Faster Pseudopolynomial Time Algorithms for Subset Sum

KONSTANTINOS KOILIARIS, University of Illinois, Urbana-Champaign, USA
CHAO XU, Yahoo! Research, USA

Given a (multi) set S of n positive integers and a target integer u, the subset sum problem is to decide if there is a subset of S that sums up to u. We present a series of new algorithms that compute and return *all* the realizable subset sums up to the integer u in $\tilde{O}(\min\{\sqrt{n}u, u^{5/4}, \sigma\})$, where σ is the sum of all elements of S and $\tilde{O}$ hides polylogarithmic factors. We also present a modified algorithm for integers modulo m, which computes all the realizable subset sums modulo m in $\tilde{O}(\min\{\sqrt{n}m, m^{5/4}\})$ time.

Our contributions improve upon the standard dynamic programming algorithm that runs in $O(nu)$ time. To the best of our knowledge, the new algorithms are the fastest deterministic algorithms for this problem. The new results can be employed in various algorithmic problems, from graph bipartition to computational social choice. Finally, we also improve a result on covering $\mathbb{Z}_m$, which might be of independent interest.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**;

Additional Key Words and Phrases: Subset sum, convolution

ACM Reference format:
Konstantinos Koiliaris and Chao Xu. 2019. Faster Pseudopolynomial Time Algorithms for Subset Sum. *ACM Trans. Algorithms* 15, 3, Article 40 (June 2019), 20 pages.
https://doi.org/10.1145/3329863

1 INTRODUCTION

Given a (multi) set S of n positive integers and an integer target value u, the *subset sum problem* is to decide if there is a subset of S that sums up to u. The subset sum is a special case of the knapsack problem [14], and it is one of Karp's original NP-complete problems [32]. The subset sum has a variety of applications, including: power indices [54], scheduling [23, 24, 45], set-based queries in databases [53], breaking precise query protocols [15], and various other graph problems with cardinality constraints [9, 10, 16, 17, 25, 34] (for a survey of further applications see Reference [33]). In some of the applications, a faster pseudopolynomial time algorithm for the subset sum would imply faster *polynomial time* algorithms.

An extended abstract of this paper appeared in SODA 2017 [35] and one algorithm appeared in an earlier manuscript [36]. This full version among other additions simplifies the $\tilde{O}(\sqrt{n}u)$ time algorithm and improves the bound from $\tilde{O}(u^{4/3})$ to $\tilde{O}(u^{5/4})$ time.
Work done in part while C. Xu attending University of Illinois, Urbana-Champaign.
Authors' addresses: K. Koiliaris, University of Illinois, Urbana-Champaign, 201 N Goodwin Ave, Urbana, IL, 61801; email: koiliar2@illinois.edu; C. Xu, Yahoo! Research, 770 Broadway, Floor 6, New York, NY, 10003; email: chao.xu@verizonmedia.com.

1549-6325/2019/06-ART40 $15.00
https://doi.org/10.1145/3329863

Table 1. Summary of *Deterministic* Pseudopolynomial Time Results on the Subset Sum Problem

Result	Time	Space	Comments
Bellman [5]	$O(nu)$	$O(u)$	original DP solution
Pisinger [44]	$O\left(\frac{nu}{\log u}\right)$	$O\left(\frac{u}{\log u}\right)$	RAM model implementation of Bellman
Pisinger [43]	$O(n \max(S))$	$O(u)$	fast if small $\max(S)$
Faaland [19], Pferschy [42]	$O(n'u)$	$O(u)$	fast for small n'
Klinz et al. [34]	$O(\sigma^{3/2})$	$O(u)$	fast for small σ, obtainable from above because $n' = O\left(\sqrt{\sigma}\right)$
Eppstein [17], Serang [48, 49]	$\tilde{O}(n \max(S))$	$O(u \log u)$	data structure
Lokshtanov et al. [38]	$\tilde{O}(n^3 u)$	$\tilde{O}(n^2)$	polynomial space
current work	$\tilde{O}(\min\{\sqrt{n'}\, u, u^{5/4}, \sigma\})$ Thm. 1.2	$O(u)$	see Section 1.1

The input S is a (multi) set of n numbers and n' distinct values, $\sigma = \sum_{x \in S} x$ and u is the target number.

The subset sum is a fundamental problem used as a standard example of a problem that can be solved in weakly polynomial time in many undergraduate algorithms and complexity classes. As a weakly NP-complete problem, there is a standard pseudopolynomial time algorithm using a dynamic programming due to Bellman [5], who solves it in $O(nu)$ time (see also [13, Chapter 34.5]). There has been extensive work done on the subset sum problem since: see Table 1 for a summary of previous deterministic pseudopolynomial time results [5, 19, 34, 38, 42–44, 48, 49]. The current state-of-the-art had until recently been improved only by a $\log u$ factor using the bit-packing technique of Reference [44].

Further results on subset sum depend on properties of the input, while others focus on data structures that maintain the set of subset sums under standard operations. In particular, when the maximum value of any integer in S is relatively small compared to the number of elements n, and the target value u lies close to one-half the total sum of the elements, then one can solve the subset sum problem in almost linear time [21, 22]. This result was improved by Chaimovich [11]. Eppstein [17] described a data structure that efficiently maintains all subset sums up to a given value u, under insertion and deletion of elements, in $O(u \log u \log n)$ time per update, which can be accelerated to $O(u \log u)$ when additional information about future updates is known. The probabilistic convolution tree, by Serang [48, 49], is also able to solve the subset sum problem in $\tilde{O}(n \max(S))$ time, where $\tilde{O}$ hides polylogarithmic factors; i.e., $\tilde{O}(T) = O(T \operatorname{polylog}(T))$.

A variant of the subset sum problem, known as the *modular subset sum*, has been especially studied in the literature of additive combinatorics [2, 18, 26, 40, 41, 51, 52, 55]. In this problem, all additions are modulo m, for some input m. Despite its popularity in combinatorics, there is no prior work on establishing efficient algorithms for it. Note, though, that the dynamic programming algorithm of Bellman [5] applies to it and solves the modular subset sum in $O(nm)$ time.

The publication of the extended abstract of this article [35] sparked a series of works on randomized pseudopolynomial time algorithms for the subset sum; we briefly survey them here. Bringmann [8] showed a randomized near-linear time algorithm $\tilde{O}(n + u)$ and a different one that runs in $\tilde{O}(nu)$ time, using only $\tilde{O}(n \log u)$ space under the Extended Riemann Hypothesis. Recently, Jin and Wu showed a simpler randomized algorithm that achieves a slightly better running time [31]. Both randomized algorithms use FFT and improve upon the best deterministic running time achieved in this article. The question of derandomizing either result remains an open problem.

