COLUMBIA UNIVERSITY

PhD THESIS PROPOSAL

Department of Computer Science

May 5, 2023

Exact Algorithms for Subset Sum & Subset Balancing Problems

Tim Randolph
twr2111
t.randolph@columbia.edu

Advisors:
Xi Chen
Rocco A. Servedio

Abstract

We propose a thesis that collects a variety of original results for Subset Sum and other subset balancing problems. The results are unified by the shared obstacle of the $O^*(2^{n/2})$-time "Meet-in-the-Middle barrier" as well as the application and adaptation of the representation method and other techniques. Concretely, we present:

1. a suite of 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;
2. a proof that 'Either-Or Subset Sum', the problem of solving either Subset Sum or Equal Subset Sum, can be solved in time $O^*(2^{(0.5-\varepsilon)n})$ in the worst case;
3. algorithms in the "log shaving" tradition that solve Subset Sum in time $2^n/poly(n)$ in the worst case in common RAM models; and
4. algorithms for Subset Sum instances with constant doubling that leverage new FPT results for classic theorems in additive combinatorics.

Contents

1 Introduction

Subset Sum is a difficult algorithmic problem with a distinguished history. It was one of Karp's 21 original NP-complete problems (as 0-1 Knapsack), it served as one of the earliest applications for exact exponential and pseudopolynomial algorithms based on dynamic programming, and it continues to inspire interest among cryptographers, complexity theorists and mathematicians today [Kar72, HS74]. Moreover, even among the many classic algorithmic problems, Subset Sum occupies a special position. Along with very close relations, such as Partition and 0-1 Knapsack, Subset Sum asks one of the most fundamental computationally hard questions about a set of integers. These "subset balancing" problems have a persistent, aggravating allure: after decades of work, when handed a bucket of numbers and asked to add them up, we are humbled by our ignorance.

This may already be sufficient reason to study the complexity of subset balancing problems. However, the study of subset balancing problems also provides connections to additive combinatorics, phase transitions in the behavior of large random sets, fine-grained and parameterized complexity, and even low-level RAM trickery, as well as opportunities for delicate algorithm design. The proposed thesis attaches these threads in the form of a collection of original results in exact algorithms for subset sum and subset balancing problems.

The question of whether Subset Sum can be solved in time $O^*(2^{(0.5-\varepsilon)n})$[1] for some constant $\varepsilon > 0$ is one of the major questions in exact algorithms. Despite many attempts, Horowitz and Sahni's classic $O^*(2^{0.5n})$-time Meet-in-the-Middle algorithm for worst-case inputs remains the benchmark after almost 50 years [HS74]. The classic Meet-in-the-Middle algorithm is presented for reference in Section 2.3 below.

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 Schroeppel and Shamir improved the $O^*(2^{n/2})$ space complexity of the meet-in-the-middle algorithm by giving an algorithm that runs in $O^*(2^{n/2})$ time and $O^*(2^{n/4})$ space [SS81]. An exciting recent result by Nederlof and Wegrzycki further improved this space complexity to $2^{0.249999n}$ [NW21], breaking the Shroeppel-Shamir space barrier but leaving the Meet-in-the-Middle barrier intact.

Another tool relevant for the proposed 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].[2] Subsequent works, including [BCJ11, Böh11, BBSS20], refined what became known as the *representation technique*, and whittled the average-case exponent down to $O^*(2^{0.283n})$. This new technique inspired a flurry of results in related settings, including better time-space tradeoffs [AKKM13, DDKS12], a faster polynomial-space algorithm [BGNV18], and fast al-

[1] We use $O^*(\cdot)$ notation to suppress factors polylogarithmic in the argument of the function; so the notation "$O^*(2^{0.5n})$" suppresses poly(n) factors. See Section 2.1.

[2] Although 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.

gorithms for large classes of sufficiently "random-like" instances [AKKN15, AKKN16]. The proposed thesis adapts the representation method to several purposes and presents the approach in greater detail in Section 2.4.

In 2019, [MNPW19] used the representation technique 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. 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]. Section 3 of this thesis covers work including further improvements for Equal Subset Sum in the average case [CJRS22].

Equal Subset Sum is also closely related to the Number Balancing problem of Karmarkar and Karp, which can be thought of as the optimization version of Equal Subset Sum [KK82]. Number Balancing considers real inputs and seeks algorithms that solve the Equal Subset Sum problem to high precision. In general, exact algorithms for sumset balancing problems can be applied to the Number Balancing problem by rounding the inputs to any degree of precision, but are less efficient than Karmarkar and Karp's approach on inputs of size $n^{O(\log(n))}$.

2 Preliminaries

In this section, we introduce notation and concepts to which we will refer back frequently. These include a generalized version of the subset sum problem, the classic Meet-in-the-Middle algorithm of Horowitz and Sahni, and a concept-focused introduction to the representation method, whose principles and limitations inform several of our algorithms.

2.1 Notation

To ease readability, we adopt the following notational conventions throughout the paper: 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 lists; and calligraphic capital letters ($\mathcal{W}$, $\mathcal{Q}$, etc.) denote collections of sets of numbers.

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

Big-O Notation. We augment standard big-O notation with a tilde ($\widetilde{O}$, $\widetilde{\Omega}$, $\widetilde{\Theta}$, etc.) to suppress $\mathrm{polylog}(n)$ factors, regardless of the argument. For example, we have $\widetilde{O}(2^n) = O(2^n \cdot \mathrm{polylog}(n))$ and $\widetilde{\Omega}(n) = \Omega(\frac{n}{\mathrm{polylog}(n)})$.

In contrast, the O-star notation (O^*, Θ^*, etc.) suppresses factors polylogarithmic in the argument. For example, $O^*(2^n) = O(2^n \cdot \mathit{poly}(n))$ and $O^*(1) = \mathit{poly}(n)$.

Probability Notation. Random variables are written in boldface. In particular, we write "$\boldsymbol{x} \sim S$" to indicate that the element $\boldsymbol{x}$ is sampled uniformly at random from the finite multiset S.

Set Notation. We write $[a : b]$ for the set of integers $\{a, a{+}1, \ldots, b\}$ and $[a]$ for $\{1, 2, \ldots, a\}$.

Given a multiset or list Y of integers, we adopt several shorthands: $\Sigma(Y) := \sum_{y \in Y} y$ denotes the sum, $w(Y) := \{\Sigma(T) \mid T \subseteq Y\}$ denotes the set of subset sums, and L_Y denotes the sorted list containing the elements of $w(Y)$.

We write $f(Y)$ for the multiset or list obtained by applying operation f element-wise, such as $Y + \alpha = \{y + \alpha \mid y \in Y\}$ and $\alpha Y = \{\alpha y \mid y \in Y\}$. The only exception is $(Y \bmod p)$, which denotes the set of *distinct* residues $\{(y \bmod p) \mid y \in Y\}$.

Stirling's Approximation for Binomial Coefficients. We use the following well-known consequence of Stirling's approximation (see e.g. [FG06, Lemma 16.19]): For each integer $j \in [0 : n/2]$,

$$\sum_{i \leq j} \binom{n}{i} \;\leq\; 2^{H(j/n) \cdot n}, \tag{1}$$

where $H(y) := -y \log(y) - (1 - y) \log(1 - y)$ is the binary entropy function.

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

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

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

2.2 Generalized Subset Sum

In sumset balancing problems, we are given a set of input numbers, usually integers, and asked to find a subset of the input that adds up to a target. As we will consider a number of variants of the problem, we state a generalized version here and refer back to this definition while making modifications as needed.

