_1, \dots, c_{m'}$ to achieve s , setting $c_j = 0$ for all $j \in [m' + 1 : n] \setminus \{i\}$, and setting c_i to $+1$ or -1 to complete the solution.

The event that $\tau + \mathbf{x}_i$ or $\tau - \mathbf{x}_i \in \mathfrak{S}(\vec{x})$ occurs with probability at least $\gamma/2$, as τ is contained in a γ -dense interval of length at least \hat{x} and $\mathbf{x}_i \in [0, \hat{x} - 1]$. The occurrence of this event is independent over each $i \in [m' + 1 : n]$, and thus the chance that no solution exists is at most $(1 - \gamma/2)^{\Omega(n)} = e^{-\Omega(n)}$. The result follows from a union bound over this failure probability and the failure probabilities from Lemmas 11 and 12. \square

Proof of Lemma 11 (Dense Interval Lemma)

Our first step in proving Lemma 11 is to bound the fraction of $\mathfrak{S}(\vec{x})$ that consists of large values. This follows easily for all $\vec{x} \in [0 : \hat{x} - 1]^\ell$ from a concentration inequality:

Claim 1. For any $\ell \in [n]$ and $\vec{x} \in [0 : \hat{x} - 1]^\ell$, we have

$$|L(\vec{x})| \leq |C|^\ell \cdot 2 \exp\left(-\frac{\varepsilon^2 n}{128d^2}\right) = |C|^\ell \cdot 2|C|^{-2\rho n}.$$

Proof. For $\ell \geq 1$, define independent random variables $\mathbf{y}_1, \dots, \mathbf{y}_\ell$ such that

$$\Pr[\mathbf{y}_i = cx_i] = \frac{1}{|C|}$$

for each $c \in C$. We have

$$\frac{|L(\vec{x})|}{|C|^\ell} \leq \Pr_{\mathbf{y}_1, \dots, \mathbf{y}_\ell} \left[\left| \sum_{i \in [\ell]} \mathbf{y}_i \right| \geq \frac{\varepsilon n \hat{x}}{8} \right],$$

from which the claim follows by Hoeffding's inequality. \square

Additionally, we would like to show that if $\S(\vec{x}_{[\ell]})$ is *not* γ -dense near the origin, $\S(\vec{x}_{[\ell+1]})$ is likely to be larger than $\S(\vec{x}_{[\ell]})$ by a factor of almost $|C|$.

Claim 2. *Let $\ell \in [m]$ and let $\vec{y} \in [0 : \hat{x} - 1]^{\ell-1}$ be a vector such that $|I \cap (\S(\vec{y}) \setminus L(\vec{y}))| < \gamma \hat{x}$ for every length- \hat{x} interval I . For $\mathbf{w} \sim [0 : \hat{x} - 1]$, we have*

$$\frac{|\S(\vec{y} \circ \mathbf{w})|}{|\S(\vec{y})|} \geq (1 - \rho/4)|C|$$

with probability at least $1 - \rho/4$, where the constant ρ is as defined in (3.4).

Proof. Fix an $\ell \in [m]$ and $\vec{y} \in [0 : \hat{x} - 1]^{\ell-1}$ such that $|I \cap (\S(\vec{y}) \setminus L(\vec{y}))| < \gamma \hat{x}$ for every length- \hat{x} interval I .

Sample $\mathbf{w} \sim [0 : \hat{x} - 1]$, and consider the random variable

$$|C| \cdot |\S(\vec{y})| - |\S(\vec{y} \circ \mathbf{w})|.$$

This variable is nonnegative, as each element in $\S(\vec{y})$ gives rise to at most $|C|$ elements in $\S(\vec{y} \circ \mathbf{w})$. If it is small, this means that the set of subset sums of y “grows” by a factor of nearly $|C|$ when \vec{y} is augmented with the input element \mathbf{w} . So we begin by proving an upper bound on the expectation

$$\mathbf{E}_{\mathbf{w}} [|C| \cdot |\S(\vec{y})| - |\S(\vec{y} \circ \mathbf{w})|]. \quad (3.8)$$

We say that two pairs $(s, c) \neq (s', c') \in \mathfrak{S}(\vec{y}) \times C$ *collide* if $s + c\mathbf{w} = s' + c'\mathbf{w}$; that is, if the two pairs indicate two ways to make the same sum in $\mathfrak{S}(\vec{y} \circ \mathbf{w})$. We can bound (3.8) by counting the number of collisions that occur. In particular, we observe that we would have $|\mathfrak{S}(\vec{y} \circ z)| = |C| \cdot |\mathfrak{S}(\vec{y})|$ if there were *no* collisions, and the random quantity $|C| \cdot |\mathfrak{S}(\vec{y})| - |\mathfrak{S}(\vec{y} \circ \mathbf{w})|$ is thus upper-bounded by the total number of collisions that occur given \mathbf{w} .

Observe that a necessary condition for the collision

$$s + c\mathbf{w} = s' + c'\mathbf{w}$$

is that

$$|s - s'| \leq 2d\hat{x}. \quad (3.9)$$

This follows from the fact that $c, c' \in C = \{-d : d\}$ and $\mathbf{w} \in [0 : \hat{x} - 1]$. To bound the number of pairs $(s, c) \neq (s', c')$ for which (3.9) holds, observe that there are at most $|\mathfrak{S}(\vec{y})|$ possibilities for s and at most $|C|^2$ possibilities for c and c' . Moreover, by assumption $\mathfrak{S}(\vec{y}) \setminus L(\vec{y})$ does not occupy a γ fraction of any length- \hat{x} interval. Hence the interval

$$[s - 2d\hat{x}, s + 2d\hat{x}]$$

contains at most $4d\hat{x}\gamma$ many elements in $\mathfrak{S}(\vec{y}) \setminus L(\vec{y})$. Finally, this interval trivially has at most $|L(\vec{y})|$ many elements in $L(\vec{y})$. Thus for each $s \in \mathfrak{S}(\vec{y})$, there are at most

$$4d\hat{x}\gamma + |L(\vec{y})| \quad (3.10)$$

elements $s' \in \mathfrak{S}(\vec{y})$ such that $|s - s'| \leq 2d\hat{x}$.

As a result, the number of pairs $(s, c) \neq (s', c')$ with $|s - s'| \leq 2d\hat{x}$ is at most

$$|\xi(\vec{y})| \cdot |C|^2 \cdot (2|C|\hat{x}\gamma + |L(\vec{y})|),$$

which follows from union-bounding (3.10) over $\xi(\vec{y}) \times C \times C$.

Each pair $(s, c) \neq (s', c')$ satisfies $s + c\mathbf{w} = s' + c'\mathbf{w}$ with probability at most $1/\hat{x}$ over the choice of $\mathbf{w} \sim [0 : \hat{x} - 1]$. Thus

$$\mathbf{E}_{\mathbf{w}} [|C| \cdot |\xi(\vec{y})| - |\xi(\vec{y} \circ \mathbf{w})|] \leq \frac{|\xi(\vec{y})| \cdot |C|^2}{\hat{x}} \cdot (2|C|\hat{x}\gamma + |L(\vec{y})|) \quad (3.11)$$

$$\leq |\xi(\vec{y})| \cdot \frac{|C|^2}{\hat{x}} \cdot (2|C|\hat{x}\gamma + |C|^{\ell-1} \cdot 2|C|^{-2\rho n}) \quad (3.12)$$

$$\leq |\xi(\vec{y})| \cdot \frac{|C|^2}{\hat{x}} \cdot (2|C|\hat{x}\gamma + |C| \cdot |C|^m \cdot 2|C|^{-2\rho n}) \quad (3.13)$$

$$\leq |\xi(\vec{y})| \cdot \frac{|C|^2}{\hat{x}} \cdot (2|C|\hat{x}\gamma + |C|B\gamma) \quad (3.14)$$

$$\leq 3|C|^3\gamma \leq \frac{\rho^2}{16}|C|. \quad (3.15)$$

Here, the first line uses linearity of expectation, the second line follows from substituting for $|L(\vec{y})|$ using Claim 1, the third line follows from observing that $|C|^{\ell-1} \leq |C| \cdot |C|^m$ by assumption, the fourth line follows from substituting $m = \lceil \rho n + \log_{|C|} B \rceil$ and observing that $2|C|^{-\rho n} < \gamma$, and the final line follows from substituting the definition of γ (3.7).

Given that the random variable in the expectation is nonnegative, it follows from Markov's inequality that

$$\Pr_{\mathbf{w}} \left[\frac{|\xi(\vec{y} \circ \mathbf{w})|}{|\xi(\vec{y})|} < (1 - \rho/4)|C| \right] \leq \frac{\rho}{4},$$

completing the proof of the claim. \square

We now have all the ingredients required to prove Lemma 11.

Proof of Lemma 11. Consider the experiment in which we draw the inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \sim [0 : \hat{x} - 1]$ in turn. For each $\ell \in [m]$, let \mathcal{X}_ℓ denote the indicator random variable that is set to 1 if either

$$|I \cap (\S(\vec{\mathbf{x}}_{[\ell]}) \setminus L(\vec{\mathbf{x}}_{[\ell]}))| \geq \gamma \hat{x} \quad (3.16)$$

for some length- \hat{x} interval I (that is, we have a dense interval close to the origin) or if

$$\frac{|\S(\vec{\mathbf{x}}_{[\ell]})|}{|\S(\vec{\mathbf{x}}_{[\ell-1]})|} \geq (1 - \rho/4)|C|. \quad (3.17)$$

(that is, the set of subset sums grows as expected when we draw \mathbf{x}_ℓ).

Then, conditioning on any outcome of $(\mathbf{x}_1, \dots, \mathbf{x}_{\ell-1})$ it follows from Claim 2 that the probability of $\mathcal{X}_\ell = 1$ is at least $1 - \rho/4$. By a Chernoff bound, the probability that

$$\sum_{\ell \in [m]} \mathcal{X}_\ell \geq (1 - \rho/2)m$$

is $1 - e^{-\Omega(n)}$. We will show that when this event occurs we *must* have (3.16) for some $\ell \in [m]$ and length- \hat{x} interval I , which in turn implies the same for $\S(\vec{\mathbf{x}}_{[m]}) \setminus L(\vec{\mathbf{x}}_{[m]})$ given that $\S(\vec{\mathbf{x}}_{[\ell]}) \subseteq \S(\vec{\mathbf{x}}_{[m]})$ for $\ell \in [m]$.

To this end we assume for contradiction that (3.16) does not hold for any $\ell \in [m]$. This implies (3.17) for every ℓ such that $\mathcal{X}_\ell = 1$. Given that $\S(\vec{\mathbf{x}}_{[\ell-1]}) \subseteq \S(\vec{\mathbf{x}}_{[\ell]})$ for all $\ell \in [m]$, it follows that

$$|\S(\mathbf{x}_{[m]})| \geq ((1 - \rho/4)|C|)^{(1-\rho/2)m} \quad (3.18)$$

$$\geq (e^{-\rho/2}|C|)^{(1-\rho/2)m} \quad (3.19)$$

$$> |C|^m \cdot |C|^{-\rho m/2} e^{-\rho m/2} \quad (3.20)$$

$$= |C|^{\rho n + \log_{|C|} \hat{x}} |C|^{-\rho m/2} e^{-\rho m/2} \quad (3.21)$$

$$\geq |C|^{\rho n + \log_{|C|} \hat{x}} \cdot |C|^{-\rho m} \quad (3.22)$$

$$\geq \hat{x} \cdot |C|^{\rho(n-m)} = \hat{x} \cdot 2^{\Omega(n)}. \quad (3.23)$$

Here the second line uses the identity $1 - \rho/4 \geq e^{-\rho/2}$, which holds as $\rho < \frac{1}{256} < 1$ (3.4). The third line regroups and drops a factor of $\exp(\rho^2 m/4)$. The fourth line substitutes $m = \lceil \rho n + \log_{|C|} \hat{x} \rceil$ according to its definition (3.3), and the fifth line uses the fact that $e < 3 \leq |C|$.

On the other hand, by Claim 1 and substitution for m using (3.3), we have

$$|L(\vec{\mathbf{x}}_{[m]})| \leq |C|^m \cdot 2|C|^{-2\rho n} = \hat{x} \cdot 2|C|^{-\rho n+1} < \hat{x} \quad (3.24)$$

by Claim 1. Combining (3.18) and (3.24) yields

$$|\S(\vec{\mathbf{x}}_{[m]}) \setminus L(\vec{\mathbf{x}}_{[m]})| = \hat{x} \cdot 2^{\Omega(n)},$$

which contradicts the fact that $\mathbb{Z} \setminus L(\vec{\mathbf{x}}_{[m]})$ spans a range of only $\varepsilon n \hat{x}/4$ integers by definition (3.6).

Thus our assumption is false and there exists a length- \hat{x} interval I with $|I \cap (\S(\vec{\mathbf{x}}) \setminus L(\vec{\mathbf{x}}))| \geq \gamma \hat{x}$ with probability $1 - e^{-\Omega(n)}$. This finishes the proof of the Dense Interval Lemma. \square

Proof of Lemma 12 (Interval Shifting Lemma)

Finally, we prove that, given a γ -dense length- \hat{x} interval not too far from the origin, drawing an additional $m' - m$ inputs ensures that we occupy a γ -fraction of an interval containing τ with very high probability.

Proof of Lemma 12. Let $\vec{x} \in [0 : \hat{x} - 1]^m$ be a vector and $I = [\alpha, \alpha + \hat{x}]$ be a length- \hat{x} interval such that

$$J := I \cap (\S(\vec{x}) \setminus L(\vec{x}))$$

satisfies

$$|J| \geq \gamma \hat{x}. \quad (3.25)$$

Because I is γ -dense in elements within $\mathbb{Z} \setminus L(\vec{x})$, we can assume that

$$I \subseteq \mathbb{Z} \setminus L(\vec{x}) \subseteq \left[-\frac{\varepsilon}{8} n \hat{x}, \frac{\varepsilon}{8} n \hat{x} \right]$$

without loss of generality.

Assume without loss of generality that $\tau \notin I$ and $\alpha > \tau$. Since $\tau = o(\hat{x}n)$ by assumption, we have

$$\alpha - \tau \leq \left(\frac{\varepsilon}{8} + o(1) \right) n \hat{x}.$$

.

Now consider the sequence of random variables $\mathbf{x}_{m+1}, \dots, \mathbf{x}_{m'} \sim [0 : \hat{x} - 1]$. The expectation of the sum of these variables is given by

$$\mathbf{E} \left[\sum_{i \in [m+1:m']} \mathbf{x}_i \right] = \frac{\hat{x} - 1}{2} \cdot (m' - m) > \frac{\varepsilon}{5} n \hat{x},$$

as

$$m' - m > \frac{\varepsilon}{2}n$$

by (3.5). Furthermore, because the random variables are independent, we have that

$$\sum_{i \in [m+1:m']} \mathbf{x}_i > n\hat{x} \cdot \frac{\varepsilon}{6} \tag{3.26}$$

with probability $1 - e^{-\Omega(n)}$ by a Hoeffding bound. In other words, the sum of $m' - m$ random inputs is greater than the distance from α to τ with very high probability.

Conditioning on (3.26), let ℓ be the smallest index satisfying

$$\sum_{i \in [m+1:\ell]} \mathbf{x}_i > \alpha - \tau$$

and write

$$\beta := \sum_{i \in [m+1:\ell]} \mathbf{x}_i$$

to simplify notation. Thus

$$\tau \in [\alpha - \beta : \alpha - \beta + \hat{x}].$$

We claim that $\mathfrak{S}(\vec{x} \circ \mathbf{x}_{m+1} \circ \cdots \circ \mathbf{x}_{m'})$ occupies at least a γ -fraction of a length- \hat{x} interval containing τ . To see this, consider what happens when we assign the -1 coefficient to the variables $\mathbf{x}_{m+1}, \dots, \mathbf{x}_\ell$, effectively translating $\mathfrak{S}(\vec{x})$ by β . We have

$$J - \beta \subseteq [\alpha - \beta : \alpha - \beta + \hat{x}] \cap \mathfrak{S}(\vec{x} \circ \mathbf{x}_{m+1} \circ \cdots \circ \mathbf{x}_{m'}).$$

This completes the proof of the lemma. □

3.2.2 When Solutions Occur in the $C = [\pm d]$ Case

The $C = \{\pm 1\}$ case has been studied in previous work by Borgs, Chayes and Pittel [BCP01]. Their results precisely determine the parameters of a phase transition within a subexponential window around $2^n/\sqrt{n}$ and prove that solutions exist with probability $1 - o(1)$ for \hat{x} below the window and with probability $o(1)$ above. For our purposes, precisely pinning down the phase transition window is less important than establishing regions within which solutions are either very likely or very unlikely to exist, so we present a corollary of their analysis ([BCP01, Theorem 2.1]) which features a larger window but more flexibility in the offset.

Corollary 3 (GSS Solution Probability on $[\pm 1]$ [BCP01]). *Let $C = [\pm 1]$ and fix any positive constant $\varepsilon > 0$. For $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$ and any integer τ satisfying $|\tau| = o(\hat{x}n)$, we have*

$$\Pr_{\vec{\mathbf{x}}} [\exists \vec{c} \in C^n : \vec{\mathbf{x}} \cdot \vec{c} \in \{\tau, \tau + 1\}] \begin{cases} \geq 1 - o(1) & \text{if } \hat{x} \leq |C|^{(1-\varepsilon)n} \\ \leq 2|C|^n/\hat{x} & \text{if } \hat{x} \geq |C|^n. \end{cases}$$

(Note that $\vec{\mathbf{x}} \cdot \vec{c}$ has the same parity as $\sum_{i \in [n]} \mathbf{x}_i$ for every $\vec{c} \in [\pm 1]^n$, so the parity of $\sum_{i \in [n]} \mathbf{x}_i$ determines a single possible target in $\{\tau, \tau + 1\}$.)

Proof. Upper Bound. Let $C = [\pm 1]$. We first consider the upper bound when $\hat{x} \geq |C|^n = 2^n$. Given any fixed $\vec{c} \in C^n$, conditioned on any values for $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n-1}$, the probability that \mathbf{x}_n is such that $\vec{\mathbf{x}} \cdot \vec{c} \in \{\tau, \tau + 1\}$ is at most $2/\hat{x}$. Union-bounding over all coefficient vectors gives the result.

Lower Bound. Let $C = [\pm 1]$ and fix $\hat{x} \leq |C|^{(1-\varepsilon)n} = 2^{(1-\varepsilon)n}$ for some $\varepsilon > 0$. We start with the case in which $|\tau| \leq \hat{x}$ and then extend our analysis to all τ such that $|\tau| = o(\hat{x}n)$.

Fix a target τ such that $|\tau| \leq \hat{x}$. Let $\mathcal{Z}_{n,\tau}$ denote the expected number of solutions $\vec{c} \in C^n$ of $\vec{c} \cdot \vec{x} = \tau$ over $\vec{x} \sim [0 : \hat{x} - 1]^n$.

In this case, [BCP01] Proposition 3.1 together with $\hat{x} \leq 2^{(1-\varepsilon)n}$ implies the following bounds on $\mathcal{Z}_{n,\tau}$:

$$\begin{aligned} \mathbf{E}[\mathcal{Z}_{n,\tau}] &= \rho_n(1 + O(n^{-1})) \quad \text{and} \\ \mathbf{E}[\mathcal{Z}_{n,\tau}^2] &= 2\rho_n^2(1 + O(n^{-1})). \end{aligned}$$

where ρ_n is defined as

$$\rho_n := \sqrt{\frac{3}{2\pi n}} \cdot \frac{2^n}{\hat{x}}.$$

Note that we cannot directly apply Chebyshev's inequality to obtain concentration of $\mathcal{Z}_{n,\tau}$ because of the extra factor of 2 in $\mathbf{E}[\mathcal{Z}_{n,\tau}]^2$. (The factor of 2 is there because of the observation that $\mathcal{Z}_{n,\tau}$ can have a solution only when the sum of \vec{x} is even, which happens with probability 1/2 over \vec{x} .)

Since we are interested in the probability that there exists $\vec{c} \in C^n$ with $\vec{c} \cdot \vec{x} \in \{\tau, \tau + 1\}$, we have

$$\begin{aligned} \mathbf{E}[\mathcal{Z}_{n,\tau} + \mathcal{Z}_{n,\tau+1}] &= 2\rho_n(1 + O(n^{-1})) \quad \text{and} \\ \mathbf{E}[(\mathcal{Z}_{n,\tau} + \mathcal{Z}_{n,\tau+1})^2] &= 4\rho_n^2(1 + O(n^{-1})). \end{aligned}$$

Here, the second line uses the fact that $\mathbf{E}[\mathcal{Z}_{n,\tau}\mathcal{Z}_{n,\tau+1}] = 0$, because for any fixed \vec{x} and τ , it is impossible to have a solution solution for *both* τ and $\tau + 1$ due to parity. It follows from

Chebyshev's inequality that

$$\Pr_{\vec{x}}[\exists \vec{c} \in C^m \mid \vec{x} \cdot \vec{c} \in \{\tau, \tau + 1\}] = 1 - o(1) \quad (3.27)$$

It remains to consider the larger target range $|\tau| = o(n\hat{x})$ as in the theorem statement. Without loss of generality, consider $\tau > 0$ and consider the experiment in which we sample the input elements $\mathbf{x}_1, \mathbf{x}_2, \dots$, one by one and assign each the coefficient -1 . Each step of this process effectively creates a new instance with one fewer input element and a smaller target τ' . It is a simple exercise to show that for some $\tau = o(n\hat{x})$, our new instance satisfies $\tau' \leq \hat{x}$ with high probability after $o(n)$ steps.⁵ Because the new GSS instance has $n' \geq (1 - \varepsilon/2)n$ elements with high probability, applying (3.27) for $\tau = \tau'$ completes the proof of the lower bound. \square

With some effort, the proof of [BCP01] can be extended to the case of $[\pm d]$ for $d > 1$:

Theorem 4 (GSS Solution Probability for $[\pm d]$, $d > 1$). *Let $C = [\pm d]$ for a fixed integer $d > 1$ and fix any constant $\varepsilon > 0$. For $\vec{x} \sim [0 : \hat{x} - 1]^n$ and any integer τ satisfying $|\tau| = o(\hat{x}n)$, we have*

$$\Pr_{\vec{x}}[\exists \vec{c} \in C^n : \vec{x} \cdot \vec{c} = \tau] \begin{cases} \geq 1 - o(1) & \text{if } \hat{x} \leq |C|^{(1-\varepsilon)n} \\ \leq |C|^n / \hat{x} & \text{if } \hat{x} \geq |C|^n. \end{cases}$$

Together, Corollary 3 and Theorem 4 cover all $d \geq 0$. (Note the slight differences between the statements due to parity in the $d = 1$ case.)

Here the challenge is again to prove the lower bound; the proof of the upper bound

⁵In fact, this is the same ‘‘Interval Shifting’’ technique as in Lemma 12.

on solution probability is trivial and follows the same argument as that in the proof of [Theorem 3](#).

To do so, fix a coefficient set $C = [\pm d]$ for a fixed constant $d > 1$ and a range bound $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ with respect to a small fixed constant $\varepsilon > 0$. We begin by introducing the quantity

$$\rho_n := \frac{|C|^{n+1/2}}{\hat{x} \sqrt{2\pi n \kappa \sum_{c \in C} c^2}} \quad (3.28)$$

where

$$\kappa := \mathbf{E}_{\mathbf{x}_i \sim [0:\hat{x}-1]} \left[\frac{\mathbf{x}_i^2}{\hat{x}^2} \right] = \frac{1}{3} - \frac{1}{2\hat{x}} + \frac{1}{6\hat{x}^2}. \quad (3.29)$$

Because $|C|$ is a fixed constant and $\kappa_n \approx \frac{1}{3}$, we have

$$\rho_n = \Theta \left(\frac{|C|^n}{\hat{x} \sqrt{n}} \right). \quad (3.30)$$

Moreover, because $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ by assumption, we also have

$$\rho_n = |C|^{\Omega(n)}. \quad (3.31)$$

The bulk of the proof consists in demonstrating that the number of solutions for a random GSS instance $\vec{\mathbf{x}} \sim [0:\hat{x}-1]^n$ is close to ρ_n with high probability.

Now, let

$$\mathbf{Z} := \mathbf{Z}(\hat{x}, C, \tau) \quad (3.32)$$

be a random variable that counts the number of solution vectors $\vec{c} \in C^n$ for a random GSS instance $\vec{\mathbf{x}} \sim [0:\hat{x}-1]^n$ with target value $|\tau| = O(\hat{x})$. Note here that in the definition of \mathbf{Z} , we consider a smaller range of τ than in the statement of [Theorem 4](#) (i.e., $O(\hat{x})$ as opposed to $o(n\hat{x})$); this is for convenience in the proofs of [Lemmas 13](#) and [14](#) and is addressed in the

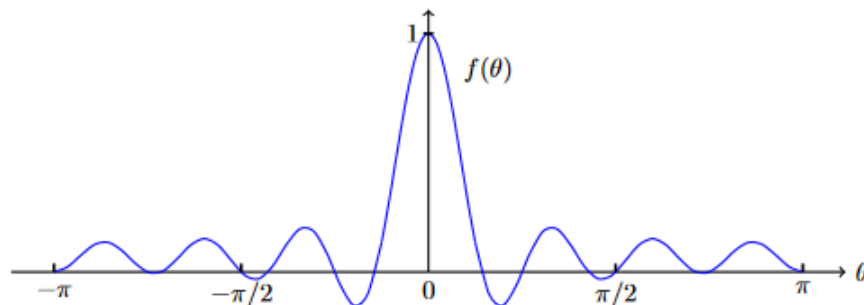
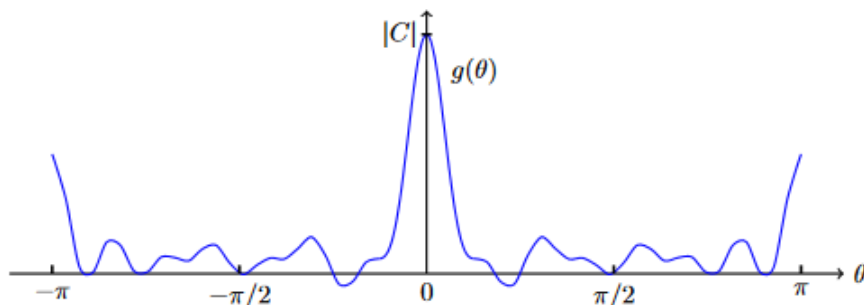
(a) f with $\hat{x} = 8$, $C = [\pm 2]$.(b) g with $\hat{x} = 8$, $C = [\pm 2]$.

Figure 3.4: Example plots of f and g . As $|C|$ increases, so does the complexity of the oscillation.

proof of [Theorem 4](#) below.

Our proof of the lower bound on solution probability when $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ generalizes Proposition 3.1 of [\[BCP01\]](#), and consists of three parts:

1. [Lemma 13](#) bounds the first moment of \mathcal{Z} . We prove that $\mathbf{E}_{\hat{x}}[\mathcal{Z}] = \rho_n \cdot (1 \pm o_n(1))$.
2. [Lemma 14](#) bounds the second moment of \mathcal{Z} . We prove that $\mathbf{E}_{\hat{x}}[\mathcal{Z}^2] \leq \rho_n^2 \cdot (1 + o_n(1))$.
3. The proof of [Theorem 4](#) uses the preceding lemmas to conclude that $\mathcal{Z} = \rho_n \cdot (1 \pm o_n(1))$ with probability $1 - o_n(1)$. The bound for $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ follows.

The first two steps are accomplished by expressing \mathcal{Z} and \mathcal{Z}^2 as integrals written in

terms of certain functions, $g : [-\pi, \pi] \rightarrow \mathbb{R}$ and $G : [-\pi, \pi]^2 \rightarrow \mathbb{R}$ respectively, to the n th power. In both cases, we show that the mass of the function is highly concentrated in a region near the origin and tightly bound the value of the function in this region. Precisely, we define

$$g(\theta) := \sum_{c \in C} f(c\theta) \quad (3.33)$$

and

$$G(\theta_1, \theta_2) := \sum_{(c_1, c_2) \in C^2} f(c_1\theta_1 + c_2\theta_2), \quad (3.34)$$

where $f : [-\pi, \pi] \rightarrow \mathbb{R}^6$ denotes the helper function

$$f(\theta) := \mathbf{E}_{\mathbf{x}_i \sim [0:\hat{x}-1]} [\cos(\theta \mathbf{x}_i)] \quad (3.35)$$

$$= \frac{1}{\hat{x}} \sum_{j \in [0:\hat{x}-1]} \cos(j\theta) \quad (3.36)$$

$$= \frac{1}{\hat{x}} \left(\frac{\sin((\hat{x} - 1/2)\theta)}{2 \sin(\theta/2)} + \frac{1}{2} \right). \quad (3.37)$$

In order to provide a rough intuition for the reader, f and g are plotted in [Figure 3.4](#). f is a symmetric waveform that takes a large positive value near 0 and whose amplitude declines as the argument increases. g is more complicated, but shares the property that the maximum over $[-\pi, \pi]$ appears at the origin. The interval close to the origin thus contains most of the mass of the functions f^n and g^n . Finally, the function G (two arguments, not plotted) is similar to g in several important respects but requires closer consideration; see further discussion in the proof of [Lemma 14](#).

⁶Our function f is defined almost identically to the function f that occurs in the proof of [\[BCP01, Proposition 3.1\]](#). The only difference is that [\[BCP01\]](#) consider inputs drawn uniformly from the set $[\hat{x}]$, while we consider inputs drawn uniformly from the set $[0 : \hat{x} - 1]$. [\(3.37\)](#) corresponds to [\[BCP01, Equation \(3.11\)\]](#).

Proof of Theorem 4 (Assuming Lemmas 13 and 14).

Upper Bound. As in the proof of Theorem 3, fix a coefficient vector $\vec{c} \in C^n \setminus \{\vec{0}\}$ and let $i^* \in [n]$ be an index such that $c_{i^*} \neq 0$. Conditioning on any outcome of $\vec{\mathbf{x}}_{[n] \setminus \{i\}}$ from $[0 : \hat{x} - 1]^{n-1}$, $\vec{\mathbf{x}} \cdot \vec{c} = \tau$ occurs with probability at most $1/\hat{x}$ over the selection of \mathbf{x}_i . Union-bounding over all $\vec{c} \in C^n \setminus \{\vec{0}\}$ gives the claimed upper bound of $|C|^n/\hat{x}$.

Lower Bound. Consider the quantity

$$\left| \frac{\mathcal{Z}}{\rho_n} - 1 \right|,$$

which captures the multiplicative difference between ρ_n and the number of solutions \mathcal{Z} of a random GSS instance.

We have

$$\begin{aligned} \mathbf{E}_{\vec{\mathbf{x}}} \left[\left| \frac{\mathcal{Z}}{\rho_n} - 1 \right| \right] &\leq \sqrt{\mathbf{E}_{\vec{\mathbf{x}}} \left[\left(\frac{\mathcal{Z}}{\rho_n} - 1 \right)^2 \right]} \\ &= \frac{1}{\rho_n} \sqrt{\mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}^2] - \mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}]^2 + \left(\mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}] - \rho_n \right)^2} \\ &\leq \frac{1}{\rho_n} \sqrt{\mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}^2] - \mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}]^2} + \frac{1}{\rho_n} \left| \mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}] - \rho_n \right|} \\ &= o_n(1). \end{aligned}$$

Here the first step holds since any random variable \mathcal{X} satisfies $\mathbf{E}[\mathcal{X}]^2 \leq \mathbf{E}[\mathcal{X}^2]$, the second step is elementary algebra, the third step holds since $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for any $a, b \in \mathbb{R}_{\geq 0}$, and the last step uses Lemmas 13 and 14 to bound $\mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}]$ and $\mathbf{E}_{\vec{\mathbf{x}}}[\mathcal{Z}^2]$, respectively.

Our instance has a solution unless $\mathcal{Z} = 0$, or, as a necessary condition, unless

$$\left| \frac{\mathcal{Z}}{\rho_n} - 1 \right| \geq 1,$$

which occurs with probability at most $o(1)$ by applying Markov's inequality to the expectation.

Fix $C = [\pm d]$ and $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ for some constant $\varepsilon > 0$. Now, over the small range $|\tau| = O(\hat{x})$, we can conclude that

$$\Pr_{\vec{x}}[\exists \vec{c} \in C^n : \vec{x} \cdot \vec{c} = \tau] = 1 - o(1). \quad (3.38)$$

It remains to consider the larger target range $|\tau| = o(n\hat{x})$, as in the theorem statement. Without loss of generality, consider $\tau > 0$ and consider the experiment in which we sample the input elements $\mathbf{x}_1, \mathbf{x}_2, \dots$, one by one and assign each the coefficient -1 . Each step of this process effectively creates a new instance with one fewer input element and a smaller target τ' . It is a simple exercise to show that when $\tau = o(n\hat{x})$, our new instance satisfies $\tau' = O(\hat{x})$ with high probability after $o(n)$ steps.⁷ Applying (3.38) for $\tau = \tau'$ completes the proof of the lower bound. \square

3.2.3 Expectation of \mathcal{Z}

Lemma 13 (Expectation of \mathcal{Z} . *Fix a coefficient set $C = [\pm d]$, $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ for some constant $\varepsilon > 0$, and a target $\tau = O(\hat{x})$. For ρ_n , f and g defined as in (3.28), (3.37) and*

⁷In fact, this is the same ‘‘Interval Shifting’’ technique as in Lemma 12.

(3.33), we have

$$\mathbf{E}[\mathcal{Z}] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta = \rho_n \cdot (1 \pm o_n(1)).$$

For any choice of $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$, we observe that

$$\begin{aligned} \mathcal{Z} &= \sum_{\vec{c} \in C^n} \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\theta(\vec{\mathbf{x}} \cdot \vec{c} - \tau)} \cdot d\theta \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-i\theta\tau} \sum_{\vec{c} \in C^n} \left(\prod_{j \in [n]} e^{i\theta c_j \mathbf{x}_j} \right) \cdot d\theta \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-i\theta\tau} \prod_{j \in [n]} \left(\sum_{c \in C} e^{i\theta c \mathbf{x}_j} \right) \cdot d\theta, \end{aligned}$$

where the first step chooses an integral to act as an indicator for the event that $\vec{\mathbf{x}} \cdot \vec{c} = \tau$, the second step factors the dot product $\vec{\mathbf{x}} \cdot \vec{c}$, and the last step exchanges the sum and the product (which is enabled by the special form of the integrand).

We note that

$$\begin{aligned} \sum_{c \in C} e^{ic\theta \mathbf{x}_j} &= \sum_{c \in [d]} (e^{ic\theta \mathbf{x}_j} + e^{-ic\theta \mathbf{x}_j}) \\ &= \sum_{c \in [d]} 2 \cos(c\theta \mathbf{x}_j) \\ &= \sum_{c \in C} \cos(c\theta \mathbf{x}_j), \end{aligned}$$

by the fact that $C = [\pm d]$, Euler's formula, and cancelling sines. Applying this identity allows us to simplify our equation for \mathcal{Z} to

$$\mathcal{Z} = \frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(\tau\theta) \prod_{j \in [n]} \left(\sum_{c \in C} \cos(c\theta \mathbf{x}_j) \right) \cdot d\theta. \quad (3.39)$$

Since the variables $\mathbf{x}_j \sim [0 : \hat{x} - 1]$ for $j \in [n]$ are independent and identically distributed., we can break down the expectation $\mathbf{E}_{\vec{x}}[\mathcal{Z}]$ as follows:

$$\begin{aligned} \mathbf{E}_{\vec{x}}[\mathcal{Z}] &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(\tau\theta) \cdot \left(\mathbf{E}_{\mathbf{x}_i \sim [0:\hat{x}-1]} \left[\sum_{c \in C} \cos(c\theta \mathbf{x}_i) \right] \right)^n d\theta \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta, \end{aligned}$$

where the last step follows by the definitions of f (3.37) and g (3.33). Later we will see that the mass of this integral is highly concentrated around the origin, where $\cos(\tau\theta) \cdot g(\theta)^n \approx |C|^n$.

Fix any constants a and b satisfying

$$\frac{|C|^2}{\varepsilon} < a < b, \quad (3.40)$$

and define the quantity

$$b_0 := 2\hat{x} \cdot \sin^{-1} \left(\frac{b}{2\hat{x}} \right), \quad (3.41)$$

which satisfies

$$b \leq b_0 \leq b \cdot \frac{\pi}{2} \quad (3.42)$$

by elementary trigonometric reasoning.

To evaluate the expectation $\mathbf{E}_{\vec{x}}[\mathcal{Z}]$, we follow the approach of [BCP01] and divide the

integral into the following three parts:⁸

$$\mathbf{E}_{\widehat{x}}[\mathcal{Z}] = \frac{1}{2\pi} \int_{|\theta| \leq \frac{\sqrt{64 \ln(n)/n}}{\widehat{x}}} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta \tag{Part I}$$

$$+ \frac{1}{2\pi} \int_{|\theta| \in \left[\frac{\sqrt{64 \ln(n)/n}}{\widehat{x}}, \frac{b_0}{\widehat{x}} \right]} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta \tag{Part II}$$

$$+ \frac{1}{2\pi} \int_{|\theta| \in \left[\frac{b_0}{\widehat{x}}, \pi \right]} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta. \tag{Part III}$$

(Part I) integrates the function close to the origin, where almost all the mass is concentrated, while (Part III) integrates the function far from the origin, where there is very little mass. (Part II) bounds a small intermediate region where the mass of the function is still small but not small enough for the rough methods that suffice for (Part III).

Claims 3 to 5 below bound (Part I), (Part II), and (Part III) respectively. Combining them completes the proof of Lemma 13.

Proof of Lemma 13. By Claims 3 to 5, we have

$$\begin{aligned} \mathbf{E}_{\widehat{x}}[\mathcal{Z}] &= \text{(Part I)} + \text{(Part II)} + \text{(Part III)} \\ &\in \text{(Part I)} \pm (|\text{(Part II)}| + |\text{(Part III)}|) \\ &= \rho_n \cdot (1 \pm o_n(1)). \end{aligned} \quad \square$$

We complete the proofs by bounding the size of (Part I), (Part II), and (Part III).

Claim 3. (Part I) = $\rho_n(1 \pm o(1))$.

⁸By (3.42), we have that

$$\frac{b_0}{\widehat{x}} \leq \frac{(\pi/2) \cdot b}{8d \cdot b} \leq \frac{\pi}{16},$$

so the interval of integration is well defined.

Proof. Following [BCP01], we set

$$y := \widehat{x}\theta$$

and consider $|y| \leq \sqrt{64 \ln(n)/n} = o_n(1)$. In this range, we observe that

$$\begin{aligned} f(\theta) &= f\left(\frac{y}{\widehat{x}}\right) \\ &= \mathbf{E}_{\mathbf{x}_i \sim [0:\widehat{x}-1]} \left[\cos\left(\mathbf{x}_i \cdot \frac{y}{\widehat{x}}\right) \right] \\ &= \mathbf{E}_{\mathbf{x}_i \sim [0:\widehat{x}-1]} \left[1 - \frac{\mathbf{x}_i^2}{2} \left(\frac{y}{\widehat{x}}\right)^2 \pm O(y^4) \right] \\ &= 1 - \frac{\kappa}{2} y^2 \pm O(y^4) \\ &= \left(1 - \frac{\kappa}{2} y^2\right) \cdot (1 \pm O(y^4)), \end{aligned} \tag{3.43}$$

where the second step uses the Taylor series approximation

$$\cos(z) = 1 - \frac{1}{2}z^2 \pm O(z^4), \tag{3.44}$$

the third step follows from the definition of κ (3.29), and the last step converts the (additive) error term into a multiplicative form. Using a similar argument, we can also write

$$\begin{aligned} g(\theta) &= g\left(\frac{y}{\widehat{x}}\right) \\ &= \sum_{c \in C} f\left(c \cdot \frac{y}{\widehat{x}}\right) \\ &= \left(|C| - \frac{\kappa}{2} \left(\sum_{c \in C} c^2\right) y^2\right) \cdot (1 \pm O(y^4)) \\ &= |C| \cdot \exp\left(-\frac{\kappa}{2|C|} \left(\sum_{c \in C} c^2\right) y^2\right) \cdot (1 \pm O(y^4)), \end{aligned} \tag{3.45}$$

where the last step uses the Taylor series approximation

$$e^{-z} = 1 - z \pm O(z^2).$$

Taking (3.45) to the power of n , it follows that

$$g\left(\frac{y}{\hat{x}}\right)^n = |C|^n \cdot \exp\left(-\frac{n\kappa}{2|C|} \left(\sum_{c \in C} c^2\right) y^2\right) \cdot (1 \pm O(ny^4)). \quad (3.46)$$

Based on (3.46), we can evaluate (Part I) as follows:

$$\begin{aligned} (\text{Part I}) &= \frac{1}{2\pi\hat{x}} \int_{|y| \leq \sqrt{64\ln(n)/n}} \cos\left(\tau \cdot \frac{y}{\hat{x}}\right) \cdot g\left(\frac{y}{\hat{x}}\right)^n \cdot dy \\ &= \frac{1}{2\pi\hat{x}} \int_{|y| \leq \sqrt{64\ln(n)/n}} (1 \pm O(y^2)) \cdot g\left(\frac{y}{\hat{x}}\right)^n \cdot dy \\ &= \frac{|C|^n}{2\pi\hat{x}} \int_{|y| \leq \sqrt{64\ln(n)/n}} (1 \pm O(y^2 + ny^4)) \cdot \exp\left(-\frac{n\kappa}{2|C|} \left(\sum_{c \in C} c^2\right) y^2\right) dy \\ &= \frac{|C|^n}{2\pi\hat{x}} \cdot (1 \pm o_n(1)) \cdot \int_{|y| \leq \sqrt{64\ln(n)/n}} \exp\left(-\frac{n\kappa}{2|C|} \left(\sum_{c \in C} c^2\right) y^2\right) dy \\ &= \frac{|C|^n}{2\pi\hat{x}} \cdot (1 \pm o_n(1)) \cdot \sqrt{\frac{2\pi|C|}{n\kappa \sum_{c \in C} c^2}} \cdot \operatorname{erf}\left(\sqrt{\frac{32\kappa}{|C|} \left(\sum_{c \in C} c^2\right) \ln(n)}\right) \\ &= \rho_n \cdot (1 \pm o_n(1)) \cdot \operatorname{erf}\left(\sqrt{\frac{32\kappa}{|C|} \left(\sum_{c \in C} c^2\right) \ln(n)}\right). \end{aligned} \quad (3.47)$$

Here the first step changes variables by substituting y for $\hat{x}\theta$. The second step follows from (3.44) and $|\tau| = O(\hat{x})$. The third step substitutes (3.46). The fourth step follows from the fact that $O(y^2 + ny^4) = o_n(1)$ when $|y| \leq \sqrt{64\ln(n)/n}$. The fifth step resolves the integral

by using the Gaussian error function

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt. \quad (3.48)$$

The last step uses the definition of ρ_n (3.28).

To finish the proof of Claim 3, it remains to quantify the Gaussian error function $\operatorname{erf}(z)$. It is known that $|1 - \operatorname{erf}(z)| \leq e^{-z^2}$ for any $z \in \mathbb{R}_{\geq 0}$ (e.g., [CCM11], Theorem 1). As a consequence, we have

$$\operatorname{erf} \left(\sqrt{\frac{32\kappa}{|C|} \left(\sum_{c \in C} c^2 \right) \ln(n)} \right) = 1 \pm n^{-32\kappa|C|^{-1}(\sum_{c \in C} c^2)} \quad (3.49)$$

$$= 1 \pm n^{-32\kappa} \quad (3.50)$$

$$= 1 \pm o_n(1), \quad (3.51)$$

where the second step holds because $|C|^{-1}(\sum_{c \in C} c^2) \geq 1$ and the last step holds because $\kappa = \Omega_n(1)$ (3.29). Substituting into (3.47) completes the proof of Claim 3. \square

Claim 4. $|(\text{Part II})| = o(\rho_n)$.

Proof. Once again we set

$$y := \hat{x}\theta.$$

Recall that (Part II) considers the range

$$|\theta| \in \left[\frac{\sqrt{64 \ln(n)/n}}{\hat{x}}, \frac{b_0}{\hat{x}} \right],$$

or equivalently,

$$|y| \in \left[\sqrt{\frac{64 \ln(n)}{n}}, b_0 \right].$$

In this range, it turns out that

$$\begin{aligned} |f(c\theta)| &= \left| f\left(c \cdot \frac{y}{\hat{x}}\right) \right| \\ &\leq \frac{1}{\sqrt[n]{n}}, \end{aligned} \tag{3.52}$$

for any (nonzero) coefficient $c \in C$. Assuming the truth of (3.52) for the moment, we have

$$|g(\theta)| = \left| g\left(\frac{y}{\hat{x}}\right) \right| \tag{3.53}$$

$$\leq \sum_{c \in C} \left| f\left(c \cdot \frac{y}{\hat{x}}\right) \right| \tag{3.54}$$

$$\leq \frac{|C|}{\sqrt[n]{n}}, \tag{3.55}$$

for any $|y| \in [\sqrt{64 \ln(n)/n}, b_0]$. Hence, the magnitude of (Part II) is at most

$$\begin{aligned} |(\text{Part II})| &= \left| \frac{1}{2\pi\hat{x}} \int_{|y| \in [\sqrt{64 \ln(n)/n}, b_0]} \cos\left(t \cdot \frac{y}{\hat{x}}\right) \cdot g\left(\frac{y}{\hat{x}}\right)^n \cdot dy \right| \\ &\leq \frac{1}{2\pi\hat{x}} \cdot 2b_0 \cdot \left(\max_{|y| \in [\sqrt{64 \ln(n)/n}, b_0]} \left| g\left(\frac{y}{\hat{x}}\right) \right| \right)^n \\ &\leq \frac{1}{2\pi\hat{x}} \cdot 2b_0 \cdot \frac{|C|^n}{n} \\ &= \frac{b_0 |C|^n}{\pi\hat{x}n} \\ &= o(\rho_n), \end{aligned}$$

where the first step changes variables using the equality $y = \hat{x}\theta$, the third step applies (3.53),

and the last step holds since $b_0 \leq \frac{\pi}{2} \cdot b$ is a constant (3.41) and $\rho_n = \Theta(\frac{|C|^n}{x\sqrt{n}})$ (3.28).

It remains to verify (3.52). Before doing so, we observe that for any $|y| \leq b_0$,

$$\begin{aligned} \left| \frac{cy}{2\hat{x}} \right| &\leq \frac{|c| \cdot b_0}{2\hat{x}} \\ &\leq \frac{|c| \cdot (\pi/2) \cdot b}{16d \cdot b} \\ &\leq \frac{|c| \cdot (\pi/32)}{d} \\ &\leq \frac{\pi}{32}, \end{aligned} \tag{3.56}$$

where the second step holds because $b_0 \leq (\pi/2) \cdot b$ (3.42) and the last step holds because $c \in C = [\pm d]$.

Thus for any $|y| \leq b_0$, we can upper-bound $|f(c\theta)|$ as follows.

$$\begin{aligned} |f(c\theta)| &= \left| f\left(c\frac{y}{\hat{x}}\right) \right| \\ &= \left| \frac{\sin(cy - cy/(2\hat{x}))}{2\hat{x} \sin(cy/(2\hat{x}))} + \frac{1}{2\hat{x}} \right| \\ &= \left| \frac{\sin(cy)}{2\hat{x} \tan(cy/(2\hat{x}))} + \frac{1 - \cos(cy)}{2\hat{x}} \right| \\ &\leq \left| \frac{\sin(cy)}{cy} \right| + \frac{1 - \cos(cy)}{2\hat{x}} \\ &\leq \left| \frac{\sin(cy)}{cy} \right| + \frac{1 - \cos(cy)}{40}. \end{aligned} \tag{3.57}$$

Here the first step changes variables by substituting y for $\hat{x}\theta$. The second step applies the identity

$$\sin(\alpha - \beta) = \sin(\alpha) \cos(\beta) - \sin(\beta) \cos(\alpha).$$

The third step holds since $|\tan(z)| \geq |z|$ for any $|z| \leq \frac{\pi}{2}$, and

$$\left| \frac{cy}{2\hat{x}} \right| \leq \frac{\pi}{4} \leq \frac{\pi}{2},$$

by (3.56).

We prove (3.52) for any nonzero $c \in C$ and any $|y| \in [\sqrt{64 \ln(n)/n}, b_0]$ in two cases.

Case I: $\sqrt{64 \ln(n)/n} \leq |y| \leq \frac{\pi}{2|c|}$. By substituting Taylor series, we have that $|\sin(z)/z| \leq e^{-z^2/6}$ for any $|z| \leq \frac{\pi}{2}$ and $\cos(z) \geq 1 - \frac{1}{2}z^2$ (3.44). Applying both facts to (3.57) gives

$$\begin{aligned} |f(c\theta)| &= \left| f\left(c \cdot \frac{y}{\hat{x}}\right) \right| \\ &\leq \left| \frac{\sin(cy)}{cy} \right| + \frac{1 - \cos(cy)}{40} \\ &\leq e^{-\frac{1}{6}c^2y^2} + \frac{1}{80}c^2y^2 \\ &\leq e^{-\frac{1}{8}c^2y^2} \\ &\leq e^{-\frac{1}{8}y^2} \\ &\leq \frac{1}{\sqrt[n]{n}}, \end{aligned}$$

where the third step holds since

$$e^{-z^2/6} + \frac{1}{80}z^2 \leq e^{-z^2/8}$$

for any $|z| \leq \frac{\pi}{2}$, the fourth step holds since $c \in C$ is a nonzero integer, and the last step holds when $|y| \geq \sqrt{64 \ln(n)/n}$.

Case II: $\frac{\pi}{2|c|} \leq |y| \leq b_0$. Following (3.57), in this range we have

$$\begin{aligned}
|f(c\theta)| &= \left| f\left(c \cdot \frac{y}{\hat{x}}\right) \right| \\
&\leq \left| \frac{\sin(cy)}{cy} \right| + \frac{1 - \cos(cy)}{40} \\
&\leq \frac{|\sin(cy)|}{\pi/2} + \frac{1 + |\cos(cy)|}{40} \\
&\leq \frac{2}{\pi} + \frac{1}{20} \\
&\leq \frac{1}{\sqrt[n]{n}},
\end{aligned} \tag{3.58}$$

where the second step applies $|y| \geq \frac{\pi}{2|c|}$, the third step holds because $|\sin(cy)| \leq 1$ and $|\cos(cy)| \leq 1$, and the last step holds because $\frac{2}{\pi} + \frac{1}{20} \approx 0.6866$ and $\frac{1}{\sqrt[n]{n}} = 1 - o_n(1)$.