Table 2. Our Contributions Compared Against the Previous Best *Deterministic* Results

Parameters	Previous best	Current work
n and u	$O(nu/\log u)$ [44]	$\tilde{O}(\min\{\sqrt{n}u,\ u^{5/4}\})$
n' and u	$O(n'u)$ [19, 42]	$\tilde{O}(\min\{\sqrt{n'}u,\ u^{5/4}\})$
σ	$O(\sigma^{3/2})$ [34]	$\tilde{O}(\sigma)$

The input S is a (multi) set of n numbers and n' distinct values, $\sigma = \sum_{x \in S} x$ and u is the target number.

Axiotis et al. showed a randomized near-linear algorithm for the modular subset sum with running time $\tilde{O}(n + m)$ through a clever use of sketching [4]. Surprisingly, their algorithm does not depend on FFT. In addition, they observed that the randomized algorithms of Reference [8] do not apply to the modular case. In fact, neither does the algorithm of Reference [31].

Finally, Abboud et al. [1] showed that it is unlikely that any subset sum algorithm runs in $O(u^{1-\varepsilon}\, 2^{o(n)})$ time for any constant $\varepsilon > 0$ and target number u, as such an algorithm would imply that the Strong Exponential Time Hypothesis (SETH) of Impagliazzo and Paturi [30] is false.

1.1 Our Contributions

The new results are summarized in Table 2—we consider the following *all subset sums* problem: Given a (multi) set S of n elements, with n' distinct values, with σ denoting the total sum of its elements, compute *all* the realizable subset sums up to a prespecified integer u. Computing all subset sums for some u also answers the standard subset sum problem with any target value less than or equal to u.

Our main contribution is a new algorithm for computing the all subset sums problem, and consequently for the subset sum problem, in $\tilde{O}(\min\{\sqrt{n}u, u^{5/4}, \sigma\})$ time. The new algorithm improves over all previous deterministic works (see Table 2). The general algorithm is a result of combining multiple approaches based on different properties of the input. As such, we have developed a series of algorithms that work well for different inputs, which we believe might be of independent interest, as they can be used as building blocks for other problems. For a high-level description, see Table 3. As part of the above, we introduce an algorithm with running time $\tilde{O}(\sqrt{n}u)$ that is surprisingly simple compared to the conference version of Reference [35]. We believe the new algorithm can be used in teaching as a simple example of a pseudopolynomial time algorithm for the subset sum problem, as well as a striking example of applying FFT to a seemingly unrelated problem.

Our second contribution is an algorithm that solves the *modular all subset sums* problem in $\tilde{O}(\min\{\sqrt{n}m, m^{5/4}\})$ time. Though the time bound is superficially similar to the first algorithm, this algorithm uses a different approach.

Both algorithms can be augmented to return the solution; i.e., the subset summing up to each number, with a polylogarithmic slowdown (see Section 6 for details).

We also improve the running time of a number of applications. For instance, the bottleneck graph partition problem on weighted graphs. This problem asks to split the vertices of a graph into two equal-sized sets, such that the value of the bottleneck (maximum-weight) edge, over all edges across the cut, is minimized. Another example is the computation of power indices in computational social choice. In both cases, we obtain significant improvements in running time. Finally, we improve the current best bound for the size of a covering set of limited-magnitude errors.

Table 3. Summary of All New Algorithms

Algorithm	Input	Output	Running time
SS_SmallInput	S, u	$\mathcal{S}_u(S)$	$\tilde{O}(\sqrt{nu})$
↳ SS_SmallSum	S	$\mathcal{S}(S)$	$\tilde{O}(\sigma)$
↳ SSC_BoundSum	S, u	$\mathcal{SC}_u(S)$	$\tilde{O}(un)$
SS_LargeInput	S, u	$\mathcal{S}_u(S)$	$\tilde{O}(u^{5/4})$
↳ SS_SmallMax	S	$\mathcal{S}(S)$	$\tilde{O}(\max(S)^{5/3})$
↳ SS_LargeMin	S, u	$\mathcal{S}_u(S)$	$\tilde{O}(u^2/\min(S))$
↳ SSC_BoundCard	S, v	$\mathcal{SC}^v(S)$	$\tilde{O}(\mathrm{diam}(S)v^2)$
SS_Mod	S, m	$\mathcal{S}(S)$ **under mod** m	$\tilde{O}(\min\{\sqrt{nm}, m^{5/4}\})$

The arrows denote algorithmic dependencies between the subroutines. The prefix **SS_** denotes the algorithm returns the set of all subset sums (Equation (†)) and **SSC_** return the set of all subset sums with their cardinalities (Equation (⋆)). The input set S has n elements, and $\sigma = \sum_{x \in S} x$.

1.2 Main Theorems

The following theorems capture our contributions.

Theorem 1.1. *(Main Theorem) Let $S \subseteq \{0, 1, \ldots, u\}$ be a set of n elements, with total sum σ, computing the set of all subset sums takes*

$$O\Big(\min\left\{\sqrt{nu}, u^{5/4}, \sigma\right\} \log^2 u\Big)$$

time.

The above result extends to the case that the input is a *multiset* as follows:

Theorem 1.2 (Main Theorem (Multiset)). *Let $S \subseteq \{0, 1, \ldots, u\}$ be a* multiset *of n' distinct elements, with total sum σ, computing the set of all subset sums takes*

$$O\Big(\min\left\{\sqrt{n'}u, u^{5/4}, \sigma\right\} \log^2 u\Big)$$

time.

The next result captures our contributions on the modular subset sum:

Theorem 1.3 (Main Theorem (Modular)). *Let $S \subseteq \{0, 1, \ldots, m-1\}$ be a set of size n, computing the set of all subset sums modulo m takes*

$$O\Big(\min\left\{\sqrt{nm}, m^{5/4}\right\} \log^2 m\Big)$$

time.

1.3 Sketch of Techniques

The straightforward divide-and-conquer algorithm for solving the subset sum problem [29] partitions the set of numbers into two sets, recursively computes their subset sums, and combines them together using FFT [17, 48, 49] (Fast Fourier Transform [13, Chapter 30]). This algorithm has a running time of $O(\sigma \log \sigma \log n)$ and is shown in Theorem 3.1.

Sketch of the $\tilde{O}(\sqrt{nu})$ Time Algorithm. Partition S into sets $S_i = \{x \in S \mid x \equiv i \pmod b\}$ for some fixed b, compute the set of subset sums (up to u) for each S_i, and combine them together. Since all the elements in the sets S_i are of the form $i + kb$ for some k, each element can be represented by the numbers k and i. Therefore, one can compute the subset sums of $S_i' = \{k \mid i + kb \in S_i\}$ maintaining the number of elements participating in the sums, and then recover S_i from it.