Problem 1: Generalized Subset Sum (GSS)

Input:

- An integer range bound M.[a]
- An input vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ or multiset $X = \{x_1, x_2, \ldots, x_n\}$. When M is specified, $\vec{x} \in [0 : M - 1]^n$.
- A set $C \subset \mathbb{Z}$ of allowed coefficients. In Subset Sum, $C = \{0, 1\}$. In Equal Subset Sum, $C = \{-1, 0, 1\}$. In *Unbounded Subset Sum*, $C = \mathbb{N}$.)
- 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 $\Sigma(S) = t$.

[a]At certain points we accept integers of arbitrary size and analyze our algorithms using a word RAM model, in which case this parameter will be left unspecified.

2.3 The Meet-in-the-Middle Barrier

Procedure `MeetInTheMiddle`(X, t)

Input: An integer multiset $X = \{x_1, x_2, \ldots, x_n\}$ and an integer target t.

```
0. Fix any partition of $X = A \cup B$ such that $|A| = |B| = n/2$.
1. Enumerate the sorted lists $L_A$ and $L_B$.[a]
2. Initialize two pointers at the smallest value in $L_A$ and the largest value in $L_B$.
    (a) If these two values sum to the target $t$, return the associated solution;
    (b) if they sum to less than $t$, then increment the pointer into $L_A$ and repeat;
    (c) and if they sum to more than $t$, then increment the pointer into $L_B$ and repeat.
   If either pointer goes past the end of its list, return "NO".
```

[a]Sorting and enumerating L_A and L_B in time $O^*(|L_A| + |L_B|)$ is trivial. In fact, it is possible to perform this step in linear time (i.e., $O(|L_A| + |L_B|)$); see [CJRS23] or Section 5.

Figure 1: The classic Meet-in-the-Middle algorithm, adapted from [HS74].

Figure 1 displays the classic *Meet-in-the-Middle algorithm* for Subset Sum. Meet-in-the-Middle was introduced by Horowitz and Sahni in 1974, and will serve as a baseline for comparison throughout the proposed thesis [HS74]. Improving on this algorithm is a major open problem in exact algorithms (see mentions in [Woe08, CFJ+14, AKKN15, AKKN16, NW21] as well numerous other works). Among the results in the proposed thesis are small *poly*(n)-factor improvements on Meet-in-the-Middle, which to the best of our knowledge are the first improvements in 50 years (Section 5). However, there is 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 $O(2^{0.337n})$, later improved to $O(2^{0.291n})$ in the work of Becker, Coron, and Joux [BCJ11]. These works pioneered the *representation method*, which continues to prove useful and is explained in the next section.

2.4 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 Section 3, Section 4 and Section 5).

In a nutshell, the technique works as follows:

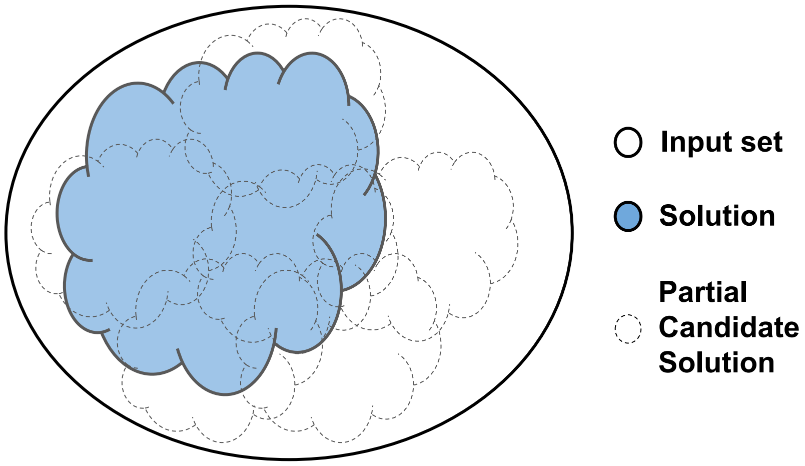

Figure 2: Partial candidate solutions. This step is virtual: in practice, there are too many partial candidate solutions to enumerate them all.

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 $n/4$, 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.

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

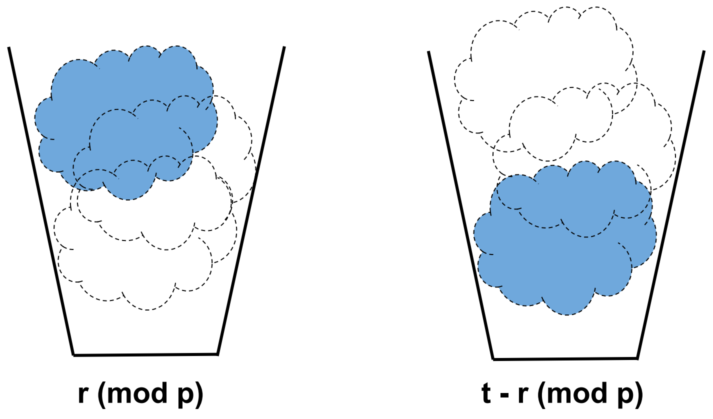

Figure 3: Filtering partial candidate solutions. At this stage, the algorithm must ensure that the selected residue classes contain an $O(1/\mathbf{p})$-fraction of partial candidate solutions.

candidate solutions are hashed by their sum into residue classes modulo a large random prime $\mathbf{p}$. The algorithm then discards most partial candidate solutions, keeping only those that fall into the residue classes r and $t - r \pmod{\mathbf{p}}$ 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 $t - r$.

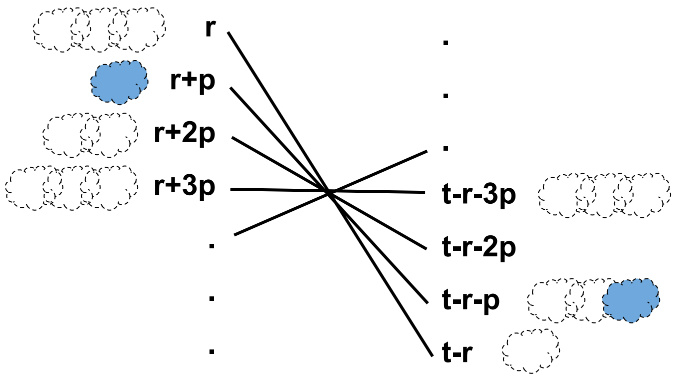

Figure 4: Comparing partial candidate solutions using Meet-in-the-Middle. Note that multiple partial candidate solutions may correspond to the same value: the algorithm must ensure that partial candidate solutions are distributed well among values and recover a pair of partial candidate solutions that sum to t and are disjoint.

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 $\binom{n}{n/4} = \Theta^*(2^{H(1/4)n}) = \Theta^*(2^{0.811n})$ input subsets of size $n/4$ by Equation (1), hashing over a prime of size $O^*(2^{0.5n})$ produces residue classes containing $(2^{0.811n-0.5n}) = O^*(2^{0.311n})$ partial candidate solutions in expectation.

However, several conditions must be met before the representation method works. First, 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. Moreover, 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. Finally, complications may arise during solution recovery. For example, in [HGJ10], partial candidate solutions may overlap, so the algorithm must ignore pairs of partial candidate solutions that add to the target but are not disjoint. Moreover, if too many partial candidate solutions have exactly the same sum, this causes a slowdown in the recovery step: if many partial candidate solutions in the r and $t - r$ residue classes (mod $\mathbf{p}$) have *exactly* the same sum, they must all be compared to each other, eliminating the speed-up from Meet-in-the-Middle.