Combining both cases together gives (3.52) and completes the proof of Claim 4. \square

Claim 5. $|(\text{Part III})| = o(\rho_n)$.

Proof. Recall that (Part III) considers the range $|\theta| \in [\frac{b_0}{\hat{x}}, \pi]$, in which we have

$$|f(\theta)| = \left| \frac{1}{\hat{x}} \left(\frac{\sin((\hat{x} - 1/2)\theta)}{2 \sin(\theta/2)} + \frac{1}{2} \right) \right| \tag{3.59}$$

$$\leq \frac{1}{b} + \frac{1}{2\hat{x}} \tag{3.60}$$

$$\leq \frac{1}{a}. \tag{3.61}$$

Here the first inequality holds as $|\sin((\hat{x} - 1/2)\theta)| \leq 1$ and

$$\left| \sin\left(\frac{\theta}{2}\right) \right| \geq \left| \sin\left(\frac{b_0}{2\hat{x}}\right) \right| = \frac{b}{2\hat{x}}$$

by (3.42).

For any $c \in C$ and $|\theta| \in [\frac{b_0}{x}, \pi]$, we know from (3.37) that

$$|f(c\theta)| = \left| \mathbf{E}_{\mathbf{x}_i \sim [0:\hat{x}-1]} [\cos(c\theta \mathbf{x}_i)] \right| \leq 1$$

(as $|C| \geq 2 \cdot |[\pm 1]| = 4$). Applying this fact and (3.59), for any $|\theta| \in [\frac{b_0}{x}, \pi]$ we have

$$|g(\theta)| \leq |f(\theta)| + |f(-\theta)| + \sum_{c \in C \setminus \{\pm 1\}} |f(c\theta)| \quad (3.62)$$

$$\leq \frac{2}{a} + (|C| - 2) \quad (3.63)$$

$$= |C|^{(1-\varepsilon_1)} \quad (3.64)$$

for some constant $\varepsilon_1 := \varepsilon_1(|C|, a) \in (0, 1)$, where the last step holds by the definition of a (3.40). Further, we know that for any integer $c \in C$ the function $f_c(\theta) := f(c\theta)$ is 2π -periodic (3.37). Thus (3.59) implies that

$$\left| \left\{ \theta \in [-\pi, \pi] : |f(c\theta)| > \frac{1}{a} \right\} \right| \leq \frac{2b_0}{\hat{x}}. \quad (3.65)$$

This, given that $g(\theta) = \sum_{c \in C} f(c\theta)$, further gives

$$\left| \left\{ \theta \in [-\pi, \pi] : |g(\theta)| > \frac{|C|}{a} \right\} \right| \leq \frac{2b_0|C|}{\hat{x}}. \quad (3.66)$$

By considering separately the subregion on which $|C|/a < |g(\theta)| \leq |C|^{(1-\varepsilon_1)}$ and the subregion on which $|g(\theta)| \leq |C|/a$, the magnitude of (Part III) is at most

$$|(\text{Part III})| = \left| \frac{1}{2\pi} \int_{|\theta| \in [\frac{b_0}{x}, \pi]} \cos(\tau\theta) \cdot g(\theta)^n \cdot d\theta \right|$$

$$\begin{aligned}
&\leq \frac{1}{2\pi} \int_{|\theta| \in [\frac{b_0}{\hat{x}}, \pi]} |g(\theta)|^n \cdot d\theta \\
&\leq \frac{1}{2\pi} \left(|C|^{(1-\varepsilon_1)n} \cdot \frac{2b_0|C|}{\hat{x}} + \left(\frac{|C|}{a}\right)^n \cdot 2\pi \right) \\
&= O\left(\frac{|C|^{(1-\varepsilon_1)n}}{\hat{x}}\right) + O\left(\frac{1}{|C|^n}\right), \tag{3.67}
\end{aligned}$$

where the third step applies (3.62) and (3.66), and the fourth step holds since (for the first term) both b_0 and $|C|$ are constants and (for the second term) $a > |C|^2/\varepsilon > |C|^2$ (3.40).

Recall that $\rho_n = |C|^{\Omega(n)}$ (3.31). Thus the two terms of (3.67) are upper-bounded by

$$O\left(\frac{|C|^{(1-\varepsilon_1)n}}{\hat{x}}\right) = \rho_n \cdot O(\sqrt{n} \cdot |C|^{-\varepsilon_1 n}) = o(\rho_n)$$

and

$$O(|C|^{-n}) = o_n(1) = o(\rho_n),$$

respectively. This completes the proof of Claim 5. \square

3.2.4 Upper Bound on the Second Moment of \mathcal{Z}

Lemma 14 (Second Moment of \mathcal{Z}). *Fix a coefficient set $C = [\pm d]$, $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ for some constant $\varepsilon > 0$, and a target $\tau = O(\hat{x})$. For ρ_n , f and G defined as in (3.28), (3.37), and (3.34) we have*

$$\begin{aligned}
\mathbf{E}_{\vec{x}}[\mathcal{Z}^2] &\leq \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2 \\
&\leq \rho_n^2 \cdot (1 + o_n(1)).
\end{aligned}$$

Squaring our identity for \mathbf{Z} (3.39) yields

$$\begin{aligned}
\mathbf{Z}^2 &= \left(\frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(\tau\theta) \prod_{j \in [n]} \left(\sum_{c \in C} \cos(c\theta \mathbf{x}_j) \right) \cdot d\theta \right)^2 \\
&= \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \cos(\tau\theta_1) \cos(\tau\theta_2) \prod_{j \in [n]} \left(\sum_{(c_1, c_2) \in C^2} \cos(c_1\theta_1 \mathbf{x}_j) \cos(c_2\theta_2 \mathbf{x}_j) \right) \cdot d\theta_1 d\theta_2 \\
&\leq \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \prod_{j \in [n]} \left| \sum_{(c_1, c_2) \in C^2} \cos(c_1\theta_1 \mathbf{x}_j) \cos(c_2\theta_2 \mathbf{x}_j) \right| \cdot d\theta_1 d\theta_2 \\
&= \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \prod_{j \in [n]} \left| \sum_{(c_1, c_2) \in C^2} (\cos(c_1\theta_1 \mathbf{x}_j) \cos(c_2\theta_2 \mathbf{x}_j) - \sin(c_1\theta_1 \mathbf{x}_j) \sin(c_2\theta_2 \mathbf{x}_j)) \right| \cdot d\theta_1 d\theta_2 \\
&= \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \prod_{j \in [n]} \left| \sum_{(c_1, c_2) \in C^2} \cos(c_1\theta_1 \mathbf{x}_j + c_2\theta_2 \mathbf{x}_j) \right| \cdot d\theta_1 d\theta_2,
\end{aligned}$$

where in the second to last line, adding the sin term does not change the equation because $C = [\pm d]$ is a symmetric set and $\sin(z)$ is an odd function, and in the last line, we use the identity

$$\cos(\alpha) \cos(\beta) - \sin(\alpha) \sin(\beta) = \cos(\alpha + \beta).$$

Since all the $\mathbf{x}_j \sim [0 : \hat{x} - 1]$ are independent and identically distributed, the following holds for the second moment $\mathbf{E}_{\hat{x}}[\mathbf{Z}^2]$:

$$\begin{aligned}
\mathbf{E}_{\hat{x}}[\mathbf{Z}^2] &\leq \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left| \sum_{(c_1, c_2) \in C^2} \mathbf{E}_{\mathbf{x}_i \sim [0 : \hat{x} - 1]} [\cos(c_1\theta_1 \mathbf{x}_i + c_2\theta_2 \mathbf{x}_i)] \right|^n \cdot d\theta_1 d\theta_2 \\
&= \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left| \sum_{(c_1, c_2) \in C^2} f(c_1\theta_1 + c_2\theta_2) \right|^n \cdot d\theta_1 d\theta_2 \\
&= \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2, \tag{3.68}
\end{aligned}$$

where the second and third steps substitute f and G using their definitions (3.37) and (3.34).

Remark 2. We will show that the mass of (3.68) is concentrated around the origin, where

$|G(\theta_1, \theta_2)|^n \approx |C|^{2n}$. Notably, this is *not* true when $C = [\pm 1]$, as considered in [BCP01]. In the $C = [\pm 1]$ case, mass is also concentrated at the points $[\pm\pi, \pm\pi]$, the corners of the region of integration. This is because in the $C = [\pm 1]$ case the function G consists of four summands that interfere constructively at these points. However, when $C = [\pm d]$ for any $d > 1$, interference from other summands ensures that mass is not concentrated at $[\pm\pi, \pm\pi]$.

We reuse the constants a , b and b_0 satisfying (3.40)

$$\left(\text{i.e., } \frac{|C|^2}{\varepsilon} < a < b\right),$$

and (3.42)

$$\left(\text{i.e., } b \leq b_0 \leq b \cdot \frac{\pi}{2}\right),$$

which were previously introduced in the proof of Lemma 13. Define the subregions $R'_\theta \subseteq R''_\theta \subseteq [-\pi, \pi]^2$ by letting

$$R'_\theta := \left[-\frac{\sqrt{64 \ln(n)}}{\hat{x}\sqrt{n}}, \frac{\sqrt{64 \ln(n)}}{\hat{x}\sqrt{n}} \right]^2 \quad \text{and}$$

$$R''_\theta := \left[-\frac{b_0}{\hat{x}}, \frac{b_0}{\hat{x}} \right]^2.$$

Similar to the approach of [BCP01], we split the formula (3.68) into the following three parts.

$$\mathbf{E}_{\hat{x}}[\mathcal{Z}] \leq \frac{1}{4\pi^2} \iint_{(\theta_1, \theta_2) \in R'_\theta} |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2 \quad (\text{Part 1})$$

$$+ \frac{1}{4\pi^2} \iint_{(\theta_1, \theta_2) \in R''_\theta \setminus R'_\theta} |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2 \quad (\text{Part 2})$$

$$+ \frac{1}{4\pi^2} \iint_{(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_\theta} |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2. \quad (\text{Part 3})$$

Once again, (Part 1) integrates the function close to the origin, where almost all the mass is concentrated, (Part 3) integrates the function far from the origin, where there is very little mass, and (Part 2) bounds a small intermediate region.

We evaluate (Part 1) and (Part 2) respectively in Claims 6 and 7. (Part 3) is rather tricky; we first prove three auxiliary results (Claims 8 to 10) and then evaluate it in Corollary 4.

Claim 6. (Part 1) = $\rho_n^2 \cdot (1 \pm o_n(1))$.

Proof. (This proof is analogous to the proof of Claim 3.) Following [BCP01], we define two variables

$$y_1 := \hat{x}\theta_1 \text{ and}$$

$$y_2 := \hat{x}\theta_2.$$

Thus, for $(\theta_1, \theta_2) \in R'_\theta$, we have

$$|y_1|, |y_2| \leq \frac{\sqrt{64 \ln(n)}}{n} = o_n(1).$$

In this range,

$$\begin{aligned} G\left(\frac{y_1}{\hat{x}}, \frac{y_2}{\hat{x}}\right) &= \sum_{(c_1, c_2) \in C^2} f\left(c_1 \frac{y_1}{\hat{x}} + c_2 \frac{y_2}{\hat{x}}\right) \\ &= \sum_{(c_1, c_2) \in C^2} \left(1 - \frac{\kappa}{2}(c_1 y_1 + c_2 y_2)^2\right) \cdot (1 \pm O(y_1^4 + y_2^4)) \\ &= \left(|C|^2 - |C| \cdot \frac{\kappa \sum_{c \in C} c^2}{2} \cdot (y_1^2 + y_2^2)\right) \cdot (1 \pm O(y_1^4 + y_2^4)) \\ &= |C|^2 \cdot \exp\left(-\frac{\kappa \sum_{c \in C} c^2}{2|C|} (y_1^2 + y_2^2)\right) \cdot (1 \pm O(y_1^4 + y_2^4)), \end{aligned}$$

where the first step changes the variables y_1 and y_2 to $\hat{x}\theta_1$ and $\hat{x}\theta_2$, the second step holds since we have $f(y/\hat{x}) = (1 - \frac{\kappa}{2}y^2) \cdot (1 \pm O(y^4))$ for small $|y|$ (3.43), the third step is elementary algebra (notice that $C = [\pm d]$ is a symmetric set, so the crossing terms $2c_1c_2y_1y_2$ get cancelled), and the last step uses the Taylor series approximation $e^{-z} = (1 - z) \cdot (1 \pm O(z^2))$.

As a consequence, the following holds for any $|y_1|, |y_2| \leq \sqrt{64 \ln(n)/n}$:

$$\begin{aligned} \left| G\left(\frac{y_1}{\hat{x}}, \frac{y_2}{\hat{x}}\right) \right|^n &= |C|^{2n} \cdot \exp\left(-\frac{n\kappa \sum_{c \in C} c^2}{2|C|} (y_1^2 + y_2^2)\right) \cdot (1 \pm O(ny_1^4 + ny_2^4)) \\ &= |C|^{2n} \cdot \exp\left(-\frac{n\kappa \sum_{c \in C} c^2}{2|C|} (y_1^2 + y_2^2)\right) \cdot (1 \pm o_n(1)). \end{aligned} \quad (3.69)$$

This allows us to bound the magnitude of (Part 1) as follows:

$$\begin{aligned} (\text{Part 1}) &= \frac{1}{4\pi^2 \hat{x}^2} \iint_{|y_1|, |y_2| \leq \sqrt{64 \ln(n)/n}} \left| G\left(\frac{y_1}{\hat{x}}, \frac{y_2}{\hat{x}}\right) \right|^n \cdot dy_1 dy_2 \\ &= \frac{|C|^{2n}}{4\pi^2 \hat{x}^2} \cdot (1 \pm o_n(1)) \cdot \iint_{|y_1|, |y_2| \leq \sqrt{64 \ln(n)/n}} \exp\left(-\frac{n\kappa \sum_{c \in C} c^2}{2|C|} (y_1^2 + y_2^2)\right) \cdot dy_1 dy_2 \\ &= \frac{|C|^{2n}}{4\pi^2 \hat{x}^2} \cdot (1 \pm o_n(1)) \cdot \frac{2\pi|C|}{n\kappa \sum_{c \in C} c^2} \cdot \operatorname{erf}\left(\sqrt{\frac{32\kappa}{|C|} \left(\sum_{c \in C} c^2\right) \ln(n)}\right)^2 \\ &= \frac{|C|^{2n}}{4\pi^2 \hat{x}^2} \cdot (1 \pm o_n(1)) \cdot \frac{2\pi|C|}{n\kappa \sum_{c \in C} c^2} \cdot (1 \pm o_n(1))^2 \\ &= \rho_n^2 \cdot (1 \pm o_n(1)). \end{aligned}$$

Here the first step changes the variables y_1 and y_2 to $\hat{x}\theta_1$ and $\hat{x}\theta_2$. The second step substitutes (3.69). The third step resolves the integral by using the Gaussian error function $\operatorname{erf}(z)$ (3.48). The fourth step uses the approximation of $\operatorname{erf}(z)$ in (3.49), and the last step uses the definition of ρ_n (3.28).

This completes the proof of Claim 6. □

Claim 7. (Part 2) $= o(\rho_n^2)$.

Proof. (This proof is analogous to the proof of Claim 4.) Once again, we set

$$y_1 := \hat{x}\theta_1 \text{ and}$$

$$y_2 := \hat{x}\theta_2$$

Recall that for (Part 2) we consider the range $(y_1, y_2) \in R''_y \setminus R'_y$, where

$$R'_y := \left[-\frac{\sqrt{64 \ln(n)}}{n}, \frac{\sqrt{64 \ln(n)}}{n} \right]^2 \quad \text{and}$$

$$R''_y := [-b_0, b_0]^2.$$

For each pair $(c_1, c_2) \in C^2$, we define the function

$$f_{c_1, c_2}^y(y_1, y_2) := \frac{1}{2} \left(f \left(\frac{c_1 y_1}{\hat{x}} + \frac{c_2 y_2}{\hat{x}} \right) + f \left(\frac{c_2 y_1}{\hat{x}} - \frac{c_1 y_2}{\hat{x}} \right) \right). \quad (3.70)$$

This family of functions allows us to decompose the function G in a convenient way: expanding G according to its definition (3.34) and regrouping, we have

$$\begin{aligned} |G(\theta_1, \theta_2)| &= \left| G \left(\frac{y_1}{\hat{x}}, \frac{y_2}{\hat{x}} \right) \right| \\ &= \left| \sum_{(c_1, c_2) \in C^2} f \left(c_1 \frac{y_1}{\hat{x}} + c_2 \frac{y_2}{\hat{x}} \right) \right| \\ &= \left| \sum_{(c_1, c_2) \in C^2} f_{c_1, c_2}^y(y_1, y_2) \right|. \end{aligned} \quad (3.71)$$

We will soon show that for any pair $(c_1, c_2) \in C^2$ and any point $(y_1, y_2) \in R''_y \setminus R'_y$, we

have

$$|f_{c_1, c_2}^y(y_1, y_2)| \leq \frac{1}{\sqrt[n]{n^2}}. \quad (3.72)$$

Assuming (3.72) for the moment, for any point $(y_1, y_2) \in R_y'' \setminus R_y'$ we have

$$\begin{aligned} |G(\theta_1, \theta_2)| &= \left| \sum_{(c_1, c_2) \in C^2} f_{c_1, c_2}^y(y_1, y_2) \right| \\ &\leq \frac{|C|^{2n}}{\sqrt[n]{n^2}}. \end{aligned} \quad (3.73)$$

Hence, we can upper-bound (Part 2) as follows.

$$\begin{aligned} (\text{Part 2}) &= \frac{1}{4\pi^2 \hat{x}^2} \iint_{y_1, y_2 \in R_y'' \setminus R_y'} \left| G\left(\frac{y_1}{\hat{x}} + \frac{y_2}{\hat{x}}\right) \right|^n \cdot dy_1 dy_2 \\ &\leq \frac{1}{4\pi^2 \hat{x}^2} \cdot |R_y''| \cdot \left(\max_{y_1, y_2 \in R_y'' \setminus R_y'} \left| G\left(\frac{y_1}{\hat{x}} + \frac{y_2}{\hat{x}}\right) \right| \right)^n \\ &\leq \frac{1}{4\pi^2 \hat{x}^2} \cdot |R_y''| \cdot \frac{|C|^{2n}}{n^2} \\ &= \frac{b_0^2 |C|^{2n}}{\pi^2 \hat{x}^2 n^2} \\ &= o(\rho_n^2), \end{aligned}$$

where the first step changes the variables y_1 and y_2 to $\hat{x}\theta_1$ and $\hat{x}\theta_2$, the third step substitutes (3.73), the fourth step is elementary algebra (notice that $|R_y''| = 4b_0^2$), and the last step holds because $b_0 \leq \frac{\pi}{2} \cdot b$ (3.42) and $\rho_n = \Theta\left(\frac{|C|^n}{\hat{x}\sqrt{n}}\right)$ (3.28).

It remains to verify (3.72). Before doing so, we observe that for any point $(y_1, y_2) \in$

$R''_y \setminus R'_y$,

$$\begin{aligned}
\left| \frac{c_1 y_1 + c_2 y_2}{2\hat{x}} \right| &\leq \frac{|c_1| + |c_2|}{2\hat{x}} \cdot b_0 \\
&\leq \frac{|c_1| + |c_2|}{16d \cdot b} \cdot \frac{\pi}{2} \cdot b \\
&\leq \frac{|c_1| + |c_2|}{d} \cdot \frac{\pi}{4} \\
&\leq \frac{\pi}{2}.
\end{aligned} \tag{3.74}$$

where the second step holds because $b_0 \leq (\frac{\pi}{2}) \cdot b$ (3.42) and the last step holds because $c \in C = [\pm d]$. Because (3.74) holds for any point $(y_1, y_2) \in R''_y \setminus R'_y$, the sequence of manipulations (3.57) can be reused as written, resulting in an analogous bound on f :

$$\left| f \left(c_1 \frac{y_1}{\hat{x}} + c_2 \frac{y_2}{\hat{x}} \right) \right| \leq \left| \frac{\sin(c_1 y_1 + c_2 y_2)}{c_1 y_1 + c_2 y_2} \right| + \frac{1 - \cos(c_1 y_1 + c_2 y_2)}{40}. \tag{3.75}$$

Similarly, for any point $(y_1, y_2) \in R''_y \setminus R'_y$, we have

$$\left| \frac{c_2 y_1 - c_1 y_2}{2\hat{x}} \right| \leq \frac{\pi}{2} \tag{3.76}$$

and

$$\left| f \left(c_2 \frac{y_1}{\hat{x}} - c_1 \frac{y_2}{\hat{x}} \right) \right| \leq \left| \frac{\sin(c_2 y_1 - c_1 y_2)}{c_2 y_1 - c_1 y_2} \right| + \frac{1 - \cos(c_2 y_1 - c_1 y_2)}{40}. \tag{3.77}$$

It remains to prove (3.72). To prove (3.72), we split the argument into three cases based on the values of c_1 , c_2 , y_1 and y_2 .

Case 1: $|c_1y_1 + c_2y_2| \geq \frac{\pi}{2}$. We deduce that

$$\begin{aligned} |f_{c_1, c_2}^y(y_1, y_2)| &= \frac{1}{2} \cdot \left| f\left(c_1 \frac{y_1}{\hat{x}} + c_2 \frac{y_2}{\hat{x}}\right) + f\left(c_2 \frac{y_1}{\hat{x}} - c_1 \frac{y_2}{\hat{x}}\right) \right| \\ &\leq \frac{1}{2} \cdot (\text{RHS of (3.75)} + 1) \\ &\leq \frac{1}{2} \cdot \left(\frac{2}{\pi} + \frac{1}{20} + 1\right) \\ &\leq \frac{1}{\sqrt[n]{n^2}}, \end{aligned}$$

where the second step uses (3.75) and the fact that $|f(\theta)| \leq 1$ for any $\theta \in \mathbb{R}$ (3.37). (Here and below, we use *RHS* to refer to the right-hand side of a previous equation when it is too cumbersome to fit on the page.) The third step holds since

$$\left| \frac{\sin(z)}{z} \right| + \frac{1 - \cos(z)}{40} \leq \frac{2}{\pi} + \frac{1}{20}$$

when $|z| \geq \frac{\pi}{2}$ (following a sequence of manipulations identical to (3.58) in Claim 4, Case II), and the last step holds since $\frac{1}{2} \cdot \left(\frac{2}{\pi} + \frac{1}{20} + 1\right) \approx 0.8433$ and $\frac{1}{\sqrt[n]{n^2}} = 1 - o_n(1)$.

Case 2: $|c_2y_1 - c_1y_2| \geq \frac{\pi}{2}$. Here we can reapply the arguments for **Case 1**.

Case 3: $|c_1y_1 + c_2y_2| \leq \frac{\pi}{2}$ **and** $|c_2y_1 - c_1y_2| \leq \frac{\pi}{2}$. Combining (3.75) and (3.77) gives

$$\begin{aligned} |f_{c_1, c_2}^y(y_1, y_2)| &= \frac{1}{2} \cdot \left| f\left(c_1 \frac{y_1}{\hat{x}} + c_2 \frac{y_2}{\hat{x}}\right) + f\left(c_2 \frac{y_1}{\hat{x}} - c_1 \frac{y_2}{\hat{x}}\right) \right| \\ &\leq \frac{1}{2} \cdot (\text{RHS of (3.75)} + \text{RHS of (3.77)}) \\ &\leq \frac{1}{2} \cdot \left(e^{-\frac{1}{8}(c_1y_1 + c_2y_2)^2} + e^{-\frac{1}{8}(c_2y_1 - c_1y_2)^2} \right), \end{aligned} \tag{3.78}$$

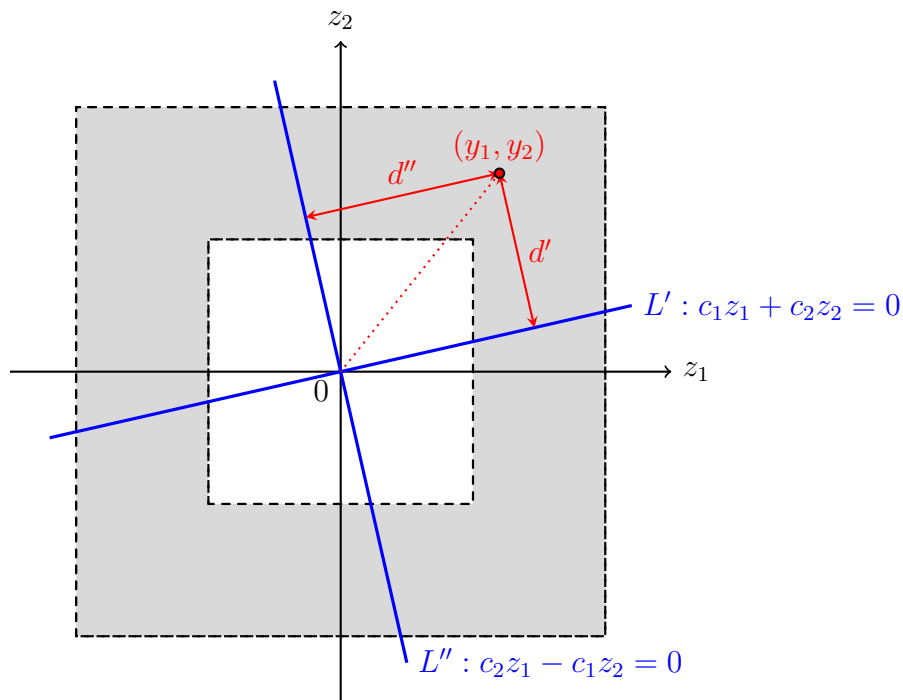


Figure 3.5: Illustration of the formula $(3.78) \geq |f_{c_1, c_2}^y(y_1, y_2)|$. The gray region represents the considered range $(y_1, y_2) \in R_y'' \setminus R_y'$, where $R_y' = [-\sqrt{64 \ln(n)/n}, \sqrt{64 \ln(n)/n}]^2$ and $R_y'' = [-b_0, b_0]^2$. The Pythagorean theorem guarantees that $d'^2 + d''^2 = y_1^2 + y_2^2$.

where the third step holds because

$$\left| \frac{\sin(z)}{z} \right| + \frac{1 - \cos(z)}{40} \leq e^{-z^2/8}$$

for any $|z| \leq \frac{\pi}{2}$, and by substituting using (3.74) and (3.76).

In order to bound (3.78), we will employ a geometric proof; Figure 3.5 is provided for supporting intuition. Consider two straight lines $L' : c_1 z_1 + c_2 z_2 = 0$ and $L'' : c_2 z_1 - c_1 z_2 = 0$. By elementary algebra, we can see that the Euclidean distance $d' := d'(y_1, y_2)$ (resp. the Euclidean distance $d'' := d''(y_1, y_2)$) from a given point $(y_1, y_2) \in \mathbb{R}$ to straight line L' (resp.

straight line L'') is given by

$$d' = \frac{|c_1 y_1 + c_2 y_2|}{\sqrt{c_1^2 + c_2^2}} \quad \text{and} \quad d'' = \frac{|c_2 y_1 - c_1 y_2|}{\sqrt{c_1^2 + c_2^2}}. \quad (3.79)$$

Moreover, L' and L'' are perpendicular (by construction), so the Pythagorean theorem ensures that $d'^2 + d''^2 = y_1^2 + y_2^2$. Accordingly, the larger distance $\max\{d', d''\}$ is lower bounded by

$$\begin{aligned} \max\{d', d''\} &\geq \frac{d' + d''}{2} \\ &= \sqrt{\frac{(d' + d'')^2}{4}} \\ &= \sqrt{\frac{d'^2 + d''^2 + 2d'd''}{4}} \\ &> \sqrt{\frac{y_1^2 + y_2^2}{4}} \\ &\geq \frac{\sqrt{32 \ln(n)}}{n}, \end{aligned} \quad (3.80)$$

$$(3.81)$$

where the last step holds since

$$(y_1, y_2) \notin R'_y = \left[-\frac{\sqrt{64 \ln(n)}}{n}, \frac{\sqrt{64 \ln(n)}}{n} \right]^2.$$

Putting everything together results in

$$\begin{aligned} |f_{c_1, c_2}^y(y_1, y_2)| &\leq (3.78) = \frac{1}{2} \cdot \left(e^{-\frac{c_1^2 + c_2^2}{8} \cdot d'^2} + e^{-\frac{c_1^2 + c_2^2}{8} \cdot d''^2} \right) \\ &\leq \frac{1}{2} \cdot \left(e^{-\frac{1}{4} \cdot d'^2} + e^{-\frac{1}{4} \cdot d''^2} \right) \\ &\leq \frac{1}{2} \cdot \left(e^{-\frac{1}{4} \cdot 0} + e^{-\frac{1}{4} \cdot 32 \ln(n)/n} \right) \end{aligned}$$

$$= \frac{1}{2} \cdot \left(1 + \frac{1}{\sqrt[n]{n^8}} \right),$$

where the first step applies (3.79), the second step follows since the coefficients $c_1, c_2 \in C$ are nonzero integers, and the third step applies (3.81) and the fact that $\min\{d', d''\} \geq 0$.

Finally, we have

$$\frac{1}{2} \cdot \left(1 + \frac{1}{\sqrt[n]{n^8}} \right) \leq \frac{1}{\sqrt[n]{n^2}}$$

for sufficiently large n . Combining all the three cases together gives (3.72) and completes the proof of Claim 7. \square

Below we use Claims 8 to 10 (as auxiliaries) to evaluate (Part 3) in Corollary 4. This evaluation together with Claims 6 and 7 immediately gives Lemma 14.

Recall that (Part 3) bounds the integral of $|G(\theta_1, \theta_2)|^n$ over a region $[-\pi, \pi]^2 \setminus R''_\theta$ that includes everything sufficiently far from the origin. In the three claims that follow, we split $[-\pi, \pi]^2 \setminus R''_\theta$ further into three *subregions*: Claim 10 proves a strong upper bound on $|G(\theta_1, \theta_2)|^n$ for all of $[-\pi, \pi]^2$ outside of a small region Q'' ; Claim 9 proves a larger upper bound for all of $[-\pi, \pi]^2$ outside of a *smaller* region $Q' \subset Q''$, and Claim 8 proves a still larger bound on all points outside of $R''_\theta \subset Q'$. Figure 3.6 provides supporting intuition for the argument.

Claim 8. *There exists a constant $\varepsilon_2 := \varepsilon_2(|C|, a) > 0$ such that, for any $(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_\theta$,*

$$|G(\theta_1, \theta_2)| \leq |C|^2 - \frac{1 - 1/a}{2} \leq |C|^{(2-\varepsilon_2)}.$$

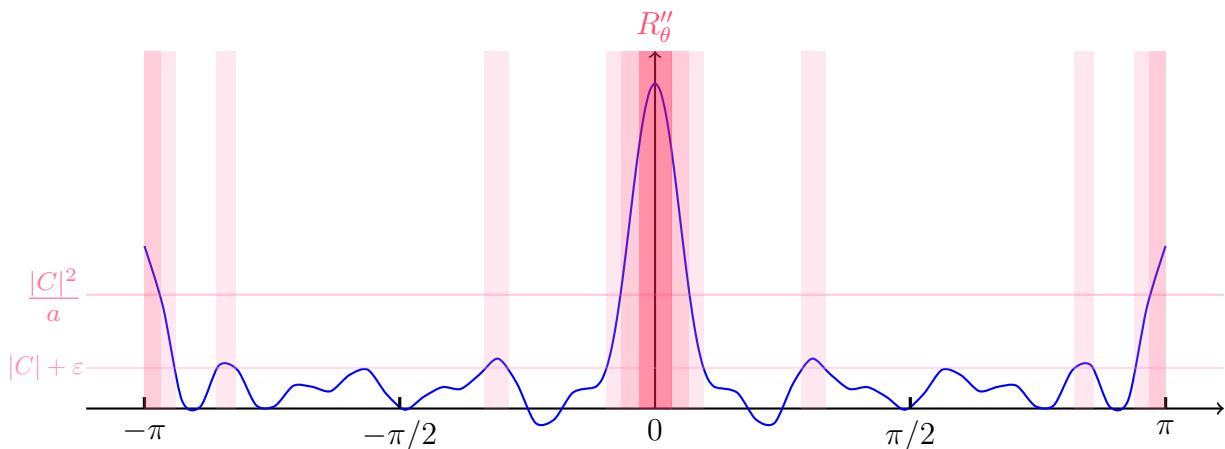


Figure 3.6: Cartoon of $|G(\theta_1, \theta_2)|^n$. (The figure is drawn in one dimension for simplicity and the function plotted is arbitrary.) The region R''_θ (darkest pink slice) is the smallest of three nested regions and contains the part of the domain nearest the origin. $Q' \supseteq R''_\theta$ (medium pink slices) denotes the larger region on which the function takes values at least $|C|^2/a$. $Q'' \supseteq Q'$ (lightest pink slices) denotes the still larger region on which the function takes values at least $|C| + \varepsilon$.

Proof. Define the function

$$f_{c_1, c_2}^\theta(\theta_1, \theta_2) := \frac{1}{2}(f(c_1\theta_1 + c_2\theta_2) + f(c_2\theta_1 - c_1\theta_2)) \tag{3.82}$$

for each pair $(c_1, c_2) \in \mathcal{C}^2$. This set of functions gives us another helpful decomposition of $G(\theta_1, \theta_2)$, akin to (3.70) but using the arguments θ_1, θ_2 in place of y_1, y_2 . Substituting into G , we have

$$\begin{aligned} |G(\theta_1, \theta_2)| &= \left| \sum_{(c_1, c_2) \in \mathcal{C}^2} f(c_1\theta_1 + c_2\theta_2) \right| \\ &= \left| \sum_{(c_1, c_2) \in \mathcal{C}^2} f_{c_1, c_2}^\theta(\theta_1, \theta_2) \right|. \end{aligned} \tag{3.83}$$

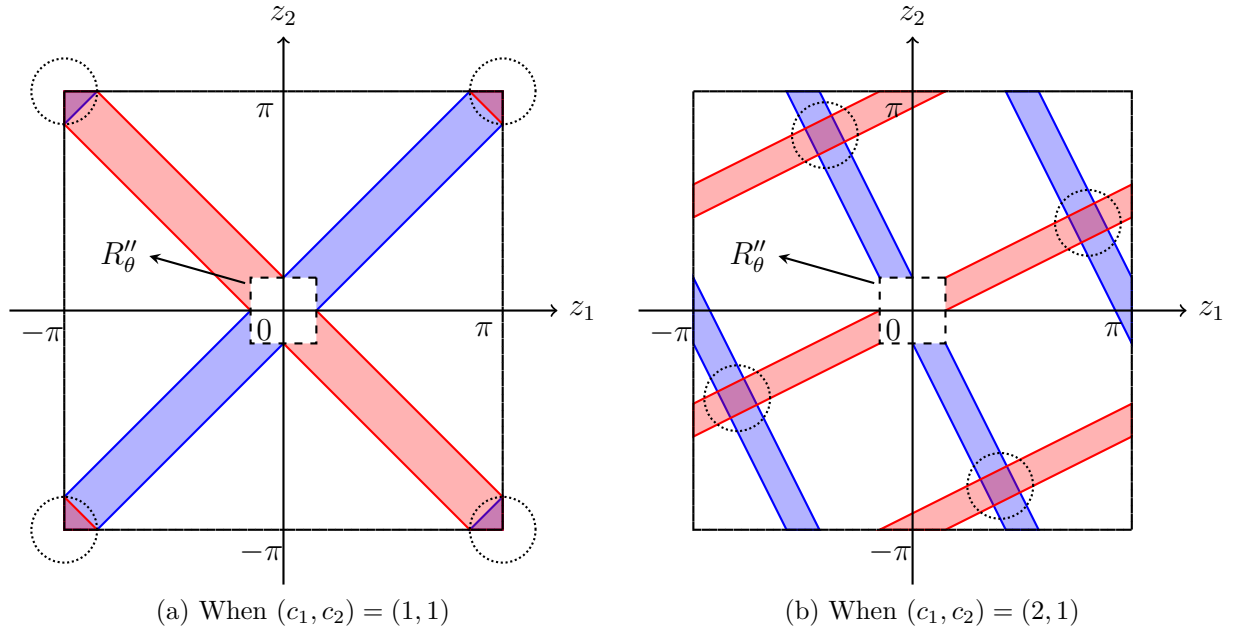


Figure 3.7: $R_{c_1, c_2} := \{(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_\theta : (c_1\theta_1 + c_2\theta_2), (c_2\theta_1 - c_1\theta_2) \in [-\frac{b_0}{x}, \frac{b_0}{x}]_\circ\}$ is the intersection of the red area (on which $(c_1\theta_1 + c_2\theta_2) \in [-\frac{b_0}{x}, \frac{b_0}{x}]_\circ$) and the blue area (on which $(c_2\theta_1 - c_1\theta_2) \in [-\frac{b_0}{x}, \frac{b_0}{x}]_\circ$). The subregion R''_θ (the dashed square) is excluded from $[-\pi, \pi]^2$. Figures 3.7a and 3.7b respectively choose $(c_1, c_2) = (1, 1)$ and $(c_1, c_2) = (2, 1)$.

Because $|f(\theta)| \leq 1$ for any $\theta \in \mathbb{R}$ by definition (3.37), we also have

$$|f_{c_1, c_2}^\theta(\theta_1, \theta_2)| \leq 1$$

for any $(\theta_1, \theta_2) \in \mathbb{R}^2$. Recall also that

$$|f(\theta)| \leq \frac{1}{a}$$

for any $|\theta| \in [\frac{b_0}{x}, \pi]$ (3.59) and that

$$\frac{b_0}{\hat{x}} \leq \frac{(\pi/2) \cdot b}{8d \cdot b} \leq \frac{\pi}{16}$$

by (3.42). Since f is 2π -periodic (3.37), only for θ within the 2π -periodic range

$$\left[-\frac{b_0}{\hat{x}}, \frac{b_0}{\hat{x}} \right]_{\circ} := \bigcup_{k \in \mathbb{Z}} \left[-\frac{b_0}{\hat{x}} + 2k\pi, \frac{b_0}{\hat{x}} + 2k\pi \right] \quad (3.84)$$

can we possibly have $|f(\theta)| > \frac{1}{a}$.

For $(c_1, c_2) \in C^2$, we thus consider the subregion

$$R_{c_1, c_2} := \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_{\theta} : (c_1\theta_1 + c_2\theta_2), (c_2\theta_1 - c_1\theta_2) \in \left[-\frac{b_0}{x}, \frac{b_0}{x} \right]_{\circ} \right\}. \quad (3.85)$$

By (3.84), this is the only region on which both components of f_{c_1, c_2}^{θ} may exceed $1/a$. In other words, for any point $(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus (R_{c_1, c_2} \cup R''_{\theta})$, we have either $|f(c_1\theta_1 + c_2\theta_2)| \leq \frac{1}{a}$ or $|f(c_2\theta_1 - c_1\theta_2)| \leq \frac{1}{a}$ or both, which (together with the fact that $|f(\theta)| \leq 1$ for any $\theta \in \mathbb{R}$) implies

$$\begin{aligned} |f_{c_1, c_2}^{\theta}(\theta_1, \theta_2)| &\leq \frac{|f(c_1\theta_1 + c_2\theta_2)| + |f(c_2\theta_1 - c_1\theta_2)|}{2} \\ &\leq \frac{1 + 1/a}{2} \end{aligned} \quad (3.86)$$

for each $(c_1, c_2) \in C^2$, on the respective regions $(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus (R_{c_1, c_2} \cup R''_{\theta})$.

Because G is the sum over f_{c_1, c_2}^{θ} for all $(c_1, c_2) \in C$, in order to bound G we would like to show that there is no point on $[-\pi, \pi]^2 \setminus R''_{\theta}$ on which the entire family of functions takes a value near 1. To see this, observe that because $C = [\pm d]$ for a fixed integer $d > 1$, G contains both $f_{1,1}^{\theta}$ and $f_{2,1}^{\theta}$ as summands.

Evaluating $f_{1,1}^{\theta}$ shows that taking the union of the $\frac{b_0}{x}$ -neighborhoods of the four points $(\pm\pi, \pm\pi)$ fully covers $R_{1,1}$ (Figure 3.7a). However, taking the union of the $\frac{b_0}{x}$ -neighborhoods of the four points $(\frac{4\pi}{5}, \frac{2\pi}{5}), (-\frac{2\pi}{5}, \frac{4\pi}{5}), (-\frac{4\pi}{5}, -\frac{2\pi}{5})$ and $(\frac{2\pi}{5}, -\frac{4\pi}{5})$ fully covers $R_{2,1}$ (Figure 3.7b).

Because $\frac{b_0}{x} \leq \frac{\pi}{16}$, the two unions of neighborhoods are disjoint, and we have

$$R_{1,1} \cap R_{2,1} = \emptyset. \quad (3.87)$$

Given the above arguments, we deduce that for any $(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_\theta$,

$$\begin{aligned} |G(\theta_1, \theta_2)| &= \left| \sum_{(c_1, c_2) \in C^2} f_{c_1, c_2}^\theta(\theta_1, \theta_2) \right| \\ &\leq (|C|^2 - 1) + \frac{1 + 1/a}{2} \\ &= |C|^2 - \frac{1 - 1/a}{2}. \end{aligned}$$

Here we begin by decomposing G according to (3.82). The second step uses the fact that the upper bound (3.86) holds for at least one of $(c_1, c_2) = (1, 1)$ or $(c_1, c_2) = (2, 1)$ because of (3.87), and the fact that $|f_{c_1, c_2}^\theta(\theta_1, \theta_2)| \leq 1$ for each pair $(c_1, c_2) \in C^2$.

Because $a > |C|^2/\varepsilon > 1$ by definition (3.40), there must exist some constant $\varepsilon_2 := \varepsilon_2(|C|, a) > 0$ such that $|C|^2 - (1 - 1/a)/2 \leq |C|^{(2-\varepsilon_2)}$. This finishes the proof of Claim 8. \square

Claim 9. *The subregion*

$$Q' := \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : |G(\theta_1, \theta_2)| > \frac{|C|^2}{a} \right\}$$

has measure at most

$$|Q'| \leq |C|^2 \cdot \frac{4\pi b_0}{\hat{x}}.$$

Proof. We begin by defining the set

$$A_{c_1, w} := \left\{ \theta_1 \in [-\pi, \pi] : |f(c_1\theta_1 + w)| > \frac{1}{a} \right\} \quad (3.88)$$

for each coefficient $c_1 \in C$ and each real offset $w \in \mathbb{R}$. We can think of this set as considering the single-dimensional cross-section of the two-dimensional function $h(\theta_1, \theta_2) := f(c_1\theta_1 + c_2\theta_2)$ at $c_2\theta_2 = w$, then selecting the subset of values for θ_1 on which this function is large.

Written in these terms, the bound (3.65) on the set of values for which f is larger than $1/a$ is equivalent to the statement that, for all $c_1 \in C$, we have

$$|A_{c_1,0}| \leq \frac{2b_0}{\hat{x}}. \quad (3.89)$$

Because f is 2π -periodic, for any integer $c \in C$, the function $f_c(\theta) := f(c\theta)$ is also 2π -periodic. Thus (3.89) implies

$$|A_{c,w}| \leq \frac{2b_0}{\hat{x}} \quad (3.90)$$

for any $c \in C$, $w \in \mathbb{R}$, as $|A_{c,w}|$ is defined with reference to an interval of length 2π , and the choice of w effectively translates f . It follows that, for any pair $(c_1, c_2) \in C^2$, we have

$$\left| \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : |f(c_1\theta_1 + c_2\theta_2)| > \frac{1}{a} \right\} \right| \leq 2\pi \cdot \frac{2b_0}{\hat{x}} \quad (3.91)$$

$$= \frac{4\pi b_0}{\hat{x}}. \quad (3.92)$$

Recall that $G(\theta_1, \theta_2)$ can be written as $\sum_{(c_1, c_2) \in C^2} f(c_1\theta_1 + c_2\theta_2)$, the sum of $|C|^2$ summands each upper-bounded by 1. It follows from the pigeonhole principle that if $|G(\theta_1, \theta_2)| > |C|^2/a$ on a point (θ_1, θ_2) , we must have $|f(c_1\theta_1 + c_2\theta_2)| > 1/a$ for at least one pair $(c_1, c_2) \in C^2$. Thus, union-bounding the measure (3.91) over all pairs $(c_1, c_2) \in C^2$ concludes the proof of **Claim 9**. □

Claim 10. *The subregion*

$$Q'' := \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : |G(\theta_1, \theta_2)| > |C| + \varepsilon \right\}$$

has measure at most

$$|Q''| \leq |C|^4 \cdot \left(\frac{2b_0}{\hat{x}} \right)^2.$$

Proof. We begin by introducing another convenient decomposition of the function G . Observe that

$$G(\theta_1, \theta_2) = \sum_{c_2 \in C} q(\theta_1, c_2 \theta_2), \text{ where}$$

$$q(\theta_1, w) := \sum_{c_1 \in C} f(c_1 \theta_1 + w).$$

In other words, $q(\theta_1, c_2 \theta_2)$ is the sum of the $|C|$ summands of G corresponding to a certain coefficient c_1 for θ_1 .

This allows us to rewrite Q'' as

$$Q'' = \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : \left| \sum_{c_2 \in C} q(\theta_1, c_2 \theta_2) \right| > |C| + \varepsilon \right\}.$$

Because q is the sum of $|C|$ summands, each upper-bounded by 1, q is upper-bounded by $|C|$. Thus for any point $(\theta_1, \theta_2) \in Q''$, there must exist two distinct coefficients $c_2, c'_2 \in C$ such that $|q(\theta_1, c_2 \theta_2)|, |q(\theta_1, c'_2 \theta_2)| > \frac{\varepsilon}{|C|}$ by the pigeonhole principle. In other words, Q'' is covered by

$$Q'' \subseteq \bigcup_{c_2 \neq c'_2 \in C} Q_{c_2, c'_2}, \tag{3.93}$$

where

$$Q_{\alpha,\beta} := \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : |q(\theta_1, \alpha\theta_2)|, |q(\theta_1, \beta\theta_2)| \geq \frac{|C|}{a} \right\}.$$

This definition of $Q_{\alpha,\beta}$ suffices because $a > |C|^2/\varepsilon$ by definition (3.40) and thus $\frac{|C|}{a} < \frac{\varepsilon}{|C|}$.

It remains to show that

$$|Q_{c_2, c'_2}| \leq |C|^2 \cdot \left(\frac{2b_0}{\hat{x}} \right)^2 \quad (3.94)$$

for each pair $(c_2, c'_2) \in C^2$ with $c_2 \neq c'_2$. (If (3.94) holds, then

$$\begin{aligned} |Q''| &\leq \left| \bigcup_{c_2 \neq c'_2 \in C} Q_{c_2, c'_2} \right| \\ &\leq |C|^2 \cdot \max_{c_2 \neq c'_2} |Q_{c_2, c'_2}| \\ &\leq |C|^4 \cdot \left(\frac{2b_0}{\hat{x}} \right)^2 \end{aligned}$$

using (3.93), completing the proof of the claim.)

To show (3.94), fix a pair $(c_2, c'_2) \in C^2$ such that $c_2 \neq c'_2$. We observe that

$$Q_{c_2, c'_2} = \left\{ (\theta_1, \theta_2) \in [-\pi, \pi]^2 : \theta_1 \in (B_{c_2\theta_2} \cap B_{c'_2\theta_2}) \right\},$$

where

$$B_w := \left\{ \theta_1 \in [-\pi, \pi] : |q(\theta_1, w)| > \frac{|C|}{a} \right\}.$$

Intuitively, B_w captures the values for which $|q(\theta_1, w)|$ is large. Because B_w is determined by the function $q(\theta_1, w) = \sum_{c \in C} f(c_1\theta_1 + w)$, it has a convenient property: for any $w, z \in \mathbb{R}$,

B_{w+z} is just the set B_w translated by $z \pmod{2\pi}$. [Figure 3.8](#) provides a visual aid for the structure of this set and the remainder of the proof.

It will be convenient to write $|Q_{c_2, c'_2}|$ in terms of the probability that a random point in $[-\pi, \pi]^2$ is contained in Q_{c_2, c'_2} . We have

$$\begin{aligned} |Q_{c_2, c'_2}| &= 4\pi^2 \cdot \Pr_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \sim [-\pi, \pi]} \left[\boldsymbol{\theta}_1 \in (B_{c_2 \boldsymbol{\theta}_2} \cap B_{c'_2 \boldsymbol{\theta}_2}) \right] \\ &= 4\pi^2 \cdot \Pr_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \sim [-\pi, \pi]} \left[\boldsymbol{\theta}_1 \in (B_0 \cap B_{(c'_2 - c_2) \boldsymbol{\theta}_2}) \right] \\ &= 4\pi^2 \cdot \Pr_{\boldsymbol{\theta}_1 \sim [-\pi, \pi]} [\boldsymbol{\theta}_1 \in B_0] \cdot \Pr_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \sim [-\pi, \pi]} \left[\boldsymbol{\theta}_1 \in B_{(c'_2 - c_2) \boldsymbol{\theta}_2} \mid \boldsymbol{\theta}_1 \in B_0 \right]. \end{aligned} \quad (3.95)$$

Here the second line follows from translating both sets by $c_2 \boldsymbol{\theta}_2 \pmod{2\pi}$, which does not change the size of the intersection. The last line uses the identity $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B \mid A]$.

Given any nonzero integer k and any fixed offset $w \in [-\pi, \pi]$, we observe that

$$\Pr_{\boldsymbol{\theta}_2 \sim [-\pi, \pi]} [w \in B_{k \boldsymbol{\theta}_2}] = \Pr_{\mathbf{y} \sim [-\pi, \pi]} [\mathbf{y} \in B_0] = \frac{|B_0|}{2\pi}, \quad (3.96)$$

because sampling $\boldsymbol{\theta}_2$ uniformly from $[-\pi, \pi]$ distributes $k \boldsymbol{\theta}_2 \pmod{2\pi}$ uniformly over $[-\pi, \pi]$. [\(3.96\)](#) holds for any nonzero integer $k \neq 0$ and any offset $w \in [-\pi, \pi]$, so $|B_0|/(2\pi)$ can be substituted for both (conditional) probabilities in [\(3.95\)](#).