Sketch of the $\tilde{O}(u^{5/4})$ Time Algorithm. For this algorithm, we combine two new algorithms: One that is fast when $\max(S)$ is small, and one that is fast when $\min(S)$ is large. In particular, when $\max(S)$ is small, we employ tools from number theory [21] to handle most instances, while for the remaining ones we apply the $\tilde{O}(\min(\sigma, \sqrt{nu}))$ time algorithms mentioned above. When $\min(S)$ is large, one can compute the subset sums quickly by ignoring most of the sums that exceed the upper bound u.

Sketch of the $\tilde{O}(\min\{\sqrt{nm}, m^{5/4}\})$ Time Algorithm. Assume m is a prime number. Using known results from number theory, we show that for any ℓ, one can partition the input set into $\tilde{O}(|S|/\ell)$ subsets, such that every such subset is contained in an arithmetic progression of the form $x, 2x, \ldots, \ell x$. The subset sums for such a set can be quickly computed by dividing and later multiplying the numbers by x. Then combine all these subset sums to get the result. Sadly, m is not always prime. Fortunately, all the numbers that are relative prime to m can be handled in the same way as above. For the remaining numbers, we use a recursive partition classifying each number, in a sieve-like process, according to which prime factors it shares with m. In the resulting subproblems, all the numbers are coprime to the moduli used, and as such the above algorithm can be used. Finally, the algorithm combines the subset sums of the subproblems.

1.4 Paper Organization

In Section 2, we provide preliminaries, including a discussion on how to consider sets over multisets without loss of generality. In Section 3, we describe the $\tilde{O}(\sigma)$ and $\tilde{O}(\sqrt{nu})$ time algorithms for finding all subset sums up to u. Section 4 covers the $\tilde{O}(u^{5/4})$ algorithm. Section 5 describes the $\tilde{O}(\min\{\sqrt{nm}, m^{5/4}\})$ time algorithm for the *modular* subset sum. In Section 6, we show how one can recover the solutions; i.e., retrieve the elements summing to each sum. Finally, Section 7 presents the impact of the results on selected applications of the problem.

2 PRELIMINARIES

Let $[x..y] = \{x, x+1, \ldots, y\}$ denote the set of integers in the interval $[x, y]$. Similarly, $[x] = [0..x]$. For a set $X \subseteq \mathbb{N}$, denote its *diameter* by $\operatorname{diam}(X) = \max\{|x - y| \mid x, y \in X\} = \max(X) - \min(X)$. For two sets $X, Y \subseteq \mathbb{N}$, let $X \oplus Y = \{x + y \mid x \in X \text{ and } y \in Y\}$, $X \oplus_u Y = (X \oplus Y) \cap [u]$, $X \otimes Y = \{xy \mid x, \in X, y \in Y\}$, and $x \otimes Y = \{xy \mid y \in Y\}$.

If $W, Z \subseteq \mathbb{N} \times \mathbb{N}$, then $W \oplus Z = \{(w_1 + z_1, w_2 + z_2) \mid w_1, w_2 \in W \text{ and } z_1, z_2 \in Z\}$, additionally define $W \oplus_u Z = (W \oplus Z) \cap ([u] \times \mathbb{N}) = \{(a, b) \mid a \leq u, (a, b) \in W \oplus Z\}$, and $W \oplus^v Z = (W \oplus Z) \cap (\mathbb{N} \times [v]) = \{(a, b) \mid b \leq v, (a, b) \in W \oplus Z\}$.

Given a set $X \subseteq \mathbb{N}$, let $\Sigma_X = \sum_{x \in X} x$. We define the *set of all subset sums of X* by

$$\mathcal{S}(X) = \{\, \Sigma_Y \mid Y \subseteq X \,\}, \tag{$\dagger$}$$

and the *set of all subset sums of X with their cardinalities* by

$$\mathcal{SC}(X) = \{\, (\Sigma_Y, |Y|) | Y \subseteq X \,\}. \tag{$\star$}$$

Moreover, we provide some additional auxiliary definitions:

- The *set of all subset sums of X up to u*:

$$\mathcal{S}_u(X) = \mathcal{S}(X) \cap [u].$$

- The *set of all subset sums of X up to u with their cardinalities*:

$$\mathcal{SC}_u(X) = \mathcal{SC}(X) \cap ([u] \times \mathbb{N}).$$

- The *set of all subset sums of X with cardinalities up to v*:

$$\mathcal{SC}^{v}(X) = \mathcal{SC}(X) \cap (\mathbb{N} \times [v]).$$

Observe, that if X and Y are two disjoint sets, then $\mathcal{S}(X \cup Y) = \mathcal{S}(X) \oplus \mathcal{S}(Y)$ and $\mathcal{SC}(X \cup Y) = \mathcal{SC}(X) \oplus \mathcal{SC}(Y)$. This simple observation also holds for the three supplementary definitions.

We note a simple observation that is used repeatedly in the analysis of running times.

Observation 1. *Let g be a positive, superadditive (i.e., $g(x + y) \geq g(x) + g(y), \forall x, y$) function. For a function $f(n, m)$ satisfying*

$$f(n, m) = \max_{m_1 + m_2 = m} \left\{ f\left(\frac{n}{2}, m_1\right) + f\left(\frac{n}{2}, m_2\right) + g(m) \right\},$$

we have that $f(n, m) = O(g(m) \log n)$.

Finally, in this work, we consider all computation under the word RAM model [20]. The word size is at least $\log L$, where L is the length of the input when all integers are expressed in unary. All operations on a single word can be completed in constant time.

2.1 Useful Tools

The following well-known lemmas describe how to compute pairwise sums between sets in almost linear time, in the size of their ranges, using FFT.