All of these necessary conditions hold with high probability over uniformly random input. 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 "solution profile" (see Section 3). Moreover, the approach can be adapted to worst-case settings, as long as the issues described above can be resolved by other means. Section 4 and Section 5 illustrate two adaptations of the representation method to worst-case input settings.

2.5 Overview and Summary of Current Progress

The proposed thesis would present a collection of results that represent the state of the art in exact algorithms for the Subset Sum problem. The different approaches are unified by the shared obstacle of the Meet-in-the-Middle barrier, which overshadows all work on exact algorithms for this problem, and by the representation method, which has proven useful in both average- and worst-case settings. The intellectual appeal of each approach comes from the insight it gives into the structure of subset balancing problems, the behavior of the integers under addition, and the techniques of exact algorithms.

2.5.1 Average-Case Algorithms for Subset Sum and Equal Subset Sum

Section 3 opens the discussion by providing motivation for Generalized Subset Sum, which generalizes almost every variant problem considered in the proposed thesis. The main message of this section is that, at least in the average case, all subset balancing problems share a common structure. The results presented were previously published as [CJRS22]. We study the behavior of average-case GSS and identify phase transitions that govern the probability of a solution, given fixed coefficient sets and input range bounds. When the coefficient set contains 0, as in the family of problems that generalize Equal Subset Sum, we show that these thresholds can be proven by a novel, intuitive approach. However, when the coefficient set does not contain 0, we are forced to fall back on higher-powered but less illuminating machinery.

We then use our structural results to develop a suite of algorithms for average-case balancing problems. These generalize the representation method to a wide variety of symmetric coefficient sets and break the natural Meet-in-the-Middle barriers that hold in the worst-case setting.

2.5.2 Complementarity of Subset Sum and Equal Subset Sum

Section 4 presents a standalone result that highlights a complementarity between the hard instances of Subset Sum and Equal Subset Sum. Surprisingly, when we relax requirements by allowing ourselves to solve *at least one* of the two problems on a certain input (allowing a negative result), the representation method can be adapted to the worst-case. The several categories of instances that cause the representation method to fail when applied to worst-case Subset Sum turn out to be relatively easy instances of Equal Subset Sum, and vice versa. The result is an algorithm that solves *Either-Or Subset Sum* in time $O(2^{(0.5-\varepsilon)n})$ for $\varepsilon > 0.03$. This work will be submitted to IPEC 2023 as [Ran23].

2.5.3 Log Shaving for Subset Sum

Is the Meet-in-the-Middle barrier absolute? One way to answer this question would be to design an $O(2^{(0.5-\varepsilon)n})$-time algorithm for Subset Sum, and another would be to prove a lower bound. However, if neither of those possibilities seem likely, a third option is to push the boundaries by improving runtime by a smaller factor.

In this section, we treat the longstanding Meet-in-the-Middle barrier with the dignity of a fine-grained complexity hypothesis and design the first algorithms that solve the Subset Sum problem in time $2^{n/2}/poly(n)$ in the worst case. We propose to continue this line of research during the next year, with the aim of achieving superpolynomial savings. This work is submitted to RANDOM/APPROX 2023 [CJRS23].

2.5.4 Subset Sum Parameterized in the Doubling Constant

The doubling constant, defined for an integer set A as the smallest K such that $|A + A| \leq K|A|$, 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. Freiman's theorem, a major result in additive combinatorics, makes this statement concrete by proving that any set A with doubling constant K is contained in a generalized arithmetic progression with dimension $d(K)$ and volume $f(K)|A|$, where d and f are functions that depend only on K. This prompts a natural question: 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 take steps in this direction 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. Moreover, we observe that the GAP formulation of Subset Sum looks suspiciously like an instance of Integer Linear Programming feasibility. We also show a FPT-reduction from Binary Integer Linear Programming feasibility to Subset Sum. As yet, we do not have a result that reduces a subset balancing problem to an ILP feasibility problem, but we propose to continue research into the precise relationship between the two problems.

3 Average-Case Algorithms for Subset Sum and Equal Subset Sum

This section paraphrases the results of [CJRS22].

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 the *average-case* Generalized Subset Sum (GSS) problem in which the input vector $\vec{\boldsymbol{x}}$ is uniformly random over $[0 : M-1]^n$.

There are two natural average-case variants of the GSS problem: in the first variant, the target value is obtained by sampling a "hidden" solution, and hence 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 that was studied by [HGJ10, BCJ11] and others. We will refer to it 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 $\vec{\boldsymbol{x}}$. 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].

3.1 Results

This section considers two families of symmetric coefficient sets, with and without zero. For a fixed positive integer $d \in \mathbb{N}$, we define $C(d) = \{\pm 1, \pm 2, \ldots, \pm d\}$ and $C_0(d) = \{0, \pm 1, \pm 2, \ldots, \pm d\}$. These two sets cover a wide range of cases, including Equal Subset Sum. We note that while the natural coefficient set corresponding to Subset Sum is $C = \{0, 1\}$, an instance $\vec{x}$ with target τ can be easily converted to GSS on $C(1) = \{\pm 1\}$ by setting a new target $\tau' := 2\tau - \sum_{i\in[n]} x_i$.[3]

Our algorithms never return false positives and succeed with high probability over uniformly distributed input vectors. Formally: given M, $\vec{x} \in [0 : M-1]^n$, $C = C(d)$ or $C_0(d)$, and t, our algorithms either return "NO" or a solution $\vec{c}$ satisfying $\vec{c} \cdot \vec{x} = \tau$ (with $\vec{c} \neq \vec{0}$ when $C = C_0(d)$ and $\tau = 0$). The algorithm fails if it returns "NO" when a solution exists; otherwise, correctness is guaranteed.

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 $C = C(d)$ or $C_0(d)$. For any constant $\zeta > 0$, there is a randomized algorithm for average-case GSS with running time*

$$O^*(|C|^{\Lambda(|C|)n+\zeta n})^{4} \qquad \textit{where} \quad \Lambda(z) = \max \begin{cases} 1 - \frac{z+1}{2z}\log_z(z+1) + \frac{1}{z}\log_z(2) \\ \frac{2}{3} - \frac{z+1}{3z}\log_z\left(\frac{z+1}{2}\right) \end{cases} .$$

Given any M and τ with $|\tau| = o(nM)$, the algorithm succeeds on $(M, \tau, \vec{\boldsymbol{x}})$ with probability at least $1 - e^{-\Omega(n)}$ when $C = C_0(d)$ and with probability at least $1 - o_n(1)$ when $C = C(d)$, over the draw of $\vec{\boldsymbol{x}} \sim [0 : M-1]^n$ and the randomness of the algorithm.

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 $|C|$. Figure 5 plots the function Λ and Table 1 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 worst-case $O^*(3^{0.488n})$ runtime of [MNPW19]:

Corollary 1 (Algorithm for Average-Case Equal Subset Sum)**.** *There exists an algorithm that solves 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 M 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 $d \in \mathbb{N}$, $C = C(d)$ or $C = C_0(d)$, $M = |C|^{\alpha n + o(n)}$ for some $\alpha \in (0,1)$ and an offset τ with $|\tau| = o(Mn)$. For any constant $\zeta > 0$, there exists an algorithm that solves average-case GSS in time*

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

where Λ is defined as in Theorem 1. The algorithm succeeds with probability at least $1-e^{-\Omega(n)}$ for $C = C_0(d)$ and with probability at least $1 - o_n(1)$ for $C = C(d)$.