Recall the definition of the set family $A_{c_1, w}$ in [\(3.88\)](#). Using [\(3.90\)](#) to union-bound [\(3.95\)](#) yields [\(3.94\)](#):

$$|Q_{c_2, c'_2}| = 4\pi^2 \cdot \left(\frac{|B_0|}{2\pi} \right)^2$$

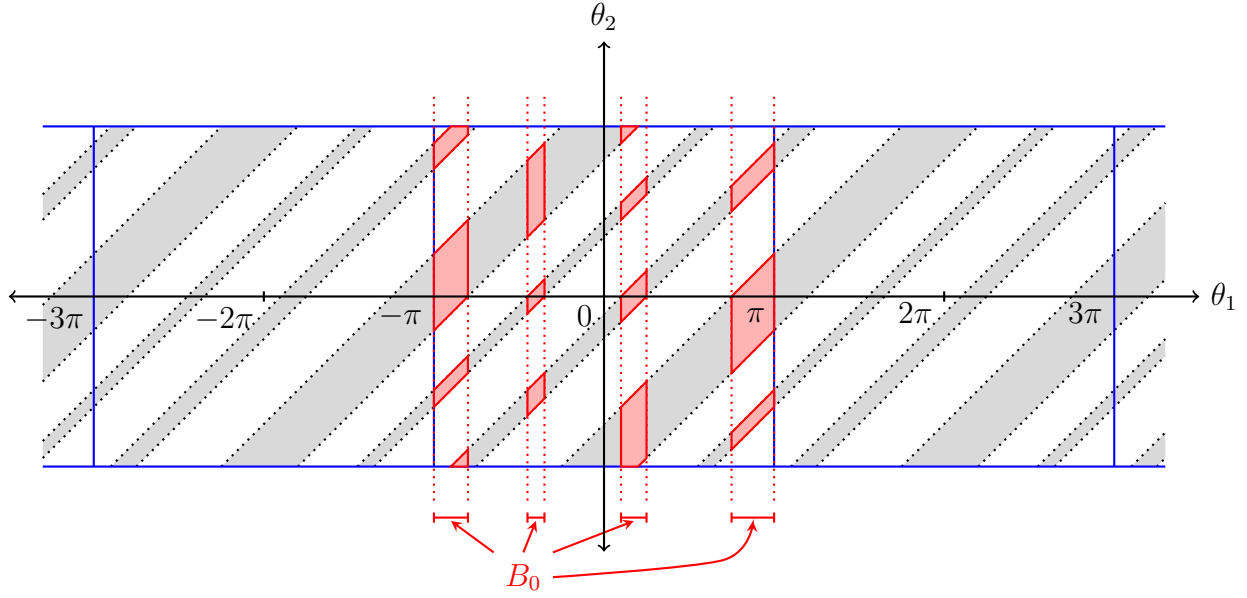


Figure 3.8: Visual aid for the proof of **Claim 10**. For any offset $\theta_2 \in [-\pi, \pi]$, the range $B_{(c'_2 - c_2)\theta_2}$ is the intersection of the line segment $(-\pi, \theta_2) - (\pi, \theta_2)$ with the gray regions. The red regions indicate $\{(\theta_1, \theta_2) \in [-\pi, \pi]^2 : \theta_1 \in B_0 \cap B_{(c'_2 - c_2)\theta_2}\}$.

$$\begin{aligned} &\leq \left(\sum_{c \in C} |A_{c,0}| \right)^2 \\ &\leq |C|^2 \cdot \left(\frac{2b_0}{\hat{x}} \right)^2. \quad \square \end{aligned}$$

Corollary 4. (**Part 3**) = $o(\rho_n^2)$.

Proof. For ease of reference, we restate the results in **Claims 8 to 10**:

- **Claim 8:** $|G(\theta_1, \theta_2)| \leq |C|^{(2-\varepsilon_2)}$ for any point $(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus R''_\theta$.
- **Claim 9:** The subregion Q' on which $|G(\theta_1, \theta_2)| > |C|^2/a$ has measure at most $|C|^2 \cdot \frac{4\pi b_0}{\hat{x}}$.
- **Claim 10:** The subregion Q'' on which $|G(\theta_1, \theta_2)| > |C| + \varepsilon$ has measure at most $|C|^4 \cdot \left(\frac{2b_0}{\hat{x}}\right)^2$.

The union of the three disjoint subregions $(Q'' \setminus R''_\theta)$ and $(Q' \setminus Q'')$ and $[-\pi, \pi]^2 \setminus Q'$ is the domain of integration for (Part 3), namely $[-\pi, \pi]^2 \setminus R''_\theta$. As a consequence, we have

$$\begin{aligned}
(\text{Part 3}) &\leq \frac{1}{4\pi^2} \left(\iint_{(\theta_1, \theta_2) \in [-\pi, \pi]^2 \setminus Q'} + \iint_{(\theta_1, \theta_2) \in Q' \setminus Q''} + \iint_{(\theta_1, \theta_2) \in Q'' \setminus R''_\theta} \right) |G(\theta_1, \theta_2)|^n \cdot d\theta_1 d\theta_2 \\
&\leq \frac{1}{4\pi^2} \left(\left(\frac{|C|^2}{a} \right)^n \cdot 4\pi^2 + (|C| + \varepsilon)^n \cdot |Q'| + |C|^{(2-\varepsilon_2)n} \cdot |Q''| \right) \\
&\leq \frac{1}{4\pi^2} \left(\left(\frac{|C|^2}{a} \right)^n \cdot 4\pi^2 + (|C| + \varepsilon)^n \cdot |C|^2 \cdot \frac{4\pi b_0}{\hat{x}} + |C|^{(2-\varepsilon_2)n} \cdot |C|^4 \cdot \left(\frac{2b_0}{\hat{x}} \right)^2 \right) \\
&= O(\varepsilon^n) + O\left(\frac{(|C| + \varepsilon)^n}{\hat{x}} \right) + O\left(\frac{|C|^{(2-\varepsilon_2)n}}{\hat{x}^2} \right). \tag{3.97}
\end{aligned}$$

Here the second line follows from Claim 8 and the definitions of Q' and Q'' . The third line applies Claims 9 and 10. The fourth line holds since (for the first term) $a > |C|^2/\varepsilon$ by definition (3.40) and (for the second and third terms) both $|C|$ and b_0 are constants. Since $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ for a constant $\varepsilon > 0$, we know from (3.28) that

$$\rho_n^2 = \Theta\left(\frac{|C|^{2n}}{\hat{x}^2 n} \right) = \Omega(|C|^{\varepsilon n}).$$

Finally, we show that each of the three terms of (3.97) is $o(\rho_n^2)$.

1. For the first term, we have $O(\varepsilon^n) = o_n(1) = o(\rho_n^2)$.
2. For the second term, we have

$$\begin{aligned}
O((|C| + \varepsilon)^n \cdot \hat{x}^{-1}) &= O(\rho_n^2) \cdot (|C| + \varepsilon)^n \cdot \frac{\hat{x}n}{|C|^{2n}} \\
&= O(\rho_n^2) \cdot O\left((|C| + \varepsilon)^n \cdot \frac{1}{|C|^{(1+\varepsilon/2)n}} \right)
\end{aligned}$$

$$\begin{aligned}
&= O(\rho_n^2) \cdot O\left(|C|^{(1+\varepsilon/4)n} \cdot \frac{1}{|C|^{(1+\varepsilon/2)n}}\right) \\
&= O(\rho_n^2) \cdot O(|C|^{-(\varepsilon/4)n}) \\
&= o(\rho_n^2).
\end{aligned}$$

Here the first line follows from the definition of ρ_n (3.28). The second line holds because $\hat{x} = O^*(|C|^{(1-\varepsilon)n})$ and thus $\hat{x}n = O(|C|^{(1-\varepsilon/2)n})$. The third line holds since

$$|C| + \varepsilon \leq |C| \cdot (1 + \varepsilon/4) < |C|^{1+\varepsilon/4},$$

given that $|C| \geq |[\pm 2]| = 4$.

3. For the third term, we have

$$\begin{aligned}
O(|C|^{(2-\varepsilon_2)n} \cdot \hat{x}^{-2}) &= O(\rho_n^2) \cdot n \cdot |C|^{-\varepsilon_2 n} \\
&= o(\rho_n^2).
\end{aligned}$$

This completes the proof of [Corollary 4](#). □

Proof of [Lemma 14](#). Putting [Claims 6](#) and [7](#) and [Corollary 4](#) together, we have

$$\mathbf{E}_{\vec{x}}[\mathcal{Z}^2] \leq (\text{Part 1}) + (\text{Part 2}) + (\text{Part 3}) \leq \rho_n^2 \cdot (1 + o_n(1)). \quad \square$$

3.3 Algorithmic Results

This section presents our main algorithm for average-case GSS, which we use to prove [Theorem 1](#).

At a high level, our structural results tell us the parameter settings for which average-case GSS instances are likely to have solutions. This allows us to prove the correctness of our algorithm, which combines new preprocessing ideas with an approach based on the Representation Method. As such, we recommend that the reader is familiar with the Representation Method ([Section 1.1.2](#)) before reading this section.

At a high level, since instances in which $\hat{x} > |C|^{(1+\varepsilon)n}$ are extremely unlikely to have a solution, we can simply ignore such instances. On the other extreme, we show how to “shrink” instances for which $\hat{x} < |C|^{(1-\varepsilon)n}$ for any constant ε while preserving the existence of a solution. The shrinking operation also gives an improved runtime for our algorithm on dense instances. This allows us to focus on the narrow range of values $\hat{x} \in |C|^{(1\pm\varepsilon)n}$ for an arbitrarily small constant $\varepsilon > 0$.

At this point, we adapt the Representation Method to Generalized Subset Sum as follows. Partial assignments of coefficients to input elements play the role of partial candidate solutions. We define the *signature* of a partial assignment to be the coefficient-weighted sum of the inputs with assigned coefficients. Then, we hash partial candidate solutions by signature into residue classes modulo a large random prime, following the general pattern described in [Section 1.1.2](#).

We prove a result ([Lemma 15](#), the “Signature Distribution Lemma”) that guarantees that with high probability over the input, the partial assignments of a solution have many distinct signatures with very high probability, satisfying the conditions of the Representation Method. Finally, we use dynamic programming to simulate a hash of partial assignments into residue classes modulo a large random prime, subsample elements from pairs of corresponding residue classes, and use a Meet-in-the-Middle approach to recover a solution.

3.3.1 Reduction to a Narrower Problem

We start by showing that it suffices to design an algorithm for the core case when $\hat{x} \in |C|^{(1 \pm \varepsilon)n}$ for a small positive constant ε and some information about the solution is known.

Definition 1 (Assignment Profile). Given a coefficient set C , we refer to $\pi = (\pi_c)_{c \in C}$, a partition of n indexed by C , as an *assignment profile*.

We define the *size* $|\pi|$ of an assignment profile to be $n - \pi_0$, the number of elements to which π assigns a nonzero coefficient.

In other words, π is a $|C|$ -tuple of positive integers that sums to n . The idea is that an assignment profile refers to a certain class of solutions: those which assign the coefficient c_0 to π_{c_0} inputs, assign the coefficient c_1 to π_{c_1} inputs, etc. Formally, for an input vector $\vec{x} = (x_1, \dots, x_n)$, a target τ and a profile π , we say $\vec{c} \in C^n$ is a solution to GSS matching the assignment profile π if $\vec{c} \cdot \vec{x} = \tau$ and the number of occurrences of c in the solution vector \vec{c} is π_c for each coefficient $c \in C$.

In order to prove [Theorem 1](#), we reduce GSS to a more specific problem: that of finding a GSS solution matching a specific profile π . Afterwards, it will remain to prove the following theorem:

Theorem 5. Fix $d \in \mathbb{N}_{\geq 1}$, $C = [\pm d]$ or $[-d : d]$. For any sufficiently small constant $\varepsilon > 0$, there exists $\delta = \delta(\varepsilon) > 0$ and a randomized algorithm with running time $|C|^{\Lambda(|C|)^{n+\varepsilon n}}$, where Λ is as defined as in [\(3.1\)](#), with the following performance guarantee:

Given any $\hat{x} \in |C|^{(1 \pm \delta)n}$, τ with $|\tau| = o(n\hat{x})$, and assignment profile π , the algorithm returns a GSS solution matching π , if one exists, with probability $1 - e^{-\Omega(n)}$ over the choice of a random input vector $\vec{x} \sim [0 : \hat{x} - 1]^n$ and randomness internal to the algorithm.

Before proving [Theorem 5](#), we show how it can be applied, along with our structural results, to prove [Theorem 1](#); the proof of [Theorem 2](#) is similar and can be found in [Section 3.3.6](#).

Proof of [Theorem 1](#). Fix a constant $\varepsilon > 0$ sufficiently small and define $\delta := \delta(\varepsilon)$ as required by the statement of [Theorem 5](#). We will show an algorithm for average-case GSS with running time $|C|^{\Lambda(|C|)^{n+\varepsilon n}}$. Given a GSS instance defined by a coefficient set C , a range bound \hat{x} , a target τ satisfying $|\tau| = o(n\hat{x})$, an assignment profile π , and an input vector $\vec{x} \in [0 : \hat{x} - 1]^n$, we split the problem into cases based on C and \hat{x} :

1. $C = [-d : d]$:

- (a) $\hat{x} \geq |C|^{(1+\delta)n}$. It follows from [Theorem 3](#) that the probability of having a solution is $e^{-\Omega(n)}$ ⁹; thus we return ‘No’ and succeed with probability $1 - e^{-\Omega(n)}$.
- (b) $\hat{x} \in |C|^{(1\pm\delta)n}$. Run the algorithm guaranteed by [Theorem 5](#) for every possible profile π and return any solution it finds; return ‘No’ if no solution is found for any profile π . Because there are $\text{poly}(n)$ assignment profiles in total, the chance that any iteration of the algorithm fails remains $e^{-\Omega(n)}$ by a union bound. (Runtime also increases by a $\text{poly}(n)$ factor.)
- (c) $\hat{x} \leq |C|^{(1-\delta)n}$. If $\hat{x} = 2^{o(n)}$, the problem can be solved in subexponential time via dynamic programming. Otherwise, set $n' = \Omega(n)$ such that

$$|C|^{(1-\delta)n'} < \hat{x} < |C|^{(1-\delta/2)n'}.$$

⁹There is a singular edge case: $\vec{0}$ is a solution if and only if $\tau = 0$. For most applications, such as Equal Subset Sum, the $\vec{0}$ solution is trivial and disallowed, but the algorithm is easily modified to handle either case.

Note that τ still satisfies $|\tau| = o(n'\hat{x})$ given $n' = \Omega(n)$.

Run the algorithm guaranteed by [Theorem 5](#) on the first n' input integers, trying all $\text{poly}(n)$ possible assignment profiles. By [Theorem 3](#), there exists a solution with probability at least $1 - e^{-\Omega(n')} = 1 - e^{-\Omega(n)}$ and thus the algorithm guaranteed by [Theorem 5](#) recovers a solution with probability $1 - e^{-\Omega(n)}$. Given that $C = [-d : d]$, we can recover a solution to the original problem by assigning the 0 coefficient to the remaining inputs.

2. $C = [\pm d]$:

- (a) $\hat{x} \geq |C|^{(1+\delta)n}$. As in Case 1(a), answer ‘No’, which is correct with probability $1 - e^{-\Omega(n)}$ by [Corollary 3](#) and [Theorem 4](#).
- (b) $\hat{x} \in |C|^{(1\pm\delta)n}$. As in Case 1(b), run the algorithm guaranteed by [Theorem 5](#) for every possible assignment profile π .
- (c) $\hat{x} \leq |C|^{(1-\delta)n}$. Unlike Case 1(c), we can no longer recover a solution to the original problem by solving a subinstance and assigning the remaining inputs the coefficient 0.

Instead, we can shrink the input instance as follows. Set n' as in Case 1(c). For each input \mathbf{x}_i , $i \in [n' + 1 : n]$, perform the following operation: if the current target τ is positive, assign -1 to \mathbf{x}_i and subtract it from τ to create a new target. If the current target τ is negative, assign $+1$ to \mathbf{x}_i and add it to τ to create a new target.

After this procedure, we are left with n' random numbers $\vec{\mathbf{x}}' = (\mathbf{x}_1, \dots, \mathbf{x}_{n'})$ and a new target τ' with $|\tau'| = o(n'\hat{x})$ such that any solution to $(\vec{\mathbf{x}}', \tau')$ can be extended to a solution to $(\vec{\mathbf{x}}, \tau)$. By [Theorem 4](#), there is a solution with probability at least

$1 - o(1)$ over the randomness of \vec{x}' in the $C = [\pm d]$, $d > 1$ case. By [Corollary 3](#), there is a solution with probability at least $1 - o(1)$ over the randomness of \vec{x}' in the $C = [\pm 1]$ case (if $\sum_i \mathbf{x}_i$ has the same parity as τ), as the shrinking procedure preserves the parity of $\sum_i \mathbf{x}_i - \tau$. This implies that running the algorithm of [Theorem 5](#) on \vec{x}' and τ' for all profiles π finds a solution with probability at least $1 - o(1)$, which can be extended to a solution to \vec{x} and τ . \square

3.3.2 Overview of the GSS Algorithm

It remains to prove [Theorem 5](#) by demonstrating an algorithm that solves GSS in time $|C|^{\Lambda(|C|)n + \varepsilon n}$, given an assignment profile π and the guarantee that $\hat{x} \in |C|^{(1 \pm \delta)n}$ for some sufficiently small $\delta := \delta(\varepsilon)$.¹⁰

Terms and Definitions

In a nutshell, our algorithm extends the Representation Method to more general coefficient sets. Recall from [Section 1.1.2](#) that the Representation Method works by first constructing exponentially many partial candidate solutions, each of which constitutes a guess at a partial solution. In the case of GSS on $C = \{0, 1\}$, in which we can assume without loss of generality that solutions contain at most $n/2$ elements, partial candidate solutions are subsets of the input of size at most $n/4$.

¹⁰In the proofs that follow, we restrain ourselves to the observation that sufficiently small $\varepsilon > 0$ and $\delta(\varepsilon) > 0$ are sufficient for the truth of the statements we wish to prove, where relevant. The constraints on δ and ε follow from the following requirements:

1. $\frac{1}{|C|} > \frac{\varepsilon}{|C-1|} + 2\varepsilon \log_2(|C|)$, from [\(3.111\)](#)
2. $\delta < f(\varepsilon)/\log_2(|C|)$, from [\(3.115\)](#), for f as in [\(3.114\)](#).

We proceed to generalize the notions of candidate solutions and partial candidate solutions.

Definition 2 (Half Coefficient Vectors (HCVs)). Given a coefficient set C , a candidate solution is specified by a coefficient vector $\vec{c} \in C^n$. Any coefficient vector \vec{c} admits numerous *half coefficient vectors (HCVs)*, each of which is an n -dimensional vector $\vec{h} \in C^n$ such that for each nonzero coefficient $c \in C \setminus \{0\}$, the set of indices assigned coefficient c satisfies

$$\{i \in [n] : \vec{h}_i = c\} \subseteq \{i \in [n] : \vec{c}_i = c\}, \text{ and} \tag{3.98}$$

$$|\{i \in [n] : \vec{h}_i = c\}| \in \frac{|\{i \in [n] : \vec{c}_i = c\}|}{2} \pm 1/2. \tag{3.99}$$

We denote the set of all HCVs of a coefficient vector \vec{c} by $HCV(\vec{c})$. The *size* of an HCV is its number of non-zero elements, which is at most $n/2 + O(|C|)$.

In other words, every non-zero element of \vec{h} matches \vec{c} , and the number of elements of \vec{h} that take the coefficient c is half the number of elements of \vec{c} that take the coefficient c (with a margin of $\pm 1/2$ added to account for parity).

We refer to two HCVs \vec{h}, \vec{h}' of the coefficient vector \vec{c} as a *matching pair* if $\vec{h} + \vec{h}' = \vec{c}$. (Note that this implies \vec{h} and \vec{h}' do not assign the same nonzero coefficient to any element.) If \vec{c} is a solution, (in which case $\vec{h} \cdot \vec{x} + \vec{h}' \cdot \vec{x} = (\vec{h} + \vec{h}') \cdot \vec{x} = \tau$) we refer to \vec{h} and \vec{h}' as a *solution pair*.

Crucial to the success of the Representation Method when $C = \{0, 1\}$ is the fact that solution pairs have complementary subset sums: if A and B are disjoint subsets of the input and $\Sigma(A) + \Sigma(B) = \tau$, then $\Sigma(A) = \tau - \Sigma(B)$. This simplifies our task, as to search for a solution pair involving B we need only look at subsets of the input that add to $\tau - \Sigma(B)$. The analogue of the sum of elements in a partial candidate solution is the signature of a

HCV:

Definition 3 (Signature). The *signature* of a (half) coefficient vector \vec{v} with respect to an input vector \vec{x} is the dot product $\vec{v} \cdot \vec{x}$.

Each assignment profile π corresponds to an exponential number of coefficient vectors. We write $C^n[\pi]$ to denote this set; that is, the set of all coefficient vectors that have π_c elements equal to the coefficient c , for each $c \in C$. Extending our previous notation, we write $HCV(C^n[\pi])$ to denote the set of all HCVs that match a coefficient vector in $C^n[\pi]$.

Our algorithm will search within $HCV(C^n[\pi])$, the set of all HCVs matching any coefficient vector with the assignment profile π , and attempt to recover a solution pair in $HCV(\vec{c})$ for some solution vector \vec{c} . Because the size of $HCV(C^n[\pi])$ and the number of solution vectors will be important to our algorithm, we introduce some notation in advance.

We introduce the shorthand

$$a(\pi) = |HCV(\vec{c})|$$

to denote the size of the set of HCVs matching an arbitrary vector \vec{c} that fits the assignment profile π . (Note that this quantity is the same for any \vec{c} that fits the assignment profile π , so identifying a specific \vec{c} is unnecessary.) When π is understood from context, we simply write a .

We have

$$\begin{aligned} a(\pi) &= \prod_{c \in C \setminus \{0\}} \Theta^*(2^{|\pi_c|}) \\ &= \Theta^*(2^{|\pi|}), \end{aligned} \tag{3.100}$$

as a consequence of Stirling's approximation (2.3).

We also introduce the shorthand

$$b(\pi) = |HCV(C^n[\pi])|$$

to denote the size of our search space, and observe that the size of $HCV(\pi)$ can be calculated by using Stirling's approximation for multinomials.

Let

$$\pi_{h0} = \pi_0 + \sum_{c \in C \setminus \{0\}} \left\lfloor \frac{\pi_c}{2} \right\rfloor$$

denote the minimum number of 0 coefficients in an HCV of a coefficient vector that fits the assignment profile π . Writing

$$\alpha := \frac{|\pi|}{n},$$

we have

$$\begin{aligned} b(\pi) &= O^* \left(\binom{n}{\left\{ \left\lfloor \frac{\pi_c}{2} \right\rfloor \right\}_{c \in C \setminus \{0\}}, \pi_{h0}} \right) \\ &= O^*(2^{H_\pi n}) \quad \text{where} \quad H_\pi = H \left(\left\{ \frac{\pi_c}{2n} \right\}_{c \in C \setminus \{0\}}, \frac{\pi_{h0}}{n} \right) \\ &\leq H \left(\frac{\alpha}{2(|C|-1)}, \dots, \frac{\alpha}{2(|C|-1)}, 1 - \frac{\alpha}{2} \right) \\ &= \frac{\alpha}{2} \cdot \log_2 \left(\frac{2(|C|-1)}{\alpha} \right) + \left(1 - \frac{\alpha}{2} \right) \cdot \log_2 \left(\frac{2}{2-\alpha} \right). \quad (3.101) \end{aligned}$$

Here, the first line uses the fact that the distribution of coefficients

$$\underbrace{\left\lfloor \frac{\pi_c}{2} \right\rfloor, \left\lfloor \frac{\pi_c}{2} \right\rfloor, \dots, \left\lfloor \frac{\pi_c}{2} \right\rfloor}_{|C|-1}, \pi_{h0}$$

maximizes entropy and thus the number of HCVs (O^* notation hides a union bound over HCVs with coefficient distribution that differs by up to $O(|C|)$), the second uses Stirling's approximation for multinomials (2.6), and the third line uses as an upper bound the assignment profile that maximizes the entropy function.

The Algorithm

Algorithm 3.9 outlines our algorithm for solving GSS instances with a fixed assignment profile. This subsection provides a high-level description of the algorithm, but in order to simplify the presentation, certain implementation details are deferred to the proofs of correctness and runtime (**Propositions 1** and **2**, below).

The input to the algorithm consists of an error parameter ε , a target τ , a coefficient set $C = [-d : d]$ or $C = [\pm d]$ for some constant d , an assignment profile π , and an input vector \vec{x} satisfying $\vec{x} \in [0 : \hat{x} - 1]^n$ for some $\hat{x} \in |C|^{(1 \pm \delta)}$, where $\delta := \delta(\varepsilon)$ is a sufficiently small constant depending on ε .

If a solution exists, the algorithm wants to recover one of $a(\pi)$ solution pairs hidden within a set of $b(\pi)$ HCVs. Following the Representation Method, we reduce the size of the search space while preserving at least one solution pair with high probability. To accomplish this, we select a large random prime \mathbf{p} and divide the set $HCV(C^n[\pi])$ into residue classes based on the signature of each HCV modulo \mathbf{p} .

Each residue class contains $b(\pi)/\mathbf{p}$ HCVs in expectation. Thus, if we set $\mathbf{p} \approx a(\pi)/n$, any solution \vec{c} that fits the assignment profile π corresponds to $\Omega(n)$ elements of $HCV(\vec{c})$ in each residue class in expectation. Moreover, if one element \vec{h} of a solution pair falls into the residue class $r \pmod{\mathbf{p}}$, this guarantees that the other element \vec{h}' of the same solution pair will fall into the residue class $\tau - r \pmod{\mathbf{p}}$, due to the fact that $\vec{h} \cdot \vec{x} + \vec{h}' \cdot \vec{x} = \tau$.

The algorithm selects $\mathbf{r} \sim [\mathbf{p}]$ uniformly at random and generates the list of HCVs whose signatures fall into the residue classes $\mathbf{r} \pmod{\mathbf{p}}$ and $\tau - \mathbf{r} \pmod{\mathbf{p}}$. Finally, we run a Meet-in-the-Middle subroutine on the two lists to recover a solution pair.

For $|C| \leq 3$, it holds that

$$b(\pi)^{2/3}a(\pi)^{-1/3} < a(\pi), \quad (3.102)$$

and turns out to be more efficient to subsample solution pairs from larger residue classes than to enumerate the entirety of smaller residue classes. In these cases, we choose $\mathbf{p} \approx b(\pi)^{2/3}a(\pi)^{-1/3}$ so that each pair of residue classes contains an exponential number of matching pairs in expectation. After choosing a pair of residue classes, we subsample each residue class to create solution lists that are likely to contain at least one matching pair. We then use a Meet-in-the-Middle algorithm on the subsampled lists to recover a solution as before.

3.3.3 Implementation Details and Signature Distribution Lemma

Although the idea behind [Algorithm 3.9](#) is relatively straightforward, a correct implementation requires the careful navigation of several overlapping technical requirements. First, in order for our algorithm to recover a solution \vec{c} , we need the solution pairs in $HCV(\vec{c})$ to have many distinct signatures. This is necessary to ensure that most pairs of residue classes contain solution pairs. Otherwise, our approach of sampling a small number of residue class pairs may not recover a solution, even if one exists.

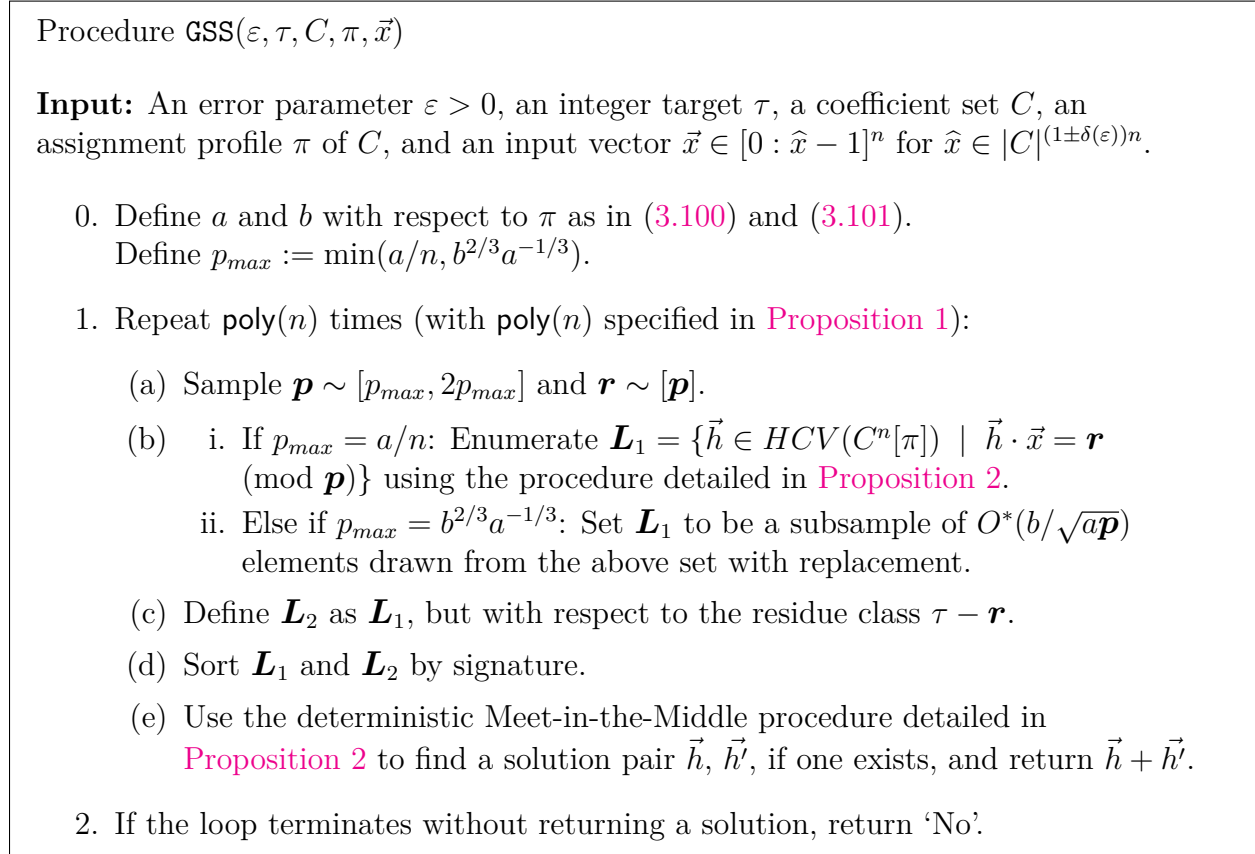


Figure 3.9: An algorithm to recover GSS solutions that fit a given assignment profile π , as required for Theorem 1. For ease of readability, precise implementation details are deferred to the proofs of correctness and runtime (Propositions 1 and 2) below.

Distribution of Solution HCV Signatures

Lemma 15 will establish that with very high probability over $\vec{x} \sim [0 : \hat{x} - 1]^n$, either there exists a solution \vec{c} such that $HCV(\vec{c})$ contains many distinct signatures, or \vec{x} contains no solutions at all. However, as a technical requirement for the lemma, we must first translate our input instance as follows.

Define

$$z := \arg \max_{c \in C \setminus \{0\}} \pi_c \quad (3.103)$$

to be the nonzero coefficient that maximizes π_z . If we consider the coefficient set

$$C' := C - z, \quad (3.104)$$

our GSS problem becomes that of finding $\vec{c} \in C'^m$ such that

$$\vec{c} \cdot \vec{x} = \tau' := \tau - z\Sigma(\vec{x}), \quad (3.105)$$

subject to the requirement that \vec{c} fits the assignment profile π' defined for each $c \in C'$ as

$$\pi'_{c-z} := \pi_c. \quad (3.106)$$

We will run [Algorithm 3.9](#) on the *translated* instance defined by τ' , C' , and π' . Any solution \vec{c} to the translated instance can be easily mapped back to the original problem by adding z to each component; indeed, solutions are preserved via a one-to-one mapping. In that case, why bother with the translation? The reason is that the translation assures a crucial property required for [Lemma 15](#): specifically, the new target τ' now depends on the input randomness. This in turn will ensure that the signatures of solution pairs distribute as intended while also maintaining an upper bound on the solution size $|\pi'|$.

Lemma 15 (Signature Distribution Lemma). *Fix a coefficient set C , a target τ , $\hat{x} \in |C|^{(1 \pm \delta)n}$ for some sufficiently small constants $\varepsilon > 0$ and $\delta = \delta(\varepsilon) > 0$, and an assignment profile π . Define C' , τ' , and π' as in [\(3.104\)](#), [\(3.105\)](#), and [\(3.106\)](#).*

With probability $1 - e^{-\Omega(n)}$ over $\vec{x} \sim [0 : \hat{x} - 1]^n$, either there exists a solution $\vec{c} \in C^m$, matching π' and satisfying $\vec{c} \cdot \vec{x} = \tau'$, such that $HCV(\vec{c})$ contains elements with $\Omega(|HCV(\vec{c})|)$ distinct signatures, or no solution exists.

Proof. We prove the lemma via a counting argument. Consider a quadruple

$$(\vec{x}, \vec{c}, \vec{h}, \vec{h}'),$$

where $\vec{x} \in [0 : \hat{x} - 1]^n$ is an input vector, $\vec{c} \in C^m$ is a coefficient vector matching the assignment profile π' , and \vec{h} and \vec{h}' are distinct HCVs of \vec{c} (not necessarily a matching pair). We refer to such a quadruple as a *signature-collision* if

$$\vec{h} \cdot \vec{x} = \vec{h}' \cdot \vec{x},$$

that is, if h and h' have the same signature with respect to x .

We proceed to bound the total number of signature-collisions. Fix $\vec{c} \in C^m$ and two distinct vectors \vec{h} and \vec{h}' that are both HCVs of \vec{c} . Because \vec{h} and \vec{h}' are distinct by assumption, there exists some index $k \in [n]$ on which \vec{h} and \vec{h}' differ. Without loss of generality, suppose that $\vec{h}_k = 0 \neq \vec{h}'_k$ (as this component can take only the value \vec{c}_k or 0).

For any fixed input vector $\vec{x}_{[n] \setminus \{k\}}$, there exists at most one value for x_k for which the quadruple $(\vec{x}, \vec{c}, \vec{h}, \vec{h}')$ is a signature collision. Thus, the number of input vectors $\vec{x} \in [0 : \hat{x} - 1]^n$ with respect to which \vec{h} and \vec{h}' have the same signature is at most \hat{x}^{n-1} . Union-bounding over the $O(a(\pi')^2)$ distinct pairs of HCVs drawn from $HCV(\vec{c})$ and $|C^n|$ possible coefficient vectors, we conclude that there are at most

$$O(|C|^n a(\pi')^2 \hat{x}^{n-1})$$

signature-collisions in total.

Call a fixed pair (\vec{x}, \vec{c}) a “bad” pair if $HCV(\vec{c})$ has $o(|HCV(\vec{c})|) = o(a(\pi'))$ distinct signatures with respect to \vec{x} . This is only possible if there are at least $\Omega(|HCV(\vec{c})|) = \Omega(a(\pi'))$ signature-collisions corresponding to the pair (\vec{x}, \vec{c}) ; otherwise, most HCVs of \vec{c} do not collide with any others. Thus any bad pair (\vec{x}, \vec{c}) “uses up” $\Omega(a(\pi'))$ signature collisions, and there can be at most $O(|C'|^n a(\pi') \hat{x}^{n-1})$ bad pairs.

Finally, we consider the pair $(\vec{\mathbf{x}}, \vec{c})$, in which \vec{c} is fixed and $\vec{\mathbf{x}} \sim [0 : \hat{x} - 1]^n$. The crucial observation is that the event that $(\vec{\mathbf{x}}, \vec{c})$ is a bad pair is independent of the draw of each \mathbf{x}_i for which $c_i = 0$.

Thus if (\vec{x}, \vec{c}) is a bad pair, this implies that \vec{c} participates in $\hat{x}^{|\pi'_0|}$ bad pairs, one for every possible assignment of the irrelevant indices of \vec{x} . On the other hand, thanks to our translation, whether \vec{c} is a solution *does* depend on the indices of $\vec{\mathbf{x}}$ for which $c_i = 0$. Because

$$\boldsymbol{\tau}' = \boldsymbol{\tau} - z \Sigma(\mathbf{x}_i)$$

for a nonzero coefficient z , $\boldsymbol{\tau}'$ depends on every component of $\vec{\mathbf{x}}$. In particular, this implies that the event $(\vec{\mathbf{x}}, \vec{c})$ is bad is independent from the event that \vec{c} is a solution of $\vec{\mathbf{x}}$. Because the event that \vec{c} is a solution of $\vec{\mathbf{x}}$ occurs with probability at most $1/\hat{x}$, it follows that there are at most

$$O(|C'|^n a(\pi') \hat{x}^{n-2})$$

bad pairs (\vec{x}, \vec{c}) such that \vec{c} is a solution for \vec{x} .

We can use this quantity as an upper bound on the number of input vectors \vec{x} that have any solution $\vec{c} \in C'^n$ such that $HCV(\vec{c})$ contains elements with $o_n(a(\pi'))$ distinct signatures.

Evaluating this expression yields

$$O(|C'|^n a(\pi') \hat{x}^{n-2}) = O^*(|C|^n 2^{|\pi'|} \hat{x}^{n-2}) \quad (3.107)$$

using the fact that $|C'| = |C|$ and substituting for $a(\pi')$ using (3.100). By Lemma 16 (proved below), we have that, conditioning on

$$|\pi'| > \left(1 - \frac{1}{|C|} + \frac{\varepsilon}{|C| - 1}\right) n, \quad (3.108)$$

the probability over a random instance $\vec{x} \sim [0 : \hat{x} - 1]^n$ that a solution exists is at most $e^{-\Omega(n)}$. Taking a union bound over the solution space, there exist at most $e^{-\Omega(n)} \cdot \hat{x}^n$ input vectors \vec{x} such that (3.108) holds and \vec{x} admits a solution. On the other hand, if (3.108) does not hold, substituting this fact into (3.107) yields

$$O^*(|C|^n 2^{|\pi'|} \hat{x}^{n-2}) \leq O^*(\hat{x}^{n-2} \cdot |C|^n \cdot 2^{(1 - \frac{1}{|C|} + \frac{\varepsilon}{|C| - 1})n}) \quad (3.109)$$

$$\leq O^*(\hat{x}^n) \cdot |C|^{2\varepsilon n} \cdot 2^{-\left(\frac{1}{|C|} - \frac{\varepsilon}{|C| - 1}\right)n} \quad (3.110)$$

$$\leq O^*(\hat{x}^n) \cdot 2^{-\left(\frac{1}{|C|} - \frac{\varepsilon}{|C| - 1} - 2\varepsilon \log_2(|C|)\right)n}, \quad (3.111)$$

where the second line uses the fact that $2^n \leq |C|^n \leq \hat{x} \cdot |C|^{\varepsilon n}$ and the third line regroups. The resulting quantity represents an $e^{-\Omega(n)}$ -fraction of the input space when $\varepsilon > 0$ is sufficiently small.

Union-bounding over (1) the number of input vectors \vec{x} for which (3.108) holds and \vec{x} still admits a solution, which is an $e^{-\Omega(n)}$ -fraction of the input space by Lemma 16, and (2) the number of input vectors \vec{x} such that (3.108) does not hold and \vec{x} has a solution \vec{c} with few distinct signatures, which is an $e^{-\Omega(n)}$ -fraction of the input space by (3.111), finishes the

proof of the lemma. □

Lemma 16 (Solution Size Bound). *Fix a coefficient set C , a target τ , $\hat{x} \in |C|^{(1 \pm \delta)^n}$ for some sufficiently small constants $\varepsilon > 0$ and $\delta = \delta(\varepsilon) > 0$, and an assignment profile π . Define C' , τ' , and π' as in (3.104), (3.105), and (3.106).*

If π' satisfies

$$|\pi'| > \left(1 - \frac{1}{|C|} + \frac{\varepsilon}{|C| - 1}\right) n, \quad (3.112)$$

then $\vec{x} \sim [0 : \hat{x} - 1]^n$ admits a solution \vec{c} that matches π' with probability at most $e^{-\Omega(n)}$.

Proof. We consider two cases, depending on whether the number of 0 coefficients assigned by the original (untranslated) solution profile π satisfies

$$\pi_0 \geq \left(\frac{1}{|C|} + \varepsilon\right) n. \quad (3.113)$$

Case 1: (3.113) holds. The number of coefficient vectors that match π' is

$$\binom{n}{\{\pi_c\}_{c \in C}} = \Theta^* \left(2^{H\left(\left\{\frac{\pi_c}{n}\right\}_{c \in C}\right)}\right)$$

by (2.6), Stirling's Approximation for multinomial coefficients. If we upper bound by the number of solutions matching the assignment profile π which maximizes the entropy function, we have that the number of coefficient vectors that match π is

$$O^*(2^{\eta}), \text{ where } \eta = H \left(\frac{1}{|C|} + \varepsilon, \underbrace{\frac{1}{|C|} - \frac{\varepsilon}{|C| - 1}, \dots, \frac{1}{|C|} - \frac{\varepsilon}{|C| - 1}}_{|C|-1} \right)$$

Significantly,

$$\eta = \log_2(|C|) - f(\varepsilon) \quad (3.114)$$

for some function f that is nonnegative and increasing on $[0, 1]$.

Each of the $O^*(2^m)$ coefficient vectors is a solution with probability at most $1/\hat{x}$ over $\vec{x} \sim [0 : \hat{x} - 1]^n$. (To see this, consider that for any fixed \vec{c} and value of $\vec{x}_{[n-1]}$, there is at most one choice of \mathbf{x}_n such that $\vec{c} \cdot \vec{x} = \tau$.) Union-bounding over every coefficient vector matching π , we have that for any

$$\delta < \frac{f(\varepsilon)}{\log_2(|C|)}, \quad (3.115)$$

the probability of having a solution matching π is

$$O^* \left(\frac{|C|^n \cdot 2^{-f(\varepsilon)n}}{\hat{x}} \right) \leq O^* \left(2^{-f(\varepsilon)n} \cdot |C|^\delta n \right) = e^{-\Omega(n)}.$$

Case 2: (3.113) does not hold. Thus we have

$$\pi_0 \leq \left(\frac{1}{|C|} + \varepsilon \right) n$$

When we translate the solution profile, π_0 becomes π'_{c-z} . Due to the choice of z as the nonzero coefficient that maximizes π_z , we have that

$$\begin{aligned} \pi'_0 &\geq \frac{1 - \left(\frac{1}{|C|} + \varepsilon \right)}{|C| - 1} n \\ &= \frac{1}{|C|} - \frac{\varepsilon}{|C| - 1} \end{aligned}$$

and thus

$$|\pi'| \leq \left(1 - \frac{1}{|C|} + \frac{\varepsilon}{|C| - 1} \right) n,$$

violating our initial assumption (3.112). □

3.3.4 Proof of Correctness

Proposition 1 (Correctness of Algorithm 3.9). *Fix $C = [\pm d]$ or $[-d : d]$, $\varepsilon > 0$, $\hat{x} \in |C|^{(1 \pm \delta)n}$ for ε and $\delta = \delta(\varepsilon)$ sufficiently small, and $\tau = o(n\hat{x})$. Define C' , π' and $\boldsymbol{\tau}'$ with respect to $\vec{x} \sim [0 : \hat{x} - 1]^n$ as in (3.104), (3.105), and (3.106).*

The algorithm outlined in Algorithm 3.9, when implemented as described below, succeeds on the input $(\varepsilon, \boldsymbol{\tau}', C', \pi', \vec{x})$ with probability $1 - e^{-\Omega(n)}$.

The algorithm succeeds automatically if the input instance does not admit a solution, as it never returns a false positive. Thus by Lemma 15 (the Signature Distribution Lemma), we can assume without loss of generality that with probability $1 - e^{-\Omega(n)}$ there exists a coefficient vector $\vec{c} \in C^m$ matching π' and satisfying $\vec{c} \cdot \vec{x}$, such that $HCV(\vec{c})$ contains elements with $\Omega(|HCV(\vec{c})|) = \Omega(a(\pi')) = \Omega(a)$ distinct signatures. Conditioning on this event, fix some such solution \vec{c} and let $\mathbf{S} := \mathbf{S}(\vec{x}, \vec{c})$ denote the set of distinct signatures of the HCVs of c .

We divide the proof into two parts:

1. We first prove that with very high probability, Algorithm 3.9 generates a “good pair” (\mathbf{p}, \mathbf{r}) , which will result in a relatively short \mathbf{L}_1 and \mathbf{L}_2 containing solution pairs, at least once during the loop in Step 1 of Lemma 17.
2. We then complete the proof of Proposition 1 by showing that, given an iteration of the loop in Step 1 which generates a “good pair”, Algorithm 3.9 recovers a solution with high probability.

Lemma 17. *With respect to Step 1 of Algorithm 3.9, an input vector \vec{x} , and a solution \vec{c} with $\Omega(|HCV(\vec{c})|) = \Omega(a)$ distinct signatures, define a good pair $(p, r) \in [p_{max} : 2p_{max}] \times [p]$*

to be one that satisfies

1. There exist $\Omega\left(\frac{a}{p}\right)$ elements of $HCV(\vec{c})$ with signatures equal to $r \pmod{p}$.
2. There exist $O\left(\frac{bn^3}{p}\right)$ elements of $HCV(C^n[\pi])$ with signatures equal to $r \pmod{p}$.

At least one iteration of the loop in Step 1 of [Algorithm 3.9](#) chooses a good pair for some solution \vec{c} with probability $1 - e^{-\Omega(n)}$ over the choice of \vec{x} , \mathbf{p} , and \mathbf{r} , conditioned on the event that a solution exists.

Proof. Condition on the event that there exists at least one solution \vec{c} with $\Omega(|HCV(\vec{c})|) = \Omega(a)$ distinct signatures, which occurs with probability $1 - e^{-\Omega(n)}$ by [Lemma 15](#) as long as some solution exists, and define the random set \mathbf{S} as above. We first observe that

$$|\mathbf{S}| \log_2(\text{diam}(\mathbf{S})) \leq a \cdot \log_2(d\hat{x}) \tag{3.116}$$

$$\leq O(an) \tag{3.117}$$

$$\leq O(n^2 p_{max}) \tag{3.118}$$

where the first line follows from our assumption, the second from the fact that $\hat{x} = |C|^{O(n)}$, and the third from the definition of p_{max} in [Algorithm 3.9](#). By [Lemma 9](#), for $\mathbf{p} \sim [p_{max} : 2p_{max}]$ the set

$$\mathbf{R} = \left\{ r \in [\mathbf{p}] \mid \left| \{s \in \mathbf{S} \mid s = r \pmod{\mathbf{p}}\} \right| \geq \frac{|\mathbf{S}|}{2\mathbf{p}} = \Omega\left(\frac{a}{p}\right) \right\}$$

of residue classes corresponding to at least a $1/2\mathbf{p}$ -fraction of the set of distinct signatures has cardinality

$$|\mathbf{R}| \geq \frac{|\mathbf{S}|}{n^3} \tag{3.119}$$

with constant probability over the choice of \mathbf{p}, \mathbf{r} .

We would also like to bound the number of residue classes that contain too many elements of $HCV(C^n[\pi'])$. For this purpose, condition on the choice of \mathbf{p}, \mathbf{r} satisfying (3.119) and observe that the set of elements in $HCV(C^n[\pi'])$ whose signatures fall into any randomly selected residue class $\mathbf{r} \in |\mathbf{R}|$ has cardinality at most

$$\begin{aligned} \left| \left\{ \vec{h} \in HCV(C^n[\pi']) \mid \vec{h} \cdot \vec{c} = \mathbf{r} \pmod{\mathbf{p}} \right\} \right| &\leq \frac{b}{|\mathbf{R}|} \\ &< \frac{bn^3}{|\mathbf{S}|} \\ &= O\left(\frac{bn^3}{a}\right) \end{aligned}$$

in expectation over the choice of \mathbf{r} , and thus is $O(\frac{bn^3}{a})$ for a constant fraction of the elements in \mathbf{R} by Markov's inequality. In other words, with constant probability over the choice of \mathbf{p} and \mathbf{r} , $\Omega(|\mathbf{R}|)$ residues correspond to a good pair and \mathbf{R} satisfies (3.119). We conclude that the conditions in the lemma statement hold with probability

$$\begin{aligned} \Omega\left(\frac{|\mathbf{R}|}{p}\right) &= \Omega\left(\frac{|\mathbf{S}|}{n^3 p}\right) \\ &= \Omega\left(\frac{a}{n^3 p}\right) \\ &= \Omega\left(\frac{a}{n^3 \min\left[\frac{a}{n}, \frac{b^{2/3}}{a^{1/3}}\right]}\right) \\ &= \Omega\left(\frac{1}{n^3}\right) \end{aligned}$$

for a randomly sampled pair (\mathbf{p}, \mathbf{r}) , where the second line uses

$$2p_{max} = 2 \min \left[\frac{a}{n}, \frac{b^{2/3}}{a^{1/3}} \right]$$

as a lower bound on \mathbf{p} and the third line uses (3.102). We can inflate the probability of choosing a good pair to $1 - e^{-\Omega(n)}$ by choosing independent pairs (\mathbf{p}, \mathbf{r}) a polynomial number of times as in Algorithm 3.9. \square

Proof of Proposition 1. By Lemma 17, if a solution exists, in some iteration of the loop in Step 1 of Algorithm 3.9, we choose a good pair (p, r) satisfying that for some solution \vec{c} ,

1. There exist $\Omega\left(\frac{a}{p}\right)$ elements of $HCV(\vec{c})$ with signatures equal to $r \pmod{p}$.
2. There exist $O\left(\frac{bn^3}{p}\right)$ elements of $HCV(C^n[\pi])$ with signatures equal to $r \pmod{p}$.

with probability $1 - e^{-\Omega(n)}$. We condition on this event and consider the probability that the algorithm recovers a solution on this iteration of the loop. We consider two cases based on the choice of p_{max} .

Case 1: $a/n < b^{2/3}a^{-1/3}$. In this case the algorithm sets $p_{max} = a/n$. The residue class pair $(r, \boldsymbol{\tau}' - r)$ corresponds to

$$\Omega\left(\frac{a}{p}\right) = \Omega(n)$$

solution pairs, so a deterministic search for solutions over the two lists guarantees solution recovery.