Lemma 2.1. *Given two sets $S, T \subseteq [u]$, one can compute $S \oplus T$ in $O(u \log u)$ time.*

Proof. Let $f_S = f_S(x) = \sum_{i \in S} x^i$ be the characteristic polynomial of S. Construct, in a similar fashion, the polynomial f_T (for the set T) and let $g = f_S * f_T$. Observe that for $i \leq u$, the coefficient of x^i in g is nonzero if and only if $i \in S \oplus T$. Using FFT, one can compute the polynomial g in $O(u \log u)$ time, and extract $S \oplus T$ from it. □

Lemma 2.2. *Given two sets of points $S, T \subseteq [u] \times [v]$, one can compute $S \oplus T$ in $O(u\, v \log(u\, v))$ time.*

Proof. As in Lemma 2.1, let $f_S = f_S(x, y) = \sum_{(i,j) \in S} x^i y^j$ and f_T be the characteristic polynomials of S and T, respectively, and let $g = f_S * f_T$. For $i \leq u$ the coefficient of $x^i y^j$ is nonzero if and only if $(i, j) \in S \oplus T$. One can compute the polynomial g by a straightforward reduction to regular FFT (see multidimensional FFT [6, Chapter 12.8]), in $O(u\, v \log(u\, v))$ time, and extract $S \oplus T$ from it. □

2.2 From Multisets to Sets

Here, we show that the case where the input is a multiset can be reduced to the case of a set. The reduction idea is somewhat standard (see [33, Section 7.1.1]) and first appeared in Reference [37]. We present it here for completeness.

For an element s in a multiset S, its *multiplicity* in S is denoted by $\chi_S(s)$. We denote by $\text{set}(S)$ the set of *distinct elements* appearing in the multiset S. The *size* of a multiset S is the number of distinct elements in S (i.e., $|\text{set}(S)|$). The *cardinality* of S, is $\text{card}(S) = \sum_{s \in S} \chi_S(s)$. We denote that a multiset S has all its elements in the interval $[x..y]$ simply by $S \subseteq [x..y]$.

Lemma 2.3. *Given a multiset S of integers, and a number $s \in S$, with $\chi_S(s) \geq 3$. Consider the multiset S' resulting from removing two copies of s from S, and adding the number $2s$ to it. Then, $\mathcal{S}_u(S) = \mathcal{S}_u(S')$. Observe that* $\text{card}(S') = \text{card}(S) - 1$.

Proof. Consider any multiset $T \subseteq S$. If T contains two or more copies of s, then replace two copies by a single copy of $2s$. The resulting subset is $T' \subseteq S'$, and $\Sigma_T = \Sigma_{T'}$, establishing the claim. □

Lemma 2.4. *Given a multiset S of integers in $[u]$ of cardinality n with n' unique values, one can compute, in $O(n' \log^2 u)$ time, a multiset T, such that:*

(1) $\mathcal{S}_u(S) = \mathcal{S}_u(T)$,
(2) $\text{card}(T) \leq \text{card}(S)$,
(3) $\text{card}(T) = O(n' \log u)$, *and*
(4) each element in T has multiplicity at most two.

Proof. Copy the elements of S into a working multiset X. Maintain the elements of $\text{set}(X)$ in a heap D, and let T initially be the empty set. In each iteration, extract the minimum element x from the heap D. If $x > u$, then we stop.

If $\chi_X(x) \leq 2$, then delete x from X and add x, with its appropriate multiplicity, to the output multiset T, and continue to the next iteration.

If $\chi_X(x) > 2$, then delete x from X, add x to the output set T (with multiplicity one), insert the number $2x$ into X with multiplicity $m' = \lfloor (\chi_X(x) - 1)/2 \rfloor$ (updating also the heap D—by adding $2x$ if it is not already in it), and set $\chi_X(x) \leftarrow \chi_X(x) - 2m'$. The algorithm now continues to the next iteration.

At any point in time, we have that $\mathcal{S}_u(S) = \mathcal{S}_u(X \cup T)$, and every iteration takes $O(\log u)$ time, and and as such overall, the running time is $O(\text{card}(T) \log u)$, as each iteration increases $\text{card}(T)$ by at most two. Finally, notice that every element in T is of the form $2^i x, x \in S$ for some i, where $i \leq \log u$, and thus $\text{card}(T) = O(n' \log u)$. □

Combining the above, we can now state the following lemma, which simplifies the upcoming analysis:

Lemma 2.5. *Given an algorithm that computes $\mathcal{S}_u(S)$ in $\mathrm{T}(n, u) = \Omega(u \log^2 u)$ time, for any* set *$S \subseteq [u]$ with n elements, then one can compute $\mathcal{S}_u(S')$ for any* multiset *$S' \subseteq [u]$, with n' distinct elements, in $O(\mathrm{T}(n' \log u, u))$ time.*

Proof. First, from S, compute the multiset T as described in Lemma 2.4, in $O(u \log^2 u)$ time. As every element in T appears at most twice, partition it into two *sets* P and Q. Then $\mathcal{S}_u(T) = \mathcal{S}_u(P) \oplus_u \mathcal{S}_u(Q)$, which is computed using Lemma 2.1, in $O(u \log u)$ time. This reduces all subset sums for multisets of n' distinct elements to two instances of all subset sums for sets of size $O(n' \log u)$. □

This section shows there is little loss in generality and running time if the input is restricted to sets instead of multisets. For simplicity of exposition, we assume the input is a *set* from here on.

3 SIMPLE IMPROVEMENTS TO FINDING $\mathcal{S}_u(S)$

In this section, we introduce the $\tilde{O}(\sigma)$ and $\tilde{O}(\sqrt{n}u)$ time algorithms for subset sum (Figure 1). Both are simple and succinct in terms of description and analysis.

First, we show how $\mathcal{S}(S)$ can be solved in $\tilde{O}(\sigma)$ time:

Theorem 3.1. *Given a set of n positive integers S with total sum σ, one can compute the set of all subset sums $\mathcal{S}(S)$ in $O(\sigma \log \sigma \log n)$ time.*

Proof. Partition S into two sets L, R of (roughly) equal cardinality, and compute recursively $L' = \mathcal{S}(L)$ and $R' = \mathcal{S}(R)$. Next, compute $\mathcal{S}(S) = L' \oplus R'$ using Lemma 2.1. The recurrence for the running time is $f(n, \sigma) = \max_{\sigma_1 + \sigma_2 = \sigma} \{f(n/2, \sigma_1) + f(n/2, \sigma_2) + O(\sigma \log \sigma)\}$, and the solution to this recurrence, by Observation 1, is $O(\sigma \log \sigma \log n)$. □

The standard divide-and-conquer algorithm of Theorem 3.1 was already known in References [17, 49]. Here, we showed a better analysis. Note that the basic divide-and-conquer algorithm without the FFT addition was known much earlier [29].

```
SS_SmallInput(S, m):
  b ← ⌊√(n log n)⌋
  for ℓ ∈ [b − 1]:
      S_ℓ ← S ∩ {x ∈ ℕ | x ≡ ℓ (mod b)}
      Q_ℓ ← {⌊x/b⌋ | x ∈ S_ℓ}
      SC_{u/b}(Q_ℓ) ← SSC_BoundSum(Q_ℓ, ⌊u/b⌋)
      R_ℓ ← { zb + ℓj | (z, j) ∈ SC_{u/b}(Q_ℓ) }
  return R_0 ⊕_u ··· ⊕_u R_{b−1}

SSC_BoundSum(S, u):
  if S = {x}:
      return {(0, 0), (x, 1)}
  T ← an arbitrary subset of S of size ⌊n/2⌋
  return SSC_BoundSum(T, u) ⊕_u SSC_BoundSum(S \ T, u)
```

Fig. 1. The $\tilde{O}(\sqrt{nu})$ time algorithm (SS_SmallInput) for the subset sum and its subroutine SSC_BoundSum: the algorithm of Theorem 3.2 for computing $\mathcal{SC}_u(S)$ in $\tilde{O}(\sqrt{nu})$ time.