[3]In fact, our results extend to all scale multiples and translations of $C(d)$ and $C_0(d)$.

[4]Note 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 $M = 2^{\omega(n)}$.

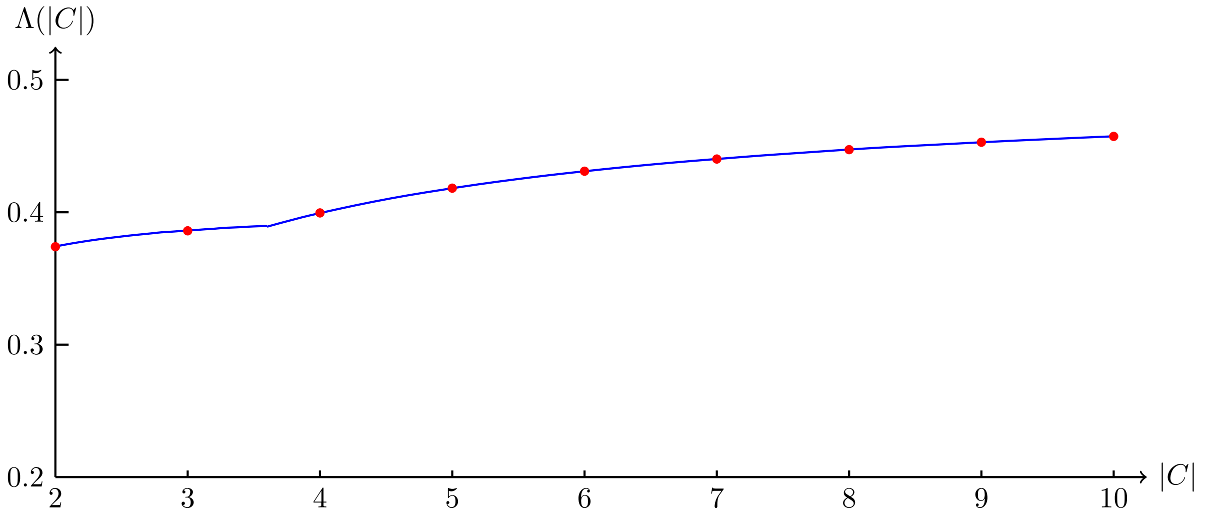

Figure 5: Plot of Λ. The red points plot $\Lambda(z)$ for $z \in [2:10]$.

$d \in \mathbb{N}$		1	2	5	10
Runtime	$C(d)$	$O^*(\lvert C\rvert^{0.375n})^\dagger$	$O^*(\lvert C\rvert^{0.400n})$	$O^*(\lvert C\rvert^{0.458n})$	$O^*(\lvert C\rvert^{0.479n})$
	$C_0(d)$	$O^*(\lvert C\rvert^{0.387n})^\ddagger$	$O^*(\lvert C\rvert^{0.419n})$	$O^*(\lvert C\rvert^{0.462n})$	$O^*(\lvert C\rvert^{0.480n})$

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

‡This runtime for $C_0(1)$ should be contrasted with the $O^*(|C|^{0.488n})$-time worst-case runtime due to [MNPW19].

Table 1: Runtime of our algorithm for average-case GSS on various coefficient sets.

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 $C = C_0(d)$ are very likely, and very unlikely, to have solutions in the average case.

Theorem 3 (GSS Solution Probability for $C = C_0(d)$). *Let $C = C_0(d)$ for some fixed $d \in \mathbb{N}$, and fix any constant $\varepsilon > 0$. For $\vec{\boldsymbol{x}} \sim [0 : M-1]^n$ and any integer τ satisfying $|\tau| = o(Mn)$, we have*

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

The preceding theorem covers the $C = C_0(d)$ case, for which our proof is simple and novel. We also prove an analogous result for the $C = C(d)$ case by a different method: in this case, we adapt the analysis of [BCP01], which results in slightly worse probability guarantees. Finally, we also show corollary results in the Number Balancing setting.

3.2 Techniques

To explain our structural and algorithmic results, we begin with some intuition for the distribution of sums that are achievable using coefficient set C. Consider a uniform random draw of the input vector $\vec{\boldsymbol{x}}$ from $[0 : M-1]^n$. Since the coefficient set C is symmetric about 0, all elements of the random set $\boldsymbol{S}_n := \{\vec{c} \cdot \vec{\boldsymbol{x}} : \vec{c} \in C^n\}$ have magnitude $O(Mn)$, and hence intuitively $\boldsymbol{S}_n$ is "tightly concentrated around the origin". Consequently, when the offset τ is not too large (the case of interest for us) it is natural to expect that a solution $\vec{c} \cdot \vec{\boldsymbol{x}} = \tau$ is likely to exist if $M \ll |C|^n$ and is unlikely to exist if $M \gg |C|^n$. Indeed, a simple counting argument establishes this for the case that $M \gg |C|^n$, but substantially more work is required to rigorously confirm the above intuition for the $M \ll |C|^n$ case. This is accomplished by our structural results, which we describe next.

3.2.1 Structural Results

Our structural results for coefficient set $C_0(d)$ (including zero) and for $C(d)$ (not including zero) are established using very different techniques. Consider the case of $C = C_0(d)$ first. We are able to show that for any positive constant $\varepsilon > 0$, given $M \leq |C|^{(1-\varepsilon)n}$ and any fixed integer offset τ with $|\tau| = o(Mn)$, the probability (over a uniform random $\vec{\boldsymbol{x}} \sim [0 : M-1]^n$) that there exists a solution $\vec{c} \cdot \vec{\boldsymbol{x}} = \tau$ with $\vec{c} \in C^n$ is *exponentially* close to 1. This extremely high probability that there exists a solution translates directly into the extremely high success probability of our algorithms in Theorem 1 and Theorem 2.

To establish this structural result, we employ a novel proof strategy that aligns with the intuition sketched earlier and is based on an iterative analysis. We define random sets $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots$ where the set $\boldsymbol{S}_\ell$ for $\ell \in [n]$ is given by $\boldsymbol{S}_\ell := \{\vec{c}' \cdot \vec{\boldsymbol{x}}' : \vec{c}' \in C^\ell\}$ with a uniform random $\vec{\boldsymbol{x}}' \sim [0 : M-1]^\ell$. We then analyze how these set sizes $|\boldsymbol{S}_\ell|$ increase as a function of ℓ by thinking of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ as being drawn in succession. We first argue that with very high probability $|\boldsymbol{S}_\ell|$ increases rapidly with ℓ until, at some value $\boldsymbol{L}_1 < n$, 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* $\sum_{i \in [\boldsymbol{L}_2]} c_i \boldsymbol{x}_i$ will hit the offset τ with very high probability, where $\boldsymbol{L}_2 \geq \boldsymbol{L}_1$ and $\boldsymbol{L}_2$ is still less than n. The remaining elements $\boldsymbol{x}_{\boldsymbol{L}_2+1}, \ldots, \boldsymbol{x}_n$ are simply assigned the 0 coefficient to complete the overall solution. Figures 6 and 7 illustrate the two principles at work.