Case 2: $a/n > b^{2/3}a^{-1/3}$. In this case the algorithm sets $p_{max} = b^{2/3}a^{-1/3}$. Let

$$s := \Omega\left(\frac{a}{p}\right) = \Omega\left(a \frac{a^{1/3}}{b^{2/3}}\right) = \Omega(n) \tag{3.120}$$

denote the number of solution pairs corresponding to the residue class pair $(r, \boldsymbol{\tau}' - r)$. In this case, the algorithm creates \mathbf{L}_1 and \mathbf{L}_2 by subsampling

$$\frac{bn^4}{\sqrt{ap}} = O^* \left(\frac{b}{\sqrt{ap}} \right)$$

elements from each residue class uniformly at random with replacement. Thus, in expectation,

$$\begin{aligned} \frac{bn^{9/5}}{\sqrt{ap}} \cdot \Omega \left(\frac{s}{\frac{bn^3}{p}} \right) &= \Omega \left(sn \sqrt{\frac{p}{a}} \right) \\ &= \Omega(n\sqrt{s}) \end{aligned}$$

elements in each list are halves of some solution pair, where the second line follows from (3.120). Because we sample each element independently with replacement, applying a Chernoff bound implies that both lists contain

$$m := \Omega(n\sqrt{s})$$

solution pair halves with probability $1 - e^{-\Omega(n)}$.

It remains to estimate the probability that our two lists contain both halves of the same matching pair. Conditioning on the event that both lists contain $m = \Omega(n\sqrt{s})$ solution pair halves, our solution pair halves are sampled independently with replacement from the set of s solution pairs. Thus the chance we recover no solution pair is at most $e^{-\Omega(n)}$ by the Birthday Bound (Lemma 1). Conditioning on the event that our lists contain a solution pair, a deterministic search over the two lists guarantees recovery. \square

3.3.5 Proof of Runtime

Proposition 2 (Runtime of [Algorithm 3.9](#)). *Fix $C = [\pm d]$ or $[-d : d]$, $\varepsilon > 0$, $\hat{x} \in |C|^{(1\pm\delta)n}$ for ε and $\delta = \delta(\varepsilon)$ sufficiently small, and $\tau = o(n\hat{x})$. Define C' , π' and τ' with respect to $\vec{x} \sim [0 : \hat{x} - 1]^n$ as in [\(3.104\)](#), [\(3.105\)](#), and [\(3.106\)](#).*

The algorithm outlined in [Algorithm 3.9](#), when implemented as described below and in the proof of [Proposition 1](#), runs in time $|C|^{\Lambda(|C|)^{n+O(\varepsilon n)}}$, where Λ is defined as in [\(3.1\)](#).

Proof. Steps 0 and 1(a) of [Algorithm 3.9](#) are subexponential. We begin by describing the implementation of steps 1(b) and 1(c), in which we create the lists \mathbf{L}_1 and \mathbf{L}_2 efficiently.

The set $HCV(C'^n[\pi'])$ is far too large for us to enumerate it in full, so we must come up with another way to enumerate HCVs whose signatures fall into a certain residue class r . We can accomplish this via dynamic programming. Construct a $n \times p$ table in which each cell $(i, j) \in [n] \times [p]$ stores the number of coefficient vectors $\vec{c} \in C'^n$ such that $\vec{c} \cdot x_{[i]} = j \pmod{p}$; that is, the number of coefficient vectors which assign nonzero coefficients to only the first i inputs and whose signatures fall into the residue class j . Within each cell, we partition the total number of coefficient vectors into a sub-array of size $\text{poly}(n)$ that indicates how many coefficient vectors match each of the $\text{poly}(n)$ possible assignment profiles.

We observe that each cell (i, j) can be filled by consulting the cells

$$(i-1, j - cx_i \pmod{p})$$

for each $c \in C'$. As a result, computing the value in each cell takes time $O_{|C'|}(\text{poly}(n))$, and filling out the entire table takes time $O^*(p)$.

The dynamic programming table allows us to sample uniformly at random from HCVs in $HCV(C'^n[\pi'])$ that have signatures that fall into the residue class $r \pmod{p}$. To do this, we

begin at cell (n, r) and consider only those coefficient vectors indexed by assignment profiles corresponding to HCVs. We sample a coefficient $c \in C'$, weighting by the number of half coefficient vectors that assign the coefficient c to x_n . We then backtrack to cell $(n-1, r - cx_n \pmod{p})$. We continue this weighted backtracking process until we recover a single HCV sampled uniformly at random from all HCVs in $HCV(C'^n[\pi'])$ whose signatures fall into the residue class $r \pmod{p}$ recovered.

Step 1(d), which sorts \mathbf{L}_1 and \mathbf{L}_2 by signature, takes time

$$O((|\mathbf{L}_1| + |\mathbf{L}_2|) \cdot \log(|\mathbf{L}_1| + |\mathbf{L}_2|)) = O^*(|\mathbf{L}_1| + |\mathbf{L}_2|).$$

This is exceeded by the deterministic solution recovery procedure performed in Step 1(e).

The final step, Step 1(e), searches $\mathbf{L}_1 \times \mathbf{L}_2$ for a matching pair. Our recovery procedure is exactly as described in [Algorithm 1.1](#), with one additional step: in addition to finding a pair of HCVs \vec{h} and \vec{h}' such that

$$\vec{h} \cdot \vec{x} + \vec{h}' \cdot \vec{x} = \vec{\tau}',$$

we must also check to make sure that \vec{h} and \vec{h}' are a solution pair; that is, \vec{h} and \vec{h}' assign nonzero coefficients to disjoint subsets of the input.

We refer to a pair of HCVs \vec{h} and \vec{h}' that satisfy $\vec{h} \cdot \vec{x} + \vec{h}' \cdot \vec{x} = \vec{\tau}'$ but are not a solution pair as a *pseudosolution pair*. Although checking and discarding a single pseudosolution pair takes time $\text{poly}(n)$, our recovery procedure must check them all. The total runtime of step 1(e) is thus

$$O^*(\max(|\mathbf{L}_1|, |\mathbf{L}_2|, \text{ps}(\mathbf{L}_1, \mathbf{L}_2))),$$

where $\text{ps}(\mathbf{L}_1, \mathbf{L}_2)$ denotes the number of pseudosolution pairs.

The total runtime of the algorithm is thus bounded by Steps 1(b) and (c), the time it takes to create the table, and Step 1(e), the time it takes to search for a solution pair. We divide the remainder of the proof into two cases based on the choice of p_{max} in Step 0 of [Algorithm 3.9](#).

Case 1: $p_{max} = a/n$. In this case, we sample all HCVs that fall into the residue classes \mathbf{r} and $\mathbf{r} - \boldsymbol{\tau}' \pmod{p}$. Under the assumption that (p, r) is a good pair as defined in [Lemma 17](#), this takes time $O^*(b/p)$. If there are more than $O(bn^2/p)$ elements in our residue class, we know that (p, r) is not a good pair and abort the loop.

To bound the number of pseudosolutions in Case 1, we can apply linearity of expectation over all distinct pairs of half coefficient vectors in $HCV(C^n[\pi'])$. The expected number of pseudosolutions over a random input \vec{x} , and uniformly random \mathbf{p} and \mathbf{r} , is

$$O^* \left(\frac{b^2}{\hat{x}p} \right) = \frac{b}{p} \cdot e^{-\Omega(n)},$$

where it follows from [\(3.101\)](#) that b is smaller than $\hat{x} \in |C^n|^{(1 \pm \varepsilon)n}$ by a factor of $e^{\Omega(n)}$. Our choice of \mathbf{p} and \mathbf{r} is not uniformly random, however: we have conditioned on the choice of a good p and r . Fortunately, we have from [Lemma 17](#) that the set of good pairs is a $\frac{1}{\text{poly}(n)}$ -fraction of the entire set of pairs, so the expectation is the same up to a $\text{poly}(n)$ factor that is absorbed by the O^* notation.

Thus, by Markov's inequality, the probability that processing pseudosolutions takes time $\Omega(b/p)$, larger than $|\mathbf{L}_1| + |\mathbf{L}_2|$ for a good residue class, is exponentially small. (If this bad event does occur, the algorithm halts and returns 'No'.)

Taking the maximum over Steps 1(b) and (c) and Step 1(e), our runtime in Case 1 is

thus

$$\begin{aligned} O^* \left(\max \left(p, \frac{b}{p} \right) \right) &= O^* \left(\max \left(a, \frac{b}{a} \right) \right) \\ &= O^* \left(\frac{b}{a} \right), \end{aligned} \tag{3.121}$$

where the first equality follows from the fact that $p = \Theta(a/n)$ in Case 1 and the second equality follows from the fact that $a/n \leq b^{2/3}a^{-1/3}$ in Case 1.

We proceed to bound (3.121). Once again let

$$\alpha := \frac{|\pi'|}{n}$$

for brevity. Recall from (3.100) and (3.101) that $a(\pi') = \Theta^*(2^{|\pi'|})$ and that $b(\pi') = O^*(2^{H_{\pi}n})$, where

$$\begin{aligned} H_{\pi} &= H \left(\left\{ \frac{\pi_c}{2n} \right\}_{c \in C' \setminus \{0\}}, \frac{\pi_{h0}}{n} \right) \\ &\leq H \left(\frac{\alpha}{2(|C'| - 1)}, \dots, \frac{\alpha}{2(|C'| - 1)}, 1 - \frac{\alpha}{2} \right) \\ &= \frac{\alpha}{2} \cdot \log_2 \left(\frac{2(|C'| - 1)}{\alpha} \right) + \left(1 - \frac{\alpha}{2} \right) \cdot \log_2 \left(\frac{2}{2 - \alpha} \right) \end{aligned}$$

It follows from (3.100) and (3.101) that $b^{2/3}a^{-1/3} \leq a/n$ when $|C'| \leq 3$, and thus we are always in Case 2 when $|C'| \leq 3$. We continue the runtime analysis of Case 1 on the assumption that $|C'| \geq 4$.

By Lemma 16, we have that α satisfies

$$\alpha \leq 1 - \frac{1}{|C'|} + \frac{\varepsilon}{|C| - 1}.$$

Applying (3.100) and (3.101) yields the following bound on b/a :

$$\frac{b}{a} := 2^{H_1 n}, \text{ where } H_1 := H_1(\alpha) = \frac{\alpha}{2} \cdot \log_2 \left(\frac{2|C'| - 2}{\alpha} \right) + \frac{2 - \alpha}{2} \cdot \log_2 \left(\frac{2}{2 - \alpha} \right) - \alpha.$$

When $|C'| \geq 4$, over the range $\alpha \in (0, 1 - 1/|C'| + O(\varepsilon)]$, the maximum of H_1 , and thus of the algorithm, is

$$H_1 \left(1 - \frac{1}{|C'|} + O(\varepsilon) \right) \leq \log_2 |C'| + \frac{1}{C'} - \frac{|C'| + 1}{2|C'|} \log_2(|C'| + 1) + O(\varepsilon^2).$$

Evaluating $\log_{|C'|}(2^{H_1})$ at this value, as $\varepsilon \rightarrow 0$, gives

$$1 - \frac{|C'| + 1}{2|C'|} \log_{|C'|}(|C'| + 1) + \frac{1}{|C'|} \log_{|C'|}(2),$$

the first of the two functions over which Λ takes the maximum.

Case 2: $p_{max} = b^{2/3}a^{-1/3}$. In this case, to create \mathbf{L}_1 and \mathbf{L}_2 we sample $O^*(b/\sqrt{ap})$ HCVs independently and with replacement to create \mathbf{L}_1 and \mathbf{L}_2 .

In Case 2, the expected number of pseudosolutions after subsampling is

$$O \left(\frac{b^2}{\hat{x}p} \cdot \left(\frac{b/\sqrt{ap}}{b/p} \right)^2 \right) \leq O \left(\frac{b}{p} \cdot \frac{p}{a} \right) \cdot e^{-\Omega(n)} \leq O \left(\frac{b}{a} \right) \cdot e^{-\Omega(n)},$$

so processing pseudosolutions takes time $O(b/a)$ with very high probability via Markov's bound. (Once again, we have conditioned on a good pair (\mathbf{p}, \mathbf{r}) , but since good pairs make up a $1/\text{poly}(n)$ fraction of the sample space, the expectation is upper-bounded by a $\text{poly}(n)$ -factor multiplied by the expectation over the entire sample space.)

Taking the maximum over Steps 1(b) and (c) and Step 1(e), our runtime in Case 2 is

thus

$$O^* \left(\max \left(p, \frac{b}{\sqrt{ap}}, \frac{b}{a} \right) \right) = O^*(b^{2/3}a^{-1/3}), \quad (3.122)$$

where the upper bound follows from substituting $p = \Omega(p_{max}) = \Omega(b^{2/3}a^{-1/3})$ and the fact that $a/n > b^{2/3}a^{-1/3}$ in Case 2.

Once again, by [Lemma 16](#), we have that α satisfies

$$\alpha \leq 1 - \frac{1}{|C'|} + \frac{\varepsilon}{|C'| - 1}.$$

Applying [\(3.100\)](#) and [\(3.101\)](#) yields the following bound on $b^{2/3}a^{-1/3}$:

$$\frac{b^{2/3}}{a^{1/3}} \leq 2^{H_2 n}, \quad \text{where } H_2 := H_2(\alpha) = \frac{\alpha}{3} \cdot \log_2 \left(\frac{2|C'| - 2}{\alpha} \right) + \frac{2 - \alpha}{3} \cdot \log_2 \left(\frac{2}{2 - \alpha} \right) - \frac{\alpha}{3}.$$

Recall that that $b^{2/3}a^{-1/3} \leq a/n$ when $|C'| \leq 3$, and thus we in Case 2 only when $|C'| \leq 3$. When $|C'| \leq 3$, over the range $(0, 1 - 1/|C'| + O(\varepsilon)]$ the maximum of H_2 , and thus of the algorithm, is

$$H_2 \left(1 - \frac{1}{|C'|} + O(\varepsilon) \right) \leq \frac{2}{3} \log_2 |C'| - \frac{|C'| + 1}{3|C'|} \log_2 \left(\frac{|C'| + 1}{2} \right) + O(\varepsilon^2).$$

Evaluating $\log_{|C'|}(2^{H_2})$ at this value as $\varepsilon \rightarrow 0$ gives

$$\frac{2}{3} - \frac{|C'| + 1}{3|C'|} \log_{|C'|} \left(\frac{|C'| + 1}{2} \right),$$

the second of the two functions over which Λ takes the maximum.

Combining the runtime of the algorithm in the $|C'| \leq 3$ and $|C'| \geq 4$ cases gives the final runtime of $|C'|^{\Lambda(n)n + O(\varepsilon n)}$. □

3.3.6 Average-Case GSS on Dense Instances: Proof of [Theorem 2](#)

Proof of [Theorem 2](#). Fix $C = [\pm d]$ or $C = [-d : d]$ and τ satisfying $|\tau| = o(\hat{x}n)$. Consider $\hat{x} = |C|^{\alpha n + o(n)}$ for some $\alpha \in (0, 1)$ and fix a constant ε as specified in the statement of [Theorem 2](#). We proceed to show that [Algorithm 3.9](#) can be used to solve this GSS instance with high probability in time $|C|^{\alpha\Lambda(|C|)n + \varepsilon n}$.

As $\hat{x} = |C|^n \cdot 2^{-\Omega(n)}$, we can use the technique described in the proof of [Theorem 1](#) to reduce the problem to a new instance \vec{x}' of average-case GSS with n' elements satisfying

$$|C|^{(1-\delta)n'} \leq \hat{x} \leq |C|^{(1-\delta/2)n'} \quad (3.123)$$

for a constant $\delta > 0$ that can be made arbitrarily small: In the $C = [-d : d]$ case, we can simply set $\vec{x}' = \vec{x}_{[n']}$ and assume the remaining elements will have their coefficients set to 0. In the $C = [\pm d]$ case, we perform the shrinking operation described in the proof of [Theorem 1](#), Case 2(c), assigning +1 and -1 coefficients to the elements of $\vec{x}_{[n'+1:n]}$ and adjusting the target τ accordingly. In both cases, we create a new instance \vec{x}' , uniformly distributed over $[0 : \hat{x} - 1]^{n'}$, with a new target τ' such that $|\tau'| = o(n'\hat{x})$, and for which any solution can be easily converted to a solution for \vec{x} .

Set ε to be sufficiently small that [Theorem 5](#) holds on \vec{x}' , and let $\delta = \delta(\varepsilon) > 0$ be the constant determined by ε in [Theorem 5](#). Run the algorithm in [Theorem 5](#) on \vec{x}' for every profile π and return any solution found (or “no solution” if no solution is found for any profile π). Because there are polynomially many assignment profiles, this takes time

$$O^*(|C|^{\Lambda(|C|)n' + \varepsilon n}) = O^*(|C|^{(\alpha\Lambda(|C|)/(1-\delta) + \varepsilon/2)n + o(n)}),$$

where we use the fact that

$$n' \leq \frac{\alpha}{1 - \delta},$$

which follows from (3.123). For sufficiently small δ , this is dominated by $|C|^{\alpha\Lambda(|C|)n+\varepsilon n}$.

Taking a union bound over the chance that the algorithm in Theorem 5 fails on any assignment profile yields a success probability of $1 - e^{-\Omega(n)}$ on \vec{x}' . In the $C = [-d : d]$ case, \vec{x}' has a solution with probability $1 - e^{-\Omega(n)}$ by Theorem 3, and thus we solve \vec{x} with probability $1 - e^{-\Omega(n)}$. In the $C = [\pm d]$, $d > 1$ case, \vec{x}' has a solution with probability at least $1 - o(1)$ by Theorem 4. By Corollary 3, there is a solution with probability at least $1 - o(1)$ over the randomness of \vec{x}' in the $C = [\pm 1]$ case if $\sum_i \mathbf{x}_i$ has the same parity as τ , as the shrinking procedure preserves the parity of $\sum_i \mathbf{x}_i - \tau$. Thus we solve \vec{x} with probability $1 - o(1)$ in this case. \square

3.4 Generalized Number Balancing

The Number Balancing problem attempts to divide n numbers in the real range $[0, 1]$ into two sets in a way that minimizes the difference between the two sums. The problem can be thought of as the optimization version of GSS on $C = [\pm 1]$. We introduce the following generalized version.

Problem 7: Generalized Number Balancing (GNB)

Input. A vector $\vec{y} = (y_1, y_2, \dots, y_n) \in [0, 1]^n$, a coefficient set $C \subset \mathbb{Z}$, and a precision parameter $\delta \in (0, 1]$.

Output. A coefficient vector $\vec{c} \in C^n$ that satisfies $|\vec{c} \cdot \vec{y}| \leq \delta$, or “no solution” if no solution exists.

In the average-case version of this problem, we consider inputs sampled uniformly at random from $[0, 1]$. In [KK82], Karmarkar and Karp achieved precision

$$\delta = n^{-\Omega(\log(n))}$$

for worst-case Number Balancing in linear time. However, a solution with exponentially small precision always exists by the pigeonhole principle. Scaling an average-case GNB instance \vec{y} by δ^{-1} and truncating the result yields a vector of integers that can be interpreted as an instance \vec{x} of GSS sampled uniformly from $[0 : \delta^{-1} - 1]^n$. A solution to \vec{x} on C with target $\tau = 0$ is then a solution to \vec{y} on C with precision δn . This insight yields the following corollaries to [Theorem 3](#), [Theorem 4](#) and [Corollary 3](#).

Corollary 5 (Optimal Precision for GNB with $C = [-d : d]$). *Fix $C = [-d : d]$ and any $\varepsilon > 0$, and consider $\vec{y} \sim [0, 1]^n$. Then we have*

$$\Pr_{\vec{y}} [\exists \vec{c} \in C^n : |\vec{y} \cdot \vec{c}| < \delta n] \begin{cases} = 1 - e^{-\Omega(n)} & \text{if } \delta = \Omega^*(|C|^{-(1-\varepsilon)n}) \\ \leq \delta |C|^n & \text{if } \delta \leq |C|^{-n}. \end{cases}$$

Corollary 6 (Optimal Precision for GNB on $[\pm d]$). *Fix $C = [\pm d]$ and any $\varepsilon > 0$, and consider $\vec{y} \sim [0, 1]^n$. Then we have*

$$\Pr_{\vec{y}} [\exists \vec{c} \in C^n : |\vec{y} \cdot \vec{c}| \leq \delta n] \begin{cases} \geq 1 - o(1) & \text{if } \delta = \Omega^*(|C|^{-(1-\varepsilon)n}) \\ \leq \delta |C|^n & \text{if } \delta \leq |C|^{-n}. \end{cases}$$

Moreover, the reduction from GNB to GSS allows us to use our algorithm to solve average-case GNB on symmetric coefficient sets.

Corollary 7 (Algorithm for Average-Case GNB). *For any $\alpha \in (0, 1)$, $C = [\pm d]$ or $C = [-d : d]$, and any constant $\varepsilon > 0$, there exists an algorithm that solves average-case GNB with precision $|C|^{-\alpha n}$ in time*

$$O(|C|^{\alpha\Lambda(|C|)n+\varepsilon n}),$$

where Λ is as defined as in (3.1) and plotted in Figure 3.1. For uniform random $\vec{y} \in [0, 1]^n$, the algorithm is correct with probability at least $1 - e^{-\Omega(n)}$ for $C = [-d : d]$ and $1 - o(1)$ for $C = [\pm d]$.

Proof. We can convert $\vec{y} \sim [0, 1]^n$ into $\vec{x} \sim [0 : n|C|^{\alpha n} - 1]^n$ by scaling and then truncating the input. (Note that this preserves uniform sampling.) Our structural results then guarantee the existence of a solution to this GSS instance with probability $1 - e^{-\Omega(n)}$ or $1 - o(1)$ depending on whether $C = [-d : d]$ or $C = [\pm d]$.¹¹ If a solution exists, it corresponds to a GNB solution with precision $|C|^{-\alpha n}$ and can be recovered by Algorithm 3.9 with probability $1 - e^{-\Omega(n)}$ in time $O(|C|^{\alpha\Lambda(|C|)n+\varepsilon n})$ by Theorem 2. \square

¹¹If $C = \{\pm 1\}$ and $|\vec{x}|_1$ has odd parity, a solution that achieves target $\tau = 1$ is fine.

Chapter 4

The Complementarity of Subset Sum and Equal Subset Sum: Solving an "Either-Or" Problem

This chapter uses material from [\[Ran23\]](#), and contains the following subsections:

- **Structure vs. Randomness and Subset Sum.** Brief introduction to the *structure vs. randomness* paradigm in mathematics and what a structure vs. randomness approach to Subset Sum might look like.
- **Summary of Results.** An $O^*(2^{(0.5-\varepsilon)n})$ -time algorithm for “Either-Or Subset Sum”, the problem that, given an instance (X, t) of (Equal) Subset Sum, requires a correct solution to either the Vanilla Subset Sum or Equal Subset Sum problems.
- **Easy ESS Instances.** Generalization of fast algorithms for unbalanced Subset Sum ([Section 2.3.3](#)) to Equal Subset Sum.

- **Subset Sum Instances That Are Easy ESS Instances.** Proof that, if a Subset Sum instance admits a solution that has few distinct subset sums, the same instance has a small Equal Subset Sum solution.
- **Proof of Theorem 6: The Algorithm for EOSS.** Proof of the main theorem.

In the previous chapter, we observed the close relationship between Vanilla Subset Sum and Equal Subset Sum: the two problems can be seen as the simplest specializations of the Generalized Subset Sum problem, in which the goal is to assign coefficients to the input vector to achieve a certain target. However, despite the seeming similarity between the two problems, we were forced to use two different analyses: it appears that the absence of the 0 coefficient is structurally significant.

The current chapter expands on the relationship between the two problems by proving a surprising result: if our goal is to find a (possibly negative) solution to *at least one* of the two problems, we can do so in time $O(2^{0.461n})$ —faster than the best worst-case algorithm for either problem, and beneath the $O^*(2^{0.5n})$ Meet-in-the-Middle barrier.

4.1 Structure vs. Randomness and Subset Sum

In the context of this thesis, this chapter serves as motivation for the *structure vs. randomness* approach to Subset Sum problems. This broad paradigm, effectively popularized by Tao [Tao07, Tao08], observes that large (or infinite) classes of mathematical objects can be effectively handled by partitioning them according to the following strategy:

- Many objects are *random-like*; that is, their important properties are similar to those that hold with high probability when we sample an element from the class uniformly

at random. If a theorem or algorithmic approach holds for a “typical” element of the class, it is extended to cover all random-like elements.

- Some objects are *structured*; that is, they have properties that sharply distinguish them from the “typical” element of the class. This structure, in turn, can often be exploited: a strong structural guarantee can outline a distinctive subclass which can be handled by an appropriate theorem or algorithmic approach.
- (Optional) Some objects that are neither structured nor random-like constitute an “error term” on which an algorithm fails or to which a theorem does not apply.

Solving an algorithmic problem over a large class of instances using the structure vs. randomness approach thus involves drawing the correct distinction between random-like and structured instances, showing that the two can be distinguished by an algorithm, and designing appropriate algorithms for each case.

In [Chapter 3](#), we saw algorithms that solve Subset Sum problems exponentially faster than Meet-in-the-Middle with very high probability over the selection of a random input vector. This is equivalent to saying that there exists an algorithm which solves almost all instances of GSS in time $O^*(|C|^{(1-\varepsilon)n/2})$ for some constant $\varepsilon > 0$. Do the few instances on which our existing algorithms fail share important structural features? If so, following the logic of the structure vs. randomness approach, we might hope to break the worst-case Meet-in-the-Middle barrier by designing a specialized approach for these structured instances.

Other authors have considered variations on this theme in the context of Subset Sum. In 2015, Austrin, Kaski, Koivisto, and Nederlof showed that Subset Sum instances in which the number of subsets adding to any given integer is small admit $O^*(2^{(1/2-\varepsilon)n})$ -time solutions for some $\varepsilon > 0$ [[AKKN15](#)]. We can view this as a Subset Sum result for a class of random-like

instances: those in which a certain “anticoncentration” of sums is guaranteed. In a follow-up work, the same authors improved on their previous results and in addition showed that instances in which many subsets add to a given integer (a structured class) can be solved in time $O^*(2^{(1/2-\varepsilon)n})$ as well [AKKN16]. Unfortunately, a large class of difficult Subset Sum instances remains.

Viewed as a structure vs. randomness result for the Vanilla Subset Sum and Equal Subset Sum problems, this chapter’s results can be summarized as follows:

1. Instances of Subset Sum that are highly structured, in the sense that they have a solution that admits few distinct subset sums, correspond to s easy instances of ESS (Proposition 3).
2. Instances of Subset Sum that are random-like, in the sense that they do not produce many sets with duplicate sums over the run of a Representation Method-style algorithm for Subset Sum, are solved with high probability over the randomness internal to the algorithm (Theorem 6).

Or, more succinctly, instances of Subset Sum that are “structured enough” to be difficult to solve in time $O^*(2^{(1/2-\varepsilon)n})$ (using known approaches) have useful features that allow ESS to be solved in time $O^*(2^{(1/2-\varepsilon)n})$.

4.2 Summary of Results

Formally, we consider the following hybrid problem:

Problem 8: Either/Or Subset Sum (EOSS)

Input: A Subset Sum instance $X = \{x_1, x_2, \dots, x_n\}, t$.

Output: *Either* a solution to the Subset Sum problem on X, t , *or* a solution to the Equal Subset Sum problem on X .

Recall the definition of GSS: given input vector \vec{x} , target t and a coefficient set C , select a coefficient vector $\vec{c} \in C^n$ such that $\vec{c} \cdot \vec{x} = t$. In Subset Sum, $C = \{0, 1\}$; in Equal Subset Sum, $C = \{-1, 0, 1\}$ (and, usually, $t = 0$). Thus Either/Or Subset Sum can be thought of as the task of solving Generalized Subset Sum on either of the two smallest natural coefficient sets.

The problem also resembles a “sum-flavored” specialization of the *Pigeon* problem, which defines the complexity class PPP [Pap94]. Pigeon asks us, given a circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}^n$, to find *either* an input that maps to 0 *or* two inputs that collide. (The existence of a solution is guaranteed by the pigeonhole principle.) Consider the problem *Sum-Pigeon*, where the input is a Subset Sum instance X, t , and the goal is to find either an input that sums to $t \pmod{2^n}$ or two inputs that collide $\pmod{2^n}$. This is equivalent to solving Pigeon on the circuit $C(X) : \{0, 1\}^n \rightarrow \{0, 1\}^n$ that sums elements of X indicated by input bits and gives their sum (less t , mod 2^n) as an output. Positive solutions to EOSS then correspond to Sum-Pigeon solutions. However, Sum-Pigeon differs from EOSS in that a solution is guaranteed by the pigeonhole principle, and negative solutions (‘No’ answers) are not allowed.

The main algorithmic result of this chapter is as follows:

Theorem 6. *EOSS can be solved in time $O^*(2^{0.461n})$ with probability $1 - o(1)$ and no false positives.*

We note that this algorithm is faster than both the Meet-in-the-Middle barrier and the best worst-case algorithm for Equal Subset Sum, the $O^*(3^{0.488n})$ -time algorithm due to Mucha, Nederlof, Pawlewicz and Węgrzycki [MNPW19].

The proof of our main result adapts the Representation Method to the worst-case setting by carefully handling unfriendly instances. As such, we recommend that the reader is familiar with [Section 1.1.2](#) before continuing. The algorithm proceeds as follows:

1. First, we use a modified Meet-in-the-Middle approach to solve the Subset Sum instance if there exists a small Subset Sum solution.
2. Second, we use a modified Meet-in-the-Middle approach to solve the Equal Subset Sum instance if there exists a small ESS solution.
3. We prove that Subset Sum instances that admit at least one solution S with $\xi_{1/2}(S)$ very small also have a small Equal Subset Sum solution ([Lemma 18](#)). Thus, if the previous step fails, it follows that our instance has a solution with many distinct small subset sums.
4. Conditioning on the existence of a Subset Sum solution S with $\xi_{1/2}(S)$ relatively large, we can adapt the Representation Method to the worst case:
 - (a) The existence of many distinct small subset sums ensures that our partial candidate solutions hash “nicely” into the residue classes of a large random prime: with high probability, many pairs $(r, t - r)$ of residue classes contain at least one solution pair.
 - (b) The solution recovery step, which uses Meet-in-the-Middle, fails only if our residue classes contain many partial candidate solutions with the same sum. However, if this occurs, we have a solution to the Equal Subset Sum problem.

4.2.1 Solving EOSS Using Techniques From [AKKN16]

After the first draft of this chapter was completed, a fruitful discussion with Jesper Nederlof revealed alternative algorithms for EOSS that can be achieved by leveraging techniques from prior work.

A solution to “Decision EOSS”, the EOSS variant in which the algorithm is not required to return a solution to Subset Sum or ESS in the ‘Yes’ case, is implicit in Theorem 1.1 of [AKKN16]. Applying Theorem 1.1 with $\varepsilon = 1/6$ in a black-box manner gives an algorithm for this problem that runs in time $O^*(2^{23n/48}) = O^*(2^{0.480n})$.

More significantly, a careful modification of Algorithm 1 in [AKKN16] gives a solution to EOSS in time $O^*(2^{0.421n})$. To achieve this runtime, set $\mu = 0.421$ and select an arbitrary input subset M of size μn . Check whether M contains any duplicate subset sums and return the resulting ESS solution if so. If M contains no duplicate subset sums, run Algorithm 1. The proof of Proposition 3.4 can then be modified with the additional assumption that the bucket size of M is 1. Substituting $\gamma = 1$ and $\mu = 0.421$ results in the claimed runtime; the optimal value of μ follows from balancing the time to enumerate $\mathfrak{S}(M)$ against the dominant $2^{(0.5-0.189\mu)n}$ term in the runtime of Algorithm 1.

Whether further improvements can be achieved by combining the two approaches is an intriguing question for future research.

4.3 Easy ESS Instances

Recall our definition of the *size* of a GSS solution \vec{c} as the number of nonzero coefficients. In the case of Equal Subset Sum, in which $C = \{-1, 0, 1\}$, this corresponds to $|A| + |B|$ for two disjoint subsets of the input such that $\Sigma(A) = \Sigma(B)$. When an instance of ESS admits

a small solution, Meet-in-the-Middle can be implemented efficiently (c.f. [Lemma 5](#)):

Lemma 18 (ESS Instances with Small Solutions are Easy; c.f. [\[MNPW19\]](#) Theorem 3.3).

Let X be an instance of Equal Subset Sum. There is a deterministic algorithm that recovers a solution (A, B) with $|A| + |B| = \alpha n$, if one exists, in time

$$O^* \left(\binom{\frac{n}{2}}{\frac{\alpha n}{4}, \frac{\alpha n}{4}, \frac{(1-\alpha)n}{2}} \right) = O^* \left(2^{\frac{(H(\alpha)+\alpha)n}{2}} \right).$$

Proof. Fix a solution (A, B) satisfying $|A| + |B| = \alpha n$. The proof follows by adapting Meet-in-the-Middle to Equal Subset Sum in the natural way.

Using [Lemma 4](#), let (Y, Z) be a partition of the input such that

$$|(A \cup B) \cap Y| = \frac{\alpha n}{2}.$$

Create the lists L_1 and L_2 by enumerating all subsets of size at most

$$\frac{\alpha n + 1}{2}$$

of Y and Z , respectively, and then enumerating every possible partition of these subsets into candidates for $(A \cap Y, B \cap Y)$ and $(A \cap Z, B \cap Z)$, respectively.

At this point we observe that

$$|L_1|, |L_2| = O^* \left(2^{\frac{(H(\alpha)+\alpha)n}{2}} \right)$$

by [\(2.6\)](#) and that our choice of the partition (Y, Z) guarantees that $L_1 \times L_2$ contains the solution. Finally, we run Meet-in-the-Middle ([Algorithm 1.1](#)) on the two lists to recover

(A, B) in in time

$$O(|L_1| + |L_2|) = O^* \left(2^{\frac{(H(\alpha)+\alpha)n}{2}} \right). \quad \square$$

4.4 Subset Sum Instances Which Are Easy ESS Instances

The concept of *additive structure* in sets is typically measured by counting the number of element pairs or subsets that add to the same sum (Section 1.1.3). We exploit the insight that the Subset Sum instances on which average-case approaches fail have solutions with duplicate subset sums, which in turn implies the existence of small Equal Subset Sum solutions.

Proposition 3. *Let (X, t) be a Subset Sum instance with a solution¹ $S \subseteq X$ such that*

$$|\mathfrak{S}_{1/2}(S)| \leq 2^{|S|-\beta n},$$

for some β, S satisfying $\beta < |S|/n$ and $\beta, |S|/n \in [0.12, 0.5]$.²

Then X has an Equal Subset Sum solution A, B satisfying

$$|A| + |B| \leq H^{-1}(1 - 2\beta)n + o(n).$$

Proof. Let X and S be as in the proposition statement. S is unusual: it has $\Theta^*(2^{|S|})$ subsets of size $|S|/2$ (“halfsets”), but few distinct “halfsums”: $|\mathfrak{S}_{1/2}(S)| \leq 2^{|S|-\beta n}$. This implies the existence of some number s such that $\Omega^*(2^{\beta n})$ halfsets add to s .

¹Strictly speaking, this holds for any subset $S \subseteq X$.

²This range is approximate. We will not need to choose $\beta, |S|/n$ at the limits of the feasible set to optimize the runtime of our algorithm.

Define the set collection

$$\mathcal{S}_{1/2}^s := \left\{ I \subset S \mid |I| = \frac{|S|}{2}, \Sigma(I) = s \right\};$$

that is, the set of halfsets of S that add to s . Associate each member $I \in \mathcal{S}_{1/2}^s$ with a Hamming ball of radius

$$\frac{H^{-1}(1 - 2\beta)n}{2}$$

in $\{0, 1\}^{|S|}$, centered at the binary indicator vector of I . Every such ball contains

$$\left(\frac{|S|}{\frac{H^{-1}(1-2\beta)n}{2}} \right) = \Theta^* \left(2^{H \left(\frac{H^{-1}(1-2\beta)n}{2|S|} \right) |S|} \right)$$

elements by Stirling's Approximation (2.3).

We would like to show that the total volume of the collection of $\Theta^*(2^{\beta n})$ balls is greater than $|\{0, 1\}^{|S|}| = 2^{|S|}$, which implies that two balls must overlap. By comparing the exponents of the two quantities, we can see that this is true when the equation

$$H \left(\frac{H^{-1}(1 - 2\beta)n}{2|S|} \right) |S| + \beta n > |S| \tag{4.1}$$

holds, or, equivalently, for $|S|$ and β such that the function

$$g(x, b) := H \left(\frac{H^{-1}(1 - 2b)}{2x} \right) x + b - x \text{ satisfies} \tag{4.2}$$

$$g \left(\frac{|S|}{n}, \beta \right) \geq 0. \tag{4.3}$$

Experimental evaluation shows that (4.3) holds when $\beta n < |S|$ and $\beta n, |S| \in [0.12, 0.5] \cdot n$. Figure [Figure 4.1](#) plots $g(|S|/n, \beta)$ as a function of $|S|/n$ for the value $\beta \approx 0.139$ we choose

for [Algorithm 4.2](#) below.

The equation is tight for any $\beta < 0.5$ when $|S| = 0.5n$, and thus adding a small sublinear factor to our radius ensures it holds in all the cases claimed.

Two overlapping Hamming balls correspond to two subsets of S that add to the same value s and differ in at most

$$2 \cdot \frac{H^{-1}(1 - 2\beta)n}{2} + o(n) = H^{-1}(1 - 2\beta)n + o(n)$$

elements. Constructing an Equal Subset Sum solution A, B from their difference gives the result. \square

4.5 Proof of [Theorem 6](#): The Algorithm for EOSS

Proof. Fix constants

$$\gamma := 0.15 \text{ and} \tag{4.4}$$

$$\beta \approx 0.139 \tag{4.5}$$

so that β satisfies the equation

$$H^{-1}(1 - 2\beta) = 0.2. \tag{4.6}$$

(β is chosen to optimize the runtime of [Algorithm 4.2](#), but γ is not tightly constrained: any value within a small constant range works equally well.) These constants define the cutoffs for two deterministic preprocessing steps designed to solve easy instances of EOSS:

1. First, we use Meet-in-the-Middle to deterministically solve the Subset Sum instance if

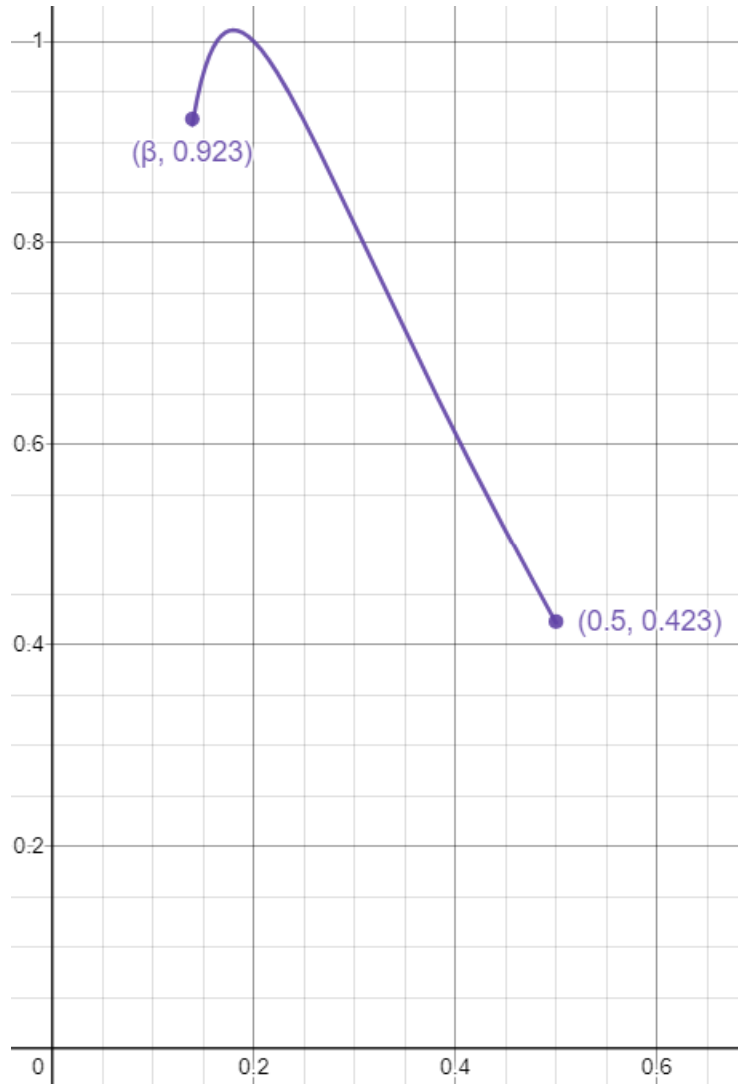
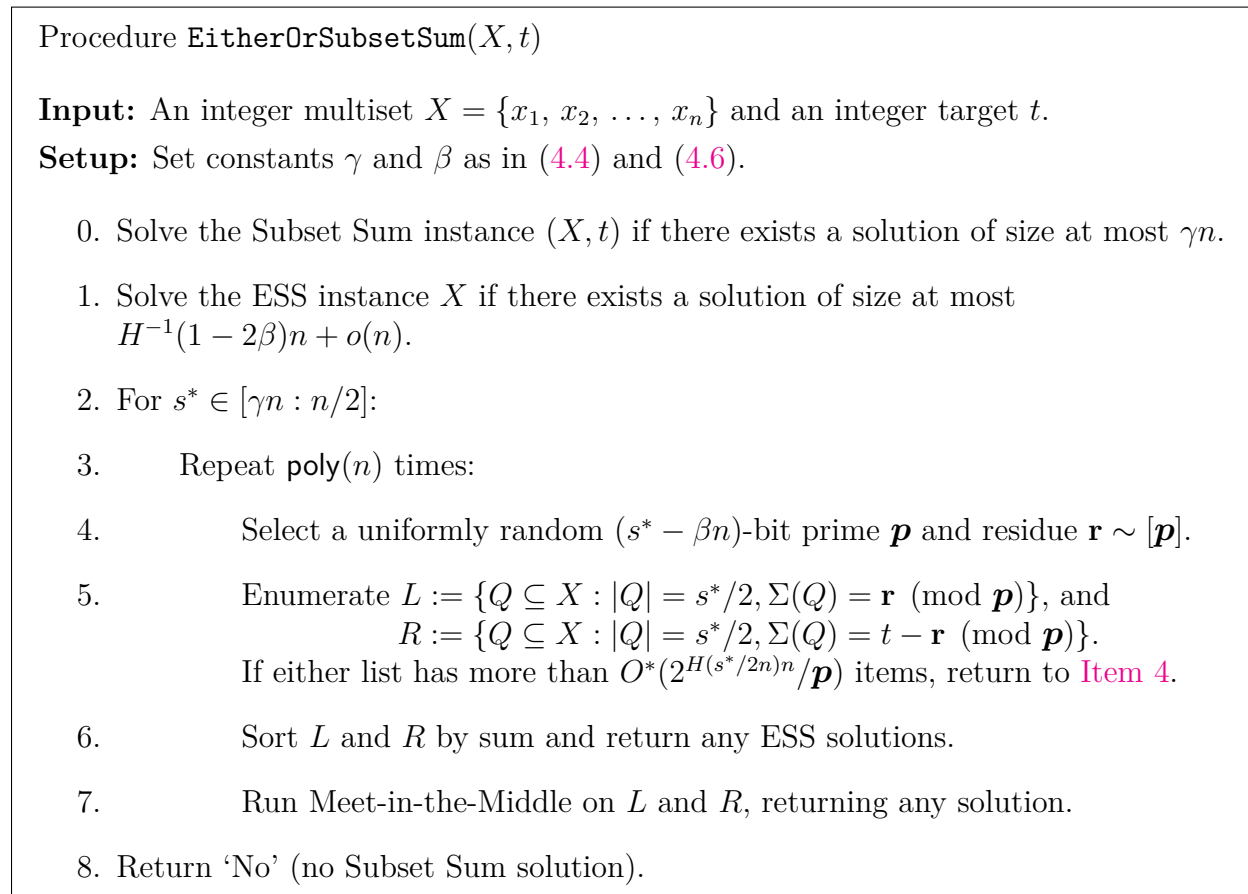


Figure 4.1: Plot of $g(|S|/n, \beta)$ as a function of $|S|/n$, for fixed constant β satisfying $H^{-1}(1 - 2\beta) = 0.2$, in the range $[\beta, 0.5]$.

Figure 4.2: The `EitherOrSubsetSum` algorithm.

there exists a solution of size at most γn . This takes time

$$O^* \left(\binom{n/2}{\gamma n/2} \right) = O^* \left(2^{H(\gamma)n/2} \right) \quad (4.7)$$

by **Lemma 5**.

2. Second, we use **Lemma 18** to deterministically solve the Equal Subset Sum instance in time

$$O^* \left(2^{\frac{1-2\beta+H^{-1}(1-2\beta)}{2}n} \right) \quad (4.8)$$

if it has an Equal Subset Sum solution of size at most $H^{-1}(1 - 2\beta)n + o(n) = 0.2 + o(n)$.

If our preprocessing steps do not find a solution, it follows from [Lemma 5](#) and [Lemma 18](#) that any Subset Sum solution $S \subseteq X$ satisfies

$$|S| \geq \gamma n \tag{4.9}$$

and

$$|\S_{1/2}(S)| \geq 2^{|S| - \beta n}, \tag{4.10}$$

as we have ruled out Subset Sum solutions with $|S| < \gamma n$ and Equal Subset Sum solutions with $|A| + |B| \leq H^{-1}(1 - 2\beta)n + o(n)$.

Fix a solution $|S|$ satisfying [\(4.9\)](#) and [\(4.10\)](#), and guess the size of a solution s^* without loss of generality (as described in [Section 2.3.1](#)) at the cost of an $O(n)$ factor in the final runtime.

Next, we apply the Representation Method. Our strategy will be to look for pairs of sets in the set collection

$$\mathcal{X}_{|S|/2n} := \left\{ I \subset X \mid |I| = \frac{|S|}{2} \right\} \tag{4.11}$$

that can be patched together to make a Subset Sum solution of size $|S|$.

Thus our search space has size

$$|\mathcal{X}_{|S|/2n}| = \Theta^* \left(2^{H(|S|/2n)n} \right)$$

by Stirling's Approximation [\(2.3\)](#). This is much larger than $2^{n/2}$ when $|S| \approx 0.5n$. However, [\(4.10\)](#) will allow us to resolve this difficulty via a Representation Method-style approach.

Let \mathbf{p} be a prime number selected uniformly at random from all $(|S| - \beta n)$ -bit primes.

Our first task is to find a way to efficiently list the elements of $\mathcal{X}_{|S|/2n}$ that add to a given residue $r \pmod{\mathbf{p}}$. We accomplish this by building a dynamic programming table T of dimension $n \times \mathbf{p}$. In the cell $T[k, r]$, for each $j \in [|S|/2]$, we store the number of subsets of $\{x_1, x_2, \dots, x_k\}$ of cardinality exactly j that sum to $r \pmod{\mathbf{p}}$. Filling out the table takes time $O^*(\mathbf{p})$, and by backtracking, we can efficiently enumerate all subsets of X of size at most $|S|/2$ that sum to $r \pmod{\mathbf{p}}$, for any residue r . Sample a residue class $\mathbf{r} \in [\mathbf{p}]$ uniformly at random and use T to enumerate all quartersets that sum to \mathbf{r} and $t - \mathbf{r}$. Call these two lists L and R , respectively.

Suppose our input (X, t) admits a Subset Sum solution S with $|S| \in [\gamma n, n/2]$, and consider the branch of the algorithm that correctly guesses $|S|$. In this case, we call \mathbf{r} a ‘good’ residue class if

1. there exists at least one *disjoint* pair of quartersets Q_1, Q_2 such that $Q_1 \cup Q_2 = S$, $\Sigma(Q_1) = \mathbf{r} \pmod{\mathbf{p}}$, and $\Sigma(Q_2) = t - \mathbf{r} \pmod{\mathbf{p}}$, and
2. the lists L and R have length $O^*(2^{H(|S|/2n)n}/\mathbf{p})$.

Claim 11. *If a solution S with $|S| \in [\gamma n, n/2]$ exists, condition (1) is true for an $\Omega(1/n^2)$ -fraction of residue classes with probability $1 - o(1)$ over \mathbf{p} .³*

Proof: Observe that by Lemma 8 that the expected number of pairs $x, y \in \mathfrak{S}_{1/2}(S) \times \mathfrak{S}_{1/2}(S)$ with $|x - y| = 0 \pmod{p}$ is

$$|\mathfrak{S}_{1/2}(S)|^2 \frac{\log(2^{O(n)})}{\mathbf{p}} = O\left(\frac{n|\mathfrak{S}_{1/2}(S)|^2}{\mathbf{p}}\right)$$

by linearity of expectation, where we use Section 2.3.4 to assume without loss of generality

³C.f. Lemma 9, which proves a similar statement for a set Y satisfying $|Y| \log_2(\text{diam}(Y)) \leq n^k p$.

that inputs have size $2^{O(n)}$. By Markov's inequality, we have that the number of collisions is

$$O\left(\frac{n^2|\xi_{1/2}(S)|^2}{\mathbf{p}}\right) \quad (4.12)$$

with probability $1 - 1/n$ over the choice of \mathbf{p} .

This in turn implies that $|\xi_{1/2}(S) \pmod{p}| = \Omega(\mathbf{p}/n^2)$. To see this, consider for contradiction the case in which the statement is false and count the minimum number of collisions which occur. Because

$$|\xi_{1/2}(S)| \geq \Omega(p)$$

by (4.10), even under the distribution of $\xi_{1/2}(S)$ over residue classes that minimizes collisions, the number of collisions would still be $\omega(n^2|\xi_{1/2}(S)|^2/\mathbf{p})$, violating (4.12). Thus (1) holds for an $\Omega(1/n^2)$ -fraction of residue classes with probability $1 - o(1)$ over the choice of \mathbf{p} . This completes the proof of the claim. \blacksquare

Conditioning on \mathbf{p} such that (1) holds for an $\Omega(1/n^2)$ -fraction of residue classes, and on choosing \mathbf{r} satisfying (1), Condition (2) occurs with probability at least $1 - 1/\text{poly}(n)$ over \mathbf{r} for an arbitrarily small polynomial by Markov's inequality. By repeating our choice of \mathbf{p} and \mathbf{r} a total of $\text{poly}(n)$ times, we can raise the chance of satisfying both (1) and (2), and thus choosing a good residue class, to $1 - e^{-\Omega(n)}$.