3.1 The $\tilde{O}(un)$ Time Algorithm for $\mathcal{SC}_u(S)$

In this section, we give an algorithm (SSC_BoundSum of Figure 1) to answer subset sum problem with cardinalities. This will be used as a subroutine for the $\tilde{O}(\sqrt{nu})$ time algorithm for finding $\mathcal{S}_u(S)$.

Theorem 3.2. *Given a set $S \subseteq [u]$ of n elements, one can compute, in $O(u\, n \log n \log u)$ time, the set $\mathcal{SC}_u(S)$, which includes all subset sums of S up to u with their cardinalities.*

Proof. Partition S into two sets S_1 and S_2 of roughly the same size. Compute $\mathcal{SC}_u(S_1)$ and $\mathcal{SC}_u(S_2)$ recursively, and observe that both $\mathcal{SC}_u(S_1)$, $\mathcal{SC}_u(S_2) \subseteq ([u] \times [\frac{n}{2}])$. Next, note that $\mathcal{SC}_u(S_1) \oplus_u \mathcal{SC}_u(S_2) = \mathcal{SC}_u(S)$. Applying Lemma 2.2 yields $\mathcal{SC}_u(S)$ in $O(un \log u)$ time. The running time follows the recursive formula $T(n) = 2 \cdot T(n/2) + O(u\, n \log u)$, which is $O(u\, n \log u \log n)$, proving the claim. □

3.2 The $\tilde{O}(\sqrt{nu})$ Time Algorithm for $\mathcal{S}_u(S)$

First, we show how to compute the subset sums of elements in a congruence class quickly.

Lemma 3.3. *Let $\ell, b \in \mathbb{N}$ with $\ell < b$. Given a set $S \subseteq \{x \in \mathbb{N} \mid x \equiv \ell \pmod b\}$ of size n, one can compute $\mathcal{S}_u(S)$ in $O((u/b)\, n \log n \log u)$ time.*

Proof. An element $x \in S$ can be written as $x = yb + \ell$. Let $Q = \{y \mid yb + \ell \in S\}$. As such, for any subset $X = \{y_1 b + \ell, \ldots, y_j b + \ell\} \subseteq S$ of size j, we have that

$$\sum_{x \in X} x = \sum_{i=1}^{j} (\, y_i b + \ell\,) = \left(\sum_{i=1}^{j} y_i\right) b + j\ell.$$

In particular, a pair $(z, j) \in \mathcal{SC}_{u/b}(Q)$ corresponds to a set $Y = \{y_1, \ldots, y_j\} \subseteq Q$ of size j, such that $\sum_i y_i = z$. The set Y in turn corresponds to the set $X = \{y_1 b + \ell, \ldots, y_j b + \ell\} \subseteq S$. By the above, the sum of the elements of X is $zb + j\ell$. As such, compute $\mathcal{SC}_{u/b}(Q)$, using the algorithm of Theorem 3.2, and return $\{\, zb + j\ell \mid (z, j) \in \mathcal{SC}_{u/b}(Q)\,\} = \mathcal{S}_u(S)$ as the desired result. □

We are ready to introduce the new algorithm for finding $\mathcal{S}_u(S)$. First, partition the input into sets by congruence. Next, compute the $\mathcal{SC}_{u/b}(T)$ for each such set T and combine the results (SS_SmallInput of Figure 1).

Theorem 3.4. *Let $S \subseteq [u]$ be a given set of n elements. One can compute the set $\mathcal{S}_u(S)$ in $O(\sqrt{n \log n}\, u \log u)$ time.*

Proof. Partition S into $b = \lfloor \sqrt{n \log n} \rfloor$ sets $S_\ell = S \cap \{x \in \mathbb{N} \mid x \equiv \ell \pmod b\}$ of size n_ℓ, for $\ell \in [b-1]$. For each S_ℓ, compute the set of all subset sums $\mathcal{S}_u(S_\ell)$ in $O((u/b)\, n_\ell \log n_\ell \log u)$ time by Lemma 3.3. The time spent to compute all $\mathcal{S}_u(S_\ell)$ is $\sum_{\ell \in [b-1]} O((u/b)\, n_\ell \log n_\ell \log u) = O((u/b)\, n \log n \log u)$. Combining $\mathcal{S}_u(S_0) \oplus_u \cdots \oplus_u \mathcal{S}_u(S_{b-1})$ takes $O(b\, u \log u)$ time. Hence, the total running time is $O((u/b)\, n \log n \log u + b\, u \log u) = O(\sqrt{n \log n}\, u \log u)$. □

4 SUBSET SUMS IN $\tilde{O}(u^{5/4})$ TIME

In this section, we describe the $\tilde{O}(u^{5/4})$ time algorithm. It is based on two subroutines: one that performs well when the input set S has a small maximum and one that performs well when it has a large minimum. The algorithm balances out the two subroutines to achieve the final running time.

4.1 The $\tilde{O}(\max(S)^{5/3})$ Algorithm for $\mathcal{S}(S)$

We begin with the first subroutine. This algorithm runs fast when the maximum element of the input set is small (SS_SmallMax of Figure 2). There are two cases. We first show how to handle the case where the set S is dense. Intuitively, when S is dense, i.e., $\tilde{\Omega}(\sqrt{\max(S)})$, one can employ number theoretical results to obtain a fast algorithm [21]. When the set is not dense, the sum of the elements has to be small, and therefore one can use the $\tilde{O}(\sigma)$ time algorithm of the previous section. Formally:

Definition 4.1 (Dense Set). A set S is called *dense* if it contains at least $1{,}000\sqrt{M} \log M$ elements, where $M = \max(S)$.