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

We extend the [BCP01] analysis and establish a similar result for general coefficient sets

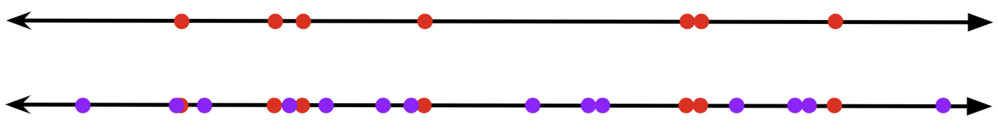

Figure 6: While $\mathbf{S}_\ell$ is sparse, it grows quickly: $|\mathbf{S}_{\ell+1}| = |\mathbf{S}_\ell + Cx_\ell| \approx |C| \cdot |\mathbf{S}_\ell|$ with high probability.

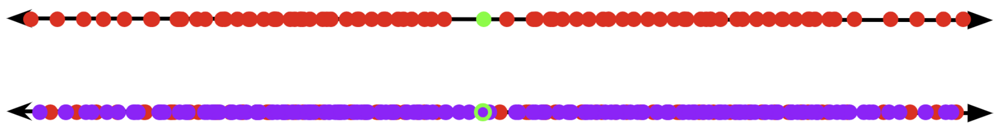

Figure 7: When $\mathbf{S}_\ell$ is dense, $\mathbf{S}_{\ell+1}$ hits any fixed **target point** with nontrivial probability.

$C = C(d)$ for arbitrary $d > 1$. We show that for any constant $d > 1$, if $M \leq |C|^{(1-\varepsilon)n}$ then given any fixed integer offset τ with $|\tau| = o(Mn)$, the probability (over a uniform draw of $\vec{\boldsymbol{x}}$ from $[0 : M-1]^n$) that there exists a solution $\vec{c} \cdot \vec{\boldsymbol{x}} = \tau$ with $\vec{c} \in C^n$ is $1 - o_n(1)$. 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.

3.2.2 Algorithmic Results

Intuitively, our structural results tell us the parameter settings for which average-case GSS instances are likely to have solutions and what those solutions will look like. This allows us to prove the correctness of our algorithm, which combines new preprocessing ideas with an approach based on the representation technique.

At a high level, since instances in which $M > |C|^{(1+\varepsilon)n}$ 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 $M < |C|^{(1-\varepsilon)n}$ 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 a narrow range of values for M.

At this point, we adapt the representation method to Generalized Subset Sum. We use partial coefficient assignments as partial candidate solutions and define the *signature* of a partial assignment as the coefficient-weighted sum of the assigned elements. Then, we hash partial candidate solutions by signature into residue classes modulo a large random prime,

```
Procedure AverageCaseGSS($M, \vec{x}, C, t$):

Input: $M \in |C|^{(1\pm\varepsilon)n}$, $\vec{x} \in [0 : M-1]^n$, $C = C(d)$ or $C_0(d)$, $t$.

0. Repeat $poly(n)$ times:
1.     Guess a "solution profile", indicating the number of times each coefficient in
       $C$ will be used in a hypothetical solution. This loop will recover only
       solutions that fit the profile.
2.     Select a large prime $\boldsymbol{p}$ and residue class $\boldsymbol{r} \in [0 : p-1]$ uniformly at random.
3.     Let $L_1$ be the set of partial candidate whose signatures sum to $\boldsymbol{r} \pmod{\mathbf{p}}$,
       or a subsample of this set if the original set is too large.
4.     Define $L_2$ as $L_1$, but with partial candidate solutions whose signatures sum
       to $t - \tau - \boldsymbol{r} \pmod{p}$, where $\tau$ is a term that adjusts for the solution profile.
5.     Sort $L_1$ and $L_2$ by signature and use Meet-in-the-Middle to find a disjoint
       solution pair whose signature sums to $t - \tau$, if one exists.
6. Return 'NO'.
```

Figure 8: Simplified sketch of the algorithm for average-case GSS. Certain elements, such as solution profiles and parameter settings, have been simplified or omitted for readability. For full details, see [CJRS22].

following the general pattern described in Section 2.4.

We prove a result that guarantees that with high probability over the input, many partial solutions have distinct signatures, which satisfies the conditions of the representation method and removes the need for heuristic assumptions. Finally, we use dynamic programming and Meet-in-the-Middle to simulate the signature-based hash, sample elements, and recover a solution.

The GSS algorithm (in extremely simplified form) is reproduced in Figure 8.

4 Complementarity of Subset Sum and Equal Subset Sum: Solving an "Either-Or" Problem

This section introduces results from [Ran23].

In the previous section, we observed the close relationship between 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 obvious similarities, we were forced to use a different structural analyses in each of the two cases.

The current section expands on the relationship between the two problems by proving a surprising result: if our goal is to find a (possibly negative) solution to *either* 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 $O^*(2^{0.5n})$ time barrier! Formally, we consider the following hybrid problem:

Problem 2: Either/Or Subset Sum (EOSS)

Input: A Subset Sum instance $X = \{x_1, x_2, \ldots, x_n\}, t$.
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 $C : \{0,1\}^n \to \{0,1\}^n$, 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 $C(X) : \{0,1\}^n \to \{0,1\}^n$ 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.

4.1 Results

The main algorithmic result of this section is as follows:

Theorem 4. *EOSS can be solved in time $O^*(2^{0.461n})$ with probability $1 - o_n(1)$ 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 Wegrzycki [MNPW19].

Our result could be considered as characterization of a large class of easy instances of Subset Sum (the complement of a class of easy Equal Subset Sum instances), or perhaps as a stepping stone toward an $O^*(2^{(0.5-c)n})$ algorithm for Subset Sum (i.e., if we do not solve Subset Sum in time $O^*(2^{(0.5-c)n})$, we have structural guarantees on the input evidenced by an Equal Subset Sum solution). However, the result might be best viewed as formally establishing a relationship between two problems: with respect to our best exact algorithmic techniques, apparently hard instances of Subset Sum are easy instances of Equal Subset Sum, and vice versa.

4.2 Techniques

The proof of our main result adapts the representation method to the worst-case setting by carefully handling unfriendly instances. The algorithm proceeds in three steps:

1. First, we use a modified Meet-in-the-Middle approach to solve the Subset Sum instance if there exists a small solution.
2. Second, we use a modified Meet-in-the-Middle approach to solve the Equal Subset Sum instance if there exists a small solution.
3. We prove that Subset Sum instances with few distinct small subset sums have a small Equal Subset Sum solution. Thus, if the previous step fails, it follows that our instance has many distinct small subset sums.
4. We use fact (3) to establish the correctness of a worst-case representation method approach.
   (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 (refer to Figure 3).
   (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 (see Figure 4). However, if this occurs, we have a solution to the Equal Subset Sum problem.

The algorithm (in simplified form) is reproduced in Figure 9. For full details, see [Ran23].

5 Beyond the Meet-in-the-Middle Barrier: Log Shaving for Subset Sum

This section introduces results from [CJRS23].