Next, we sort the elements of L and R by their sum, in anticipation of running Meet-in-the-Middle. If either list contains two sets with the same sum, we can halt and return these sets as a solution to Equal Subset Sum. If neither L nor R contains duplicate sums, we run Meet-in-the-Middle on the two lists in time

$$O^*(|L| + |R|) = O^*(2^{H(|S|/2n)n-\mathbf{p}}),$$

recovering a solution successfully if we have chosen a good residue class. Thus if the algorithm completes the loop over all $\text{poly}(n)$ choices of s^* , \mathbf{p} , and \mathbf{r} without finding a Subset Sum or ESS solution, it can return ‘No’, indicating that there is no Subset Sum solution, and be correct with high probability.

The total runtime of the algorithm is the sum of the three main subprocedures, with (2) and (3) carefully balanced by the choice of β :

1.

$$O^*(2^{H(\gamma)n/2}) = O^*(2^{0.305n}),$$

the time taken to check for small Subset Sum solutions (Step 0),

2.

$$O^*(2^{\frac{1-2\beta+H^{-1}(1-2\beta)}{2}n}) = O^*(2^{0.461n}),$$

the time taken to check for small Equal Subset Sum solutions (Step 1), and

3.

$$\max_{|S| \in [\gamma n, n/2]} O^*(2^{H(|S|/2n) - \frac{|S|}{n} + \beta}n) = O^*(2^{0.4609n}),$$

the runtime of Meet-in-the-Middle (Steps 2-8). □

Chapter 5

Beyond the Meet-in-the-Middle

Barrier: Log Shaving for Subset Sum

This section introduces results from [CJRS23]. Although what follows has been refurbished for inclusion in this thesis, the arguments presented represent the collaborative efforts of the four original authors: Xi Chen, Yaonan Jin, Rocco A. Servedio, and I.

This chapter contains the following subsections:

- **Circuit RAM and Word RAM.** An introduction to two common memory models in the context of log shaving for Subset Sum.
- **Summary of Results.** Three algorithms that shave factors of $n^{0.5}/\log(n)$, $n^{0.01}$, and $n^{0.502}$ off the $O(2^{n/2})$ running time of Meet-in-the-Middle
- **$\Omega(n^{0.5}/\log n)$ -Factor Speedup via Bit Packing.** Adaptation of the Bit Packing approach of [BDP05] to the Subset Sum problem.

- **$\Omega(n^{0.01})$ -Factor Speedup via Orthogonal Vectors and the Representation Method.** A small polynomial speed-up achieved by memoizing very small instances of the Orthogonal Vectors problem.
- **Subset Sum in Time $O(2^{n/2} \cdot n^{-0.5023})$.** A careful combination of the previous two approaches.

Given the longstanding difficulty of achieving a $2^{(1/2-\varepsilon)n}$ -time worst-case algorithm for Subset Sum, it is natural to consider the relaxed goal of achieving *some* nontrivial speedup over Meet-in-the-Middle. This chapter overviews the first results of this type. We give three different randomized algorithms for worst-case Subset Sum, each of which runs in time $O(2^{n/2} \cdot n^{-\gamma})$ for a specific constant $\gamma > 0$ in standard word RAM and circuit RAM models. Our fastest algorithm, which combines techniques from our other two algorithms, runs in time $O(2^{n/2} \cdot n^{-0.5023})$.

The improvements we achieve over the $O(2^{n/2})$ runtime of Meet-in-the-Middle are analogous to “log shaving” improvements on the runtimes of polynomial-time algorithms. As noted in the introduction, this progress comes with an important caveat: unlike other barriers to which log shaving techniques have been applied, there is no strong evidence that an $O(2^{(1/2-\varepsilon)n})$ -time algorithm for Subset Sum is impossible. In our view, the modest goals of this chapter are justified by the fifty years of effort that have passed without improvements on the runtime of our best algorithm for worst-case Subset Sum. Moreover, we remain optimistic that the techniques demonstrated below will eventually lead to future, perhaps superpolynomial, improvements.

5.1 Circuit RAM and Word RAM

Unlike in previous chapters of this thesis, in this chapter we are concerned with subexponential improvements in runtime. Specifically, the factors by which we improve on Meet-in-the-Middle are comparable to the bit length of the input (in both cases, $\text{poly}(n)$). For this reason, we pause to explain our memory model in more detail.

Given that Subset Sum inputs are upper-bounded by the size of the target t without loss of generality, it is natural to adopt a memory model with word length $\ell = \Theta(\log t)$ so that each input integer can be stored in a single word. This is the framework used in the work of Pisinger [Pis03], who studied dynamic programming approaches for Subset Sum in the word RAM model. We also note that this memory model is analogous to the standard RAM model that is commonly used for problems such as 3SUM (see e.g. [BDP05]), where it is assumed that each input value is at most $\text{poly}(n)$ and hence fits into a single $O(\log n)$ -bit machine word.

This model lets us consider arbitrary input instances of Subset Sum with no constraints on the size of the input integers. If $t = 2^{o(n)}$, standard dynamic programming algorithms [Bel66] solve the problem in time $O(nt) = 2^{o(n)}$, which supersedes our $\text{poly}(n)$ -factor improvements over Meet-in-the-Middle; hence throughout the chapter we assume $t = 2^{\Omega(n)}$. It is arguably most natural to think about instances in which $t = 2^{\Theta(n)}$, in which case $\ell = \Theta(n)$, and we encourage the first-time reader to imagine $\ell = \Theta(n)$ for easy digestion. However, we can use the bit length reduction described in Section 5.3.1 to reduce much larger inputs to $O(n)$ bits without loss of generality. If inputs are extremely large (for example, 2^{2^n}), a word length of $\Theta(\log t)$ dwarfs the time it takes to solve the problem.

We consider runtime in two standard variants of the RAM model. The first is *circuit*

RAM; in this model, any operation that maps a constant number of words to a single word and has a $\text{poly}(\ell)$ -size circuit with unbounded fan-in gates can be performed in time proportional to the depth of the circuit. Consequently, in the circuit RAM model, AC^0 operations on a constant number of words can be performed in constant time, and multiplying, performing modular division, etc., on two ℓ -bit words can be performed in time $O(\log \ell)$. The second is *word RAM*, in which the usual arithmetic operations, including multiplication, are assumed to take unit time, but arbitrary AC^0 operations are not atomic operations on words. We present each of our algorithms for the stronger circuit RAM model,¹ and explain adaptations that give corresponding word RAM algorithms.

5.2 Summary of Results

Our first result improves on the $O(2^{n/2})$ -time implementation of Meet-in-the-Middle described in [Algorithm 1.1](#) using a “bit packing” approach.

Theorem 7. *There exists an algorithm that solves Vanilla Subset Sum with constant probability in time $O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$ in the circuit RAM model and $\tilde{O}(2^{n/2} \cdot n^{-1/2})$ in the word RAM model.*

The algorithm, sketched in simplified form in [Algorithm 5.1](#), works by adapting the *bit packing* trick, a technique developed by Baran, Demaine, and Pătraşcu [[BDP05](#)] for the 3SUM problem, to Meet-in-the-Middle. The idea is to compress the two lists of partial subset sums used in Meet-in-the-Middle by packing hashes of multiple values into a machine word, while preserving enough information to make it possible to run a Meet-in-the-Middle-style

¹Note that any algorithm in the word RAM model can be simulated in the circuit RAM model with no more than an $O(\log \ell)$ -factor slowdown.

algorithm on the lists of hashed and packed values. This results in a runtime savings over performing Meet-in-the-Middle on the original lists because processing every pair of words, each of which contains multiple hashed values, takes constant expected time in the circuit RAM model and can be memoized to take constant time in the word RAM model.

However, even if the initial hashing and packing step takes time linear in the size of the lists, this is still too slow if the lists of partial subset sums have length $\Omega(2^{n/2})$. To get around this, we set aside a small set D before constructing the list of partial subset sums. We then solve many Subset Sum instances, each with target $t - d$ for a certain $d \in \mathfrak{S}(D)$, re-using the packed lists each time. Setting $|D| = \log(n)$ results in the best possible runtime.

Theorem 8. *There exists an algorithm that solves Vanilla Subset Sum with constant probability in time $O(2^{n/2} \cdot n^{-\gamma})$ for a constant $\gamma > 0.01$ in the circuit RAM and word RAM models.*

Our second algorithm achieves a speedup of $\Omega(n^\gamma)$ over Meet-in-the-Middle for a constant $\gamma > 0.01$. While this is a smaller speedup than that achieved by the bit packing approach, our second algorithm is distinguished by not using “bit tricks”. Instead, it combines Meet-in-the-Middle with the Representation Method to reduce the Subset Sum problem to many small instances of Orthogonal Vectors.

The algorithm begins by partitioning the input $X = A \sqcup B \sqcup C$ into two large subsets A and B and one small subset C of size approximately $|C| \approx \log n$. We then use sets consisting of $|A|/2$ elements of A and $|C|/4$ elements of C (respectively, $|B|/2$ elements of B and $|C|/4$ elements of C) as our partial candidate solutions and apply the Representation Method (see [Section 1.1.2](#)), producing two lists of partial candidate solutions that fall into complementary residue classes modulo a large random prime \mathbf{p} .

The key to adapting the Representation Method to the worst case, for this algorithm, is

the fact that partial candidate solutions can overlap only in elements of C . C is intentionally very small, which lets us ensure that it has the properties required for the Representation Method via a deterministic preprocessing step. This results in a lower bound on the probability that our two lists of partial candidate solutions contain a solution pair, if a solution exists.

The recovery phase applies Meet-in-the-Middle, with the only snag occurring when many partial candidate solutions have the same sum. In this case, we need to search for a matching pair of partial candidate solutions that do not use any element of C multiple times. We can do this by observing that the problem of searching for a disjoint pair in a collection of sets of size $O(\log(n))$ is equivalent to solving a small instance of Orthogonal Vectors. By considering Orthogonal Vectors on inputs of size at most $\varepsilon \log n$ as a Boolean function, we can solve the problem by memoization: we simply tabulate all possible outputs in time $O(2^{\varepsilon n})$.

Theorem 9. *There exists an algorithm that solves Vanilla Subset Sum with constant probability in time $O(2^{n/2} \cdot n^{-(1/2+\gamma)})$ for a constant $\gamma > 0.0023$ in the circuit RAM and word RAM models.*

Our fastest algorithm uses a delicate combination of the techniques from the previous two results to obtain a runtime of $O(2^{n/2} \cdot n^{-0.5023})$. While the runtime improvement over the previous theorem is not large, this algorithm demonstrates that by leveraging insights specific to the Subset Sum problem, we can achieve time savings beyond what is possible with more generic log shaving techniques.

This algorithm is complex due to a difficulty that arises in combining the previous two algorithms. Our bit packing approach saves time by removing a subset $D \subseteq X$ from the input, then running a Meet-in-the-Middle variant on the resulting subinstance multiple times.

In contrast, our second approach runs a Meet-in-the-Middle variant on multiple subinstances indexed by their residue class modulo a random prime \mathbf{p} . This presents a problem: to get the time savings from bit packing, we would like to reuse Subset Sum subinstances multiple times, but to get the time savings from the Representation Method we need to build separate subinstances with respect to each residue class $(\bmod \mathbf{p})$ that contains elements of $\xi(D)$. To solve this problem, we construct D in a way that ensures the elements of $\xi(D)$ fall into few residue classes.

5.3 $\Omega(n^{0.5}/\log n)$ -Factor Speedup via Bit Packing

For our first algorithm we need the concept of a *pseudolinear hash function*. Such a function maps a large range onto a much smaller domain while preserving linearity: collisions are unlikely, and simple additive relationships continue to hold.

Given an integer $m \leq \ell$, we write $\mathbf{h}_m : [2^\ell] \rightarrow [2^m]$ to denote the random hash function defined as

$$\mathbf{h}_m(y) := (\mathbf{u} \cdot y \pmod{2^\ell}) \gg \ell - m.$$

Here the input y is an ℓ -bit integer, \mathbf{u} is selected uniformly at random from all *odd* ℓ -bit integers, and \gg denotes a bit shift to the right, i.e., dividing $\mathbf{u} \cdot y \pmod{2^\ell}$ by $2^{\ell-m}$ and then truncating the result so that only the higher-order m bits remain.

This hash function $\mathbf{h}_m(y)$ can be evaluated in time asymptotically equivalent to multiplication: $O(\log \ell) = O(\log n)$ in the circuit RAM model and $O(1)$ in the word RAM model. Further, \mathbf{h}_m has the following useful properties.

Lemma 19 (Pseudolinear Hashing [DHKP97, BDP05]). *The following hold for \mathbf{h}_m :*

1. **Pseudolinearity.** For any two ℓ -bit integers y, z and any outcome of \mathbf{h}_m ,

$$\mathbf{h}_m(y) + \mathbf{h}_m(z) \in \mathbf{h}_m(y + z) - \{0, 1\} \pmod{2^m}.$$

2. **Pseudouniversality.** For any two ℓ -bit integers y, z with $y \neq z$,

$$\Pr[\mathbf{h}_m(y) = \mathbf{h}_m(z)] = O(2^{-m}).$$

We are now prepared to present the first algorithm of the chapter.

Theorem 10. *Algorithm 5.1 is a zero-error randomized algorithm for the Subset Sum problem with expected runtime*

$$O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell) = O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$$

*in the circuit RAM model.*²

Algorithm 5.1 works by packing ℓ/m hashed values into a single word via our pseudolinear hash function \mathbf{h}_m , for $m = 3 \log \ell$, while preserving enough information to run Algorithm 1.1 on the lists of hashed and packed values. This allows us to compare two length- (ℓ/m) sublists of $\vec{\xi}(A)$ and $\vec{\xi}(B)$ in constant expected time in the circuit RAM model, since each hashed and packed sublist fits into a constant number of words, instead of in time $O(\ell/m)$. So far, this is essentially the approach taken by [BDP05] in their bit-packing algorithm for 3SUM. However, in our context the $O(\ell/m)$ speedup described above is offset by the following issue:

²We can easily convert Theorem 10 into a one-sided Monte Carlo algorithm such as that described in Theorem 7 by halting and returning ‘No’ if runtime exceeds $\alpha \cdot 2^{n/2} \cdot n^{-1/2} \cdot \log n$ for a large enough constant $\alpha > 0$.

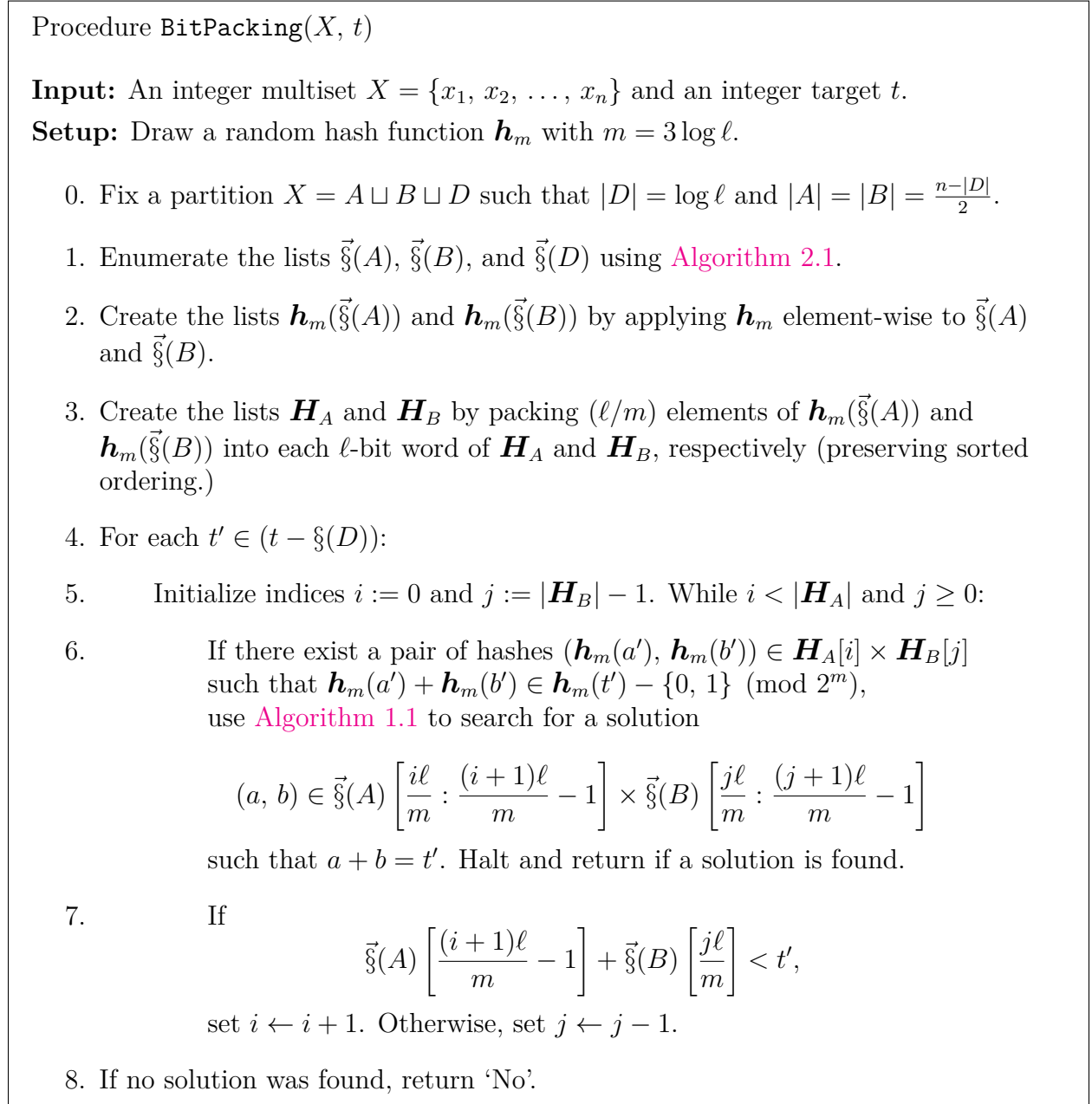


Figure 5.1: The Bit-Packing algorithm.

if we follow the original [Algorithm 1.1](#) setup and take $|A| = |B| = \frac{n}{2}$, the lists $\vec{\xi}(A)$ and $\vec{\xi}(B)$ may have length $\Omega(2^{n/2})$, increasing the runtime.

To deal with this, we set aside a small set $D \subseteq X$ of $|D| = \log \ell$ many input elements and solve the remaining subinstance $X \setminus D$ for each shifted target $t' \in (t - \xi(D))$. Removing the elements in D shortens the lists L_A and L_B , which are now formed from the elements in $X \setminus D$, and allows us to enumerate and pack them quickly. Balancing the overhead of solving each of these subinstances against the savings described earlier, we get the claimed speedup $\Omega(\ell^{1/2}/\log \ell) \geq \Omega(n^{1/2}/\log n)$.

Steps 6 and 7 implement a subprocedure that we refer to, informally, as *Block Meet-in-the-Middle*, which increments two pointers in a manner closely analogous to [Algorithm 1.1](#). The key difference is that each pointer indicates a *word* containing multiple packed hashes. If Step 6 discovers two hashes that appear to correspond to a solution, we return to the original (unpacked) list and use [Algorithm 1.1](#) to recover a solution.

Proof of Correctness for [Algorithm 5.1](#). The algorithm outputs “Yes” if and only if it recovers a triple

$$(a, b, t') \in \vec{\xi}(A) \times \vec{\xi}(B) \times (t - \xi(D))$$

satisfying $a + b = t'$, so it never returns a false positive.

It remains to show that for any $t' \in (t - \xi(D))$, we are guaranteed to find a shifted solution $(a, b) \in \vec{\xi}(A) \times \vec{\xi}(B)$ such that $a + b = t'$, if one exists. Without loss of generality, we consider two sublists

$$\vec{\xi}(A) \left[\frac{i\ell}{m} : \frac{(i+1)\ell}{m} - 1 \right] \quad \text{and} \quad \vec{\xi}(B) \left[\frac{j\ell}{m} : \frac{(j+1)\ell}{m} - 1 \right]$$

that contain such an (a, b) and that correspond to two packed words $\mathbf{H}_A[i]$ and $\mathbf{H}_B[j]$ for some indices i and j .

By the same principle as the original Meet-in-the-Middle Algorithm ([Algorithm 1.1](#)), the

condition in Line 7 ensures that the algorithm will not step past either the packed word $\mathbf{H}_A[i]$ or the packed word $\mathbf{H}_B[j]$ before reaching the other, so the algorithm must compare these two packed words at some point.

By Lemma 19, we have

$$\mathbf{h}_m(a) + \mathbf{h}_m(b) \in \mathbf{h}_m(t') - \{0, 1\} \pmod{2^m},$$

satisfying the condition in Line 6. Thus we are guaranteed to find the shifted solution (a, b) by running Algorithm 1.1 to check all pairs in

$$\vec{\S}(A) \left[\frac{i\ell}{m} : \frac{(i+1)\ell}{m} - 1 \right] \times \vec{\S}(B) \left[\frac{j\ell}{m} : \frac{(j+1)\ell}{m} - 1 \right]. \quad \square$$

Proof of Runtime for Algorithm 5.1.

- Line 1 takes time $O(2^{|A|} + 2^{|B|} + 2^{|D|}) = O(2^{n/2} \cdot \ell^{-1/2} + \ell) = O(2^{n/2} \cdot \ell^{-1/2})$ by Lemma 2.
- Line 2 takes time $(|\vec{\S}(A)| + |\vec{\S}(B)|) \cdot O(\log \ell) = O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell)$, where $O(\log \ell)$ bounds the time for each evaluation of the hash function \mathbf{h}_m .
- Line 4 (the outer loop) is performed for at most $|\S(D)| \leq 2^{|D|} = \ell$ iterations.
- Line 5 (the inner loop) is performed for at most

$$(|\vec{\S}(A)| + |\vec{\S}(B)|) \cdot \frac{1}{\ell/m} = O(2^{n/2} \cdot \ell^{-3/2} \cdot \log \ell)$$

iterations, since each iteration either increments i or decrements j .

- Line 6: Checking whether $\mathbf{H}_A[i]$ and $\mathbf{H}_B[j]$ contain a pair of hashes corresponding to a potential solution requires a single AC^0 operation on three words, which takes constant time in the circuit RAM model.

If the two packed words contain at least one potential solution, we index back into the unpacked lists $\vec{\xi}(A)$ and $\vec{\xi}(B)$ and run [Algorithm 1.1](#) on the two length- (ℓ/m) sublists. This takes time $O(\ell/m) = O(\ell/\log \ell)$.

The amount of time taken searching through $\vec{\xi}(A)$ and $\vec{\xi}(B)$ depends on the number of potential solutions (essentially, hash collisions), which in turn is bounded by the guarantees of our pseudolinear hash function. Note that the sequence of pairs of words $(\mathbf{H}_A[i], \mathbf{H}_B[j])$ we compare is completely determined by L_A and L_B and is thus independent of the random hash function \mathbf{h}_m . Thus by [Lemma 19](#), each of the $(\ell/m)^2$ pairs of hashed values in $(\mathbf{H}_A[i], \mathbf{H}_B[j])$ incurs a collision with probability $O(2^{-m})$. By a union bound, the expected time taken for Line 6 because of hash collisions is at most

$$(\ell/m)^2 \cdot O(2^{-m}) \cdot O(\ell/m) = O(1/\log^3(\ell)) = o(1)$$

since $m = 3 \log \ell$ and $\ell = \Omega(n)$.

Consequently, [Algorithm 5.1](#) has expected runtime

$$\begin{aligned} \text{TIME}(n, \ell) &= O(2^{n/2} \cdot \ell^{-1/2} + \ell) + O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell) && \text{Lines 1 and 2} \\ &+ 1 \cdot (O(1) + O(\ell/\log \ell) + O(1)) && \text{Lines 4-7; solution} \\ &+ \ell \cdot O(2^{n/2} \cdot \ell^{-3/2} \cdot \log \ell) \cdot (O(1) + o_n(1) + O(1)) && \text{Lines 4-7; collisions} \\ &= O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell). && \square \end{aligned}$$

5.3.1 Adapting Algorithm 5.1 to Word RAM

Adapting Algorithm 5.1 to the word RAM model requires replacing the constant-time AC^0 circuit RAM operation used to detect hash collisions in Line 6. We can accomplish this using a memoization strategy, which we will re-use with slight modifications for our subsequent algorithms (Algorithm 5.2 and Algorithm 5.5).

Corollary 8. *Algorithm 5.1 is a zero-error randomized algorithm for the Subset Sum problem with expected runtime*

$$O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell) = O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$$

in the word RAM model.

Proof. The main change required to achieve expected runtime $O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$ in the word RAM model is to run Algorithm 5.1 as if the word length were $\ell' := 0.1n$. Following a similar conversion from [BDP05]:

Run Algorithm 5.1 as if the word length were $\ell' = 0.1n$, which results in the modified parameters $m' = 3 \log \ell'$, $|D'| = \log \ell'$, and $|A'| = |B'| = (n - |D'|)/2$. We make two more modifications to the algorithm:

1. Line 2 packs $q' := \min\{\ell', \ell\}/m' = \Theta(\frac{n}{\log n})$ many m' -bit hashes into each word of $\mathbf{H}_{A'}$, $\mathbf{H}_{B'}$, so every (ℓ'/m') hashes are stored in $\ell'/(m'q') = \Theta_n(1)$ words rather than a single word.
2. Before Line 6, create a table that memoizes the result of every comparison of two ℓ' -bit

strings in time $(2^{\ell'})^2 \cdot \text{poly}(\ell') = O(2^{0.21n})$.³ This table can then be accessed via a $2^{\ell'}/(m'q') = \Theta_n(1)$ -word index in constant time. Line 6 replaces the constant-time AC^0 circuit RAM operation on two ℓ' -bit strings with a constant-time lookup into this table.

Compared with running Algorithm 5.1 itself for $\ell' = 0.1n$, the only difference of this variant is that $\mathbf{H}_{A'}$ and $\mathbf{H}_{B'}$ are stored in $\Theta_n(1)$ times as many words, so correctness follows from the same argument. The expected time taken for the collisions in each execution of Line 6 is $(q')^2 \cdot O(2^{-m'}) \cdot O(q') = o(1)$. Thus, the overall runtime is as claimed:

$$\underbrace{O(2^{|A'|} + 2^{|B'|} + 2^{|D'|} + 2^{0.21n})}_{\text{Lines 1 and 2}} + \underbrace{O(q') + O\left(2^{|D'|} \frac{2^{|A'|} + 2^{|B'|}}{q'}\right)}_{\text{Lines 4-7; potential solutions}} = O\left(2^{n/2} \cdot \frac{\log n}{n^{1/2}}\right). \quad \square$$

5.4 $\Omega(n^{0.01})$ -Factor Speedup via Orthogonal Vectors and the Representation Method

Our second algorithm, Algorithm 5.2, achieves a speedup of $\Omega(\ell^\gamma) \geq \Omega(n^\gamma)$ over Algorithm 1.1 for a constant $\gamma > 0.01$.

While this is a smaller speedup than that achieved by Algorithm 5.1, Algorithm 5.2 does not use “bit tricks”. Instead, Algorithm 5.2 combines Algorithm 1.1 with the Representation Method (Section 1.1.2) so as to reduce the Subset Sum problem to many small instances of the Orthogonal Vectors (OV) problem: namely, instances with $O(\ell/\log \ell)$ many binary

³Note that we can reduce this runtime to $2^{\varepsilon n}$ for any $\varepsilon > 0$, as necessary. We will exploit the expanded power of memoization in the exponential-time setting in our next algorithm.

vectors of dimension $\Theta(\log \ell) = \Theta(\log n)$. Such instances of OV can be solved quickly through a single AC^0 word operation in the circuit RAM model or through constantly many operations in the word RAM model after an initial memoization step. This results in our speedup.

The high-level idea behind our algorithm is to partition the input $X = A \sqcup B \sqcup C$ into two large subsets A and B and one small subset C of size $|C| = \Theta(\log \ell)$, and to run [Algorithm 1.1](#) on two lists formed from subsets of $(A \cup C)$ and $(B \cup C)$. Roughly speaking, the three lists are created by modifying the Representation Method ([Section 1.1.2](#)) to ensure that the lists are not too long. In order to eliminate the false positives due to overlapping subsets of C , we exploit a fast implementation of a function that computes a batch of small instances of Orthogonal Vectors.⁴ Before giving more details we provide some helpful notation:

5.4.1 Definitions and Notation

For the sake of brevity in the sections that follow, we introduce some new notation. Write

$$\mathcal{Q}(C) := \left\{ T \subseteq C \mid |T| \leq \frac{|C|}{4} \right\},$$

the collection of sets satisfying

$$\mathfrak{S}_{1/4}(C) = \{\Sigma(Q) \mid Q \in \mathcal{Q}(C)\}.$$

Here, we use the symbol \mathcal{Q} to indicate the fact that the elements of $\mathcal{Q}(C)$ contain one quarter of the elements of C . Accordingly, we refer to the elements of $\mathcal{Q}(C)$ as “quartersets” below.

⁴To see the relevance of the Orthogonal Vectors problem in this context, note that a solution to an instance of OV on k -dimensional Boolean vectors corresponds to two disjoint subsets of the set $[k]$.

For technical reasons, we will sometimes need to add a small “margin of error” to $\mathcal{Q}(C)$. Given a small constant $\varepsilon > 0$, we write

$$\mathcal{Q}^{+\varepsilon}(C) := \left\{ T \subseteq C \mid |T| \leq (1 + \varepsilon) \frac{|C|}{4} \right\}.$$

Formally, the Orthogonal Vectors problem is as follows:

Problem 9: Orthogonal Vectors (OV)

In: Two sets $V = \{\vec{v}_1, \dots, \vec{v}_n\}$, $W = \{\vec{w}_1, \dots, \vec{w}_n\}$ of vectors in $\{0, 1\}^d$.

Out: A pair $(v_i, w_j) \in V \times W$ such that $v_i \cdot w_j = 0$, or ‘No’ if no solution exists.

The following 2ℓ -bit boolean function is equivalent to Orthogonal Vectors on two sets containing $\ell/|C|$ boolean vectors of $|C|$ bits each. Define

$$\text{OV} : (\{0, 1\}^{|C|})^{\ell/|C|} \times (\{0, 1\}^{|C|})^{\ell/|C|} \rightarrow \{0, 1\}$$

as

$$\text{OV}\left((x^1, \dots, x^{\ell/|C|}), (y^1, \dots, y^{\ell/|C|})\right) := \begin{cases} 1 & \text{if } \exists(i, j) : x^i \cdot y^j = 0, \\ 0 & \text{otherwise.} \end{cases} \quad (5.1)$$

Because OV is an AC^0 operation on two ℓ -bit words, it takes constant time in the circuit RAM model.

5.4.2 Proof of Theorem 11

We begin with an overview of Algorithm 5.2. For the partial solutions used in the Representation Method, we use the list $\mathcal{Q}^{+\varepsilon_2}(C)$ of all “near-quartersets”, where ε_2 is a small constant

defined below (refer to (5.2) and (5.3)). If a certain Subset Sum solution $S \subseteq X$ satisfies $|S \cap C| \leq (1 + \varepsilon_2) \frac{|C|}{2}$, the set $S \cap C$ can be written as the union of many different pairs of disjoint quartersets $Q_1, Q_2 \in \mathcal{Q}^{+\varepsilon_2}(C)$.

We then perform the filtering step of the Representation Method: given a random prime \mathbf{p} we construct two lists of “sum-subset pairs” by selecting sublists of

$$\vec{\S}(A) \times \mathcal{Q}^{+\varepsilon_2}(C) \text{ and } \vec{\S}(B) \times \mathcal{Q}^{+\varepsilon_2}(C)$$

corresponding to two residue classes that add to $t \pmod{\mathbf{p}}$. This reduces the size of the search space while ensuring that we retain some pair of near-quartersets $Q_1, Q_2 \in \mathcal{Q}^{+\varepsilon_2}(C)$ satisfying $Q_1 \cup Q_2 = S \cap C$ and $Q_1 \cap Q_2 = \emptyset$.

Finally, we use a modified Meet-in-the-Middle procedure to search for a solution, i.e., two sum-subset pairs

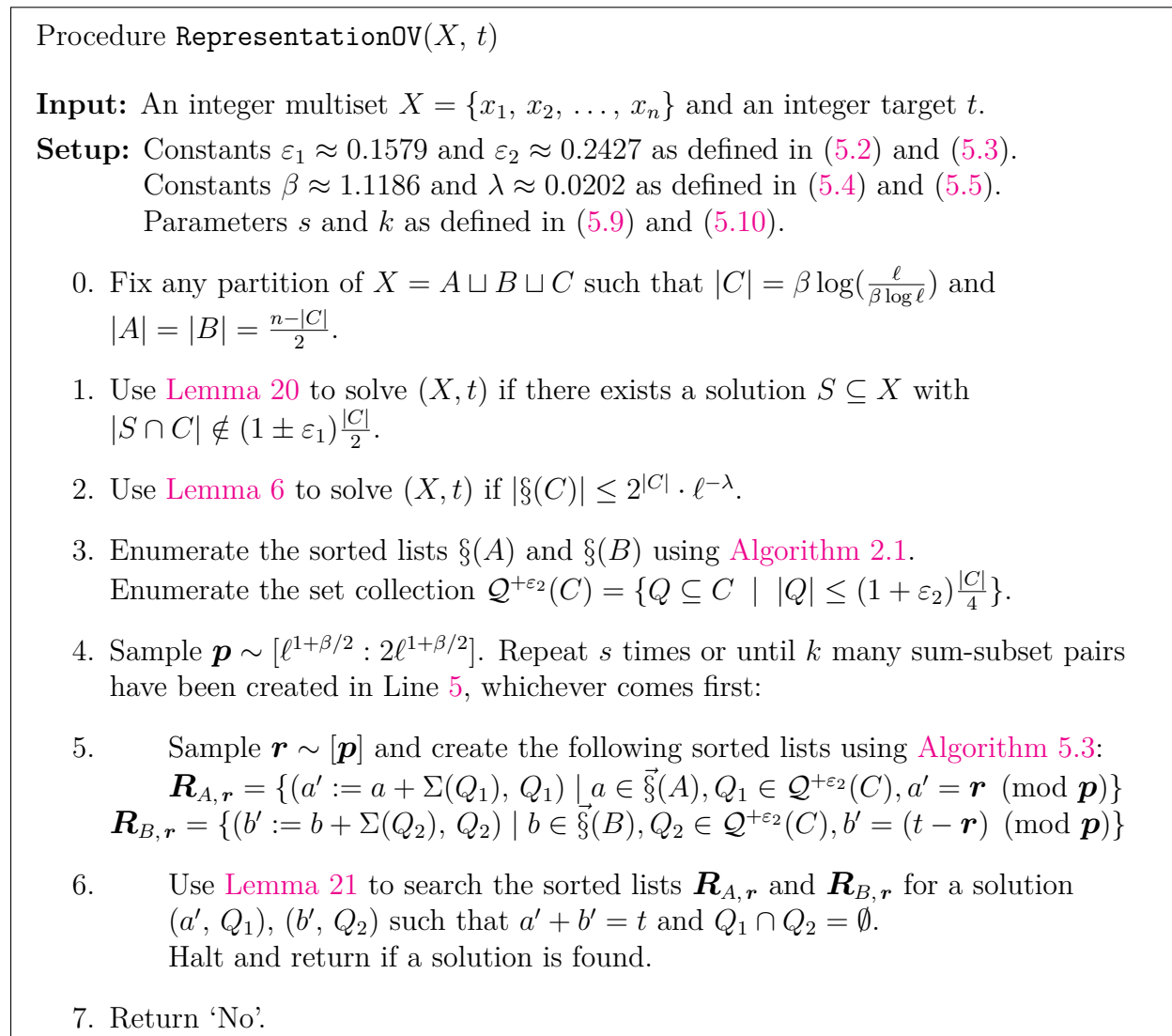
$$\begin{aligned} (a, Q_1) &\in (\vec{\S}(A) \times \mathcal{Q}^{+\varepsilon_2}(C)), \\ (b, Q_2) &\in (\vec{\S}(B) \times \mathcal{Q}^{+\varepsilon_2}(C)) \end{aligned}$$

satisfying

$$\begin{aligned} a + \Sigma(Q_1) + b + \Sigma(Q_2) &= \Sigma(S \cap A) + \Sigma(S \cap B) + \Sigma(S \cap C) \\ &= \Sigma(S) = t. \end{aligned}$$

We verify that $Q_1 \cap Q_2 = \emptyset$ via the boolean function \mathcal{OV} .

We are now ready to prove correctness and runtime for [Algorithm 5.2](#).

Figure 5.2: The `Representation0V` algorithm.

Theorem 11. *Algorithm 5.2 solves the Vanilla Subset Sum problem with constant success probability in time $O(2^{n/2} \cdot \ell^{-\gamma}) \leq O(2^{n/2} \cdot n^{-\gamma})$ for some constant $\gamma > 0.01$ in the circuit RAM model.*

Proof of Correctness for Algorithm 5.2. We define the constants $\varepsilon_1 \approx 0.1579$ and $\varepsilon_2 \approx 0.2427$ to be the solutions to the following equations (which will be useful later):

$$\frac{1 - H\left(\frac{1-\varepsilon_1}{2}\right)}{(1-\varepsilon_1)/2} = 1 - H\left(\frac{1-\varepsilon_2}{2}\right), \quad (5.2)$$

$$1 + \varepsilon_1 - 3H\left(\frac{1-\varepsilon_1}{2}\right) = -2H\left(\frac{1+\varepsilon_2}{4}\right). \quad (5.3)$$

ε_1 will serve to mark the cut-off point where we solve the instance by alternative methods if $|S \cap C|$ is far from $|C|/2$. ε_2 defines our margin of error for the set of near-quartersets $\mathcal{Q}^{+\varepsilon_2}$.

We also define the constants

$$\beta := \frac{1}{H((1+\varepsilon_2)/4)} \approx 1.1186 \text{ and} \quad (5.4)$$

$$\lambda := (1 - 10^{-5}) \cdot \frac{1-\varepsilon_1}{2} \cdot \beta \cdot \left(1 - H\left(\frac{1-\varepsilon_2}{2}\right)\right) \approx 0.0202. \quad (5.5)$$

β determines the size of the set C , and λ will be used to balance the runtime of the algorithm steps. Ultimately, we will save a factor of $n^\gamma = n^{\lambda/2}$ over Meet-in-the-Middle.

Lines 1 and 2 preprocess the instance (X, t) , solving it deterministically via Lemma 6 or Lemma 20 unless both of the following conditions hold:

Condition 1. (X, t) is either a “Yes” instance such that

$$|S \cap C| \in \left[(1-\varepsilon_1) \frac{|C|}{2} : \frac{|C|}{2} \right]$$

for each solution $S \subseteq X$, or has no solutions.

Condition 2. $|\xi(C)| > 2^{|C|} \cdot \ell^{-\lambda}$. Note that this implies

$$|\xi(T)| > 2^{|T|} \cdot \ell^{-\lambda} \text{ for each } T \subseteq C.$$

Lines 3-7 accept only if in Line 6 we find a solution satisfying

$$a' + b' = (a + \Sigma(Q_1)) + (b + \Sigma(Q_2)) = t$$

for $a \in \vec{\xi}(A)$, $b \in \vec{\xi}(B)$ and $Q_1, Q_2 \in \mathcal{Q}^{+\varepsilon_2}(C)$ with $Q_1 \cap Q_2 = \emptyset$.

It remains to show that Lines 4-6 accept a “Yes” instance (X, t) with constant probability if (X, t) satisfies Conditions 1 and 2. Consider a solution $S \subseteq X$ and let W denote the set of all distinct sums of “ ε_2 -balanced” subsets of $S \cap C$:

$$W := \left\{ \Sigma(Q) \mid Q \subseteq (S \cap C) \text{ and } |Q| \in (1 \pm \varepsilon_2) \frac{|S \cap C|}{2} \right\}. \quad (5.6)$$

We say that a residue $i \in [\mathbf{p}]$ is *good* if it satisfies

$$i - \Sigma(S \cap A) \in (W \bmod \mathbf{p}),$$

namely, if there exists a subset $Q_1 \subseteq (S \cap C)$ of size $|Q_1| \in (1 \pm \varepsilon_2) \frac{|S \cap C|}{2}$ such that

$$\Sigma(S \cap A) + \Sigma(Q_1) = i \pmod{\mathbf{p}}.$$

On sampling a good residue $\mathbf{r} = i$ in Line 5, both Q_1 and $Q_2 := (S \cap C) \setminus Q_1$ are of size

at most $(1 + \varepsilon_2) \frac{|S \cap C|}{2} \leq (1 + \varepsilon_2) \frac{|C|}{4}$, so they are included in the collection $\mathcal{Q}^{+\varepsilon_2}(C)$ and, respectively, in the lists $\mathbf{R}_{A,\mathbf{r}}$ and $\mathbf{R}_{B,\mathbf{r}}$ created in Line 5. If this event occurs the algorithm is guaranteed to recover the solution $S = (S \cap A) \cup (S \cap B) \cup (Q_1 \cup Q_2)$ by Lemma 21, proved below.

Hence it suffices to (i) lower bound the probability that at least one of the s samples $\mathbf{r} \sim [\mathbf{p}]$ is good, and (ii) upper bound the probability that these samples generate a total of k or more sum-subset pairs over all samples of \mathbf{r} .

(i). To show (i), we begin by proving bounds on $|W|$:

Claim 12. *We claim that*

$$|W| \leq 2^{|S \cap C|} \leq \ell^{\beta/2} \text{ and} \quad (5.7)$$

$$|W| = \Omega(2^{|S \cap C|} \cdot \ell^{-\lambda}) = \tilde{\Omega}(\ell^{(1-\varepsilon_1)\beta/2-\lambda}). \quad (5.8)$$

Proof: The upper bound follows from applying the definition of W (5.6), Condition 1, and the fact that $|C| := \beta \log(\frac{\ell}{\beta \log \ell})$ in Algorithm 5.2:

$$\begin{aligned} |W| &\leq 2^{|S \cap C|} \\ &\leq 2^{|C|/2} \\ &\leq 2^{\frac{\beta}{2} \log(\frac{\ell}{\beta \log \ell})} \\ &\leq \ell^{\beta/2} \cdot o_\ell(1). \end{aligned}$$

The lower bound follows from a combination of two observations. First, the set $(S \cap C)$ has at least $2^{|S \cap C|} \cdot \ell^{-\lambda}$ many distinct subset sums, by Condition 2. Second, to bound

$\S(S \cap C) \setminus W$, we observe that the number of subsets $Q \subseteq (S \cap C)$ of size $|Q| \notin (1 \pm \varepsilon_2) \frac{|S \cap C|}{2}$ is at most

$$2 \cdot 2^{H\left(\frac{1-\varepsilon_2}{2}\right) \cdot |S \cap C|} = o(2^{|S \cap C|} \cdot \ell^{-\lambda}),$$

by Stirling's approximation (2.3) and the technical condition

$$\frac{|S \cap C|}{\log \ell} \cdot \left(1 - H\left(\frac{1-\varepsilon_2}{2}\right)\right) \geq (1 - o_n(1)) \cdot \frac{1-\varepsilon_1}{2} \cdot \beta \cdot \left(1 - H\left(\frac{1-\varepsilon_2}{2}\right)\right) > \lambda,$$

which is true for our choice of constants; see (5.2), (5.3), (5.4), and (5.5). Finally, the fact that $2^{|S \cap C|} = \tilde{\Omega}(\ell^{1-\varepsilon_1} \cdot \beta/2)$ follows from Condition 1 and the fact that $|C| := \beta \log\left(\frac{\ell}{\beta \log \ell}\right)$ in

Algorithm 5.2. ■

By (5.7) and the fact that the elements of W are bounded by $t \leq 2^\ell$ without loss of generality, we have that

$$\begin{aligned} |W| \log_2(\text{diam}(W)) &\leq \ell^{\beta/2} \log_2(2^\ell) \\ &\leq \ell^{1+\beta/2}. \end{aligned}$$

Because $\mathbf{p} \sim [\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$, we can apply the Second Prime Distribution Lemma (Lemma 10) to conclude that there are $|W \bmod \mathbf{p}| = \Omega(|W|) = \tilde{\Omega}(\ell^{(1-\varepsilon_1) \cdot \beta/2 - \lambda})$ many good residues with constant probability over the choice of \mathbf{p} .

Conditioned on this event, taking

$$s := s(\ell) = \tilde{\Theta}(\ell^{1+\lambda+\varepsilon_1 \cdot \beta/2}) \geq \tilde{\Omega}\left(\frac{\mathbf{p}}{|(W \bmod \mathbf{p})|}\right) \quad (5.9)$$

many samples $\mathbf{r} \sim [\mathbf{p}]$ yields at least one good residue with constant probability.

(ii). All the sorted lists of sum-subset pairs taken together (that is, $\{\mathbf{R}_{A,i}\}_{i \in [p]}$, $\{\mathbf{R}_{B,i}\}_{i \in [p]}$) have a total of

$$(|\vec{\S}(A)| + |\vec{\S}(B)|) \cdot |\mathcal{Q}^{+\varepsilon_2}(C)| \leq 2 \cdot 2^{(n-|C|)/2} \cdot 2^{|C|/\beta} = O(2^{n/2} \cdot \ell^{1-\beta/2})$$

sum-subset pairs, by construction (Line 3), the choices of $|A|$, $|B|$, $|C|$, and Stirling's approximation (2.3). For any prime modulus $p = \Theta(\ell^{1+\beta/2})$ in the range $[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$, taking s samples $\mathbf{r} \sim [p]$ generates a total of

$$O(2^{n/2} \cdot \ell^{1-\beta/2}) \cdot \frac{s}{p} = \tilde{O}(2^{n/2} \cdot \ell^{-((1-\varepsilon_1/2) \cdot \beta - (1+\lambda))})$$

sum-subset pairs in expectation. By setting a large enough cutoff

$$k := \tilde{\Theta}\left(2^{n/2} \cdot \ell^{-((1-\varepsilon_1/2) \cdot \beta - (1+\lambda))}\right), \quad (5.10)$$

we reduce the probability that k or more sum-subset pairs are generated to an arbitrarily small constant.

Subtracting the chance that we generate k or more sum-subset pairs from the probability that we achieve a good residue implies that our algorithm succeeds with constant probability.

□

Proof of Runtime for Algorithm 5.2.

- Line 1 takes time $\tilde{O}(2^{n/2} \cdot \ell^{-(1-H((1-\varepsilon_1)/2)) \cdot \beta/2})$ by Lemma 20.
- Line 2 takes time $\tilde{O}(2^{n/2} \cdot \ell^{-\lambda/2})$, by Lemma 6.

- Line 3 takes time

$$O(2^{|A|} + 2^{|B|} + 2^{|C|}) \cdot O(\log \ell) = \tilde{O}(2^{n/2} \cdot \ell^{-\beta/2}),$$

by Lemma 2, the choices of $|A|$, $|B|$, and $|C|$, and the fact that the modulo operation takes time $O(\log \ell)$ time in the circuit RAM model.

- Lines 4-6 take time

$$\tilde{O}(k) = \tilde{O}(2^{n/2} \cdot \ell^{-((1-\varepsilon_1/2)\cdot\beta-(1+\lambda))}),$$

because a single iteration (Lemma 21) takes time $\tilde{O}(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$, and, by construction, we create a total of at most

$$\sum_r (|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|) \leq k$$

many sum-subset pairs.

The runtime of Line 3 is dominated by that of Line 1, so the bottleneck occurs in Line 1, Line 2, or Lines 4-6. For the choices of constants given in the algorithm, we achieve a speedup of $\Omega(\ell^\gamma)$ for any constant $\gamma \in (0, \gamma_*)$, where

$$\begin{aligned} \gamma_* &:= \min \left\{ \frac{\lambda}{2}, \left(1 - H\left(\frac{1-\varepsilon_1}{2}\right)\right) \cdot \frac{\beta}{2}, \left(1 - \frac{\varepsilon_1}{2}\right) \cdot \beta - (1 + \lambda) \right\} \\ &= \frac{\lambda}{2} \approx 0.0101. \end{aligned} \quad \square$$

5.4.3 Auxiliary Lemmas

A useful preprocessing lemma shows that if the algorithm chooses a small subset with respect to which some solution S is “unbalanced”, we can exploit this fact to solve Vanilla Subset Sum quickly. This is a straightforward recombination of ideas from [Lemmas 5](#) and [6](#) tailored to our specific needs.

Lemma 20 (Solving Subset Sum with an Unbalanced Subset, c.f. [Lemma 6](#)). *Fix a Subset Sum instance (X, t) , and let $Y \subseteq X$ be a set of cardinality*

$$|Y| = \beta \log \left(\frac{\ell}{\beta \log \ell} \right),$$

for β as in [\(5.4\)](#), known to the algorithm, such that some solution $S \subseteq X$ satisfies

$$|S \cap Y| \notin (1 \pm \varepsilon) \frac{|Y|}{2}$$

for a constant $\varepsilon > 0$. Then S can be found deterministically in time $\tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$, where the constant

$$\delta := \left(1 - H \left(\frac{1 - \varepsilon}{2} \right) \right) \cdot \beta.$$

Proof. Fix a Subset Sum Instance (X, t) , and let Y and S be a subset of X and a solution, respectively, as in the lemma statement.

Assume without loss of generality that the solution S satisfies $|S \cap Y| \leq \frac{|Y|}{2}$ ([Section 2.3.3](#)).

This implies

$$|S \cap Y| \leq (1 - \varepsilon) \frac{|Y|}{2}.$$

Then the sorted list $\vec{\xi}(Y')$ of

$$Y' := \left\{ \Sigma(T) \mid T \subseteq Y \text{ and } |T| \leq (1 - \varepsilon) \frac{|Y|}{2} \right\}$$

can be enumerated in time $O(2^{|Y|}) = O(\ell^\beta) = \text{poly}(n)$ by [Algorithm 2.1](#) (restricting the algorithm to subsets of Y of size at most $(1 - \varepsilon) \frac{|Y|}{2}$).