Next, we state a lemma from Reference [21] that shows how the dense structure of a set S can be exploited to compute a large portion of the set $\mathcal{S}(S)$ fast.[1]

Lemma 4.2 (References [21, 22], see also Reference [39, Theorem C.4]). *Given a dense set S of n elements with $M = \max(S)$, one can compute $\mathcal{S}(S) \cap (L, \sigma - L)$ in $O(n + (M \log(M)/n)^2)$ time, where σ is the sum of S and $L = 100\, M^2/n$.*

Lemma 4.3. *Given a set S of n elements, one can compute $\mathcal{S}(S)$ in $O(M^{\frac{5}{3}} \log^2 M)$ time, where $M = \max(S)$.*

Proof. Consider the following two cases:

- When $n \leq M^{2/3}$, compute $\mathcal{S}(S)$ in $O(\sigma \log \sigma \log n) = O(M^{5/3} \log^2 M)$ time, using the algorithm of Theorem 3.1.
- When $n > M^{2/3}$, the input set S is dense. Let $L = 100M^2/n$. Compute $\mathcal{S}(S) \cap (L, \sigma - L)$ in $O(n + (M \log(M)/n)^2)$ time, using Lemma 4.2. Since $x \in \mathcal{S}(S) \iff \sigma - x \in \mathcal{S}(S)$, it suffices to compute $\mathcal{S}(S) \cap [L]$. Computing $\mathcal{S}(S) \cap [L]$ via the algorithm of Theorem 3.4 takes $O(\sqrt{n \log n}\, L \log L) = O(M^2 \sqrt{\log n} \log(M/n)/\sqrt{n})$ time. The $O((M \log M)^2/\sqrt{n}) = O(M^{5/3} \log^2 M)$ running time dominates both. □

4.2 The $\tilde{O}(u^2/\min(S))$ Algorithm for $\mathcal{S}_u(S)$

The second subroutine computes the set of subset sums fast when the minimum element of the input set is large (SS_LargeMin of Figure 2). As part of this algorithm, we first describe a subroutine that computes the set of subset sums with cardinalities $\mathcal{SC}^v(S)$ fast when the diameter of the input set S is small (SSC_BoundCard of Figure 2).

[1] The same idea was used by Reference [39] in their approximation algorithm for the subset sum.

SS_LargeInput(S, u):
  $r \leftarrow u^{3/4}$
  $S^- \leftarrow \{ s \in S \mid s < r \}$
  $S^+ \leftarrow S \setminus S^-$
  return SS_SmallMax$(S^-) \oplus_u$ SS_LargeMin(S^+, u)

SS_SmallMax(S):
  $M \leftarrow \max(S)$
  $L \leftarrow 100\, M^2/|S|$
  $\sigma \leftarrow \sum_{x \in S} x$
  if $|S| \le M^{2/3}$:
    return SS_SmallSum(S)
  else:
    $T \leftarrow$ *SS_Dense(S) via Lemma 4.2*
    $T' \leftarrow$ SS_SmallInput$(S \cap [L], L)$
    return $T \cup T' \cup \{\sigma - x \mid x \in T'\}$

SS_LargeMin(S, u):
  $\mu \leftarrow \min(S)$
  $\mu_i \leftarrow \lfloor 2^i \mu \rfloor$, for all i
  $k \leftarrow$ the smallest i such that $\mu_i > u$
  for i in $[1..k]$:
    $S_i \leftarrow S \cap [\mu_{i-1}..\mu_i - 1]$
    $T'_i \leftarrow$ SSC_BoundCard$(S_i, \lceil u/\mu_{i-1} \rceil)$
    $T_i \leftarrow \{x \mid (x, y) \in T'_i\} \cap [u]$
  return $T_1 \oplus_u \ldots \oplus_u T_k$

SSC_BoundCard(S, v):
  if $S = \{x\}$:
    return $\{(0, 0), (x, 1)\}$
  $\mu, \delta \leftarrow \min(S)$, median$(S)$
  $S^- \leftarrow \{s \mid s \in S, s < \delta\}$
  $S^+ \leftarrow S \setminus S^-$
  $T^-, T^+ \leftarrow$ SSC_BoundCard(S^-, v), SSC_BoundCard(S^+, v)
  $f \leftarrow$ the linear map $(i, j) \mapsto (i - \mu j, j)$
  return $f^{-1}(f(T^-) \oplus^v f(T^+))$

Fig. 2. The $\tilde{O}(u^{5/4})$ time algorithm (SS_LargeInput) along with the three new algorithms for the subset sum introduced in this section used as subroutines. SS_SmallMax is the $\tilde{O}(M^{5/3})$ time algorithm of Lemma 4.3, SS_LargeMin is the $\tilde{O}(u^2/\mu)$ time algorithm of Lemma 4.6, and SSC_BoundCard is the $\tilde{O}(\ell v^2)$ time algorithm of Lemma 4.5. Note that SS_Dense is the algorithm of Lemma 4.2, which we use as black box.

Lemma 4.4. *Given two disjoint sets $A, B \subseteq [x..x + \ell]$ and $\mathcal{SC}^v(A)$, $\mathcal{SC}^v(B)$, one can compute $\mathcal{SC}^v(A \cup B)$ in $O(\ell v^2 \log(\ell v))$ time.*

Proof. Consider the linear map f defined as $(i, j) \mapsto (i - xj, j)$. Let $X = f(\mathcal{SC}^v(A))$ and $Y = f(\mathcal{SC}^v(B))$. If $(i, j) \in \mathcal{SC}^v(A) \cup \mathcal{SC}^v(B)$, then $i = jx + y$ for $y \in [0..\ell j]$. Hence, $X, Y \subseteq [0..\ell v] \times [0..v]$.

Computing $X \oplus^{v} Y$ using the algorithm of Lemma 2.2 can be done in $O(\ell v^2 \log(\ell v))$ time. Let $Z = X \oplus^{v} Y$. The set $\mathcal{SC}^{v}(A \cup B)$ is then precisely $f^{-1}(Z)$, where f^{-1} is the inverse map of f, which is $(a, b) \mapsto (a + xb, b)$. Applying f^{-1} to Z takes an additional $O(\ell v^2 \log(\ell v))$ time. □

Lemma 4.5. *Given a set $S \subseteq [x..x + \ell]$ of n elements, one can compute $\mathcal{SC}^{v}(S)$ in $O(\ell v^2 \log(\ell v) \log n)$ time.*