Given the longstanding difficulty of achieving a $2^{(1/2-c)n}$-time worst-case algorithm for Subset Sum, it is natural to consider the relaxed goal of achieving *some* nontrivial speedup

```
Procedure EitherOrSubsetSum($X, t$)

Input: An integer multiset $X = \{x_1, x_2, \dots, x_n\}$ and an integer target $t$.
Setup: Fix constants $\gamma = 0.15$, $\beta \approx 0.139$ satisfying $H^{-1}(1-2\beta) = 0.2$.

0. Solve the Subset Sum instance $(X, t)$ if there exists a solution of size at most $\gamma n$.
1. Solve the ESS instance $X$ if there exists a solution of size at most
   $H^{-1}(1-2\beta)n + o(n)$.
2. (Guess solution size by brute force.) For $|S| \in [\gamma n : n/2]$:
3.     Randomly pick a $(|S| - \beta n)$-bit prime $\boldsymbol{p}$ and repeat $poly(n)$ times:
4.         Select a residue class $\mathbf{r} \sim [\boldsymbol{p}]$ uniformly at random.
5.         Enumerate $L := \{Q \subseteq X : |Q| = |S|/2, \Sigma(Q) = \mathbf{r}\}$, and
                     $R := \{Q \subseteq X : |Q| = |S|/2, \Sigma(Q) = t - \mathbf{r}\}$.
           If either list has more than $O^*(2^{H(|S|/2n)n - \boldsymbol{p}})$ items, return to Item 4.
6.         Sort $L$ and $R$ by sum and return any ESS solutions.
7.         Run Meet-in-the-Middle on $L$ and $R$, returning any solution.
8. Return 'NO' (no Subset Sum solution).
```

Figure 9: The Either/Or Subset Sum algorithm.

over Meet-in-the-Middle. This section 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 or 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 the meet-in-the-middle algorithm for Subset Sum are analogous to "log shaving" improvements on the runtimes of well-known and simple polynomial-time algorithms for various problems which have resisted attempts at polynomial-factor improvements. There is a substantial strand of research along these lines (see [Cha13b, Cha13a] for a non-exhaustive overview); indeed, Abboud and Bringmann [AB18] have recently stated that: "A noticeable fraction of Algorithms papers in the last few decades improve the runtime of well-known algorithms for fundamental problems by logarithmic factors." In our setting, since the Meet-in-the-Middle algorithm runs in exponential time, saving a poly(n) factor is analogous to log shaving. Indeed, our first and most straightforward algorithm is based on “bit-packing” techniques that were used by Baran, Demaine, and Pătraşcu [BDP05] to shave log factors from the standard $O(n^2)$-time algorithm for the 3SUM problem.

5.1 Circuit RAM and Word RAM

A Subset Sum instance is parameterized by the number of inputs n and the size of the target value t (without loss of generality, $x_1, x_2, \ldots, x_n \le t$). Thus it is natural to adopt a memory model with word length $\ell = \Theta(\log t)$ such that each input integer can be stored in a single word (see, e.g., Pisinger's work on dynamic programming approaches for Subset Sum [Pis03]). 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 $\mathrm{poly}(n)$ and hence fits into a single $O(\log n)$-bit machine word.

This framework 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 $\mathrm{poly}(n)$-factor improvements over meet-in-the-middle; hence throughout the paper we assume $t = 2^{\Omega(n)}$. We make the assumption throughout the paper that the word size $n = \mathrm{poly}(n)$.

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 $\mathrm{poly}(n)$-size circuit with unbounded fan-in gates can be performed in time proportional to the depth of the circuit. 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.

5.2 Results and Techniques

As a preliminary result, we show how to implement Meet-in-the-Middle in time $O(2^{n/2})$ in the word and circuit RAM models, shaving an $O(n)$ extra factor which is traditionally absorbed by the O^* notation. Although a complete presentation of this result does not appear to exist elsewhere in the literature, it follows straightforwardly from folklore results and we believe it is implicit in prior work.

Proposition 1. *Carefully implemented, Meet-in-the-Middle runs in time $O(2^{n/2})$.*

However, we can improve this further:

Theorem 5. *There exists a zero-error randomized algorithm for the Subset Sum problem with expected runtime $O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$ in the circuit RAM model and $\widetilde{O}(2^{n/2} \cdot n^{-1/2})$ in the word RAM model.*

By halting and returning “NO” if runtime exceeds $C \cdot 2^{n/2} \cdot n^{-1/2} \cdot \log n$ for a large enough constant $C > 0$, we can get a one-sided error algorithm with success probability $3/4$.

```
Procedure BitPacking($X$, $t$):

Input: An integer multiset $X = \{x_1, x_2, \ldots, x_n\}$ and an integer target $t$.
Setup: Let $\boldsymbol{h}$ be a pseudolinear hash function that maps $O(n)$ bits to $O(\log(n))$ bits.

0. Fix any partition of $X = A \cup B \cup D$ such that $|D| = \log n$ and $|A| = |B| = \frac{n-|D|}{2}$.
1. Enumerate the set $w(D)$ and the sorted lists $L_A$, $L_B$ in linear time.
2. Create lists $\boldsymbol{h}(L_A)$ and $\boldsymbol{h}(L_B)$ by applying $\boldsymbol{h}$ element-wise to $L_A$ and $L_B$.
   Let $\boldsymbol{H}_A$ be the list obtained from $\boldsymbol{h}(L_A)$ by packing $O(n/\log(n))$ elements of
   $\boldsymbol{h}(L_A)$ into each $n$-bit word of $\boldsymbol{H}_A$, and likewise for $\boldsymbol{H}_B$.
3. For each $t' \in (t - w(D))$:
4.     Initialize indices $i := 0$ and $j := |\boldsymbol{H}_B| - 1$. While $i < |\boldsymbol{H}_A|$ and $j \geq 0$:
5.         If $\boldsymbol{H}_A[i]$ and $\boldsymbol{H}_B[j]$ contain a pair of hashes $(\boldsymbol{h}(a'), \boldsymbol{h}(b'))$
           such that $\boldsymbol{h}(a') + \boldsymbol{h}(b') \in \boldsymbol{h}(t') - \{0, 1\} \pmod{2^m}$, use
           Meet-in-the-Middle to search the appropriate portion of $L_A \times L_B$ for a
           solution $(a, b)$ with $a + b = t'$. Halt returning "yes" if a solution is found.
6.         If $L_A[(i+1)n/m - 1] + L_B[jn/m] < t'$, increment $i \leftarrow i + 1$.
           Otherwise, decrement $j \leftarrow j - 1$.
7. Return "NO" (i.e., no solution was found for any $t' \in (t - w(D))$).
```

Figure 10: The bit packing algorithm for log shaving, in simplified form. For full details, see [CJRS23].

Our first and simplest algorithm, sketched in simplified form in Figure 10,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 (an adaptation of) Meet-in-the-Middle on the lists of hashed and packed values. This results in a runtime savings over performing Meet-in-the-Middle on the original lists (without hashing and packing), because processing a pair of words, each containing 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 $d \in w(D)$, re-using the packed lists each time. Setting $|D| = \log(n)$ results in the best possible runtime.

For our final two algorithms, we omit algorithm figures for the sake of readability and refer the reader to the full paper [CJRS23].

Theorem 6. *There exists a randomized algorithm for the Subset Sum problem with one-sided error, success probability at least 1/3 and worst-case runtime $O(2^{n/2} \cdot n^{-\gamma})$ 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 $X = A \cup B \cup C$ 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 2.4), 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 necessary desirable properties, such as many distinct subset sums, by brute force in a preprocessing stage. 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 don't re-use any element of C. 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. Fortunately, we can consider Orthogonal Vectors on inputs of size at most $\varepsilon \log n$ as a Boolean function and tabulate all possible outputs in time $O(2^{\varepsilon n})$.