Fix an arbitrary set A of cardinality

$$\frac{n + (\delta/\beta)|Y|}{2} = \frac{n + (1 - H(\frac{1-\varepsilon}{2}))|Y|}{2}$$

satisfying

$$Y \subseteq A \subseteq X.$$

In the regime $\ell = \text{poly}(n)$, we can use $\vec{\xi}(Y')$ and [Lemma 3](#) to create the sorted list $\vec{\xi}(A')$ of

$$A' := \left\{ \Sigma(T) \mid T \subseteq A \text{ and } |T \cap Y| < (1 - \varepsilon) \frac{|Y|}{2} \right\}$$

in time

$$\begin{aligned} \tilde{O}(|\vec{\xi}(A')|) &\leq \tilde{O}(2^{|A \setminus Y|} \cdot |\vec{\xi}(Y')|) \\ &\leq \tilde{O}(2^{|A \setminus Y|} \cdot 2^{H(\frac{1-\varepsilon}{2}) \cdot |Y|}) \\ &\leq \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2}), \end{aligned}$$

using Stirling's approximation [\(2.3\)](#) and plugging in $|Y|$ and $|A|$. We can then use [Lemma 2](#) to enumerate the sorted list $L_{X \setminus A}$ in time $O(2^{|X \setminus A|}) = \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$.

Provided $|S \cap Y| < (1 - \varepsilon) \frac{|Y|}{2}$, running [Algorithm 1.1](#) on $\vec{\xi}(A')$ and $\vec{\xi}(X \setminus A)$ solves (X, t)

in time

$$O(|\vec{\mathfrak{S}}(A)| + |\vec{\mathfrak{S}}(X \setminus A)|) = \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2}).$$

The overall runtime is $\tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$. □

Subroutine **ResidueCoupleList**($\{L_{A,i}\}_{i \in [p]}$, $\mathcal{Q}^{+\varepsilon}(C)$, r)

Input: A collection of $p = \text{poly}(\ell)$ sorted sublists $\vec{\mathfrak{S}}(A) = \bigcup_{i \in [p]} L_{A,i}$, for some subset $A \subseteq X$, indexed by residue class modulo p ; also a collection $\mathcal{Q}^{+\varepsilon}(C)$ of near-quartersets.

Output: A sorted sum-subset list $R_{A,r}$ with elements in the sum-subset pair format $(a', \mathcal{Q}_{a'})$.

5(a) For each $Q \in \mathcal{Q}^{+\varepsilon}(C)$, create the sum-subset sublist $R_Q := f_Q(L_{A,j(Q)})$ by applying f_Q , the element-to-couple operation $a \mapsto (a' := a + \Sigma(Q), Q)$, to the particular input sublist of index $j(Q) := (r - \Sigma(Q)) \pmod{p}$.

5(b) Let $R_{A,r}$ be the list obtained by merging $\{R_Q\}_{Q \in \mathcal{Q}^{+\varepsilon}(C)}$, sorted by sums $a' = a + \Sigma(Q)$.

For each distinct sum a' , compress all couples $(a', Q_1), (a', Q_2), \dots$ with the same first element a' into a single data object $(a', \mathcal{Q}_{a'} := \{Q_1, Q_2, \dots\})$.

Figure 5.3: The **Residue-Couple-List** subroutine.

Lemma 21 (Implementation of [Algorithm 5.2](#) Lines 5 and 6).

1. Line 5 of [Algorithm 5.2](#) creates the sorted lists $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ in time $\tilde{O}(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$.

2. Line 6 finds a solution pair $(a', Q_1) \in \mathbf{R}_{A,r}$, $(b', Q_2) \in \mathbf{R}_{B,r}$, if one exists, in time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$.

Proof. Line 5 creates $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ using the subroutine **ResidueCoupleList** ([Algorithm 5.3](#)), which we proceed to analyse. First, we claim that [Algorithm 5.3](#) takes time $\tilde{O}(|\mathbf{R}_A|)$ for $\ell = \text{poly}(n)$:

- Line 5(a) takes time

$$O\left(\sum_{Q \in \mathcal{Q}^{+\varepsilon_2}(C)} |\mathbf{R}_Q|\right) = O(|\mathbf{R}_{A,r}|),$$

since all shifts $\Sigma(Q)$ for $Q \in \mathcal{Q}^{+\varepsilon_2}(C)$ can be precomputed and memoized when the collection $\mathcal{Q}^{+\varepsilon_2}(C)$ is created in Line 3.

- Line 5(b) enumerates $\mathbf{R}_{A,r}$ using the collection of sorted sublists $\{\mathbf{R}_Q\}_{Q \in \mathcal{Q}^{+\varepsilon_2}(C)}$, which contains

$$|\mathcal{Q}^{+\varepsilon_2}(C)| \leq 2^{|C|/\beta} = \ell/(\beta \log \ell) = \text{poly}(n)$$

sorted sublists. If we do so using a merge sort approach as in Section 2.3.2, this takes time $O(|\mathbf{R}_{A,r}| \cdot \log n)$.

After Line 5 creates the sorted lists $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$, Line 6 uses Meet-in-the-Middle (Algorithm 1.1) on the two lists, using the sum a' as the index of each sum-subset sublist $(a', \mathcal{Q}_{a'}) \in \mathbf{R}_{A,r}$. This takes time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$.

The Meet-in-the-Middle procedure ensures that we discover every pair $(a', \mathcal{Q}_{a'}), (b', \mathcal{Q}_{b'}) \in \mathbf{R}_{A,r} \times \mathbf{R}_{B,r}$ such that $a' + b' = t$. Such a pair yields a solution if and only if it contains two disjoint near-quartersets $Q_1 \in \mathcal{Q}_{a'}, Q_2 \in \mathcal{Q}_{b'}$ with $Q_1 \cap Q_2 = \emptyset$, which we can check in constant time by one call of the boolean function \mathcal{OV} . (Because each near-quarterset $Q \in \mathcal{Q}^{+\varepsilon_2}(C)$ is stored in $|C| < \beta \log \ell$ bits, each collection $\mathcal{Q}_{a'}, \mathcal{Q}_{b'} \subseteq \mathcal{Q}^{+\varepsilon_2}(C)$ can be stored a single word $|C| \cdot |\mathcal{Q}^{+\varepsilon_2}(C)| \leq \ell$. Thus one call of \mathcal{OV} suffices to check a given pair $(a', \mathcal{Q}_{a'}), (b', \mathcal{Q}_{b'})$.) Overall, Line 6 takes time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$. \square

5.4.4 Adapting Algorithm 5.2 to Word RAM

Analogously to Section 5.3.1, adapting Algorithm 5.2 to word RAM requires adapting the operation OV , which we can accomplish via memoization.

Theorem 12. *Algorithm 5.2 solves the Vanilla Subset Sum problem with constant success probability in time $O(2^{n/2} \cdot \ell^{-\gamma}) \leq O(2^{n/2} \cdot n^{-\gamma})$ for some constant $\gamma > 0.01$ in the word RAM model.*

Proof. Similar to the strategy used in Section 5.3, our word RAM variant of Algorithm 5.2 runs as if the word length were $\ell' := 0.1n$ (using sets A' , B' , C' , etc.) We require three nontrivial modifications to the algorithm:

1. In lines 3 and 5(b), the collection $\mathcal{Q} \subseteq \mathcal{Q}^{+\varepsilon_2}(C')$ may no longer fit into a single word due to the new restriction in word size. Instead, we can store the collection as a bit string it in at most

$$\left\lceil |\mathcal{Q}^{+\varepsilon_2}(C')| \cdot \frac{|C'|}{\ell} \right\rceil \leq \left\lceil \frac{\ell'}{\ell} \right\rceil = \Theta(1)$$

separate words.

2. In Line 3, after creating the collection $\mathcal{Q}^{+\varepsilon_2}(C')$, create a lookup table OV' that memoizes the input-output result of the boolean function OV on each subcollection pair $\mathcal{Q}_{a'}, \mathcal{Q}_{b'} \subseteq \mathcal{Q}^{+\varepsilon_2}(C')$ in time

$$(2^{|\mathcal{Q}^{+\varepsilon_2}(C')|})^2 \cdot \text{poly}(|\mathcal{Q}^{+\varepsilon_2}(C')|) \leq (2^{\ell'})^2 \cdot \text{poly}(\ell') = O(2^{0.21n}).$$

This table can then be accessed using a $2^{\lceil \ell'/\ell \rceil} = \Theta(1)$ -word index in constant time.

3. Line 6 replaces the boolean function OV (namely a constant-time AC^0 circuit RAM operation) with constant-time lookup into the table OV' . \square

5.5 Subset Sum in Time $O(2^{n/2} \cdot n^{-0.5023})$

The algorithm in this section is a delicate combination of [Algorithm 5.1](#) and [Algorithm 5.2](#) that requires significant constant-balancing and other bookkeeping. The reason for persisting in the exercise is not the small polynomial time savings over [Algorithm 5.1](#), but the conceptual significance of the fact that [Algorithm 5.1](#) and [Algorithm 5.2](#) are complementary: [Theorem 13](#) demonstrates that problem-specific features of Subset Sum can be exploited to obtain an additional time savings beyond what can be achieved by applying generic bit-packing tricks to Meet-in-the-Middle. Because the bookkeeping is intricate, the first-time reader is encouraged to first trace the application of the ideas from [Section 5.3](#) and [Section 5.4](#) and return later to the details which arise when combining them.

To gain a high-level understanding of the issues that occur in the attempt to combine [Algorithm 5.1](#) and [Algorithm 5.2](#), consider the way in which each algorithm saves time over Meet-in-the-Middle:

- [Algorithm 5.1](#) saves time by removing a subset $D \subseteq X$ from the input, then running a Meet-in-the-Middle variant on the resulting subinstance multiple times.
- [Algorithm 5.2](#) runs a Meet-in-the-Middle variant on multiple subinstances indexed by residue classes modulo a random prime p .

This presents a problem: to get the time savings from bit packing, we would like to reuse two sorted lists of sums multiple times, but to get the time savings from the Representation

Method, we would like to build separate subinstances with respect to each residue class $(\bmod \mathbf{p})$ that contains elements of $\xi(D)$.

To solve this problem, we construct D in a way that ensures the elements of $\xi(D)$ fall into few residue classes. Specifically, we fix a prime modulus p and construct D based on the distribution of $X \pmod{p}$:

- **Case I:** the elements of X fall into few residue classes $(\bmod p)$. In this case, it is possible to choose a small set D such that the elements of D (and $\xi(D)$) fall into very few residue classes $(\bmod p)$. As a result, we need to construct only $|\xi(D) \bmod p| = \text{polylog}(n)$ distinct subproblems.
- **Case II:** the elements of X fall into many residue classes $(\bmod p)$. In this case, a carefully selected D satisfies the weaker bound $|\xi(D) \bmod p| = \tilde{O}(n^\delta)$ for a small constant $\delta > 0$, which does increase the runtime by a polynomial factor. However, the fact that the elements of X fall into many residue classes $(\bmod p)$ allows us to select a larger set C such that the subset sums in $\xi(S \cap C)$ distribute well $(\bmod p)$ for *any* solution S , which in turn offsets the increase in runtime.

Similar to the AC^0 operation OV in [Algorithm 5.2](#), the core of [Algorithm 5.5](#) is the AC^0 operation PackedOV . This new operation also takes two ℓ -bit words as input, but may solve *multiple* very small Orthogonal Vectors instances. We define

$$\text{PackedOV} : \{0, 1\}^\ell \times \{0, 1\}^\ell \rightarrow \{0, 1\}$$

as

$$\text{PackedOV}\left(\left(\mathbf{h}_m(s_1), (\vec{x}^{s_1,1}, \dots)\right), \left(\mathbf{h}_m(s_2), (\vec{x}^{s_2,2}, \dots)\right), \dots; \left(\mathbf{h}_m(r_1), (\vec{y}^{r_1,1}, \dots)\right), \dots\right)$$

$$:= \begin{cases} 1 & \text{if } \exists(i, j, k) : \vec{x}^{s_i, j} \cdot \vec{y}^{r_i, k} = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (5.11)$$

The difference between **OV** and **PackedOV** is that the bit vectors in each input word may come from any one of multiple lists, each indexed by the m -bit hash $\mathbf{h}_m(s)$ of a corresponding subset sum s . The reader may think of **PackedOV** as asking: “for any of these small **OV** instances, indexed by the sums $(s_1, r_1), (s_2, r_2)$, etc., such that $s_i + r_i = t$ for all pairs, does there exist an orthogonal pair corresponding to a pair of disjoint quartersets?”

We also define a *density constant*

$$\rho := \frac{\log n}{\log \ell}, \quad (5.12)$$

with respect to the cardinality of the input set n and the bit length ℓ of a word in memory.

Note that ρ satisfies

$$0 < \rho \leq 1 + \Theta\left(\frac{1}{\log n}\right)$$

under the assumptions that $\ell = \text{poly}(n)$ and $\ell = \Omega(n)$.⁵

In the “low density regime”, in which ρ satisfies

$$0 < \rho \leq \frac{1}{2 - H(1/4)} \approx 0.8412,$$

Algorithm 5.5 can achieve the claimed speedup by naively simulating **BitPacking** (**Algo-**

⁵Another density parameter, $n/\log t$, is used in the literature (e.g., [AKKN16]). This is equivalent to n/ℓ using our notation, as we assume $\ell = \Omega(\log t)$. In comparison, our density constant $\rho = \frac{\log n}{\log \ell}$ is more “fine-grained”.

gorithm 5.1), which achieves a time savings of

$$\tilde{\Omega}(\ell^{1/2}) = \tilde{\Omega}(n^{1/(2\rho)}) \geq \Omega(n^{0.5943}) \geq \Omega(n^{0.5023}).$$

Thus it suffices to consider the “medium-density range”, in which

$$\frac{1}{2 - H(1/4)} < \rho \leq 1 + \Theta\left(\frac{1}{\log n}\right). \quad (5.13)$$

As in Algorithm 5.2, we will employ the constants ε_1 and ε_2 , which take the same values and serve the same functions as previously. β and λ also play the same roles as their counterparts in the previous proof, but are now defined with respect to the density parameter ρ in order to optimize runtime. The parameters s and k still denote our bounds on the number of residue class samples and the total number of sum-subset pairs, respectively, although in contrast to β and λ , the fixed constants, they take different values in Case I and Case II to ensure the correctness of the algorithm in both cases. Finally, because we construct the set D differently in Case I and Case II, we will use different threshold values to deal with $|S \cap C|$ and the set of near-quartersums in each case; this results in the use of a new constant $\varepsilon'_1 := \varepsilon'_1(\rho)$ in Case I and endogenous values e_1 and e_2 set differently based on whether the algorithm enters Case I or Case II.

Procedure PackedRepresentationOV(X, t)

Input: A set X and an integer target t .

Setup: Constants $\varepsilon_1 \approx 0.1579$ and $\varepsilon_2 \approx 0.2427$ as defined in (5.2) and (5.3).

Constants λ, β , and ε'_1 , defined in (5.23) and (5.24) with respect to ρ .

Parameters s and k defined with respect to n and ℓ in the proof of correctness.

0. Sample a random prime $\mathbf{p} \sim [\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$. Construct the random sets \mathbf{C} and \mathbf{D} according to one of two procedures:

(a) **Case I:** $|X \bmod \mathbf{p}| > \ell^\rho / \log \ell$.

Set $e_1 := \varepsilon'_1$ and $e_2 := 0$.

Use [Lemma 22](#) to sample $\mathbf{C} \subseteq X$ and [Lemma 23](#) to sample $\mathbf{D} \subseteq (X \setminus \mathbf{C})$.

(b) **Case II:** $|X \bmod \mathbf{p}| \leq \ell^\rho / \log \ell$.

Set $e_1 := \varepsilon_1$ and $e_2 := \varepsilon_2$.

Set \mathbf{D} to be an arbitrary subset of X with cardinality $\log \ell$ such that every element falls into the same residue class: $|\mathbf{D} \bmod \mathbf{p}| = 1$. Sample

$$\mathbf{C} \sim \left\{ T \subseteq (X \setminus \mathbf{D}), \left| |T| = \beta \log \left(\frac{\ell}{\beta \log \ell} \right) \right. \right\}.$$

1. Let $A \sqcup B$ be an arbitrary partition of $X \setminus (\mathbf{C} \cup \mathbf{D})$ satisfying

$$|\mathbf{A}| = |\mathbf{B}| = \frac{n - |\mathbf{C}| - |\mathbf{D}|}{2}.$$

2. Use [Lemma 25](#) to solve (X, t) if there exists a solution S with

$$|S \cap \mathbf{C}| \notin (1 \pm e_1) \frac{|\mathbf{C}|}{2}.$$

3. Use [Lemma 24](#) to solve (X, t) if

$$|\S(\mathbf{C})| \leq 2^{|\mathbf{C}|} \cdot \ell^{-\lambda}.$$

4. Enumerate $\xi(\mathbf{D})$ and the collection of near-quartersets

$$\mathcal{Q}^{+e_2}(\mathbf{C}) := \left\{ Q \subseteq \mathbf{C} \mid |Q| \leq (1 + e_2) \frac{|\mathbf{C}|}{4} \right\}.$$

Enumerate the sorted lists $\vec{\xi}(\mathbf{A})$ and $\vec{\xi}(\mathbf{B})$ using [Algorithm 2.1](#).

5. For each adjusted target $t' \in (t - \xi(\mathbf{D}) \pmod{\mathbf{p}})$:
6. Repeat Line 7 s times or until k many sum-subset pairs have been created:
7. Sample a residue $\mathbf{r} \sim [\mathbf{p}]$, then create the sorted lists

$$\mathbf{R}_{\mathbf{A}, t', \mathbf{r}} := \{(a + \Sigma(Q_1), Q_1) \mid (a, Q_1) \in \vec{\xi}(\mathbf{A}) \times \mathcal{Q}^{+e_2}(\mathbf{C}), a + \Sigma(Q_1) = \mathbf{r} \pmod{p}\}$$

$$\mathbf{R}_{\mathbf{B}, t', \mathbf{r}} := \{(b + \Sigma(Q_2), Q_2) \mid$$

$$(b, Q_2) \in \vec{\xi}(\mathbf{B}) \times \mathcal{Q}^{+e_2}(\mathbf{C}), b + \Sigma(Q_2) = (t' - \mathbf{r}) \pmod{p}\}$$

8. Define

$$\mathbf{R}_{\mathbf{A}, t'} := \bigcup_{\mathbf{r}} \mathbf{R}_{\mathbf{A}, t', \mathbf{r}},$$

with each pair $(a + \Sigma(Q), Q)$ sorted by sum $a + \Sigma(Q)$; do likewise for $\mathbf{R}_{\mathbf{B}, t'}$.

Use [Lemma 26](#) to create the hashed and packed lists $\mathbf{H}_{\mathbf{A}, t'}$ and $\mathbf{H}_{\mathbf{B}, t'}$ from $\mathbf{R}_{\mathbf{A}, t'}$ and $\mathbf{R}_{\mathbf{B}, t'}$, respectively.

9. For each $t'' \in (t - \xi(\mathbf{D}))$:
10. Use [Lemma 27](#) to search the sorted lists $\mathbf{H}_{\mathbf{A}, (t'' \pmod{p})}$, $\mathbf{H}_{\mathbf{B}, (t'' \pmod{p})}$ for a solution pair (a', Q_1) , (b', Q_2) such that $a' + b' = t''$ and $Q_1 \cap Q_2 = \emptyset$.
Halt and return if a solution is found.

11. Return ‘No’.

Theorem 13. *Algorithm 5.5 solves the Vanilla Subset Sum problem with constant success probability in time $O(2^{n/2} \cdot n^{-(1/2+\gamma)})$ for some constant $\gamma > 0.0023$ in the circuit RAM model.*

Proof of Correctness. Lines 2 and 3 solve (X, t) in specific easy cases. Line 2 solves the instance if there exists a solution S that is significantly unbalanced with respect to the randomly selected set C , and Line 3 solves the instance if C has few subset sums.

Conditioning on the event that neither Line 2 nor 3 solves the instance, our instance satisfies conditions analogous to those in the proof of Theorem 11:

Condition 1’. (X, t) is either a “Yes” instance such that

$$|S \cap C| \in \left[(1 - e_1) \frac{|C|}{2} : \frac{|C|}{2} \right] \quad (5.14)$$

holds for every solution $S \subseteq X$, or has no solutions.

Condition 2’.

$$|\S(C)| > 2^{|C|} \cdot \ell^{-\lambda}, \text{ which immediately implies} \quad (5.15)$$

$$|\S(T)| > 2^{|T|} \cdot \ell^{-\lambda} \text{ for all } T \subseteq C. \quad (5.16)$$

In other words, the preprocessing in Lines 2 and 3 serves to ensure that C behaves like a random set in two specific, helpful ways.

It remains to show that if a solution exists, Lines 4-10 recover it with constant probability. Fix any outcome of the partition $X = \mathbf{A} \sqcup \mathbf{B} \sqcup \mathbf{C} \sqcup \mathbf{D} = A \sqcup B \sqcup C \sqcup D$. With respect to a fixed solution $S \subseteq X$, define the set

$$W' := \left\{ \Sigma(Q) \mid Q \subseteq (S \cap C) \text{ and } |Q| \in (1 \pm e_2) \frac{|S \cap C|}{2} \right\} \subseteq \S(S \cap C).$$

W' , like the set W in the proof of [Theorem 11](#), is almost the same as $\S(S \cap C)$, with the additional condition that we consider only the sums of subsets that are “ e_2 -balanced”, where e_2 is the small constant fixed in [Step 0](#) of [Algorithm 5.5](#).

We say that a residue $i \in [\mathbf{p}]$ is *good* if it satisfies

$$i - \Sigma(S \cap A) \in (W' \bmod \mathbf{p}),$$

in which case there exists a subset $Q_1 \subseteq (S \cap C)$ of size $|Q_1| \in (1 \pm e_2) \frac{|S \cap C|}{2}$ such that $\Sigma(S \cap A) + \Sigma(Q_1) = i \pmod{\mathbf{p}}$.

Consider the iteration

$$t_* := t - \Sigma(S \cap D) \pmod{\mathbf{p}}$$

of the loop in [Line 5](#), which is the iteration in which we may recover the solution S . If we condition on sampling a good residue \mathbf{r} in [Line 7](#), both sets Q_1 and $Q_2 := (S \cap C) \setminus Q_1$ have size at most

$$(1 + e_2) \frac{|S \cap C|}{2} \leq (1 + e_2) \frac{|C|}{4},$$

so they are included in $\mathcal{Q}^{+e_2}(C)$ and, respectively, in the lists $R_{A, t_*, \mathbf{r}}$ and $R_{B, t_*, \mathbf{r}}$ created in [Line 7](#). As a consequence, in iteration t_* of [Line 9](#), we recover a solution pair (a', Q_1) , (b', Q_2) with constant probability by [Lemma 27](#).

To complete the lemma it remains to reason about the iteration of [Line 5](#) that selects t_* , and prove

- **(i)** a lower bound on the probability that at least one of the s many samples $\mathbf{r} \sim [\mathbf{p}]$ in [Line 7](#) is good and
- **(ii)** an upper bound on the probability that these samples generate a total of k or more

sum-subset pairs.

We divide into two cases corresponding to Case I and Case II of Step 0 and prove (i) and (ii) for each. (Recall that whether the algorithm enters Case I or Case II is determined by whether $|X \pmod{p}| \geq \ell^\rho / \log \ell$ and subsequently determines the manner in which we sample the small random sets \mathbf{C} and \mathbf{D} .)

In Case I:

(i). By Lemma 22, proved below, we have $|\xi(\mathbf{C}) \pmod{p}| = 2^{|\mathbf{C}|}$, which implies that $|\xi(S \cap \mathbf{C}) \pmod{p}| = 2^{|\mathbf{S} \cap \mathbf{C}|}$ and thus

$$\begin{aligned} |W' \pmod{p}| &= \tilde{\Omega} \left(\binom{|\mathbf{S} \cap \mathbf{C}|}{|\mathbf{S} \cap \mathbf{C}|/2} \right) \\ &= \tilde{\Omega}(2^{|\mathbf{S} \cap \mathbf{C}|}) \\ &\geq \tilde{\Omega}(2^{(1-\varepsilon'_1) \cdot |\mathbf{C}|/2}) \\ &= \tilde{\Omega}(\ell^{(1-\varepsilon'_1) \cdot \rho/4}), \end{aligned}$$

in which the first line follows from Stirling's approximation (2.3) and the remaining lines follow from Condition 1' (5.14) and the choice of $|\mathbf{C}| = \frac{1}{2} \log(\ell^\rho / \log \ell)$ in Lemma 22, $e_1 = \varepsilon'_1$, and $e_2 = 0$ in Case I.

Thus there exists a choice of

$$s = s_I := \tilde{\Theta}(\ell^{1+\beta/2-(1-\varepsilon'_1) \cdot \rho/4}) \tag{5.17}$$

that ensures we obtain a good residue with constant probability.

(ii). All candidate lists $\{R_{A,t^*,i}\}_{i \in [p]}$, $\{R_{B,t^*,i}\}_{i \in [p]}$ contain a total of

$$(|\vec{\mathfrak{S}}(A)| + |\vec{\mathfrak{S}}(B)|) \cdot |\mathcal{Q}^{+e_2}(C)| = \tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{(H(1/4)-1/2) \cdot \rho/2 - (1-\rho+\beta/2)/2})$$

many sum-subset pairs, by construction (Line 4), the choices of $|A|$, $|B|$, and $|C|$, $e_2 = 0$, and Stirling's approximation (2.3). Thus, a taking $s = s_I$ many samples $\mathbf{r} \sim [p]$ creates

$$(|\vec{\mathfrak{S}}(A)| + |\vec{\mathfrak{S}}(B)|) \cdot |\mathcal{Q}^{+e_2}(C)| \cdot \frac{s(\ell)}{p} \leq \frac{1}{3}k$$

sum-subset pairs in expectation, and choosing

$$k = k_I := \tilde{\Theta}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-((1-H(1/4)-\epsilon'_1/2) \cdot \rho/2 + (1-\rho+\beta/2)/2)}), \quad (5.18)$$

allows us to reduce the probability that k or more sum-subset couples are generated to an arbitrarily small constant by Markov's inequality. Thus, in Case I, the algorithm succeeds with constant probability.

In Case II:

Consider the collection $\mathcal{C}'(X)$ of size- $(\beta \log(\frac{\ell}{\beta \log \ell}))$ subsets $C' \subseteq X$ that satisfy Conditions 1' (5.14) and 2' (5.15):

$$\mathcal{C}'(X) := \{C' \subseteq X \mid (5.14) \text{ and } (5.15) \text{ hold, and } |C'| = \beta \log(\frac{\ell}{\beta \log \ell})\}$$

Consider also the restriction of this set to $(X \setminus D)$:

$$\mathcal{C}'(X \setminus D) := \{C \subseteq (X \setminus D) \mid (5.14) \text{ and } (5.15) \text{ hold, and } |C| = \beta \log(\frac{\ell}{\beta \log \ell})\}.$$

Conditioning on the event that we are in Case II and both (5.14) and (5.15) hold, the random subset \mathbf{C} is distributed as $\mathbf{C} \sim \mathcal{C}'(X \setminus D)$.

Let

$$P_2 \subseteq [\ell^{1+\beta/2} : 2\ell^{1+\beta/2}] \quad (5.19)$$

denote the set of prime values for \mathbf{p} such that, with respect to our input instance X , the algorithm enters Case II (the ‘‘Case II primes’’). Similarly, let P_1 denote the set values for \mathbf{p} such that the algorithm enters Case I (‘‘Case I primes’’). Without loss of generality, we assume that

$$|P_2| = \Omega(|P_1|), \quad (5.20)$$

that is, that Case II occurs with constant probability over $\mathbf{p} \sim [\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$. (Otherwise, the algorithm enters Case I and succeeds with constant probability, so analysis of Case II is not necessary.) Conditioning on the algorithm entering Case II, the random modulus \mathbf{p} is distributed as $\mathbf{p} \sim P_2$.

(i). Each subset $C' \in \mathcal{C}'(X)$ satisfies the technical condition

$$\begin{aligned} \frac{|S \cap C'|}{\log \ell} \cdot \left(1 - H\left(\frac{1 - \varepsilon_2}{2}\right)\right) &\geq (1 - o_n(1)) \cdot \frac{1 - \varepsilon_1}{2} \cdot \beta \cdot \left(1 - H\left(\frac{1 - \varepsilon_2}{2}\right)\right) \\ &> \lambda, \end{aligned}$$

which can be verified for our chosen λ (5.23), β (5.24), ε_1 (5.2), ε_2 (5.3), and the choice of $|C'| = \beta \log(\frac{\ell}{\beta \log \ell})$ using Condition 1' (5.14).

Following identical arguments to the proof of Claim 12, we can conclude that

$$|W'| \leq 2^{|S \cap C'|} \leq \ell^{\beta/2} \text{ and} \quad (5.21)$$

$$|W'| = \Omega(2^{|S \cap C'|} \cdot \ell^{-\lambda}) = \tilde{\Omega}(\ell^{(1-\varepsilon_1) \cdot \beta/2 - \lambda}). \quad (5.22)$$

Thus we have

$$|W'| \log_2(\text{diam}(W')) \leq \ell^{1+\beta/2},$$

which allows us to apply the Second Prime Distribution Lemma ([Lemma 10](#)) to conclude that there are

$$|W' \bmod \mathbf{p}| = \Theta(|W'|) = \tilde{\Omega}(\ell^{(1-\varepsilon_1) \cdot \beta/2 - \lambda})$$

good residues with constant probability over $\mathbf{p} \sim (P_1 \cup P_2) = [\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$ and $\mathbf{C}' \sim \mathcal{C}'(X)$. Moreover, because a constant fraction of choices for \mathbf{p} are Case II primes by [\(5.20\)](#), we conclude that

$$\Pr_{(\mathbf{p}, \mathbf{C}') \sim P_2 \times \mathcal{C}'(X)} \left[|\mathbf{W}' \bmod \mathbf{p}| = \tilde{\Omega}(\ell^{(1-\varepsilon_1) \cdot \beta/2 - \lambda}) \right] = \Omega(1).$$

So far, we have performed our analysis for a random subset \mathbf{C}' sampled uniformly at random from $\mathcal{C}'(X)$, rather than the *actual* support $\mathcal{C}(X \setminus D)$. Fortunately, as \mathbf{D} is very small, the restriction of the sample to $X \setminus D$ removes a negligible fraction of subsets:

$$\begin{aligned} \frac{|\mathcal{C}'(X) \setminus \mathcal{C}'(X \setminus D)|}{|\mathcal{C}'(X)|} &\leq 1 - \frac{\binom{n-|D|}{|\mathcal{C}'|}}{\binom{n}{|\mathcal{C}'|}} \\ &\in \frac{\beta \log^2(\ell)}{n} \cdot (1 \pm o_n(1)) \\ &= o_n(1). \end{aligned}$$

Thus, a uniform random $\mathbf{C} \sim \mathcal{C}(X \setminus D)$ yields a large residue set $(\bmod \mathbf{p})$ with constant

probability. Conditioned on this event, choosing a sufficiently large

$$s := s_{II} = \Omega\left(\frac{\mathbf{p}}{|\mathbf{W}' \bmod \mathbf{p}|}\right) = \tilde{\Omega}(\ell^{1+\varepsilon_1 \cdot \beta/2+\lambda})$$

yields at least one good residue with constant probability.

(ii). By substituting the Case II values of $|A|$, $|B|$ and $|C| = \beta \log(\frac{\ell}{\beta \log \ell})$ into the proof of (ii) from Case I, we conclude that the probability of generating more than k sum-subset pairs in total is an arbitrarily small constant for an appropriate choice of

$$k = k_{II} := \tilde{\Theta}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-((1-H(\frac{1+\varepsilon_2}{4}))-\varepsilon_1/2) \cdot \beta - \lambda}).$$

We conclude that, in Case II, the algorithm succeeds with constant probability. Combining this fact with our proof for Case I concludes the proof of correctness for [Theorem 13](#). \square

Proof of Runtime for [Algorithm 5.5](#). Line 0 takes time $\text{poly}(\ell) = \text{poly}(n)$ regardless of whether the algorithm enters Case I ([Lemmas 22](#) and [23](#)) or Case II. This is easily dominated by runtime of the rest of our algorithm. To analyse the runtime of the remaining steps, we divide our analysis into two parts based on whether the algorithm enters Case I or Case II in Step 0 of [Algorithm 5.5](#).

Case I. Depending on the choice of parameters, the runtime bottleneck occurs in Lines [2-3](#) or Lines [5-10](#):

- Lines [2-3](#) take time

$$\tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-(1-H(\frac{1-\varepsilon'_1}{2})) \cdot \rho/4})$$

by [Lemma 25](#) and the choice of $|C|$.

- Line 4 takes time

$$O(2^{|A|} + 2^{|B|} + 2^{|C|} + 2^{|D|}) = \tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-(2-\rho+\beta)/4})$$

by Lemma 2 and the choices of $|A|$, $|B|$, $|C|$, and $|D|$. This runtime is dominated by that of Line 2 as

$$\left(1 - H\left(\frac{1 - \varepsilon'_1}{2}\right)\right) \rho < 2 - \rho + \beta$$

for β and ε'_1 as defined in (5.24) as $\rho \leq 1 + o(1)$ by (5.12).

- Lines 5-10 take time

$$\tilde{O}(k_I(n, \ell) \cdot \ell^{1-\rho+\beta/2}) = \tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-((4-2H(1/4)-\varepsilon'_1)\cdot\rho-2-\beta)/4}) :$$

- Line 5 iterates Lines 6-8 a total of $|\S(D) \bmod \mathbf{p}| = \tilde{O}(\ell^{1-\rho+\beta/2})$ times, by Lemma 23. Each iteration takes time

$$\tilde{O}\left(\sum_{\mathbf{r}} (|\mathbf{R}_{A,t',r}| + |\mathbf{R}_{B,t',r}|)\right) = \tilde{O}(k_I)$$

by Lemma 26.

- Line 9 iterates Line 10 a total of $2^{|D|} = \ell^{2-\rho+\beta/2}$ times. Each iteration takes time $\tilde{O}(k_I \cdot \ell^{-1})$ by Lemma 27. (Note that we can convert the expected runtime guarantee of Lemma 27 into a runtime upper bound with arbitrarily small failure probability using a runtime cutoff, by Markov's inequality.)

Taking the maximum runtime over lines 2-3 and lines 5-10, Algorithm 5.5 achieves a speedup

of $\tilde{\Omega}(\ell^{1/2+\gamma'_*})$ over Meet-in-the-Middle in Case I, where

$$\gamma'_* := \min \left\{ \frac{1 - H((1 - \varepsilon'_1)/2)}{4} \cdot \rho, \frac{4 - 2H(1/4) - \varepsilon'_1}{4} \cdot \rho - \frac{1}{2} - \frac{\beta}{4} \right\}.$$

Case II. Once again, depending on the choice of parameters, the bottleneck occurs in Lines 2-3 or Lines 5-10:

- Lines 2-3 take time

$$\tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-\lambda/2}) + \tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-(1-H(\frac{1-\varepsilon_1}{2}))\cdot\beta/2})$$

by Lemmas 24 and 25.

- Line 4 takes time $O(2^{|A|} + 2^{|B|} + 2^{|C|} + 2^{|D|}) = \tilde{O}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-\beta/2})$, by Lemma 2 and the choices of $|A|$, $|B|$, $|C|$, and $|D|$. This runtime is dominated by that of Line 2.
- Lines 5-10 take time

$$\tilde{O}(k_{II}) = \tilde{\Theta}(2^{n/2} \cdot \ell^{-1/2} \cdot \ell^{-((1-H(\frac{1+\varepsilon_2}{4})-\varepsilon_1/2)\cdot\beta-\lambda)}).$$

- Lines 5 iterates Lines 6-8 a total of

$$|\S(D) \bmod \mathbf{p}| \leq |D| + 1 = O(\log \ell) = O(\log n)$$

times, because all integers in \mathbf{D} are congruent modulo \mathbf{p} by construction

(Line 0). Each iteration takes time

$$\tilde{O}\left(\sum_{\mathbf{r}} (|\mathbf{R}_{A,t',\mathbf{r}}| + |\mathbf{R}_{B,t',\mathbf{r}}|)\right) = \tilde{O}(k_{II})$$

by Lemma 26.

- Line 9 iterates Line 10 a total of $2^{|D|} = \ell$ times, by the choice of $|D|$. Each iteration takes time $\tilde{O}(k_{II} \cdot \ell^{-1})$, by Lemma 27. (Note that we can convert the expected runtime guarantee of Lemma 27 into a runtime upper bound with arbitrarily small failure probability using a runtime cutoff, by Markov's inequality.)

Taking the maximum runtime over Lines 2-3 and Lines 5-10, we achieve a speedup of $\tilde{\Omega}(\ell^{1/2+\gamma''})$ over Meet-in-the-Middle in Case II, where

$$\begin{aligned} \gamma''_* &:= \min \left\{ \frac{\lambda}{2}, \frac{1 - H((1 - \varepsilon_1)/2)}{2} \cdot \beta, \left(1 - H\left(\frac{1 + \varepsilon_2}{4}\right) - \frac{\varepsilon_1}{2}\right) \cdot \beta - \lambda \right\} \\ &= \frac{\lambda}{2} = (1 - 10^{-5}) \cdot \frac{1 - \varepsilon_1}{4} \cdot \left(1 - H\left(\frac{1 - \varepsilon_2}{2}\right)\right) \cdot \beta \geq 9.0324 \times 10^{-3} \cdot \beta. \end{aligned}$$

Set

$$\lambda = \lambda(\rho) := (1 - 10^{-5}) \cdot \frac{1 - \varepsilon_1}{2} \cdot \left(1 - H\left(\frac{1 - \varepsilon_2}{2}\right)\right) \cdot \beta \approx 1.8065 \times 10^{-2} \cdot \beta. \quad (5.23)$$

so that $\frac{\lambda}{2}$ minimizes the expression above for our chosen values of ε_1 and ε_2 , as in the proof of runtime for Algorithm 5.2.

Constant Balancing. We set the constants $\gamma_* = \gamma_*(\rho) := \min\{\gamma'_*, \gamma''_*\}$, $\beta = \beta(\rho)$, and

$\varepsilon'_1 = \varepsilon'_1(\rho)$ to satisfy the equations

$$\gamma_*(\rho) = 9.0324 \times 10^{-3} \cdot \beta(\rho) \quad (5.24)$$

$$= \frac{1 - H\left(\frac{1-\varepsilon'_1(\rho)}{2}\right)}{4} \cdot \rho \quad (5.25)$$

$$= \frac{4 - 2H(1/4) - \varepsilon'_1(\rho)}{4} \cdot \rho - \frac{1}{2} - \frac{\beta(\rho)}{4}. \quad (5.26)$$

Alternatively, we can write the speedup as $\Omega(n^\alpha)$ for

$$\alpha = \frac{1 + 2\gamma}{2\rho} \in (0, \alpha_*(\rho)) \text{ and}$$

$$\alpha_*(\rho) := \frac{1 + 2\gamma_*(\rho)}{2\rho}.$$

In the density range

$$\frac{1}{2 - H(1/4)} < \rho \leq 1 + \Theta\left(\frac{1}{\log n}\right),$$

which we assume without loss of generality (5.12), the minimum speedup of Algorithm 5.5 over Meet-in-the-Middle is a factor of $\tilde{\Omega}(n^{\alpha_*(1)}) \geq \Omega(n^{0.5023})$ when word length is linear $\ell = \Theta(n)$. \square

The result listed is the speed-up achieved for worst-case ℓ ; for other density values, the speed-up varies. We plot the values $\alpha_*(\rho)$, $\gamma_*(\rho)$, $\beta(\rho)$, and $\varepsilon'_1(\rho)$ over the density range $\frac{1}{2-H(1/4)} < \rho \leq 1 + \Theta\left(\frac{1}{\log n}\right)$ in Figure 5.4. The maximum speed-up, a factor of

$$\tilde{\Omega}(n^{(2-H(1/4))/2}) \geq \Omega(n^{0.5943}),$$

occurs when word length is the slightly superlinear $\ell = \Theta(n^{2-H(1/4)}) \approx \Theta(n^{1.1887})$.

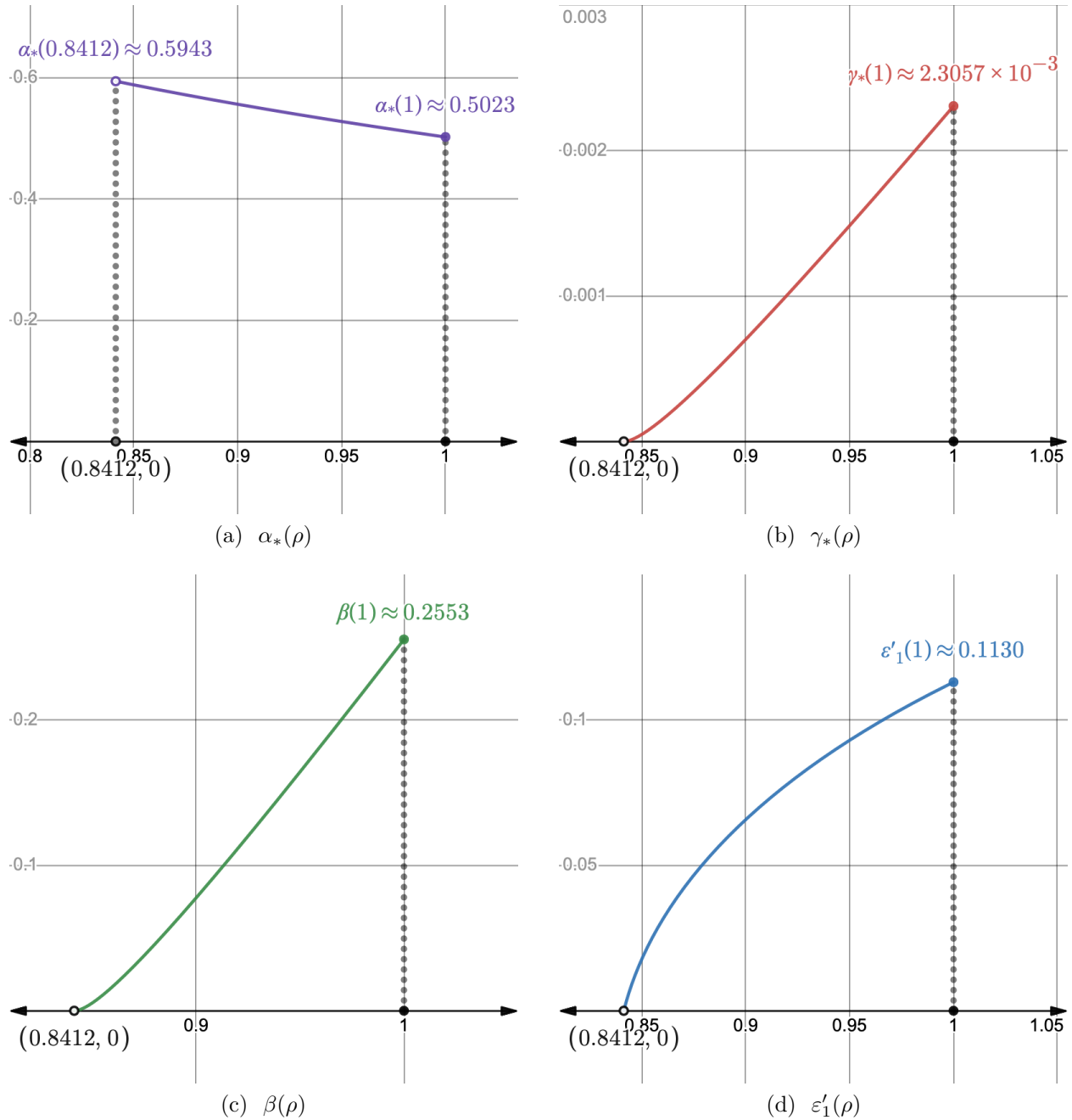


Figure 5.4: Plot of $\alpha_*(\rho)$, $\gamma_*(\rho)$, $\beta(\rho)$, and $\varepsilon'_1(\rho)$ in the range $\frac{1}{2-H(1/4)} < \rho \leq 1 + \Theta(\frac{1}{\log n})$.

5.5.1 Auxiliary Lemmas

Lemma 22 (Finding a Subset C with Large $\S(C)$). *Fix any integer p and multiset Y of cardinality n satisfying*

$$|Y \bmod p| > \frac{\ell^p}{\log \ell}.$$

There exists a subset $C \subseteq Y$ of cardinality

$$|C| = \frac{1}{2} \log \left(\frac{\ell^p}{\log \ell} \right)$$

satisfying $|\S(C) \bmod p| = 2^{|C|}$. Moreover, we can recover C in time $\text{poly}(\ell) = \text{poly}(n)$.

Proof. We create the subset $C \subseteq Y$ using a simple greedy algorithm, as follows:

1. Define $C_0 := \emptyset$.
2. For $i \in [\frac{1}{2} \log(\ell^p / \log \ell)]$:
 - (a) Select any $y_i \in (Y \setminus C_{i-1})$ which satisfies $y_i \not\equiv (c' - c'') \pmod{p}$ for any two $c', c'' \in \S(C_{i-1})$.
3. Set $C_i := C_{i-1} \cup \{y_i\}$.

The existence of a satisfactory value y_i in step 2(a) is guaranteed because the intermediate subset C_{i-1} has at most $|\S(C_{i-1})| \leq 2^{i-1}$ subset sums, and thus

$$|\{(c' - c'') \bmod p \mid c', c'' \in C_{i-1}\}| \leq \frac{\ell^p}{\log \ell} < |Y \bmod p|. \quad \square$$

Lemma 23 (Finding a Subset D with Small $\S(D)$). *Fix a integer $p = \Theta(\ell^{1+\beta/2})$ and a multiset Y of cardinality $\Theta(n)$. There exists a subset $D \subseteq Y$ of cardinality*

$$\left(2 - \rho + \frac{\beta}{2}\right) \log \ell$$

such that

$$|\S(D) \bmod p| = \tilde{O}(\ell^{1-\rho+\beta/2}).$$

Moreover, we can recover D in time $\text{poly}(\ell) = \text{poly}(n)$.

Proof. For a sufficiently small quantity

$$q = \Theta\left(\frac{n}{\log \ell}\right) = \Theta\left(\frac{\ell^\rho}{\log \ell}\right),$$

partition $[p] = \bigsqcup_{j \in [q]} P_j$ into the collection of arithmetic progressions

$$P_j := \{r \in [p] \mid r = j \pmod{q}\} \text{ for } j \in [q],$$

each of which contains $\lceil p/q \rceil$ or $\lfloor p/q \rfloor$ many residues. By the pigeonhole principle, for at least one of these progressions P_{j^*} the set

$$Y_{j^*} := \{y \in Y \mid y \in P_{j^*} \pmod{p}\}$$

has cardinality at least

$$|Y_{j^*}| \geq \frac{|Y|}{q} > \left(2 - \rho + \frac{\beta}{2}\right) \log \ell.$$

We can find such a progression P_{j^*} and the corresponding subset Y_{j^*} in time $\text{poly}(\ell) =$

$\text{poly}(n)$. Let $D = \{y_i\}_{i \in [|D|]}$ be an arbitrary subset of Y_{j^*} containing $(2 - \rho + \frac{\beta}{2}) \log \ell$ elements. Each selected y_i falls into the residue class

$$(y_i \bmod p) = q \cdot k_i + j^*,$$

where

$$k_i := \left\lfloor \frac{y_i \pmod{p}}{q} \right\rfloor \leq \frac{p}{q}.$$

Thus D satisfies

$$\begin{aligned} |\S(D) \bmod p| &\leq |\S(\{k_i\}_{i \in [|D|]})| \cdot |D| \\ &\leq \Sigma(\{k_i\}_{i \in [|D|]}) \cdot |D| \\ &\leq \frac{p}{q} \cdot |D|^2 \\ &= O(\ell^{1-\rho+\beta/2} \cdot \log^3(\ell)) \end{aligned}$$

distinct subset sums modulo p , as desired. \square

Below, [Lemmas 24](#) and [25](#) refine [Lemmas 6](#) and [20](#), respectively, to leverage bit packing techniques in the specific context of [Algorithm 5.5](#).

Lemma 24 (Speedup via Additive Structure). *Let (X, t) be a Subset Sum instance. Given a subset $Y \subseteq X$ of size*

$$|Y| \leq \frac{n - \log \ell}{2}$$

such that

$$|\S(Y)| \leq 2^{|Y|} \cdot \ell^{-\varepsilon}$$

for some constant

$$0 < \varepsilon < \frac{|Y|}{\log \ell}$$

The instance (X, t) can be solved in time $\tilde{O}(2^{n/2} \cdot \ell^{-(1+\varepsilon)/2})$ with arbitrarily high constant probability and with no false positives.

Proof. Fix any partition $X = A \sqcup B \sqcup D$ such that

$$|D| = \log \ell, \quad |A| = \frac{n - (1 - \varepsilon) \log \ell}{2} \text{ with } A \supseteq Y, \text{ and } |B| = \frac{n - (1 + \varepsilon) \log \ell}{2}.$$

We have

$$\begin{aligned} |\mathfrak{S}(A)| &\leq 2^{|A \setminus Y|} \cdot |\mathfrak{S}(Y)| \\ &\leq 2^{n/2} \cdot \ell^{-(1+\varepsilon)/2} \end{aligned}$$

and

$$|\mathfrak{S}(B)| \leq 2^{n/2} \cdot \ell^{-(1+\varepsilon)/2}.$$

Applying [Lemmas 2](#) and [3](#) and [Theorem 7](#), it takes expected time $O(2^{n/2} \cdot \ell^{-(1+\varepsilon)/2} \cdot \log \ell)$ to create the sorted lists $\vec{\mathfrak{S}}(A), \vec{\mathfrak{S}}(B)$ and to run [BitPacking \(Algorithm 5.1\)](#).

Cutting off the algorithm if it takes longer than a large constant times the expected runtime yields a Monte Carlo algorithm with one-sided error by Markov's inequality. \square

Lemma 25 (Speedup via Unbalanced Solutions). *Let (X, t) be a Subset Sum instance, and let $Y \subseteq X$ be a subset of size $|Y| = c \log \ell$ such that some solution $S \subseteq X$ satisfies $|S \cap Y| \notin (1 \pm \varepsilon) \frac{|Y|}{2}$ for some constants $c > 0, \varepsilon \in (0, 1)$. Then S can be found in time*

$$\tilde{O}(2^{n/2} \cdot \ell^{-(1+\delta)/2}),$$

where

$$\delta := \left(1 - H\left(\frac{1 - \varepsilon}{2}\right)\right) \cdot c,$$

with arbitrarily high constant probability.