Proof. Compute the median of S, denoted by δ, in linear time. Next, partition S into two sets $S^- = S \cap [\delta]$ and $S^+ = S \setminus S^-$. Compute recursively $T^- = \mathcal{SC}^{v}(S^-)$ and $T^+ = \mathcal{SC}^{v}(S^+)$, and combine them into $\mathcal{SC}^{v}(S^- \cup S^+)$ using Lemma 4.4. The recurrence for the running time is:

$$f(n, \ell) = \max_{\ell_1 + \ell_2 = \ell} \left\{ f\left(\frac{n}{2}, \ell_1\right) + f\left(\frac{n}{2}, \ell_2\right) + O\left(\ell v^2 \log(\ell v)\right) \right\},$$

which takes $O(\ell v^2 \log(\ell v) \log n)$ time, by Observation 1. □

Lemma 4.6. *Given a set S of n elements, one can compute $\mathcal{S}_u(S)$ in $O(u^2/\mu \log^2 u)$ time, where $\mu = \min(S)$.*

Proof. We first partition S into $S_1, \ldots, S_k$ as follows: $S_i = S \cap [\mu_{i-1}..\mu_i - 1]$, where $\mu_i = \lfloor 2^i \mu \rfloor$. The resulting partition is composed of $k = O(\log u)$ sets $S_1, \ldots, S_k$, and can be computed in $O(n \log n)$ time. Indeed, sort the numbers in S and add them into the sets, in the obvious fashion.

For each S_i, we will compute $T_i = \mathcal{S}_u(S_i)$. The idea is as follows: The sets S_i contain numbers at least as large as μ_{i-1}. Moreover, each set S_i is contained in an interval of length $\ell_i = \mu_i - \mu_{i-1} = \mu_{i-1}$. Apply Lemma 4.5 by setting $v = \lfloor u/\mu_{i-1} \rfloor$ to get $T_i' = \mathcal{SC}^{v}(S_i)$. This can be done in $O((u/\mu_{i-1})^2 \ell_i \log(\ell_i u/\mu_{i-1}) \log n_i) = O(\frac{u^2}{\mu_{i-1}} \log^2 u)$ time. Let $T_i = \{x \mid (x, y) \in T_i'\} \cap [u]$, and observe that $T_i = \mathcal{S}_u(S_i)$. Summing this, for $i = 1, \ldots, k$, results in $O(\frac{u^2}{\mu} \log^2 u)$ running time. Finally, combining all the T_i to obtain $\mathcal{S}_u(S)$ takes an additional $O(u \log^2 u)$ time. □

4.3 The $\tilde{O}(u^{5/4})$ Time Algorithm for $\mathcal{S}_u(S)$

Combining the above results yields a $\tilde{O}(u^{5/4})$ time algorithm with straightforward description; all that remains is deciding when to apply which algorithm (SS_LargeInput of Figure 2).

Theorem 4.7. *Let $S \subseteq [u]$ be a set of n elements. Computing the set of all subset sums $\mathcal{S}_u(S)$, takes $O(u^{5/4} \log^2 u)$ time.*

Proof. Fix an $0 < r \leq u$. Partition S into $S^- = \{ s \in S \mid s < r \}$, and $S^+ = S \setminus S^-$. Compute $\mathcal{S}_u(S^+)$ in $O(u^2/r \log^2 u)$ time via the algorithm of Lemma 4.6 and $\mathcal{S}_u(S^-)$ in $O(r^{5/3} \log^2 r)$ time using the algorithm of Lemma 4.3. Compute $\mathcal{S}_u(S) = \mathcal{S}_u(S^-) \oplus_u \mathcal{S}_u(S^+)$ in $O(u \log u)$ time using Lemma 2.1. The running time is $O(r^{5/3} \log^2 r + u^2 \log^2 u/r + u)$. Setting $r = u^{3/4}$ proves the theorem. □

5 THE MODULAR SUBSET SUM

In this section, we demonstrate how to extend some of the previous ideas to work for the modular subset sum problem, where all additions take place modulo m. Previous algorithms could ignore many sums that fell outside of $[u]$; the challenge here is that this can no longer be done, since these sums get "wrapped back in" and as such must be accounted for.

For any positive integer m, the set of integers *modulo* m with the operation of addition forms a finite group $\mathbb{Z}_m = \{0, 1, \ldots, m - 1\}$ of *order* m. We denote by $\mathbb{Z}_m^* = \{x \in \mathbb{Z}_m \mid \gcd(x, m) = 1\}$ the *set of units* of $\mathbb{Z}_m$ (also known as the multiplicative group of integers modulo m) and by $\varphi(m) = |\mathbb{Z}_m^*|$

```
SS_Mod(S, m):
    return SS_Mod'(S, m, m)

SS_Mod'(Γ, μ, τ):
    if τ = 1:
        return SS_ModUnit(Γ, μ)
    q ← smallest prime factor of τ
    r ← maximum r such that q^r | τ
    for i in [r]:
        T_i ← SS_Mod'((Γ/q^i)⧸q, μ/q^i, τ/q^r)
    U_r ← T_r
    for i from r − 1 to 0:
        U_i ← T_i ⊕ (q ⊗ U_{i+1})
    return U_0

SS_ModUnit(S, m):
    n ← |S|
    if n ≥ 2√m:
        S ← a subset of 2√m elements of S
    ℓ ← m/n
    L_0, ..., L_k ← SetCover(S, {(x ⊗ [ℓ]) ∩ S | x ∈ Z*_m})
    Partition S into S_0, ..., S_k such that S_i ⊆ L_i
    for i in [k]:
        T_i ← {x (mod m) | x ∈ SS_SmallSum(S_i)}
    return T_0 ⊕ ··· ⊕ T_k
```

Fig. 3. The $O(\min\{\sqrt{n}m, m^{5/4}\}\log^2 m)$ time algorithm (SS_Mod) for the *modular* subset sum, along with its subroutine SS_ModUnit: the algorithm of Lemma 5.5 for computing the set of subset sums when input is a subset of $\mathbb{Z}_m^*$ in $O\left(\min\left\{\sqrt{n}m, m^{5/4}\right\}\log m \log n\right)$ time. Here, SetCover$(S, \mathcal{T})$ is the greedy algorithm that returns a sequence of sets in $\mathcal{T}$ that covers S.

Euler's totient function capturing the *number of units* of $\mathbb{Z}_m$. Two integers x, y with $\gcd(x, y) = 1$ are called *coprime* (or relatively prime). We call the finite arithmetic progression

$$x \otimes [\ell] = \{0, x, 2x, \ldots, \ell x\}$$

a *segment* of x of *length* ℓ. Finally, let $S/x = \{s/x \mid s \in S \text{ and } x \mid s\}$ and $S \not| x = \{s \in S \mid x \nmid s\}$, where $x \mid s$ and $x \nmid s$ denote that "s divides q" and "s does not divide q," respectively.