Theorem 7. *There exists a randomized algorithm for the Subset Sum problem with one-sided error, success probability at least 1/12, and with worst-case runtime $O(2^{n/2} \cdot n^{-(1/2+\gamma)})$ 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 residue classes modulo a random prime $\boldsymbol{p}$. 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 $\pmod{\boldsymbol{p}}$ that contains elements of $w(D)$. To solve this problem, we construct D in a way that ensures the elements of $w(D)$ fall into few residue classes $\pmod{\boldsymbol{p}}$. For full details, refer to [CJRS23].

6 Subset Sum Parameterized in the Doubling Constant

This section summarizes work from [RW23] as well as ongoing work.

The field of parameterized complexity tackles hard problems by solving instances whose structural complexity is captured by a certain numeric parameter. Usually, 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 the natural measure. Alternatively, parameters may be chosen because they effectively capture the complexity of the problem, or 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 comprehensive introductions to parameterized complexity.

The main obstacle to parameterizing Subset Sum is choosing the correct parameter. Searching for a small solution makes the problem easier in a predictable way,[5] but does not appear to yield interesting theoretical insights into the problem. However, there is one parameter frequently used in the field of additive combinatorics to measure additive structure. This is the *doubling constant*: for a set A of integers, we define

$$K := K(A) = \frac{|A+A|}{|A|}, \tag{2}$$

where $A + A$ denotes the set $\{a_1 + a_2 \ : \ a_1, a_2 \in A\}$. Formally, we can parameterize Subset Sum in the doubling constant as follows.

Problem 3: K-Subset Sum

Input: A subset sum instance $(X = \{x_1, x_2, \ldots, x_n\}, t)$ such that $|X + X| \leq K|X|$.
Output: $S \subseteq X$ such that $\Sigma(S) = t$, or 'NO' if no solution exists.

[5]Parameterizing by solution size k immediately yields algorithms that run in time $O(2^{H(2k)n/2})$ using Meet-in-the-Middle.

Sets with constant doubling have a variety of structural properties that make them desirable to work with. 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 the feasibility of binary integer programs.

6.1 Results and Techniques

Recall the definition of a Generalized Arithmetic Progression P as an integer set

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

defined by the integer vector $y = \{y_1, y_2, \ldots, y_d\}$ and the dimension bounds $L_1, L_2, \ldots, L_d$. We can think of P as a projection of a d-dimensional parallelepiped onto the line. Given such a GAP, by "P can be explicitly constructed (in time $f(n)$)", we mean that an algorithm can compute y_i and L_i for all $i \in [d]$. The results in this section involve several constructions; in general, when we write that an object can be constructed in time $O_K(f(n))$, we mean that in time $O_K(f(n))$ we can either write down a representation of the object explicitly or represent the object in such a way that any relevant information about it can be accessed in time $O_K(1)$. (Here, O_K notation suppresses factors that depend only on K.)

Our first result provides an algorithmic statement of a classic result from the field of additive combinatorics that allows us to efficiently transform instances of Subset Sum with constant doubling into an alternative representation. We prove a constructive variant of Freiman's Theorem: given X with $|X + X| \leq K|X|$, we show how to explicitly construct a GAP P of dimension $d(K)$ and volume $f(K)$, containing X, in time $O_K(\text{poly}(n))$. We achieve $d(K) = 2^{K^{O(1)}}$ and $v(K) = 2^{2^{K^{O(1)}}}$, as in the original statement of Freiman's Theorem.[6] We do not attempt to optimize these functions, but we expect that techniques used to optimize d and v in subsequent proofs of Freiman's Theorem (see [Cha02, Sch11, San12, San13]) could be used to improve the dependence on K in our results.

Theorem 8 (Freiman's Theorem is Fixed-Parameter Tractable in the Doubling Constant)**.** *Let A be a finite integer set with $|A + A| \leq K|A|$. There exist computable functions d, v, and f such that we can construct an arithmetic progression*

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

with dimension $d(K)$ and volume $v(K)|A|$ in time $f(K) \cdot \widetilde{O}(n) \cdot \text{polylog}(M)$, where M is the largest input integer, such that $P \supseteq A$ with probability $1/2$.

Our proof follows Zhao's exposition of Freiman's Theorem [Zha22], making each component constructive in FPT time. The existing results to which we refer are well-known in additive combinatorics; for a general reference, see [TV06], and for precise statements of the results proved, refer to [RW23]. We prove the following fixed-parameter variants on the way to the proof that Freiman's Theorem is FPT:

[6] The original proof of the theorem can be found in [Fre64]. For a modern presentation, see [Zha22].

1. **Ruzsa's Covering Lemma.** Given input subsets A and B, the subset $T \subseteq B$ guaranteed by Ruzsa's Covering Lemma can be computed in FPT time $\widetilde{O}(K|A - A + B|)$ (this was previously shown by [AB18]).

2. **Ruzsa's Modeling Lemma**. Given a finite set A and an integer $m \geq 4|sA - sA|$ for some integer $s \geq 2$, we can construct an s-Freiman isomorphism from $\mathbb{Z}$ to $\mathbb{Z}/m\mathbb{Z}$ satisfying the conditions of Ruzsa's Modeling Lemma in FPT time $\widetilde{O}(polylog(max(A)) + |A|)$.

3. **Bogolyubov's Lemma in $\mathbb{Z}/m\mathbb{Z}$** ([Zha22] Theorem 7.8.5). Given $A \subseteq \mathbb{Z}/m\mathbb{Z}$ with $|A| = \alpha m$, we can compute a Bohr set satisfying the conditions of Bogolyubov's Lemma in $\mathbb{Z}/m\mathbb{Z}$ in FPT time $\widetilde{O}(|A|)$.

4. **Finding a large GAP in a Bohr set** ([Zha22] Theorem 7.10.1). 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 $P \subset Bohr(R, \varepsilon)$ with dimension at most d and volume at least $(\varepsilon/d)^d$ in FPT time $\widetilde{O}_d(|A|)$.

This result gives us a powerful tool: given an instance of Subset Sum with inputs bounded by M and doubling constant K, we can construct a GAP P with $vol(P) = O_K(|A|)$ and dimension $dim(P) = O_K(1)$ using Freiman's Theorem in time $O_K(n \cdot \text{polylog}(n, M))$.

This "GAP representation" of the subset sum instance allows us to write each input element in terms of the parameters of the GAP, as a $d(K)$-tuple with strongly bounded components (Freiman's bound on the volume of the GAP implies that the product of the components is $O_K(n)$). The $d(K)$-tuple is not unique, but further results exist that contain a given GAP within a *proper* GAP (e.g., [TV06] 3.40); a fixed-parameter version of this result would give us unique representations of the subset sum input elements. The problem that results bears similarities both to vector sum and Binary Integer Linear Programming feasibility, as we discuss in the next section.

6.2 Ongoing Work

In further work, we hope to show a tight connection between Subset Sum and Integer Linear Programming feasibility, another well-studied problem in parameterized complexity.

Problem 4: Integer Linear Programming (ILP) Feasibility

In: An integer linear program of the form $\{Ax = b, x \geq 0\}$ specified by the integer matrix $A \in \mathbb{Z}^{m \times n}$ and the integer vector $b \in \mathbb{Z}^m$.
Out: A vector $x \in \mathbb{Z}^n_{\geq 0}$ such that $Ax = b$, or 'NO' if no solution exists.