Proof. Combine the proofs of [Lemmas 20](#) and [24](#). □

Subroutine **SampleList**($R_{A,t'}, h_m$)

Input: A sorted list $R_{A,t'}$ containing elements in the sum-collection couple format $(a', \mathcal{Q}_{a'})$, and a hash function h_m .

Output: A hashed and packed list $H_{A,t'}$.

7(a) Initialize indices $i, j := 0$. While $i < |R_{A,t'}|$, set three words of $H_{A,t'}$ as follows:

7(b) $H_{A,t'}[3j] := a'_i$ stores the integer a'_i , the smallest not-yet-packed sum.

7(c) $H_{A,t'}[3j + 1] := ((h_m(a'_i), \mathcal{Q}_{a'_i}), (h_m(a'_{i+1}), \mathcal{Q}_{a'_{i+1}}), \dots)$ packs as many hash-collection couples as will fit into a single ℓ -bit word.
Update i to the index of the next not-yet-packed sum in $R_{A,t'}$.

7(d) $H_{A,t'}[3j + 2] := a'_{i-1}$ stores the integer a'_{i-1} , the largest already-packed sum.

7(e) Increment $j \leftarrow j + 1$.

Figure 5.5: The **Sample-Packing** subroutine.

Lemma 26 (Hashing and Packing; Line 8 of [Algorithm 5.5](#)).

1. Line 8 of [Algorithm 5.5](#) enumerates the lists $\mathbf{H}_{A,t'}$ and $\mathbf{H}_{B,t'}$ in time $\tilde{O}(k)$.
2. The lists $\mathbf{H}_{A,t'}$ and $\mathbf{H}_{B,t'}$ use $|\mathbf{H}_{A,t'}| + |\mathbf{H}_{B,t'}| = \tilde{O}(k \cdot \ell^{-1})$ words.

Proof. Line 8 creates $\mathbf{H}_{A,t'}$, $\mathbf{H}_{B,t'}$ using the subroutine **SampleList** ([Algorithm 5.5](#)). The

claimed runtime of $\tilde{O}(k)$ follows from the fact that the input list $\mathbf{R}_{\mathcal{A},t'}$ contains

$$|\mathbf{R}_{\mathcal{A},t'}| \leq \sum_{a'} |\mathcal{Q}_{a'}| \leq k$$

sum-collection couples by construction (Lemma 21 and Line 6), and Algorithm 5.5 goes through it once, spending time $O(\log \ell)$ on each couple to compute the hash $\mathbf{h}_m(a')$.

To see that the output list $\mathbf{H}_{\mathcal{A},t'}$ is stored in $\tilde{O}(k \cdot \ell^{-1})$ many ℓ -bit words, we first observe that a single hash-collection couple $(\mathbf{h}_m(a'), \mathcal{Q}_{a'})$ takes at most

$$\begin{aligned} m + |\mathcal{Q}_{a'}| \cdot |\mathbf{C}| &\leq m + |\mathcal{Q}^{+e_3}(\mathbf{C})| \cdot |\mathbf{C}| \\ &\leq m + 2^{|\mathbf{C}|} \cdot |\mathbf{C}| \\ &= \tilde{O}(\ell^{1/2}) \\ &= o(\ell) \end{aligned}$$

many bits, for $m = 3 \log \ell$ and

$$\begin{aligned} |\mathbf{C}| &= \max \left\{ \frac{1}{2} \log \left(\frac{\ell^p}{\log \ell} \right), \beta \log \left(\frac{\ell}{\beta \log \ell} \right) \right\} \\ &\leq \frac{1}{2} \log \ell \end{aligned}$$

(Line 0 and Figure 5.4c). That is, a single word $\mathbf{H}_{\mathcal{A},t'}[3j+1]$ created in Line 7(c) has at most $o(\ell)$ many unused bits, which is negligible compared to the word length. Moreover, all of the $|\mathbf{R}_{\mathcal{A},t'}| \leq \sum_{a'} |\mathcal{Q}_{a'}| \leq k(\ell)$ many hash-collection couples $(\mathbf{h}_m(a'), \mathcal{Q}_{a'})$ take a total of

at most

$$\begin{aligned} |\mathbf{R}_{A,t'}| \cdot m + \sum_{a'} |\mathcal{Q}_{a'}| \cdot |\mathbf{C}| &\leq k \cdot 3 \log \ell + k \cdot \frac{1}{2} \log \ell \\ &= \tilde{O}(k) \end{aligned}$$

many bits. Accordingly, a total of $\tilde{O}(k) \cdot \ell^{-1} = \tilde{O}(k \cdot \ell^{-1})$ many words $\mathbf{H}_{A,t'}[3j + 1]$ are created throughout all the executions of Line 7(c). The entire output list $\mathbf{H}_{A,t'}$ has three times as many words because of the additional elements storing the highest and the lowest values hashed into each word $\mathbf{H}_{A,t'}[3j + 1]$. \square

Subroutine **SearchList**($t'', R_{A,t'}, R_{B,t'}, H_{A,t'}, H_{B,t'}$)

Input: A shifted target t'' and sorted lists $R_{A,t'}, R_{B,t'}, H_{A,t'}, H_{B,t'}$ for $t' = (t'' \bmod p)$.

9(a) Initialize indices $i := 0$ and $j := |H_{B,t'}| - 1$. While $i < |H_{A,t'}|$ and $j \geq 0$:

9(b) If the indexed words $H_{A,t'}[3i + 1]$, $H_{B,t'}[3j + 1]$ contain hash-collection pairs $(\mathbf{h}_m(a'), \mathcal{Q}_{a'})$, $(\mathbf{h}_m(b'), \mathcal{Q}_{b'})$ with

$$\mathbf{h}_m(a') + \mathbf{h}_m(b') \in \mathbf{h}_m(t'') - \{0, 1\} \pmod{2^m}$$

and $Q_{a'} \cap Q_{b'} = \emptyset$ for some $Q_{a'} \in \mathcal{Q}_{a'}$, $Q_{b'} \in \mathcal{Q}_{b'}$, use **Lemma 21** to search the corresponding sublists of $R_{A,t'}, R_{B,t'}$ for a solution as described in the proof of **Lemma 27**. Halt and return “yes” if a solution is found.

9(c) If $H_{A,t'}[3i] + H_{B,t'}[3j + 2] < t''$, set $i \leftarrow i + 1$. Otherwise, set $j \leftarrow j - 1$.

Figure 5.6: The **Sample-Searching** subroutine.

Lemma 27 (Searching for Solutions; Line 10 of **Algorithm 5.5**).

1. *Correctness:* Line 10 of **Algorithm 5.5** finds a solution if the lists $\mathbf{R}_{A,(t'' \bmod p)}$ and $\mathbf{R}_{B,(t'' \bmod p)}$ contain one.

2. *Runtime:* Line 10 of Algorithm 5.5 finds a solution in expected time $\tilde{O}(k/\ell)$.

Proof. Correctness. Algorithm 5.6 adapts Lines 5-7 of Algorithm 5.1 for the lists $\mathbf{H}_{A,t'}$, $\mathbf{H}_{B,t'}$ to address two additional issues: First, not only a hash collision but also a pair of overlapping near-quartersets in $\mathbf{H}_{A,t'}$, $\mathbf{H}_{B,t'}$ may incur a false positive. Second, each word in $\mathbf{H}_{A,t'}$, $\mathbf{H}_{B,t'}$ packs an uncertain amount of hash-collision pairs (while Algorithm 5.1 processes exactly (ℓ/m) hashes per iteration).

Line 9(b) settles the first issue by modifying the “If” test in line 5 of Algorithm 5.1. This new test is implemented by Packed0V. Recall that Packed0V it takes as input two packed words

$$u = ((\mathbf{h}_m(a'_1), \mathcal{Q}_{a'_1}), (\mathbf{h}_m(a'_2), \mathcal{Q}_{a'_2}), \dots) \text{ and}$$

$$v = ((\mathbf{h}_m(b'_1), \mathcal{Q}_{b'_1}), (\mathbf{h}_m(b'_2), \mathcal{Q}_{b'_2}), \dots),$$

where each $\mathbf{h}_m(a')$, $\mathbf{h}_m(b')$ is a hashed value and each $\mathcal{Q}_{a'}$, $\mathcal{Q}_{b'} \subseteq \mathcal{Q}^{+e_3}(\mathbf{C})$ is a collection of near-quartersets. Packed0V(u, v) returns 1 if and only if two conditions hold: first, there is a pair of hash-collection couples $(\mathbf{h}_m(a'), \mathcal{Q}_{a'})$, $(\mathbf{h}_m(b'), \mathcal{Q}_{b'})$ with $\mathbf{h}_m(a') + \mathbf{h}_m(b') \in \mathbf{h}_m(t'') - \{0, 1\} \pmod{2^m}$; second, there are two disjoint near-quartersets $Q_{a'} \cap Q_{b'} = \emptyset$, for $Q_{a'} \in \mathcal{Q}_{a'}$, $Q_{b'} \in \mathcal{Q}_{b'}$, in the packed collections indexed by $\mathbf{h}_m(a')$, $\mathbf{h}_m(b')$. Packed0V is an AC^0 operation on two words and takes constant time to evaluate in the circuit RAM model, since all hash-collection couples can be checked in parallel.

Now on comparing any two words $\mathbf{H}_{A,t'}[3i+1]$, $\mathbf{H}_{B,t'}[3j+1]$, by the second condition above, overlapping near-quartersets never incur false positives. Hence like Algorithm 5.1, the new “If” test in Line 9(b) is passed by every correct solution and (some of) the hash collisions. We further verify such a potential solution by returning back to the (unhashed)

lists $\mathbf{R}_{A,t'}$, $\mathbf{R}_{B,t'}$ and checking all the sum-collection couples that are packed into $\mathbf{H}_{A,t'}[3i+1]$, $\mathbf{H}_{B,t'}[3j+1]$, namely the two sorted sublists

$$\{(a', \mathcal{Q}_{a'}) \mid \mathbf{H}_{A,t'}[3i] \leq a' \leq \mathbf{H}_{A,t'}[3i+2]\} \text{ and}$$

$$\{(b', \mathcal{Q}_{b'}) \mid \mathbf{H}_{B,t'}[3j] \leq b' \leq \mathbf{H}_{B,t'}[3j+2]\}.$$

Either sublist is stored in at most (ℓ/m) words by construction (see [Algorithm 5.5](#), Line 7(c)). Hence by [Lemma 21](#), this verification process takes time $O(\ell/m) = \text{poly}(n)$.

Line 9(c) settles the second issue by replacing the test in Line 7 of [Algorithm 5.1](#) with a new test: whether

$$\mathbf{H}_{A,t'}[3i] + \mathbf{H}_{B,t'}[3j+2] < t''$$

holds. By construction ([Algorithm 5.5](#)), $\mathbf{H}_{A,t'}[3i]$ and $\mathbf{H}_{B,t'}[3j+2]$ are the exact values of the smallest sum packed into $\mathbf{H}_{A,t'}[3i+1]$ and the largest sum packed into $\mathbf{H}_{B,t'}[3j+1]$, respectively. By the same argument as in the proof of correctness for [Algorithm 5.1](#), in a single scan of $\mathbf{H}_{A,t'}$ and $\mathbf{H}_{B,t'}$, we never miss a pair of words that packs a correct solution.

Runtime. [Algorithm 5.6](#) performs a single scan of $\mathbf{H}_{A,t'}$ and $\mathbf{H}_{B,t'}$, plus the verification of at most one correct solution versus the hash collisions. Using the same argument as in the proof of runtime for [Algorithm 5.1](#) (Line 6), the expected verification time

$$\text{poly}(n) + (|\mathbf{H}_{A,t'}| + |\mathbf{H}_{B,t'}|) \cdot o_n(1)$$

is dominated by the scan time, which is

$$O(|\mathbf{H}_{A,t'}| + |\mathbf{H}_{B,t'}|) = \tilde{O}(k/\ell),$$

by [Lemma 26](#). □

5.5.2 Adapting [Algorithm 5.5](#) to Word RAM

Corollary 9. *[Algorithm 5.5](#) solves the Vanilla Subset Sum problem with constant success probability in time $O(2^{n/2} \cdot n^{-(1/2+\gamma)})$ for some constant $\gamma > 0.0023$ in the word RAM model.*

Proof. As in [Section 5.4](#), to adapt [Algorithm 5.5](#) to the word RAM model, we run the algorithm as if the word length were $\ell' := 0.1n$ and memoize `PackedOV` to speed up the evaluation of this function. There are three additional modifications to the algorithm:

1. Line [7\(a\)](#) sets $\lceil \ell'/\ell \rceil + 2$ words (rather than three words) in a single iteration, i.e., the single word set in Line [7\(c\)](#) is replaced with $\lceil \ell'/\ell \rceil = \Theta(1)$ words because a single word with ℓ bits may be insufficient.
2. Before the execution of Line [9](#), create a table `PackedOV'` that memoizes all input-output results of the boolean function `PackedOV`, in time $(2^{\ell'})^2 \cdot \text{poly}(\ell') = O(2^{0.21n})$. We also replicate the table `OV'` described in [Section 5.4.4](#). Then either table can be accessed using a $2\lceil \ell'/\ell \rceil = \Theta(1)$ -word index in constant time.
3. Line [9\(b\)](#) replaces the functions `PackedOV` (used in [Lemma 27](#)) and `OV` (used in [Lemma 21](#)) with constant-time lookup into the above memoized tables `PackedOV'` and `OV'`, respectively.

Compared with running [Algorithm 5.5](#) itself for $\ell' = 0.1n$, the only significant difference of this word RAM variant is that the number of packed words in $\mathbf{H}_{A,\ell'}$ and $\mathbf{H}_{B,\ell'}$ increases by a $\Theta(1)$ factor, so we can easily check correctness and that runtime remains the same up to an $\Theta(1)$ factor. □

Chapter 6

Subset Sum Parameterized in the Doubling Constant

This section includes work from [RW23]. Although what follows has been refurbished for inclusion in this thesis, the arguments presented represent the collaborative efforts of the two original authors, Karol Węgrzycki and I.

This chapter contains the following subsections:

- **Summary of Results.**
- **The Constructive Freiman’s Theorem.** Given a set with constant doubling, the generalized arithmetic progression guaranteed by Freiman’s Theorem can be explicitly constructed in time FPT in the doubling constant.
- **\mathcal{C} -Integer Programming.** When the column set of the constraint matrix of an integer program has doubling constant \mathcal{C} , an instance \mathcal{I} of Integer Programming with n bounded variables can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \text{poly}(|\mathcal{I}|)$.

- **\mathcal{C} -Subset Sum.** Subset Sum parameterized in the doubling constant admits an XP-algorithm. Binary Integer Linear Programming Feasibility can be reduced to Subset Sum. Hyperplane-Constrained Binary Integer Linear Programming feasibility can be solved in time $\Delta^{O(m)} \cdot \text{poly}(n)$ if and only if Subset Sum is FPT in the doubling constant.
- **\mathcal{C} -Unbounded Subset Sum.** Unbounded Subset Sum parameterized in the doubling constant admits an algorithm that runs in time $n^{O_{\mathcal{C}}(\log \log \log n)}$, or in time $n^{O_{\mathcal{C}}(1)}$ if ILP instances on v variables can be solved in time $2^{O(n)} \text{poly}(|\mathcal{I}|)$.
- **(\mathcal{C}, k) -SUM with Constant Doubling.** k -SUM is Fixed-Parameter Linear when parameterized in the doubling constant \mathcal{C} in addition to k ; this result is tight for $k = 4$ and nearly tight for larger k .

The field of parameterized complexity tackles hard problems by solving instances whose structural complexity is captured by a certain numeric parameter. Often, this parameter is a natural part of the problem. For example, when searching a large object for a substructure such as a clique or subgraph or for a global superstructure such as a set cover or dominating set, the size of the structure is a natural measure. Alternatively, parameters may be chosen because they effectively capture the complexity of the problem, because they are often bounded in instances arising from real-world applications, or both (for example, treewidth and twin-width). See, e.g., [CFK⁺15, DF12] for introductions to parameterized complexity.

Both pseudopolynomial Subset Sum and k -SUM can be viewed as parameterized versions of the problem (in terms of the size of the target t and the number of elements k in the solution, respectively). Both parameterizations allow more efficient solutions when the relevant parameter is small, and come with their own runtime barriers ($O(t^{1-\varepsilon} 2^{\delta n})$ and $O(t^{1-\varepsilon} n^{\delta k})$ for

δ 's that depend on ε , and perhaps also $O(n^{\lceil k/2 \rceil})$ (see [ABHS22] for the former two claims, and [CWX22, FKP23] for some recent work on the 3-SUM hypothesis).

It is natural to consider small numbers and small solutions, but it is also natural to ask: when is it possible to efficiently find large solutions on instances with large inputs? Fortunately, there is another natural parameter, frequently used in the field of additive combinatorics to measure additive structure. This is the *doubling constant*. Sets with constant doubling have a variety of structural properties that make them desirable to work with (see Section 1.1.3, Additive Structure). In this section, we show that existing results from additive combinatorics, made constructive, can be used to establish a close connection between Subset Sum and Integer Programming, and to design more efficient algorithms for both problems.

In addition, this chapter takes inspiration from an emerging trend in the literature of fine-grained complexity and algorithms: the “import” of results from additive combinatorics. In several recent works, researchers have achieved breakthroughs by taking central existential results from the field of additive combinatorics and modifying their proofs to make the results explicitly and efficiently constructive.

For example, in 2015 Chan and Lewenstein proved a version of the Balog-Szemerédi-Gowers (BSG) theorem that allows certain sets guaranteed by the theorem to be constructed algorithmically [CL15]. 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 [ABF23]. 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 [JX23]. More broadly, these works reflect the increasing role of additive combinatorics in theoretical computer science over the last few decades; for general references, see [Tre09, Vio11, Bib13, Lov17].

6.1 Summary of Results

A Constructive Freiman’s Theorem. We begin by unlocking a new tool to help us manipulate problem instances 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. We make this theorem efficiently constructive by showing how an algorithm can obtain such an arithmetic progression in time $\tilde{O}_C(n)$ (Theorem 15). Later in the chapter, we use this theorem to reduce Subset Sum with constant doubling to a constrained integer programming problem and to design efficient algorithms for Unbounded Subset Sum. We hope that, like the constructive BSG theorem discussed above, the constructive statement of Freiman’s Theorem may find other independent applications.

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

$$\max \left\{ \vec{c} \cdot \vec{x} \mid A\vec{x} = \vec{b}, \vec{x} \in \mathbb{Z}_{\geq 0}^n \right\},$$

where $A \in \mathbb{Z}^{m \times n}$, $\vec{c} \in \mathbb{Z}^n$ and $\vec{b} \in \mathbb{Z}^m$. An integer program specified by a constraint matrix $A \in \mathbb{Z}^{m \times n}$ and solution vector $\vec{b} \in \mathbb{Z}^m$ is *feasible* if there exists a solution $x \in \mathbb{Z}_{\geq 0}^n$ such that $A\vec{x} = \vec{b}$.

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 that an instance \mathcal{I} of ILP feasibility with constant doubling and n binary variables can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \text{poly}(|\mathcal{I}|)$. This follows from Freiman's Theorem (without construction) and a dynamic programming algorithm. The theorem also holds in the case where the variables x_1, x_2, \dots, x_n have upper and lower bounds of magnitude $\text{poly}(n)$.

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 11](#)).

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 $O_{\mathcal{C}}(\text{poly}(n))$? We show that this result appears unlikely by way of an interesting connection to the feasibility of integer programs with binary variables. We prove that Subset Sum with constant doubling can be solved in time $O_{\mathcal{C}}(\text{poly}(n))$ if and only if an instance \mathcal{I} of *Hyperplane-Constrained Binary ILP* (HBILP) can be solved in time $\Delta^{O(m)} \cdot O_m(\text{poly}(|\mathcal{I}|))$.

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 $A\vec{x} \cdot \vec{s} = t$ for a certain target t and “step vector” $\vec{s} \in \mathbb{Z}^m$ orthogonal to a hyperplane. The best existing algorithm solves HBILP Feasibility in time $O_m(|\mathcal{I}|) + \Delta^{O(m^2)}$ ([DLRV23], Corollary 2). However, as previously noted in [DLRV23], 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 [EW19]. Our result implies that finding an FPT algorithm for Subset Sum with constant doubling would be equivalent to solving this problem.

We can also reduce ILP Feasibility with bounded variables to HBILP feasibility (Lemma 36), which means that an FPT algorithm for Subset Sum with constant doubling would further imply a $\Delta^{O(m)} \cdot \text{poly}(n)$ algorithm for ILP Feasibility with bounded variables (Corollary 12). One of the most significant open questions in the parameterized complexity of integer programming is 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 [EW19]. Our results imply that an FPT algorithm for Subset Sum with bounded doubling would allow us to determine the feasibility of ILPs with bounded variables in a similar runtime, which would be a significant breakthrough in the area [JR18, KPW20].

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 $\Delta = n^{O(1/d(c))}$ 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 $n^{O_c(\log \log \log n)}$ using the current best algorithm for solving ILPs, or in time $n^{O_c(1)}$ under the hypothesis that a v -variable ILP \mathcal{I} can be solved in time $2^{O(v)} \text{poly}(|\mathcal{I}|)$ (Theorem 18).

***k*-SUM with Constant Doubling.** The application of recent algorithms for sparse nonnegative convolution [BFN22] allows us to solve *k*-SUM with constant doubling in time $\tilde{O}(\mathcal{C}^{\lceil k/2 \rceil} \cdot 2^{O(k)} \cdot n)$ (Theorem 19).

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 $\tilde{O}(\mathcal{C}n)$, which is optimal up to polylogarithmic factors under the *k*-SUM conjecture.

6.2 Freiman’s Theorem Made Constructive

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

Theorem 14 (Freiman’s Theorem, [Fre64], see [Zha22] for a modern presentation). *Any finite integer set X with $|X + X| \leq \mathcal{C}|X|$ is contained in a GAP P of dimension $d(\mathcal{C})$ and size (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 $\text{polylog}(n)$ factor and a (large) function of \mathcal{C} .

Theorem 15 (FPT Freiman’s Theorem). *Let A be a set of n integers satisfying $|A + A| \leq \mathcal{C}|A|$. There exists an $\tilde{O}_{\mathcal{C}}(n)$ algorithm that, with probability $1 - n^{-\gamma}$ for an arbitrarily large*

constant $\gamma > 0$, returns¹ 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} .

Making Freiman’s Theorem constructive is not difficult from a strictly algorithmic perspective. However, verifying the result requires close attention to the structure of the original proof and requires concepts from additive combinatorics, group theory and the geometry of numbers along the way. For this reason, we closely follow Zhao’s recent exposition of a proof due to Ruzsa [Ruz94], making modifications where necessary. We wish to emphasize that *neither the existential results nor the overall proof structure below are novel*. Our contribution is the introduction of algorithmic techniques required to make the proof constructive in near-linear time.

Given a set X of cardinality n , our proof constructs a GAP P of dimension $d(\mathcal{C}) = 2^{c^{O(1)}}$ and volume $v(\mathcal{C}) = 2^{2^{c^{O(1)}}} n$, as in the original statement of Freiman’s Theorem. We do not attempt to optimize these functions, but we suspect that techniques used to optimize d and v in subsequent proofs of Freiman’s Theorem (e.g., [Cha02, Sch11, San12, San13]) could be used to improve the dependence on \mathcal{C} in our results.

At several points, we make use of the Plünnecke-Ruzsa Inequality, a useful bound on the additive “growth rate” of integer sets with small doubling constant:

Lemma 28 (Plünnecke-Ruzsa Inequality). *If X is a finite subset of an abelian group and*

¹Specifically, we compute the values $x_1, x_2, \dots, x_{d(\mathcal{C})}$ and $L_1, L_2, \dots, L_{d(\mathcal{C})}$.

$|X + X| \leq C|X|$ for a constant C , then for all nonnegative integers s and t ,

$$|sX - tX| \leq C^{s+t}|X|.$$

6.2.1 Ruzsa’s Modeling Lemma

A core ingredient in Freiman’s theorem is Ruzsa’s Modeling Lemma. This allows us to take an integer set A and map a large piece of it to small, finite group (specifically, the prime cyclic group $\mathbb{Z}/q\mathbb{Z}$) in such a way that additive structure is “preserved”: the image in $\mathbb{Z}/q\mathbb{Z}$ behaves isomorphically to the preimage in \mathbb{Z} under addition, up to a certain fixed number s of additions. The size q of the prime cyclic group is controlled by the size of $|sA - sA|$, which is related to the doubling constant by the Plünnecke-Ruzsa Inequality.

A map that preserves additive structure in this way is called a *Freiman s -isomorphism*:

Definition 4 (Freiman Homomorphism and Isomorphism). Given subsets A and B of two (possibly different) abelian groups and a positive integer $s \geq 2$, $\phi : A \rightarrow B$ is a *Freiman s -homomorphism* if

$$\phi(a_1) + \cdots + \phi(a_s) = \phi(a'_1) + \cdots + \phi(a'_s)$$

for all pairs of s -tuples in A satisfying $a_1 + \cdots + a_s = a'_1 + \cdots + a'_s$. ϕ is a *Freiman s -isomorphism* if ϕ is a bijection and both ϕ and ϕ^{-1} are Freiman s -homomorphisms.

Lemma 29 (Constructive Ruzsa’s Modeling Lemma, c.f. [Zha22] Theorem 7.7.3). *Let A be a set of n integers with $|A + A| \leq C|A|$, set $\Delta = \max_{a \in A} |a|$, let $s \geq 2$ be a fixed constant, and set $m = 4C^{2s}n$. There exists an $O(n + \text{polylog}(\Delta))$ -time algorithm that:*

1. with probability at least $1/2$, returns a mapping $\psi : \mathbb{Z} \rightarrow \mathbb{Z}/m\mathbb{Z}$ and a set $A' \subset A$ with $|A'| \geq |A|/s$ such that ψ is a s -Freiman isomorphism from A' to $\psi(A')$, and

2. with probability at most $1/2$, returns ‘failure’.

Proof. Fix any prime $q > \max(sA - sA)$. As $|\max(sA - sA)| \leq s\Delta$, we can find a prime of this size with high probability in time $O_s(\text{polylog}(\Delta))$ by repeatedly guessing and testing primality [Agr04].

For each value $\lambda \in [q]$, let ϕ_λ denote the map $\phi_\lambda : \mathbb{Z} \rightarrow \mathbb{Z}/q\mathbb{Z} \rightarrow \mathbb{Z}/q\mathbb{Z} \rightarrow \mathbb{Z}$ that

1. first maps $a \in \mathbb{Z}$ to $a' = a \pmod{q} \in \mathbb{Z}/q\mathbb{Z}$,
2. computes $a'' = \lambda a'$ in $\mathbb{Z}/q\mathbb{Z}$,
3. then maps a'' back to $[0 : q - 1] \subset \mathbb{Z}$ via the identity map.

Choose $\lambda \in [q - 1]$ uniformly at random. Since q is prime, any element $r \in [q - 1]$ is a generator for $\mathbb{Z}/q\mathbb{Z}$, and thus for any $r \in [q - 1]$, $\phi_\lambda(r)$ is uniformly distributed over $[q - 1]$. Since $q > \max(sA - sA)$, for any nonzero integer $c \in sA - sA$, $c \in [q - 1]$ and thus $\phi_\lambda(c)$ is uniformly random over $[q - 1]$.

Thus for any nonzero $c \in sA - sA$ the probability that $\phi_\lambda(c)$ is divisible by m is less than $2/m$. As $m = 4\mathcal{C}^{2s}n \geq 4|sA - sA|$ by Lemma 28, $|sA - sA| \leq m/4$ and the probability that m evenly divides *any* nonzero element in $sA - sA$ is less than $1/2$ by a union bound. Compute $\phi_\lambda(A)$ in time $O(n)$ and output “failure” if m divides any element in this set. Otherwise, continue.

Let A' be a subset of A such that $|A'| \geq n/s$ and $\text{diam}(\phi_\lambda(A')) \leq q/s$. Note that the existence of A' is guaranteed by the pigeonhole principle. We can compute A' in time $O(n)$ by partitioning $[q]$ into evenly-sized intervals and computing $\phi_\lambda(A)$.

Finally, we define $\psi_\lambda : \mathbb{Z} \rightarrow \mathbb{Z}/m\mathbb{Z}$ by $\psi_\lambda(x) = \phi_\lambda(x) \pmod{m}$ and observe that ψ_λ is a s -isomorphism from A' to $\psi_\lambda(A')$ as m does not divide any nonzero element in $sA - sA$.

This follows from the final two paragraphs of the proof of Theorem 7.7.3 in [Zha22], with the argument unchanged. \square

6.2.2 Bogolyubov’s Lemma in $\mathbb{Z}/m\mathbb{Z}$

Given a relatively large set $B \in \mathbb{Z}/m\mathbb{Z}$, Bogolyubov’s Lemma states that $2B - 2B$ contains a set of points that behaves “like a subspace” in the sense that each point is “close to orthogonal” to a certain set $R \in \mathbb{Z}/m\mathbb{Z}$. Specifically, we employ the concept of a *Bohr set*, defined as

$$\text{Bohr}_m(R, \varepsilon) := \{x \in \mathbb{Z}/m\mathbb{Z} : \|rx/m\|_{\mathbb{R}/\mathbb{Z}} \leq \varepsilon, \text{ for all } r \in R\}.$$

(Recall that the norm $\|\cdot\|_{\mathbb{R}/\mathbb{Z}}$ denotes distance from the nearest integer.) We refer to $|R|$ as the *dimension* and ε as the *width* of the Bohr set.

A Bohr set is analogous to a subspace of codimension $|R|$, in the sense that if we add together several elements of a Bohr set, their sum is still close to a multiple of m when scaled by any $r \in R$. In this sense, we can view Bogolyubov’s lemma as a statement that sets of the form $2B - 2B \in \mathbb{Z}/m\mathbb{Z}$ contain subsets with group-like structure.

Lemma 30 (Constructive Bogolyubov’s lemma for $\mathbb{Z}/m\mathbb{Z}$, c.f. [Zha22] Theorem 7.8.5). *Given $B \subseteq \mathbb{Z}/m\mathbb{Z}$ with $|B| = \alpha m$, we can compute $R \subseteq \mathbb{Z}/m\mathbb{Z}$ of dimension $|R| < 1/\alpha^2$ such that $\text{Bohr}(R, 1/4) \subseteq 2B - 2B$ in time $\tilde{O}(m)$.*

Proof. To make Bogolyubov’s lemma in $\mathbb{Z}/m\mathbb{Z}$ constructive, it suffices to observe that R is defined explicitly as

$$R = \{r \in \mathbb{Z}/m\mathbb{Z} \setminus \{0\} : |\widehat{\mathbb{1}_B}(r)| > \alpha^{3/2}\}.$$

Here $\widehat{\mathbb{1}_B}$ is the finite group Fourier transform of $\mathbb{1}_B$, the membership function of B :

$$\widehat{\mathbb{1}_B}(r) = \frac{1}{m} \sum_{x \in \mathbb{Z}/m\mathbb{Z}} \mathbb{1}_B(x) e^{-\frac{2\pi i r x}{m}}.$$

Computing R directly using the Fast Fourier Transform takes time $O(m \log m)$. □

6.2.3 Finding a GAP in a Bohr Set

The structured nature of the Bohr set is instrumental in constructing a generalized arithmetic progression: in fact, we can show that every Bohr set contains a large GAP. In order to prove this, we need to introduce definitions from the geometry of numbers.

Definition 5 (Successive Minima and Directional Basis). Let $\Lambda \subseteq \mathbb{R}^d$ be a lattice and $T \subseteq \mathbb{R}^d$ be a centrally symmetric convex body.

For $i \in [d]$, the *ith successive minimum* λ_i of T with respect to Λ is the minimum value such that $\Lambda \cap \lambda_i \cdot T$ contains i linearly independent lattice vectors.

A *directional basis* of T with respect to Λ is a basis $\{\vec{b}_1, \vec{b}_2, \dots, \vec{b}_d\}$ of \mathbb{R}^d such that for each $i \in [d]$, $\vec{b}_i \in \lambda_i T$.

In visual terms, we can imagine constructing a directional basis by gradually scaling the convex body T outward from the origin. Every time T engulfs a new lattice vector \vec{v} , we add \vec{v} to our directional basis if and only if \vec{v} is linearly independent from the current set of basis vectors.

Lemma 31 (Constructing a Large GAP in a Bohr Set, c.f. [Zha22] Theorem 7.10.1). *Let m be a prime. Given a set $R \subseteq \mathbb{Z}/m\mathbb{Z}$ of size $|R| = d$ and $\varepsilon \in (0, 1)$, we can compute a proper GAP $P \subseteq \text{Bohr}(R, \varepsilon)$ with dimension at most d and volume at least $(\varepsilon/d)^d m$ in time $\tilde{O}_d(m)$.*

Proof. Let $R = \{r_1, r_2, \dots, r_d\}$ be a subset of $\mathbb{Z}/m\mathbb{Z}$ (recall that m is a prime). We can directly compute the vector $\vec{v} = (\frac{r_1}{m}, \dots, \frac{r_d}{m}) \in \mathbb{R}^d$ to define the lattice

$$\Lambda = \mathbb{Z}^d + \mathbb{Z}\vec{v} \subseteq \mathbb{R}^d.$$

Note that the lattice vectors are not necessarily integral, and we have not yet computed a lattice basis: the set $\{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_d, \vec{v}\}$, where \vec{e}_i denotes the standard basis vector in dimension i , spans the lattice but is not linearly independent.

Let r' be any nonzero element of R . Since $\mathbb{Z}/m\mathbb{Z}$ is a cyclic group of prime order, r' generates $\mathbb{Z}/m\mathbb{Z}$. Thus, because one component of \vec{v} is r'/m , the translations of the integer lattice $\{\mathbb{Z}^d + \gamma\vec{v}\}_{\gamma \in [m]}$ are all disjoint. Since

$$\Lambda = \mathbb{Z}^d + \mathbb{Z}\vec{v} \subseteq \bigcup_{\gamma \in [m]} \mathbb{Z}^d + \gamma\vec{v},$$

we have that Λ is the disjoint union of m translates of the integer lattice. This implies that there are exactly m lattice points of Λ within each translate of the unit cube, and, equivalently, that $\det(\Lambda) = 1/m$.

As a result, we can enumerate the set

$$C := \{ \|\gamma r'/m\|_{\mathbb{R} \setminus \mathbb{Z}} : \gamma \in [m] \} - \{0, 1\}^d,$$

the set of all $2^d \cdot m$ lattice points in the cube $\Lambda \cap [-1, 1]^d$, in time $O(2^d \cdot m) = O_d(m)$.

Next, we sort the set C according to the L_∞ metric, which takes time $\tilde{O}_d(m)$. This coincides with our definition of the successive minima of a cube centered on the origin with respect to Λ : if λ_i is the i th successive minima of $[-\varepsilon, \varepsilon]^d$ with respect to Λ , then the i th

directional basis vector satisfies $\|\vec{b}_i\|_\infty \leq \lambda_i[-\varepsilon, \varepsilon]^d$.

Construct the successive minima $\lambda_1, \dots, \lambda_d$ and the directional basis $\vec{b}_1, \dots, \vec{b}_d$ of $[-\varepsilon, \varepsilon]^d$ with respect to Λ by greedily adding independent lattice vectors to our basis from short to long according to the L_∞ metric. Checking whether each subsequent lattice vector is independent from the previous set takes time $O_d(1)$ using Gaussian elimination. Because $[-1, 1]^d$ contains d linearly independent lattice vectors (consider the standard basis), our directional basis is guaranteed to be contained in $\Lambda \cap [-1, 1]^d = C$.

To complete the construction of the GAP in [Zha22, Theorem 7.10.1], we observe that the proper GAP P is defined explicitly in terms of the directional basis of $[-\varepsilon, \varepsilon]^d$ with respect to Λ that we have just constructed. Specifically, we have

$$P = \{\ell_1 x_1 + \dots + \ell_d x_d \mid \forall i \in [d], \ell_i \in [L_i]\},$$

where each x_i is the unique element in $[0 : m-1]$ such that $\vec{b}_i \in x_i \vec{v} + \mathbb{Z}^d$ and $L_i := \lceil 1/(\lambda_i d) \rceil$. Each L_i can be computed directly, and each x_i can be computed in time $O(m)$. \square

6.2.4 Ruzsa's Covering Lemma

Ruzsa's covering lemma states that if the sumset $|Y + Z|$ is small relative to $|Y|$, it is possible to cover Z with a small number of translates of $Y - Y$. A rough intuition for this result is that it is a statement about the ‘‘conservation of additive structure’’: if Y and Z have ‘‘common additive structure’’ (captured by the condition that $|Y + Z| \leq \mathcal{C}|Y|$), then Z is ‘‘similar’’ to $Y - Y$ (in the sense that Z is covered by few translates of $Y - Y$).

The fact that Ruzsa's covering lemma can be made efficiently constructive was previously observed by Abboud, Bringmann, and Fischer [ABF23]:

Lemma 32 (Constructive Ruzsa’s Covering Lemma, [ABF23] Lemma 4.7). *Let Y, Z be nonempty finite subsets of an abelian group. If $|Y + Z| \leq C|Y|$, then there exists a subset $X \subseteq Z$ with $|X| \leq C$ and $Z \subseteq Y - Y + X$. Moreover, X can be computed in time $\tilde{O}\left(\frac{|Y-Y+Z| \cdot |Y+Z|}{|Y|}\right) = \tilde{O}(C|Y - Y + Z|)$.*

6.2.5 Proof of Theorem 15: The Constructive Freiman’s Theorem

Proof. Combining the ingredients from the previous subsections allows us to prove Theorem 15. Let A be a finite integer set with $|A + A| \leq C|A| = Cn$. By Lemma 28, $|8A - 8A| \leq C^{16}|A|$.

Choose a prime $m = O_C(n)$ satisfying $4C^{16}n < m < 16C^{16}n$, which can be accomplished in time $O_C(\text{polylog}(n))$ with high probability by guessing and testing primality [Agr04]. Then, apply Lemma 29 with $s = 8$ to compute a set $A' \subseteq A$ with $|A'| \geq |A|/8$ and a mapping ψ such that ψ is a Freiman 8-isomorphism from A' to $B := \psi(A') \subseteq \mathbb{Z}/m\mathbb{Z}$ with probability at least $1/2$ in time $\tilde{O}(n)$. We can increase the success probability of this step by repetition: for any integer constant $\gamma > 0$, running the algorithm $\gamma \log(n)$ times lowers the failure probability to $n^{-\gamma}$.

Apply Lemma 30 to B with

$$\alpha = \frac{|B|}{m} = \frac{|A'|}{m} \geq \frac{|A|}{8m} = \frac{1}{O_C(1)}.$$

This gives us $R \subseteq \mathbb{Z}/m\mathbb{Z}$ of size $1/\alpha^2 = O_C(1)$ such that $\text{Bohr}(R, 1/4) \subseteq 2B - 2B$ in time $\tilde{O}(m) = \tilde{O}(n)$. Then, apply Lemma 31 to R to compute the proper GAP $P \subset \text{Bohr}(R, \varepsilon) \subseteq 2B - 2B$ with dimension at most $|R| = O_C(1)$ and volume at least $(1/4|R|)^{|R|}m = m/O_C(1)$.

Following the proof of Theorem 7.11.1 (Freiman’s Theorem) in [Zha22], we have that

$Q := \psi^{-1}(P)$ is a GAP of the same dimension and volume satisfying

$$Q \subseteq 2A' - 2A' \subseteq 2A - 2A. \quad (6.1)$$

Thus

$$|Q + A| \leq |2A - 2A + A| = |3A - 2A| \leq \mathcal{C}^5 |A| = O_{\mathcal{C}}(1) \cdot |Q|,$$

where the second inequality uses [Lemma 28](#). Using the fact that $|Q + A| = O_{\mathcal{C}}(1) \cdot |Q|$, we apply [Lemma 32](#) to Q and A to get a set X of size $|X| = O_{\mathcal{C}}(1)$ satisfying $A \subseteq Q - Q + X$ in time

$$\tilde{O}(\mathcal{C}|Q - Q + A|) = \tilde{O}(\mathcal{C}|2A - 2A - (2A - 2A) + A|) = \tilde{O}(\mathcal{C}|5A - 4A|) = \tilde{O}(\mathcal{C}^9|A|),$$

where the final equality uses [Lemma 28](#). We conclude with the observation that $Q - Q + X$ is a GAP of dimension $O_{\mathcal{C}}(1)$ and volume $O_{\mathcal{C}}(1) \cdot |Q| = O_{\mathcal{C}}(1) \cdot |A|$, containing A . Note that each step in the proof takes $\tilde{O}_{\mathcal{C}}(n)$ time. \square

6.2.6 Bounding GAP Coefficients

The following observation further simplifies [Theorem 15](#).

Observation 1. In the GAP P guaranteed by [Theorem 15](#), without loss of generality we can assume

$$L_i \leq n^{2/d(\mathcal{C})}$$

for all $i \in [d(\mathcal{C})]$, where $d(\mathcal{C})$ denotes the dimension of P .

Proof. Suppose $L_i > n^{1/d(\mathcal{C})}$ for some $i \in [d(\mathcal{C})]$, and let $\alpha_i := \alpha_i(\mathcal{C})$ be the solution to

$L_i \leq n^{\alpha_i/d(\mathcal{C})}$. (Note that $\alpha_i = O_{\mathcal{C}}(1)$, as $|A| = O_{\mathcal{C}}(n)$.)

Let $\hat{\alpha}_i := \alpha_i - \lfloor \alpha_i \rfloor$ denote the decimal part of α_i , and observe that

$$\{y_i \ell_i : \ell_i \in [L_i]\} \subseteq \tag{6.2}$$

$$\{y_{i,1} \ell_{i,1} + \dots + y_{i,\lfloor \alpha_i \rfloor} \ell_{i,\lfloor \alpha_i \rfloor} + y_{i,\lceil \alpha_i \rceil} \ell_{i,\lceil \alpha_i \rceil} : \forall j \in [\lfloor \alpha_i \rfloor], \ell_{i,j} \in [\lceil L_i^{1/d(\mathcal{C})} \rceil], \ell_{i,\lceil \alpha_i \rceil} \in [n^{\hat{\alpha}_i/d(\mathcal{C})}]\}; \tag{6.3}$$

that is, we can replace one dimension of our arithmetic progression with $\lceil \alpha_i \rceil$ new dimensions, each bounded by $n^{1/d(\mathcal{C})}$. As

$$\text{vol}(P) = \prod_{i \in [d]} L_i = O_{\mathcal{C}}(n)$$

by [Theorem 15](#), performing this operation for each $L_i > n^{1/d(\mathcal{C})}$ results in a new gap P' with dimension $d'(\mathcal{C}) \leq 2d(\mathcal{C})$ and volume $O_{\mathcal{C}}(n)$. □

6.3 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 $A\vec{x}$; 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. Formally, our definition of an integer program with constant doubling is as follows:

Problem 10: \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 $\vec{b} \in \mathbb{Z}^m$, such that the column set $\mathcal{A} := \mathcal{A}(A)$ satisfies $|\mathcal{A} + \mathcal{A}| \leq \mathcal{C}|\mathcal{A}|$ for a constant \mathcal{C} independent of m and n .

Out: A vector $\vec{x} \in \mathbb{Z}_{\geq 0}^n$ such that $A\vec{x} = \vec{b}$, or ‘No’ if no solution exists.

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 $\vec{x} \in \{0, 1\}^n$ yields **\mathcal{C} -Binary ILP Feasibility**.

Remark 3. Bounded ILPs with n variables and $|\ell_i|, |u_i| = O(\text{poly}(n))$ for $i \in [n]$ can be converted into equivalent binary ILPs with $\text{poly}(n)$ variables by duplicating the columns of A .

6.3.1 \mathcal{C} -Binary ILP Feasibility

Given a constraint matrix with constant doubling, Freiman’s Theorem bounds the number of possible values for $A\vec{x}$ 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.²

²The constructive Freiman’s theorem will be required later, specifically in [Lemma 34](#) and [Theorem 18](#). The current result emphasizes the usefulness of parameterization in the doubling constant.

Theorem 16. *An instance \mathcal{I} of \mathcal{C} -Binary ILP Feasibility on n variables can be solved in time $n^{O_{\mathcal{C}}(1)} \cdot \text{poly}(|\mathcal{I}|)$.*

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

Let $\S(\mathcal{A})$ 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 $A\vec{x}$ for any $\vec{x} \in \{0, 1\}^n$. Our first goal is to bound $|\S(\mathcal{A})|$.

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

$$\mathcal{A} \subseteq P = \{\vec{x}_1\ell_1 + \vec{x}_2\ell_2 + \cdots + \vec{x}_{d(\mathcal{C})}\ell_{d(\mathcal{C})} : \forall i, \ell_i \in [L_i]\},$$

where $\vec{x}_i \in \mathbb{Z}^m$ 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-free³ commutative groups ([Ruz09], Theorem 8.1).

Thus $\S(\mathcal{A})$ is contained in the GAP

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

which implies

$$|\S(\mathcal{A})| \leq |P'| \leq n^{d(\mathcal{C})}|P| = n^{d(\mathcal{C})}v(\mathcal{C})n = n^{O_{\mathcal{C}}(1)}. \quad (6.4)$$

To complete the proof, we claim that we can enumerate $\S(\mathcal{A})$ efficiently via dynamic programming, using the following procedure: Initially, we set $\S(\mathcal{A})_1 = A[1, \cdot]$. Then, we iterate $i = 2, 3, \dots, n$. In the i th iteration, we construct the sorted list $\S(\mathcal{A})_i$, defined as:

³That is, groups in which only the identity element has finite order.

$$\mathfrak{S}(\mathcal{A})_i := \mathfrak{S}(\mathcal{A})_{i-1} \cup \{a + A[i, \cdot] \mid a \in \mathfrak{S}(\mathcal{A})_{i-1}\}.$$

Finally, we return list $\mathfrak{S}(\mathcal{A}) = \mathfrak{S}(\mathcal{A})_n$. Correctness of the above algorithm follows immediately by construction. For the running time, observe that $\mathfrak{S}(\mathcal{A})_i$ can be constructed in $O(|\mathfrak{S}(\mathcal{A})_i|)$ time. Because each of the n iterations of the subprocedure takes time $O(|\mathfrak{S}(\mathcal{A})_i|) = O(|\mathfrak{S}(\mathcal{A})|)$, the total runtime is, by (6.4), at most $n \cdot O(|\mathfrak{S}(\mathcal{A})|) = n^{O_c(1)}$. \square

6.3.2 \mathcal{C} -Bounded ILP Feasibility

In general, ILPs with polynomially bounded variables can be converted to ILPs with binary variables (see Remark 3); 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 10. *An instance \mathcal{I} of \mathcal{C} -Bounded ILP Feasibility such that $\ell_i \leq x_i \leq u_i$ and $|\ell_i|, |u_i| = \text{poly}(n)$ for $i \in [n]$ can be solved in time $n^{O_c(1)} \cdot \text{poly}(|\mathcal{I}|)$.*

Proof. Modify the proof of Theorem 16 by considering the list $\mathfrak{S}(\mathcal{A})'$ of all possible outputs $A\vec{x}$ 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 $|\mathfrak{S}(\mathcal{A})'|$.

Observe that $\mathfrak{S}(\mathcal{A})'$ 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 $|\mathfrak{S}(\mathcal{A})'| = n^{O_c(1)}$. We can enumerate $\mathfrak{S}(\mathcal{A})'$ by modifying the procedure given above so that Step 2 merges a polynomial number of lists, one for each variable assignment. \square

6.4 Subset Sum with Constant Doubling

We now consider the useful applications of parameterization in the doubling constant to Subset Sum. \mathcal{C} -Subset Sum is equivalent to \mathcal{C} -Binary ILP with a single constraint. As a result, [Theorem 16](#) yields the following corollary for Subset Sum with n variables:

Corollary 11 (\mathcal{C} -Subset Sum is in XP). *\mathcal{C} -Subset Sum can be solved in time $n^{O_c(1)}$.*

At this point, it is natural to wonder whether \mathcal{C} -Subset Sum can be solved in time $O_c(1) \cdot n^{O(1)}$: that is, whether Subset Sum is in FPT with 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.

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

Recent generalizations of Integer Programming consider the problem of optimizing the value $g(A\vec{x})$ in place of $A\vec{x}$, where $g : \mathbb{R}^m \rightarrow \mathbb{R}$ is a low-dimensional objective function [[DLRV23](#)]. 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 \vec{x} satisfying $A\vec{x} \cdot \vec{s} = t$ for a specific “step vector” \vec{s} . Formally, our problem is as follows:

Problem 11: Hyperplane-Constrained Binary ILP (HBILP) Feasibility

In: An integer matrix $A \in \mathbb{Z}^{m \times n}$, a step vector $\vec{s} \in \mathbb{Z}^m$, and a target integer t . We let $\Delta := \|A\|_\infty$, the magnitude of A 's largest entry.

Out: A vector $\vec{x} \in \{0, 1\}^n$ such that $A\vec{x} \cdot \vec{s} = t$, or 'NO' if no solution exists.

The reduction from \mathcal{C} -Subset Sum to HBILP Feasibility (Lemma 34) is straightforward but relies crucially on our constructive Freiman's Theorem.

Lemma 33. *For any fixed instance (Z, t) of \mathcal{C} -Subset Sum, there exists a HBILP Feasibility instance given by $A \in \mathbb{Z}^{d(\mathcal{C}) \times n}$, $\vec{s} \in \mathbb{Z}^{d(\mathcal{C})}$, and t for some function $d(\mathcal{C})$ such that the vector $\vec{x} \in \{0, 1\}^n$ satisfies*

$$A\vec{x} = t \quad \text{if and only if} \quad \sum_{i: x_i=1} z_i = t.$$

Moreover, $\Delta := \|A\|_\infty \leq n^{2/d(\mathcal{C})}$, and the reduction can be computed in time $\tilde{O}_{\mathcal{C}}(n)$ 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 $|Z + Z| \leq \mathcal{C}|Z|$ and an integer target t . Apply Theorem 15, which fails with probability $n^{-\gamma}$ and otherwise produces a GAP

$$P = \{y_1 \ell_1 + y_2 \ell_2 + \dots + y_d \ell_d : \forall i, \ell_i \in [L_i]\}$$

of dimension $d := d(\mathcal{C})$ and volume $v(\mathcal{C})n$ containing Z .

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

$$y_1 v_1 + y_2 v_2 + \dots + y_d v_d = z_i \text{ and } \forall i \in [d], v_i \in [L_i].$$

We can think of $\vec{v}(z_i)$ as the d -dimensional ‘‘GAP coordinates’’ of the input element z_i . $\vec{v}(z_i)$ 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, $\vec{v}(z_i)$ is not guaranteed to be unique.

To complete the reduction, set

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

set $\vec{s} := (y_1, y_2, \dots, y_d)$ and preserve the same target t . Note that $\|A\|_{\infty} \leq n^{2/d(\mathcal{C})}$ without loss of generality by [Observation 1](#).

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

$$A\vec{x} \cdot \vec{s} = \sum_{i: x_i=1} z_i. \tag{6.5}$$

To see this, observe that

$$\begin{aligned} A\vec{x} \cdot \vec{s} &= \sum_{i \in [d]} y_i \sum_{x_j=1} A[i, j] \\ &= \sum_{x_j=1} y_1 A[1, j] + y_2 A[2, j] + \dots + y_d A[d, j] \\ &= \sum_{x_j=1} \vec{y} \cdot \vec{v}(z_j) \\ &= \sum_{j: x_j=1} z_j. \end{aligned}$$

Thus $A\vec{x} \cdot \vec{s} = t$ 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. \square

6.4.2 Equivalence Between HBILP Feasibility and Subset Sum

Theorem 17. *\mathcal{C} -Subset Sum can be solved in time $O_{\mathcal{C}}(\text{poly}(n))$ if and only if Hyperplane-Constrained Binary ILP (HBILP) can be solved in time $\Delta^{O(m)} \cdot O_m(\text{poly}(|\mathcal{I}|))$, where $|\mathcal{I}|$ is the size of the instance.*

Theorem 17 follows immediately from the next two lemmas, which show reductions in both directions. The first is a consequence of the reduction in [Section 6.4.1](#):

Lemma 34. *If HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(|\mathcal{I}|)$, then \mathcal{C} -Subset Sum can be solved in time $O_{\mathcal{C}}(\text{poly}(n))$ with success probability $1 - n^{-\gamma}$ for an arbitrarily large constant $\gamma > 0$.*

Proof. Recall that we can preprocess an instance of Subset Sum and produce an equivalent one such that all integers are bounded by $2^{O(n)}$ with very high probability (see [Lemma 7](#)). Next, we use the reduction given in [Lemma 33](#), which takes time $\tilde{O}_{\mathcal{C}}(n)$ and succeeds with probability $1 - n^{-\gamma}$, and solve the resulting HBILP instance in time

$$\Delta^{O(m)} \cdot O_m(|\mathcal{I}|) = (n^{2/d(\mathcal{C})})^{O(d(\mathcal{C}))} \cdot O_{\mathcal{C}}(\text{poly}(n)) = O_{\mathcal{C}}(\text{poly}(n)). \quad \square$$

Lemma 35. *If \mathcal{C} -Subset Sum admits an $O_{\mathcal{C}}(\text{poly}(n))$ -time algorithm, HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(\text{poly}(n))$.*

Proof. Fix an instance of HBILP Feasibility given by the matrix $A \in \mathbb{Z}^{m \times n}$, the vector $\vec{s} \in \mathbb{Z}^m$, 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', \vec{s}', t' with the property that every column $A'[\cdot, j]$ of A' has a unique dot product $A'[\cdot, j] \cdot \vec{s}'$. We then reduce A', \vec{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. Moreover, by [Observation 2](#), proved below, we can assume that each entry of A is non-negative and that any solution vector \vec{x} has fixed support exactly q for some $q = \Theta(n)$.

Step 1: Self-reduction. In order to construct the instance A', \vec{s}', t' , define $M := nm\Delta\|\vec{s}\|_\infty + 1$, which satisfies

$$M > A\vec{x} \cdot \vec{s} \tag{6.6}$$

for any $\vec{x} \in \{0, 1\}^n$ 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 $\Delta \cdot J_{k \times n} - R$.

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 \tilde{A} is distinct because each column in R is distinct and no column in A is all 0's.

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

reduction. Pictorially, we can represent A' and \vec{s}' as follows:

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

Claim 13. *For every distinct pair of indices $i, j \in [2n]$, $A'[\cdot, i] \cdot \vec{s}' \neq A'[\cdot, j] \cdot \vec{s}'$.*

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' :

$$A'[\cdot, i] \cdot \vec{s}' = A[\cdot, i] \cdot \vec{s} + R[\cdot, i] \cdot \vec{v}.$$

First, observe that $R[\cdot, i] \cdot \vec{v}$ is distinct for every $i \in [n]$ by construction, due to the fact that each component of $R[\cdot, i]$ is less than Δ , and the components of v increase by factors of Δ .

Second, because

$$0 < A[\cdot, i] \cdot \vec{s} \leq M,$$

the $A[\cdot, i] \cdot \vec{s}$ term of the dot product $A'[\cdot, i] \cdot \vec{s}'$ is not large enough to interfere with the $R[\cdot, i] \cdot \vec{v}$ term, and thus the first n columns of A' have distinct dot products with \vec{s}' .

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

Claim 14. *The ILP instance (A', \vec{s}', t') has a solution if and only if the instance (A, \vec{s}, t) has a solution (and the solution to A, \vec{s}, t can be recovered efficiently from the solution of A', \vec{s}', t').*

Proof: Suppose \vec{x} satisfies $A\vec{x} \cdot \vec{s} = t$. Recall that \vec{x} has support exactly q by [Observation 2](#) without loss of generality. Thus the vector \vec{x}' created by concatenating two copies of x satisfies

$$A'\vec{x}' \cdot \vec{s}' = t + q\Delta\|\vec{v}\|_1 = t'.$$

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

$$A(y'_1, y'_2, \dots, y'_n) \cdot \vec{s} = t$$

by construction. This is because the $[R \mid \bar{R}]$ submatrix of A' can contribute to t' only in multiples of M , so because $A(y'_1, y'_2, \dots, y'_n) \cdot \vec{s} < M$ by [\(6.6\)](#), this product must evaluate to t . ■

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

$$\vec{z} := (\vec{s}' \cdot A'[\cdot, 1], \vec{s}' \cdot A'[\cdot, 2], \dots, \vec{s}' \cdot A'[\cdot, 2n]) \tag{6.7}$$

and let

$$Z = \{z_1, z_2, \dots, z_{2n}\}$$

denote the set containing of the components of \vec{z} . (Note that Z is a proper set and contains no duplicates, by [Claim 13](#).) We proceed to consider Z, t as an instance of Subset Sum.

Because $A'\vec{x} \cdot \vec{s}' = t'$ if and only if $\vec{x} \cdot \vec{z} = t'$ by construction [\(6.7\)](#), 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 $\vec{x} \in \{0, 1\}^{2n}$ such that $A'\vec{x} \cdot \vec{s}' = t'$, which can be used to recover a solution for the original instance A, \vec{s}, t by [Claim 14](#). It remains to show that an $O_{\mathcal{C}}(\text{poly}(n))$ 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 $z_j = \vec{s} \cdot A'[\cdot, j]$ for all $j \in [n]$, and thus Z is a subset of the GAP

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

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

$$|Z| = \Omega(|Y|) \tag{6.8}$$

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

$$\begin{aligned} |Z + Z| &\leq |Y + Y| \\ &\leq 2^{m+1} \cdot |Y| \\ &= O_m(1) \cdot |Z|, \end{aligned}$$

where the first line follows from the fact that $Z \subseteq Y$, the second line follows from the fact that $|Y|$ has dimension $m + 1$, and the third follows from (6.8).

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 $O_C(\text{poly}(n))$ solves Z, t in time $O_m(\text{poly}(\Delta^m \cdot n)) = \Delta^{O(m)} \cdot O_m(\text{poly}(n))$ as claimed. \square

6.4.3 Non-negativity for HBILP Feasibility

Observation 2. Let \mathcal{I} be an instance of HBILP feasibility given by the constraint matrix $A \in \mathbb{Z}^{m \times n}$, the step vector $\vec{s} \in \mathbb{Z}^m$, 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 $q = \Theta(n)$.

Proof. Given an HBILP feasibility instance (A, \vec{s}, t) , we create a new HBILP feasibility instance $(\tilde{A}, \tilde{\vec{s}}, \tilde{t})$ as follows. Recall that $J_{m \times n}$ denotes the m by n matrix of 1's and add $\Delta \cdot J_{m \times n}$ to A . Then append the matrix $\Delta \cdot J_{m \times n}$ to the right-hand side A . Finally, add a row of 1's to the bottom of the matrix.

Define

$$M := \|\vec{s}\|_\infty \cdot 3n\Delta m,$$

and note that, by construction, we have

$$(A + 2\Delta J_{m \times n})\vec{x} \cdot \vec{s} \leq 3n\Delta J_{m \times 1} \cdot \vec{s} < M. \tag{6.9}$$

Create $\tilde{\vec{s}} \in \mathbb{Z}^{m+1}$ by appending M to \vec{s} , and set $\tilde{t} = t + n\Delta\|\vec{s}\|_1 + nM$. Written as block matrices, we have:

$$\tilde{A} := \left(\begin{array}{c|c} A + \Delta \cdot J_{m \times n} & \Delta \cdot J_{m \times n} \\ \hline J_{1 \times n} & J_{1 \times n} \end{array} \right) \quad \tilde{\vec{s}} := \left(\begin{array}{c} \vec{s} \\ \hline M \end{array} \right)$$

Observe that every entry in \tilde{A} is positive and that the maximum entry in \tilde{A} is at most $2\Delta = O(\Delta)$. Because the top m rows of \tilde{A} contribute a total value less than M to the dot product $\tilde{A}\tilde{x} \cdot \tilde{s}$ by (6.9), any solution to $(\tilde{A}, \tilde{s}, \tilde{t})$ must have support exactly n so that the resulting dot product contains the term nM .

It remains to prove correctness:

Claim 15. *A vector $\vec{y} \in \{0, 1\}^{2n}$ satisfies $\tilde{A}\vec{y} \cdot \tilde{s} = \tilde{t}$ if and only if $\text{supp}(\vec{y}) = n$ and the first half of \vec{y} , the vector $\vec{y}' = (y_1, y_2, \dots, y_n)$, satisfies $A\vec{y}' \cdot \vec{s} = t$.*

Proof: Suppose some vector $\vec{y}' \in \{0, 1\}^n$ satisfies $A\vec{y}' \cdot \vec{s} = t$. Then the vector \vec{y} created by adding an arbitrary n -bit string with support $n - \text{supp}(\vec{y}')$ satisfies

$$\tilde{A}\vec{x}' \cdot \tilde{s} = \tilde{t}$$

and is a valid solution to $(\tilde{A}, \tilde{s}, \tilde{t})$.

Now suppose some vector $\vec{y} \in \{0, 1\}^{2n}$ satisfies $\tilde{A}\vec{y} \cdot \tilde{s} = \tilde{t}$. As previously noted, we must have $\text{supp}(\vec{y}) = n$ to create the nM term in the product $\tilde{A}\vec{y} \cdot \tilde{s}$. The additional Δ factors added to every component in each of the first m rows of \tilde{A} create the $n\Delta\|\vec{s}\|_1$ term in the product. If we remove these two terms, the remainder of the equation $\tilde{A}\vec{y} \cdot \tilde{s} = \tilde{t}$ is $A(y'_1, y'_2, \dots, y'_n) \cdot \vec{s} = t$. ■

This concludes the proof of [Observation 2](#). □

Reduction from BILP Feasibility to HBILP Feasibility

An FPT algorithm for \mathcal{C} -Subset Sum further implies a $\Delta^{O(m)} \cdot O_m(\text{poly}(n))$ 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 12. *If \mathcal{C} -Subset Sum can be solved in $O_{\mathcal{C}}(\text{poly}(n))$, then Bounded ILPs defined by $A \in \mathbb{Z}^{m \times n}$, $\vec{b} \in \mathbb{Z}^m$ with $\Delta := \|A\|_{\infty}$ and each variable x_i bounded by $\text{poly}(n)$ can be solved in time $\Delta^{O(m)} \cdot O_m(\text{poly}(n))$.*

Corollary 12 is a straightforward corollary of Lemma 36, which reduces ILP Feasibility with binary variables to HBILP feasibility, and the fact that ILPs with polynomially bounded variables can be reduced to binary ILPs (Remark 3).

Lemma 36. *If HBILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(|\mathcal{I}|)$, Binary ILP Feasibility can be solved in time $\Delta^{O(m)} \cdot O_m(|\mathcal{I}|)$.*

Proof. Fix an instance $A \in \mathbb{Z}^{m \times n}$, $\vec{b} \in \mathbb{Z}^n$ of Binary ILP Feasibility with $\Delta := \|A\|_{\infty}$. By Observation 3, proved below, we can assume without loss of generality that the entries of A are non-negative.

Define $q := q(n, \Delta) = n\Delta + 1$ and create a new instance (A, \vec{s}, t) of HBILP feasibility by setting

$$\begin{aligned} \vec{s} &:= (q^0, q^1, \dots, q^{m-1}) \text{ and} \\ t &:= \vec{b} \cdot \vec{s} = \vec{b}_1 \cdot q^0 + \vec{b}_2 \cdot q^1 + \dots + \vec{b}_m \cdot q^{m-1}, \end{aligned}$$

effectively using t to store m registers of $\log_2(q) = \log_2(n\Delta + 1)$ bits each.

We claim that $\vec{x} \in \{0, 1\}^n$ solves A, \vec{b} if and only if it solves (A, \vec{s}, t) . If $A\vec{x} = \vec{b}$, $A\vec{x} \cdot \vec{s} = t$ follows immediately from the definition of t .

Now suppose $A\vec{x} \cdot \vec{s} = t$. Because $q > A[i, \cdot]\vec{x}$ for any row $i \in [m]$ by construction, the only way to achieve t is if $A[i, \cdot]\vec{x} = b_i$ for each $i \in [m]$. \square

Observation 3. Let \mathcal{I} be an instance of ILP feasibility with binary variables given by the constraint matrix $A \in \mathbb{Z}^{m \times n}$ and the target vector $\vec{b} \in \mathbb{Z}^m$. Without loss of generality, we

can assume that entries of A are non-negative and that every solution has fixed support q for some $q = \Theta(n)$.

Proof. Construct a new constraint matrix \tilde{A} as follows: Recall that $J_{m \times n}$ denotes the m by n matrix of 1's, and add $\Delta \cdot J_{m \times n}$ to the matrix A . Then we append an additional n columns to A , where each column consists only of Δ entries only. Finally, we append a row of 1's.

To create $\tilde{\vec{b}}$, add $n\Delta \cdot J_{m \times 1}$ to \vec{b} and append a single entry with the value n .

$$\tilde{A} := \left(\begin{array}{c|c} A + \Delta \cdot J_{m \times n} & \Delta \cdot J_{m \times n} \\ \hline J_{1 \times n} & J_{1 \times n} \end{array} \right) \quad \tilde{\vec{b}} := \left(\begin{array}{c} \vec{b} + n\Delta \cdot J_{m \times 1} \\ \hline n \end{array} \right)$$

Observe that every entry in \tilde{A} is positive and that the maximum entry in \tilde{A} is at most $2\Delta = O(\Delta)$. For correctness, note that the last row ensures that any solution to $\tilde{A}\vec{x} = \tilde{\vec{b}}$ with $\vec{x} \in \{0, 1\}^{2n}$ has support exactly n . This implies that the additional Δ factors added to every component in each of the first m rows add a total of $n\Delta$ to each component of $A\vec{x}$. Thus $\tilde{A}\vec{x} = \tilde{\vec{b}}$ if and only if $A\vec{x} = \vec{b}$. \square

6.5 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 following lemma, which proves that feasible ILP instances with bounded entries admit solutions with small support, follows a standard pattern of argument (see, e.g., [Gom69] Theorem 1, adapted in [Kle22] Corollary 2.1).

Lemma 37 (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 $\vec{b} \in \mathbb{Z}^m$ corresponding to at least one solution $\vec{x} \in \mathbb{Z}_{\geq 0}^n$ with $A\vec{x} = \vec{b}$, there exists a small-support solution $\vec{y} \in \mathbb{Z}_{\geq 0}^n$ satisfying*

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

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

Claim 16. *Let $A \in \mathbb{Z}^{m \times n}$ with $\Delta = \|A\|_\infty$, and let $\vec{y} \in \mathbb{Z}_{\geq 0}^n$ be the lexicographically minimal vector such that $A\vec{y} = \vec{b}$ for some $\vec{b} \in \mathbb{Z}^m$. Then $|\text{supp}(\vec{y})| \leq m \log_2(2n\Delta + 1)$.*

Proof: Assume for contradiction that

$$2^{|\text{supp}(\vec{y})|} > (2n\Delta + 1)^m.$$

Because $A\vec{x} \leq (2n\Delta + 1)^m$ for any $\vec{x} \in \{0, 1\}^n$, there must exist two different vectors $\vec{v}, \vec{w} \in \{0, 1\}^n$ such that (i) $\text{supp}(\vec{v}), \text{supp}(\vec{w}) \subseteq \text{supp}(\vec{y})$, and (ii) $A\vec{v} = A\vec{w}$, by the pigeonhole principle.

Let $\vec{y}_1 = \vec{y} - \vec{w} + \vec{v}$ and $\vec{y}_2 = \vec{y} + \vec{w} - \vec{v}$. Observe that $A\vec{y}_1 = A\vec{y}_2$ and $\vec{y}_1, \vec{y}_2 \in \mathbb{Z}_{\geq 0}^n$ because $\text{supp}(\vec{v}), \text{supp}(\vec{w}) \subseteq \text{supp}(\vec{y})$. Moreover, because $\vec{v} \neq \vec{w}$ we have that one of \vec{y}_1 or \vec{y}_2 is lexicographically smaller than \vec{y} , which contradicts the assumption that \vec{y} is lexicographically

minimal. ■

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

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

where $M = 4n\Delta$. Note that this can be solved in $(n\Delta)^{O(m)}$ time by [EW19, Theorem 2.3] for each \vec{b} . Hence, the set \mathcal{X} can be constructed in the claimed time. Finally, it remains to show that for any feasible \vec{b} , there exists a solution \vec{y} with small support in \mathcal{X} .

Claim 17. *Let $\vec{b} \in \mathbb{Z}^m$ be any vector for which there exists $\vec{x} \in \mathbb{Z}_{\geq 0}^n$ with $A\vec{x} = \vec{b}$. Then there also exists $\vec{y} \in \mathbb{Z}_{\geq 0}^n$ such that $A\vec{y} = \vec{b}$ and $\text{supp}(\vec{y}) \in \mathcal{X}$.*

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

Hence it remains to show that \hat{z} is the lexicographically minimal vector for which $A\hat{z} = \hat{b}$. Assume for contradiction that there exists $\hat{y} \in \mathbb{Z}_{\geq 0}^n$ such that $A\hat{y} = \hat{b}$ and \hat{y} is lexicographically smaller than \hat{z} . Consider a vector $\vec{y} = \vec{z} - \hat{z} + \hat{y}$. Note, that $\text{supp}(\hat{z}) \subseteq \text{supp}(\vec{z})$ so $\vec{y} \in \mathbb{Z}_{\geq 0}^n$. Clearly $A\vec{y} = A\vec{z} = \vec{b}$. Moreover, because \hat{y} is lexicographically smaller than \hat{z} , it follows that \vec{y} is lexicographically smaller than \vec{z} . This contradicts our assumption that \vec{z} is lexicographically minimal. ■

Thus the set \mathcal{X} satisfies the property stated in Lemma 37, concluding the proof. □

With Lemma 37 in hand, let us present our algorithm for \mathcal{C} -Unbounded Subset Sum.

Theorem 18 (Near-XP algorithm for \mathcal{C} -Unbounded Subset Sum). *\mathcal{C} -Unbounded Subset Sum can be solved in time $n^{O_c(1)}$ if an ILP instance \mathcal{I} on v variables can be solved in time $2^{O(v)} \text{poly}(|\mathcal{I}|)$.*

Using the current best algorithm [RR23], \mathcal{C} -Unbounded Subset Sum can be solved in $n^{O_c(1) \log \log \log(n)}$ time.

Proof. Following the steps of our reduction from \mathcal{C} -Subset Sum to HBILP feasibility (Lemma 33), we can use the constructive Freiman’s theorem (Theorem 14) 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 $\vec{\ell}$, and target t .

We then use Lemma 37 to construct a set \mathcal{X} of candidate supports in $(n\Delta)^{O(m)} = n^{O_c(1)}$ time. For each support vector $\vec{x}^* \in \mathcal{X}$, we reduce the ILP to variables in \vec{x}^* . This gives us a program with $|\vec{x}^*| = O(m \log_2(n\Delta)) = O_c(\log(n))$ variables. Now, we encode this problem as the ILP

$$\left\{ \vec{x} \in \mathbb{Z}_{\geq 0}^n \mid \sum_{j=1}^d \ell_j \sum_{i \in \vec{x}^*} A[i, j] x_i = t \right\}.$$

Observe that this is equivalent to an instance of Unbounded Subset Sum with $O_c(\log(n))$ 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 $n^{O_c(1)}$ time algorithm for \mathcal{C} -Unbounded Subset Sum. Using the best known algorithm for unbounded ILPs, which runs in time $(\log v)^{O(v)}$ [RR23], we get an $n^{O_c(1) \log \log \log(n)}$ -time algorithm. □

6.6 k -SUM with Constant Doubling

Our next contribution concerns (\mathcal{C}, k) -SUM. We prove [Theorem 19](#) and observe that the same approach gives an algorithm for 4-SUM that is tight up to subpolynomial factors, assuming the k -SUM conjecture.

We note that [\[ABF23\]](#) and [\[JX23\]](#) 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 small doubling.

Theorem 19. *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 $\{x_1, \dots, x_k\} \subseteq X$ such that $x_1 + \dots + x_k = t$ in deterministic time $\tilde{O}(\mathcal{C}^{\lceil k/2 \rceil} \cdot 2^{O(k)} \cdot n)$.*

Proof. Let X be an integer set of size n , and let $\{x_1, \dots, x_k\} \subseteq X$ denote a set of k integers that sum to t . We make use of the sparse convolution algorithm of Bringmann et al. [\[BFN22\]](#).

Lemma 38 (Theorem 1 in [\[BFN22\]](#)). *Given two sets $A, B \subseteq [\Delta]$, the set $A + B := \{a + b \mid a \in A, b \in B\}$ can be constructed deterministically in $\tilde{O}(|A + B| \cdot \text{polylog}(\Delta))$ time.*

We use [Lemma 38](#) to enumerate two sets:

$$\mathcal{L} := \underbrace{X + \dots + X}_{\lfloor k/2 \rfloor} \quad \text{and} \quad \mathcal{R} := \underbrace{X + \dots + X}_{k - \lfloor k/2 \rfloor}.$$

Both \mathcal{L} and \mathcal{R} can be computed deterministically in $\tilde{O}(k \cdot (|\mathcal{L}| + |\mathcal{R}|))$ time by repeatedly applying [Lemma 38](#). Next, with both \mathcal{L} and \mathcal{R} in hand, we sort the sets by value and apply

Meet-in-the-Middle ([Algorithm 1.1](#)) to recover a solution in time $\tilde{O}_k(|\mathcal{L}| + |\mathcal{R}|)$ if one exists. 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 definition of the sets \mathcal{L} and \mathcal{R} . It remains to bound the runtime. Since the other steps of the algorithm take time $\tilde{O}_k(n)$, the bottleneck occurs in the Meet-in-the-Middle step, which takes time $\tilde{O}_k(|\mathcal{L}| + |\mathcal{R}|)$. Therefore it remains to bound the sizes of \mathcal{L} and \mathcal{R} . Without loss of generality, consider $|\mathcal{R}|$, and observe

$$|\mathcal{R}| = |\underbrace{X + \dots + X}_{k - \lfloor k/2 \rfloor}| \leq \lceil k/2 \rceil |X| = \mathcal{C}^{\lceil k/2 \rceil} |X|,$$

where the final step applies Plünnecke-Ruzsa ([Lemma 28](#)). This concludes the proof of [Theorem 19](#). □

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 13. *$(\mathcal{C}, 4)$ -SUM can be solved in expected time $\tilde{O}(\mathcal{C}n)$. Moreover, for any constant $\varepsilon > 0$, $(\mathcal{C}, 4)$ -SUM cannot be solved in $O((\mathcal{C}n)^{1-\varepsilon})$ 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 19](#). Recall that the bottleneck is determined by $|\mathcal{L}| + |\mathcal{R}|$, 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 $O((\mathcal{C}n)^{1-\varepsilon})$ would yield an algorithm for 4-SUM that runs in $O(n^{2-\varepsilon})$ time. \square

Remark 4. Applying the same lower bound argument to the more general case of k -SUM gives a lower bound of $\Omega(\mathcal{C}^{\lceil k/2 \rceil - 1} n)$ for (\mathcal{C}, k) -SUM under the k -SUM conjecture, leaving an $O(\mathcal{C})$ -factor gap.

In [Pet11], Petridis gives the slightly improved bound

$$|hA| = O(\mathcal{C}^{h-1} |A|^{2-\frac{2}{h}}),$$

for finite sets in commutative groups, improving on Plünnecke-Ruzsa. This narrows the gap between our upper and lower bounds slightly, although the result is still not tight for $k > 4$. Further improvements, either by non-trivially leveraging the small doubling constant of the input set to achieve a better algorithmic result for large k or by improving on Plünnecke-Ruzsa, would be both interesting and surprising.

Chapter 7

Future Work

What comes next for the Subset Sum problem? In this brief concluding chapter, we review what is known and offer some ideas for future work related to the topics considered in this thesis.

7.1 The Meet-in-the-Middle Barrier, Reconsidered

The most significant open problem remains to break the Meet-in-the-Middle barrier.

Question 4. *Can (Vanilla) Subset Sum be solved in time $O(2^{(0.5-\varepsilon)n})$ for any constant $\varepsilon > 0$?*

Informed opinion seems optimistic on this question. Referring to recent advancements, Abboud, Bringmann, Hermelin and Shabtay commented in 2022, “All this progress leads to the feeling that a positive resolution to [Question 4] might be just around the corner” [ABHS22]. As such, it might be more appropriate to restate the open question as a conjecture:

Conjecture 1 (Meet-in-the-Middle Conjecture). *There exists a constant $\varepsilon > 0$ such that (Vanilla) Subset Sum can be solved in time $O(2^{(0.5-\varepsilon)n})$.*

Improvements on certain existing algorithms would imply the conjecture. For example, in [NW21], Nederlof and Węgrzycki show that any improvement on their algorithm for Weighted Orthogonal Vectors would imply **Conjecture 1**, as would improvement on a certain node-weighted graph problem. Likewise, “color-coding” reductions from Subset Sum to k -SUM imply that an $n^{k/2-\varepsilon}$ algorithm achieved for any k would also prove the Meet-in-the-Middle Conjecture. Other results implying **Conjecture 1** are more simply stated as results for large classes of Subset Sum instances; see below.

The following subsections review the classes of worst-case Subset Sum instances that *can* be solved in time $2^{(0.5-\varepsilon)n}$, leaving us with a stubborn core of instances that appear (for now) to be harder. Two papers by Austrin, Kaski, Koivisto, and Nederlof [AKKN15, AKKN16] contributed significantly to this way of thinking about the problem, and are cited repeatedly below.

7.1.1 Unbalanced Subset Sum Instances

Recall that a Subset Sum instance (X, t) is *unbalanced* if it or its complement $(X, \Sigma(X) - t)$ admits a solution of size at most αn for some constant $\alpha < 0.5$. Such an instance can be solved deterministically in time

$$O^* \left(2^{H(\alpha)n/2} \right) = 2^{(0.5-\Omega(1))n}$$

by **Lemma 5**.

7.1.2 Random-Like Instances.

As previously noted in [Chapter 4](#), an algorithm for average-case Subset Sum that solves the problem with high probability over the draw of a random input is equivalent to an algorithm that solves most worst-case Subset Sum instances.

It is technically correct to say that the apparently difficult set of Subset Sum instances is a subset of those instances on which the Representation Method fails. However, in order for this classification to be helpful, we must exploit some useful property that they have in common. [Theorem 6](#), interpreted as a statement of this type, effectively says that the hard instances of Subset Sum either have a small Equal Subset Sum solution or construct an Equal Subset Sum solution while executing the Representation Method. However, this is only directly helpful if we are interested in obtaining solutions to Equal Subset Sum, as in Either-Or Subset Sum.

The two crucial conditions for the Representation Method are that

- partial candidate solutions fall into many residue classes (so that choosing a few residue classes recovers a solution with high probability) and
- the number of *pseudosolutions*, pairs of partial candidate solutions which add to the target but cannot be combined to create a solution (usually, because of overlapping elements) is not large enough to spoil the runtime of the algorithm.

(See [Section 1.1.2](#) for background on the representation method, and refer to [Algorithm 3.9](#), [Algorithm 4.2](#), and [Algorithm 5.2](#) for three ways in which the problems of solution distribution and pseudosolutions can be overcome.)

The following instance types are known to have properties that allow Representation Method-style approaches to succeed:

Instances With No Very Frequent Sums. Given a Subset Sum instance (X, t) , let β_s denote the *bin size* of the sum s :

$$\beta_s(X) := |\{Y \subseteq X \mid \Sigma(Y) = s\}|,$$

and let β denote the maximum bin size:

$$\beta := \max_s \beta_s.$$

Subset Sum instances in which $\beta = 2^{(0.5-\Omega(1))n}$ can be solved in time $2^{(0.5-\Omega(1))n}$ [AKKN16].

Instances with Many Sums. A corollary to Lemma 1.3 of [AKKN16] is that instances with many distinct Subset Sums ($\S(X) \geq 2^{0.997n}$) can be solved in time $2^{(0.5-\Omega(1))n}$. This follows from an information-theoretic argument which proves that the former condition implies $\beta < 2^{0.4996n}$.

7.1.3 Structured Instances

Certain properties related to additive structure are also sufficient to allow Subset Sum instances to be solved in time faster than $2^{n/2}$. **Constant doubling** (or extremely small doubling), explored in Chapter 6, is one of these. Others include the following:

Instances with Structured Subsets. If the algorithm can identify a small subset with few subset sums, it can exploit this directly to speed up Meet-in-the-Middle. Specifically, given a set $Y \subseteq X$ of cardinality at most αn for some $\alpha < 0.5$ and with $\S(Y) = 2^{(1-\Omega(1))n}$, Subset Sum can be solved in time $2^{(0.5-\Omega(1))n}$ by [AKKN16], Lemma 3.2. (Lemma 6, a version

of the same lemma that achieves a polynomial-time speedup, is used in [Chapter 5](#).)

Duplicate Elements and C -Collisions. Duplicate elements and small sets with equal sums tend to make Subset Sum easier: for example, if for some element s the multiset $\{s, s, s\} \subseteq X$, we can replace the subset $\{s, s, s\}$ with $\{s, 2s\}$ to shrink the instance without changing $\S(X)$.

More generally, for any constant $C \in \mathbb{Z}_{\geq 0}$, we can shrink the instance using the following procedure:

1. Enumerate all sums of at most C elements of X in time $O(n^C)$.
2. If two sets of size at most C have the same sum, there exists a subset $Y \subseteq X$ with $|Y| \leq 2C$ and $|\S(Y)| \leq 2^{|Y|} - 1$. Replace (X, t) with the instance family

$$\mathcal{X}_Y := \{(X \setminus Y, t - y) \mid y \in \S(Y)\}.$$

(X, t) has a solution if and only if at least one instance in \mathcal{X}_Y has a solution.

Repeat this procedure until no C -collisions exist. Two outcomes are possible. In the first, we repeat the procedure $o(n)$ times; in this case, we are left with a set family of size $(2^{2C} - 1)^{o(n)} = 2^{o(n)}$ and an instance of size $n - o(n)$ with no C -collisions. By Instance Splitting and the assumption, without loss of generality, of one-sided error, this is no harder than the original problem ([Section 2.3.1](#)).

On the other hand, if we repeat the procedure $\Omega(n)$ times, we can obtain a structured subset. Because each removed subset Y satisfies $|\S(Y)| \leq 2^{|Y|} - 1$, we can take the union of $\Theta(n)$ such subsets to create a new set Z with $|Z| = \Omega(n)$, $|Z| \leq 0.5n$, and $|\S(Z)| = 2^{(1-\Omega(1))|Z|}$. Z is a structured subset in the sense of the previous subheading, so we can exploit it to solve

the instance in time $2^{(0.5-\Omega(1))n}$.

Instances With One Very Frequent Sum. [AKKN16] also demonstrates that Subset Sum instances in which $\beta > 2^{0.661n}$ can be solved in time $2^{(0.5-\Omega(1))n}$.

Reviewing the tractable cases above suggests the conclusion that to break the Meet-in-the-Middle barrier, we need a better understanding of additive structure in the regime in which $|\mathfrak{S}(Y)| = 2^{\alpha n}$ for constants $\alpha < 1$, independent of n . Unfortunately, most existing results for problems of this type apply only to sets with much stronger guarantees of additive structure, about which we can draw more precise structural inferences (see, e.g., Section 1.1.3). In our regime, the problem appears difficult even to experts. For example, Tao observes, “The study of sets with close to maximal doubling appears to be hopeless at present” ([TV06], Section 2.2).

In other words, if we are to prove Conjecture 1, we may have to do so without strong statements about sets with “intermediate additive structure”. However, even in the absence of such helpful tools, several promising avenues remain open:

1. Reducing Subset Sum to other algorithmic problems or primitives that admit surprisingly fast algorithms. Algorithms that made creative use of fast matrix multiplication or the fast Fourier transform would be examples of this approach.
2. Progress on existing problems already used as ingredients in fast algorithms for Subset Sum, for example, Orthogonal Vectors or k -SUM.
3. Results that exploit aspects of additive structure specific to hard instances of Subset Sum. Such an approach might use information-theoretic or other arguments, even roughly, to infer better information about the occurrence of pseudosolutions when

average-case approaches are applied to hard Subset Sum instances.

4. Finally, perhaps most frustratingly and most tantalizingly, there remains the possibility of achieving exponentially faster algorithms for Subset Sum using only elementary methods. At present, the best “evidence” against such a breakthrough is only that many previous attempts have failed.

7.2 Open Questions from This Thesis

Of course, breaking the Meet-in-the-Middle barrier is far from the only open problem in the area of Subset Sum problems. Other related problems are interesting in their own right—and could, of course, lead indirectly to progress on [Conjecture 1](#). Those most closely related to the topics in this thesis include:

1. **Breaking the Meet-in-the-Middle Barrier on Other Coefficient Sets.** Given [\[MNPW19\]](#)’s result for Equal Subset Sum, it is natural to wonder whether there exist faster algorithms for GSS on other coefficient sets. There do not appear to be fundamental obstacles preventing the previous approach from being generalized to $C = [d : d]$, but the simplest arguments of this type are slightly too weak to break the Meet-in-the-Middle barrier. We are optimistic that a careful application of the representation method could result in $|C|^{(1/2 - \Omega_d(1))n}$ -time algorithms for all coefficient sets of the form $C = [-d : d]$, if not $C = [\pm d]$. This could help address the question of which variant of the Meet-in-the-Middle barrier is truly “natural” for GSS.
2. **Decision Either-Or Subset Sum.** It is unclear whether the decision variant of the Either-Or Subset Sum problem is easier than the search problem. If an algorithm

for EOSS is required only to give a single Yes/No answer for either Subset Sum or Equal Subset Sum, without producing a solution, can this problem be solved as fast as average-case Vanilla Subset Sum?

3. **Further Applications of Additive Structure.** The largest open question from [Chapter 6](#) is whether Subset Sum is FPT in the doubling constant. It would also be interesting to develop an algorithm for Integer Programming with unbounded variables parameterized in the doubling constant, effectively combining our progress on Bounded ILP feasibility and Unbounded Subset Sum. We suspect such a result could be derived from a constructive Freiman's theorem for torsion-free commutative groups, or simply for integer vectors of constant dimension.

References

- [ABF23] 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*, pages 391–404, 2023.
- [ABHS22] 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.
- [Agr04] Agrawal, Manindra and Kayal, Neeraj and Saxena, Nitin. PRIMES is in P. *Annals of Mathematics*, pages 781–793, 2004.
- [AKKM13] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space–Time Tradeoffs for Subset Sum: An Improved Worst Case Algorithm. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 45–56. Springer, 2013.
- [AKKN15] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset Sum in the Absence of Concentration. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.
- [AKKN16] 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.
- [BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved Classical and Quantum Algorithms for Subset-Sum. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 633–666. Springer, 2020.

- [BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved Generic Algorithms for Hard Knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 364–385. Springer, 2011.
- [BCP01] Christian Borgs, Jennifer Chayes, and Boris Pittel. Phase Transition and Finite-Size Scaling for the Integer Partitioning Problem. *Random Structures & Algorithms*, 19(3-4):247–288, 2001.
- [BDP05] Ilya Baran, Erik D Demaine, and Mihai Patrascu. Subquadratic Algorithms for 3SUM. In *Workshop on Algorithms and Data Structures*, pages 409–421. Springer, 2005.
- [Bel66] Richard Bellman. Dynamic Programming. *Science*, 153(3731):34–37, 1966.
- [BFN22] 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.
- [BGNV18] Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster Space-Efficient Algorithms for Subset Sum, k-SUM, and Related Problems. *SIAM J. Comput.*, 47(5):1755–1777, 2018.
- [Bib13] Khodakhast Bibak. Additive Combinatorics: With a View Towards Computer Science and Cryptography—An Exposition. In Jonathan M. Borwein, Igor Shparlinski, and Wadim Zudilin, editors, *Number Theory and Related Fields*, pages 99–128. Springer, 2013.
- [Böh11] E. Böhme. Verbesserte Subset-Sum Algorithmen, 2011.
- [Bri17] 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.
- [Bri23] Karl Bringmann. Knapsack with Small Items in Near-Quadratic Time. *arXiv:2308.03075*, 2023.
- [BW21] 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)*, pages 1777–1796. SIAM, 2021.

- [CCM11] Seok-Ho Chang, Pamela C. Cosman, and Laurence B Milstein. Chernoff-Type Bounds for the Gaussian Error Function. *IEEE Transactions on Communications*, 59(11):2939–2944, 2011.
- [CFG89] Mark Chaimovich, Gregory Freiman, and Zvi Galil. Solving Dense Subset-Sum Problems by Using Analytical Number Theory. *Journal of Complexity*, 5(3):271–282, 1989.
- [CFJ⁺14] Marek Cygan, Fedor Fomin, Bart M.P. Jansen, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Open Problems for FPT School. Available at <https://fptschool.mimuw.edu.pl/opl.pdf>, 2014.
- [CFK⁺15] Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 5.4. Springer, 2015.
- [Cha02] Mei-Chu Chang. A Polynomial Bound in Freiman’s Theorem. *Duke Math. J.*, 115(1):399–419, 2002.
- [Cha19] Timothy M Chan. More Logarithmic-Factor Speedups for 3SUM, (median,+)-Convolution, and Some Geometric 3SUM-Hard Problems. *ACM Transactions on Algorithms (TALG)*, 16(1):1–23, 2019.
- [CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The Complexity of Satisfiability of Small Depth Circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.
- [CJRS22] Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A Servedio. Average-Case Subset Balancing Problems. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–778. SIAM, 2022.
- [CJRS23] Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset Sum in Time $2^{n/2}/\text{poly}(n)$. In *27th International Conference on Randomization and Computation (RANDOM/APPROX)*, 2023.
- [CL15] Timothy M Chan and Moshe Lewenstein. Clustered Integer 3SUM Via Additive Combinatorics. In *Proceedings of the Forty-Seventh Annual ACM Symposium on the Theory of Computing (STOC 2015)*, pages 31–40, 2015.

- [CLMZ24] Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster Algorithms for Bounded Knapsack and Bounded Subset Sum via Fine-Grained Proximity Results. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4828–4848. SIAM, 2024.
- [CWX22] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Hardness for Triangle Problems Under Even More Believable Hypotheses: Reductions From Real APSP, Real 3SUM, and OV. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, page 1501–1514, New York, NY, USA, 2022. Association for Computing Machinery.
- [DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In *Annual Cryptology Conference*, pages 719–740. Springer, 2012.
- [DF12] Rodney G Downey and Michael Ralph Fellows. *Parameterized Complexity*. Springer Science & Business Media, 2012.
- [DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics Over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.
- [DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A Reliable Randomized Algorithm for the Closest-Pair Problem. *Journal of Algorithms*, 25(1):19–51, 1997.
- [DLRV23] 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.
- [EW19] 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.
- [FKP23] Nick Fischer, Piotr Kaliciak, and Adam Polak. Deterministic 3SUM-Hardness. In *Information Technology Convergence and Services*, 2023.
- [Fre64] Gregory A Freiman. On the Addition of Finite Sets. In *Doklady Akademii Nauk*,

- volume 158.5, pages 1038–1041. Russian Academy of Sciences, 1964.
- [Fre17] Ari Freund. Improved Subquadratic 3SUM. *Algorithmica*, 77:440–458, 2017.
- [GM91] Zvi Galil and Oded Margalit. An Almost Linear-Time Algorithm for the Dense Subset-Sum Problem. *SIAM Journal on Computing*, 20(6):1157–1189, 1991.
- [GO95] Anka Gajentaan and Mark H Overmars. On a Class of $O(n^2)$ Problems in Computational Geometry. *Computational Geometry*, 5(3):165–185, 1995.
- [Gom69] Ralph E Gomory. Some Polyhedra Related to Combinatorial Problems. *Linear Algebra and Its Applications*, 2(4):451–558, 1969.
- [GP18] Allan Grønlund and Seth Pettie. Threesomes, Degenerates, and Love Triangles. *J. ACM*, 65(4):22:1–22:25, 2018.
- [GS15] Omer Gold and Micha Sharir. Improved Bounds for 3SUM, K-SUM, and Linear Degeneracy. In *Embedded Systems and Applications*, 2015.
- [HGJ10] Nick Howgrave-Graham and Antoine Joux. New Generic Algorithms for Hard Knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.
- [HS74] Ellis Horowitz and Sartaj Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.
- [IP01] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- [JR18] Klaus Jansen and Lars Rohwedder. On Integer Programming and Convolution. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [JX23] 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.
- [Kar72] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

-
- [KK82] Narendra Karmarkar and Richard M Karp. *The Differencing Method of Set Partitioning*. Computer Science Division (EECS), University of California Berkeley, 1982.
- [Kle22] Kim-Manuel Klein. On the Fine-Grained Complexity of the Unbounded Subset-Sum and the Frobenius Problem. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3567–3582. SIAM, 2022.
- [KLM19] Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-Optimal Linear Decision trees for k-SUM and Related Problems. *Journal of the ACM (JACM)*, 66(3):1–18, 2019.
- [KPW20] 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.
- [KX19] Konstantinos Koiliaris and Chao Xu. Faster Pseudopolynomial Time Algorithms for Subset Sum. *ACM Transactions on Algorithms (TALG)*, 15(3):1–20, 2019.
- [Lov17] Shachar Lovett. *Additive Combinatorics and Its Applications in Theoretical Computer Science*. Number 8 in Graduate Surveys. Theory of Computing Library, 2017.
- [Lue98] George S Lueker. Exponentially Small Bounds on the Expected Optimum of the Partition and Subset Sum Problems. *Random Structures & Algorithms*, 12(1):51–62, 1998.
- [MNPW19] Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki. Equal-Subset-Sum Faster Than Meet-in-the-Middle. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 73:1–73:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [NSS95] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and Near-Optimal Derandomization. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 182–191. IEEE, 1995.
- [NW21] Jesper Nederlof and Karol Węgrzycki. Improving Schroepel and Shamir’s Algorithm for Subset Sum Via Orthogonal Vectors. In *Proceedings of the 53rd*

- Annual ACM SIGACT Symposium on Theory of Computing*, pages 1670–1683, 2021.
- [Pap94] Christos H Papadimitriou. On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.
- [Pet11] Giorgis Petridis. Upper bounds on the cardinality of higher sumsets. *Acta Arithmetica*, 158, 01 2011.
- [Pis03] David Pisinger. Dynamic Programming on the Word RAM. *Algorithmica*, 35(2):128–145, 2003.
- [PRW21] Adam Polak, Lars Rohwedder, and Karol Wkegrzycki. Knapsack and Subset Sum with Small Items. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2021.
- [Ran23] Tim Randolph. A Hybrid Algorithm for Subset Sum and Equal Subset Sum. 2023.
- [RR23] Victor Reis and Thomas Rothvoss. The Subspace Flatness Conjecture and Faster Integer Programming. In *64th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 974–988. IEEE, 2023.
- [Ruz94] Imre Z. Ruzsa. Generalized Arithmetical Progressions and Sumsets. *Acta Mathematica Hungarica*, 65(4):379–388, 1994.
- [Ruz09] Imre Z Ruzsa. Sumsets and Structure. *Combinatorial Number Theory and Additive Group Theory*, pages 87–210, 2009.
- [RW23] Tim Randolph and Karol Wegryzcki. Subset Sum with Constant Doubling. 2023.
- [San12] Tom Sanders. On the Bogolyubov–Ruzsa Lemma. *Analysis & PDE*, 5(3):627–655, 2012.
- [San13] Tom Sanders. The Structure Theory of Set Addition Revisited. *Bulletin of the American Mathematical Society*, 50(1):93–127, 2013.
- [Sch11] Tomasz Schoen. Near Optimal Bounds in Freiman’s Theorem. *Duke Mathemat-*

- ical Journal*, 158(1):1–12, 2011.
- [SS81] Richard Schroepel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM Jeournal on Computing*, 10(3):456–464, 1981.
- [Tao07] Terence Tao. Structure and Randomness in Combinatorics. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 3–15. IEEE, 2007.
- [Tao08] Terence Tao. *Structure and Randomness: Pages From Year One of a Mathematical Blog*. American Mathematical Soc., 2008.
- [Tre09] Luca Trevisan. Additive Combinatorics and Theoretical Computer Science. *ACM SIGACT News*, 40:50–66, 2009.
- [TV06] Terence Tao and Van H Vu. *Additive Combinatorics*, volume 105. Cambridge University Press, 2006.
- [Vio11] Emanuele Viola. *Selected Results in Additive Combinatorics: An Exposition*. Number 3 in Graduate Surveys. Theory of Computing Library, 2011.
- [Woe08] Gerhard J Woeginger. Open Problems Around Exact Algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.
- [Zha22] Yufei Zhao. Graph Theory and Additive Combinatorics. *Notes for MIT*, 18:49–58, 2022.

Appendix A

Appendix to Chapter 2

A.1 Reduction of Multilist k -SUM to Single-List k -SUM

The **Multilist k -SUM** problem is defined similarly to the k -SUM problem (**Problem 2**), but with the single input set replaced by k input sets X_1, X_2, \dots, X_k . Instead of selecting k elements from the single input set, we are now required to select 1 element from each of the k input sets.

Multilist k -SUM can be reduced to k -SUM as follows.

Fix a Multilist k -SUM instance $(X_1, X_2, \dots, X_k, t)$. Let b and c be the smallest powers of 2 satisfying

$$b \geq \max_{i \in k} \|X_i\|_\infty,$$

that is, larger than the largest input integer, and

$$c \geq k.$$

For $i \in [k]$, create the list

$$X'_i := X_i + bc^i$$

by translating X_i by bc^i . We can think of this operation as adding a ‘flag’ to the bit representation of each element to mark its membership in the original list X_i . The large value b serves as an offset to prevent the sum of original input integers from affecting the flag bits, while the multiplication of each flag bit by c^i prevents interference between the flag bits when at most k elements are added together.

Define a new k -SUM instance with the input list

$$X := \bigcup_{i \in [k]} X'_i$$

and target

$$t' := t + b \sum_{i \in [k]} c^i.$$

The use of ‘flag’ bits ensures that X contains no duplicate elements. The equivalence of the new problem follows from observing the one-to-one relationships between solutions to the Multilist k -SUM instance $(X_1, X_2, \dots, X_k, t)$ and the k -SUM instance (X, t') .

The new k -SUM instance contains kn inputs of size at most $O_k(1)$ times larger than elements of the original instance. Thus an $n^{f(k)}$ -time algorithm for k -SUM can be used to solve the original Multilist k -SUM problem in time $O_k(n^{f(k)})$.

A.2 Reduction of Multiset k -SUM to k -SUM

The **Multiset k -SUM** problem is defined similarly to the k -SUM problem ([Problem 2](#)), but with set input replaced with *multiset* input; i.e., the input list is allowed to contain duplicates. Multiset k -SUM can be reduced to k -SUM as follows.

We begin by recalling a definition and theorem from [\[NSS95\]](#).

Definition 6 (Splitter). An (n, k, ℓ) -splitter \mathcal{F} is a family of functions from $[n]$ to $[\ell]$ such that for every set $S \subseteq [n]$ of size k , there exists $f \in \mathcal{F}$ such that for every $1 \leq j, j' \leq \ell$, the values $|f^{-1}(j) \cap S|$ and $|f^{-1}(j') \cap S|$ differ by at most 1.

In other words, for every $S \subseteq [n]$ of size k , some $f \in \mathcal{F}$ partitions $[n]$ into ℓ subsets in a way that splits S as evenly as possible. The special case of an (n, k, k) -splitter is called (n, k) -perfect hash family. A result due to Naor, Schulman, and Srinivasan allows us to construct (n, k) -perfect hash families in time $O_k(n \log n)$.

Theorem 20 ([\[NSS95\]](#)). *For any $n, k \geq 1$, it is possible to construct an (n, k) -perfect hash family of size $e^k k^{O(\log k)} \log n$ in time $e^k k^{O(\log k)} n \log n$.*

Fix a Multiset k -SUM instance (X, t) , let \mathcal{F} be an (n, k) -perfect hash family, and consider the set of $O_k(\log n)$ instances of Multilist k -SUM created by partitioning \vec{x} according to the elements of \mathcal{F} . (Note that, without loss of generality, we can assume the Multilist k -SUM instances do not contain duplicate elements in the same list. Because we must choose exactly one element from each of the k input lists in Multilist k -SUM, we can remove duplicates from the same list without changing whether a solution exists.)

By [Definition 6](#), at least one of these new Multilist k -SUM instances has a solution if and only if our original instance has a solution. We complete the reduction by converting

each new Multilist k -SUM instance into a k -SUM instance using the reduction given in [Appendix A.1](#).

As a result, given an $n^{f(k)}$ time algorithm for k -SUM, we can construct and solve the $O_k(\log(n))$ Multilist k -SUM instances and recover a solution to the original Multiset k -SUM instance, if one exists, in time $O_k(n^{f(k)} \cdot \log n)$.