The variant problem *Binary* ILP feasibility restricts the solution vector x to the set $\{0, 1\}^n$. Previous work on parameterized algorithms for ILP feasibility solves the problem in time parameterized by m and the magnitudes $\Delta := ||A||_\infty$, $||b||_\infty$ of the largest entries of A and b, respectively [JR18, KPW20]:

Theorem 9 ([JR18] Theorem 6)**.** *ILP Feasibility can be solved in time $O(m\Delta)^m \cdot log(\Delta) \cdot \log(\Delta + ||b||_\infty)$.*

The GAP formulation of the Subset Sum problem described in the previous section appears structurally similar to ILP feasibility. In ongoing work, we are investigating the connection between the two. If it were possible to transform an instance of Unbounded Subset Sum into an appropriately-sized ILP feasibility instance, this would show that Unbounded Subset Sum, the variant in which the coefficient set $C = \mathbb{N}$, is FPT in the doubling constant.

Reductions in the other direction also appear plausible. Indeed, we can show that Binary ILP feasibility can be FPT-reduced to Subset Sum:

Proposition 2. *If we can solve K-Subset Sum in time $O_K(\text{poly}(n))$, we can solve Binary ILP feasibility instances $\{Ax = b,\ x \in \{0,1\}^n\}, A \in \mathbb{Z}^{m\times n}$ in time $O_m(\Delta^{O(m)})$.*

Whether or not Binary ILP feasibility can be solved in time $O_m(\Delta^{O(m)})$ is an open problem. It would be interesting to show a converse reduction, which would imply that Subset Sum is FPT in the doubling constant if and only if binary ILP feasibility can be solved in time $O_m(||A||_\infty^{O(m)})$.

7 Future Work

The research projects outlined in Sections 3 to 5 are largely complete, while work on Section 6 is still ongoing. Over the next year, I plan to focus my thesis research on the following areas:

1. **Parameterized Algorithms for Subset Sum.** Notwithstanding the progress described in Section 6, the project of linking the Subset Sum problem to well-studied problems in parameterized complexity is still incomplete. Parameterization in the doubling constant has already proved useful, in the sense that it allows many tools from additive combinatorics to apply to the problem, but it would be more satisfying if this lead to concrete tractability results or stronger links to existing problems. This line of work is currently ongoing and will determine the contents of Section 6.

2. **Improved Log Shaving and Shaving Beyond Logs.** The tools described in Section 5 may not lead to further progress in log shaving, but other approaches remain to be tried. In particular, I would like to better understand the contouring approach described in [GP18]. Progress on this front would expand Section 5.

3. **Confronting the Meet-in-the-Middle Barrier.** A valid objection to the project of log shaving for Subset Sum is that, unlike certain other conjectures in fine-grained complexity, there is no compelling evidence that the Meet-in-the-Middle barrier is unbreakable (except lack of progress). I am currently interested in an approach that attempts to reduce Subset Sum to node-weighted triangle detection, a problem for which there are nontrivial algorithms based on fast matrix multiplication.

I do not expect to break the Meet-in-the-Middle barrier by an exponential margin, but given the nature of this thesis, it seems appropriate to give the problem a good try. Substantial progress on this problem would substantially change the thesis, adding a new section and diminishing the significance of Section 5.

8 Research Plan

Table 2 summarizes my plan for completion of my thesis.

Task	**Timeline**
Average-Case Algorithms for Subset Sum and Equal Subset Sum	Completed
Complementarity of Subset Sum and Equal Subset Sum	Completed
Log Shaving for Subset Sum	Main body completed Extensions wrapped by Feb '24
Subset Sum Parameterized in the Doubling Constant	Project stable by Aug '23 Extensions wrapped by Feb '24
Breaking the Meet-in-the-Middle Barrier	Fall '23 & winter '24, time permitting Wrapped by spring '24
Thesis Writing	Winter '24 Edits wrapped by spring '24
Thesis defense	May '24

Table 2: Thesis completion timeline

References

[AB18] Amir Abboud and Karl Bringmann. Tighter Connections Between Formula-SAT and Shaving Logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 5, 1

[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*, pages 45–56. Springer, 2013. 1

[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. 1, 2.3

[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. 1, 2.3

[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. 1, **??**

[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. 1, 2, 2.3, 3, **??**

[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. 3, 3.1, 3.2.1

[BDP05] Ilya Baran, Erik D Demaine, and Mihai Patraşcu. Subquadratic algorithms for 3SUM. In *Workshop on Algorithms and Data Structures*, pages 409–421. Springer, 2005. 5, 5.1, 5.2

[Bel66] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. 5.1

[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. 1

[Böh11] E. Böhme. Verbesserte subset-sum algorithmen, 2011. 1, ??

[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. 2.3

[CFK+15] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. 6

[Cha02] Mei-Chu Chang. A polynomial bound in freiman's theorem. *Duke Math. J.*, 115(1):399–419, 2002. 6.1

[Cha13a] Timothy Chan. The art of shaving logs. Presentation at WADS 2013, slides available at https://tmc.web.engr.illinois.edu/talks/wads13_talk.pdf, 2013. 5

[Cha13b] Timothy M. Chan. The art of shaving logs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, volume 8037 of *Lecture Notes in Computer Science*, page 231. Springer, 2013. 5

[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. 1, 2.5.1, 3, 8

[CJRS23] Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset sum in time $2^{n/2}/poly(n)$, 2023. *a*, 2.5.3, 5, 10, 5.2, 5.2

[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. 1

[DF12] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012. 6

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. 2.1

[Fre64] Gregory A Freiman. On the addition of finite sets. In *Doklady Akademii Nauk*, volume 158, pages 1038–1041. Russian Academy of Sciences, 1964. 6

[GP18] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. 2

[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. 1, 2.3, 2.4, 2.4, 2.4, 3, **??**

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974. 1, 1, 2.3

[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. 6.2, 9

[Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. 1

[KK82] Narendra Karmarkar and Richard M Karp. *The differencing method of set partitioning*. Computer Science Division (EECS), University of California Berkeley, 1982. 1

[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. 6.2

[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. 3

[MNPW19] Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Wegrzycki. Equal-subset-sum faster than the 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. 1, 3.1, **??**, 4.1

[NW21] Jesper Nederlof and Karol Wegrzycki. Improving Schroeppel and Shamir's algorithm for subset sum via orthogonal vectors. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. 1, 2.3

[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. 4

[Pis03] David Pisinger. Dynamic programming on the word RAM. *Algorithmica*, 35(2):128–145, 2003. 5.1

[Ran23] Tim Randolph. A hybrid algorithm for subset sum and equal subset sum. 2023. 2.5.2, 4, 4.2

[RW23] Tim Randolph and Karol Wegryzcki. Subset sum with constant doubling. 2023. 6, 6.1

[San12] Tom Sanders. On the bogolyubov–ruzsa lemma. *Analysis & PDE*, 5(3):627–655, 2012. 6.1

[San13] Tom Sanders. The structure theory of set addition revisited. *Bulletin of the American Mathematical Society*, 50(1):93–127, 2013. 6.1

[Sch11] Tomasz Schoen. Near optimal bounds in freiman's theorem. *Duke Mathematical Journal*, 158(1):1–12, 2011. 6.1

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM journal on Computing*, 10(3):456–464, 1981. 1

[TV06] Terence Tao and Van H Vu. *Additive combinatorics*, volume 105. Cambridge University Press, 2006. 6.1, 6.1

[Woe08] Gerhard J Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008. 1, 2.3

[Zha22] Yufei Zhao. Graph theory and additive combinatorics. *Notes for MIT*, 18:49–58, 2022. 6.1, 6, 3, 4