5.1 Subset Sums When All Numbers Are Coprime to m

First, we consider the special case of computing $\mathcal{S}(S)$ modulo m when all the numbers in S are coprime to m (SS_ModUnit of Figure 3). The idea is to partition S into a small number of short segments, compute the subset sums of each segment, and combine them. We will use properties of $\mathbb{Z}_m^*$ to show that one can indeed find such a covering.

Lemma 5.1. *For a set $S \subseteq \mathbb{Z}_m$ of size n, such that $S \subseteq x \otimes [\ell]$, the set $\mathcal{S}(S)$ can be computed in $O(n\ell \log(n\ell) \log n)$ time.*

Proof. All elements of $x \otimes [\ell]$ are multiples of x, and thus $S' = S/x \subseteq [\ell]$ is a well-defined set of integers. Next, compute $\mathcal{S}(S')$ in $O(n\ell \log(n\ell) \log n)$ time using the algorithm of Theorem 3.1 (over the integers). Finally, compute the set $\{\sigma x \pmod m \mid \sigma \in \mathcal{S}(S')\} = \mathcal{S}(S)$ in linear time. □

Lemma 5.2. *Let $S \subseteq \mathbb{Z}_m$ be a set of size n covered by segments $x_1 \otimes [\ell], \ldots, x_k \otimes [\ell]$, formally $S \subseteq \bigcup_{i=1}^{k} x_i \otimes [\ell]$, then the set $\mathcal{S}(S)$ can be computed in $O(km \log m + n\ell \log(n\ell) \log n)$ time.*

Proof. Partition, in $O(kn)$ time, the elements of S into k sets $S_1, \ldots, S_k$, such that $S_i \subseteq x_i \otimes [\ell]$, for $i \in [k]$. Next, compute the subset sums $T_i = \mathcal{S}(S_i)$ using the algorithm of Lemma 5.1 for $i \in [k]$. Then, compute $T_1 \oplus T_2 \oplus \ldots \oplus T_k = \mathcal{S}(S)$ by $k-1$ applications of Lemma 2.1. The resulting running time is $O((k-1)m \log m + \sum_i |S_i|\ell \log(|S_i|\ell) \log |S_i|) = O(km \log m + n\ell \log(n\ell) \log n)$. □

Next, we show how to acquire such a covering, via an application of set cover. The set cover problem takes as input a collection of sets in a universe, and outputs a small number of sets that cover the universe. Covering $S \subseteq \mathbb{Z}_m^*$ by segments can be framed as a covering problem for the universe S with sets $\{(x \otimes [\ell]) \cap S \mid x \in \mathbb{Z}_m^*\}$. The textbook greedy set cover algorithm finds an approximate cover in linear time [13, Section 35.3].

We state the following lemma, which stems from the proof of Theorem 3.1 and Remark 3.4 in Reference [7].

Lemma 5.3 (Reference [7]). *Let G be a group of m elements and let $A \subseteq G$ be a subset of k elements. Define a set cover instance with universe $S \subseteq G$ and sets $\mathcal{T} = \{S \cap (x \oplus A) \mid x \in G\}$. Then, the greedy set cover algorithm yields a solution $\mathcal{B} = \{S \cap (b \oplus A) \mid b \in B\} \subseteq \mathcal{T}$, where $|\mathcal{B}| = |B| \leq \frac{m}{k}(\log(k+1))$.*

We apply the above lemma to algorithmically produce a set of segments that covers a set $S \subseteq \mathbb{Z}_m^*$.

Lemma 5.4. *Let $S \subseteq \mathbb{Z}_m^*$ be a set of size n. There exists a constant $c > 0$ such that for any $\ell \geq 2^{\frac{c \ln m}{\ln \ln m}}$ there is a collection $\mathcal{L}$ of $O(\frac{m \ln \ell}{\ell})$ segments, each of length ℓ, such that $S \subseteq \bigcup_{x \in \mathcal{L}} x \otimes [\ell]$. Furthermore, such a collection can be computed in $O((n + \log m)\,\ell)$ time.*

Proof. There are two steps in the proof. First, we need to bound $k = |\mathbb{Z}_m^* \cap [\ell]|$ with ℓ. Recall that $|\mathbb{Z}_m^*| = \phi(m)$. Let $\theta(m)$ be the number of distinct square-free divisors of m. It is known that $k \geq \ell \frac{\phi(m)}{m} - c_0\theta(m)$ for some c_0 [50, Equation (1.4)]. For $m \geq 3$, there are constants $c_1, c_2 > 0$ such that $\theta(m) \leq 2^{\frac{c_1 \ln m}{\ln \ln m}}$ [46] and $\phi(m) \geq c_2 \frac{m}{\ln \ln m}$ [27, Theorem 328]. Hence, there is some constant $c > 0$ for which the following holds:

$$c_0\theta(m) \leq 2^{\frac{c_1 \ln m}{\ln \ln m}} \leq \frac{1}{2} 2^{\frac{c \ln m}{\ln \ln m}} \frac{c_2}{\ln \ln m} \leq \frac{1}{2} \ell \frac{\phi(m)}{m}.$$

Therefore, have $k = \Omega(\ell \frac{\phi(m)}{m})$.

Second, we need to show the running time. Note that $\mathbb{Z}_m^*$ is the multiplicative group of integers modulo m. By Lemma 5.3, there exists a $B \subseteq U$ of size $\frac{\phi(m)}{k}(1 + \ln k) = O(\frac{m}{\ell} \log \ell)$ such that $S \subseteq A \otimes B$ and can be found with the greedy set cover algorithm over $\mathcal{T} = \{(x \otimes [\ell]) \cap S \mid x \in \mathbb{Z}_m^*\}$. So, we need to show that the sets in $\mathcal{T}$ can be computed quickly. To implement this efficiently, in the preprocessing stage, compute the modular inverses of every element in $[\ell]$ using the extended Euclidean algorithm in $O(\ell \log m)$ time [13, Section 31.2]. Then, for every $b \in S$ and every $i \in A$, find the unique x for which $ix \equiv b \pmod m$ using the inverse i^{-1} in $O(1)$ time. This indicates that b is in $(x \otimes [\ell]) \cap S$. Hence, the greedy algorithm computes $(x \otimes [\ell]) \cap S$, for all x, in time $O(n\ell + \ell \log m)$. □

Putting everything together completes the algorithm.

Lemma 5.5. *Let $S \subseteq \mathbb{Z}_m^*$ be a set of size n, computing the set of all subset sums $\mathcal{S}(S)$ takes $O(\min\{\sqrt{n}m, m^{5/4}\} \log m \log n)$ time.*

Proof. If $n < 2\sqrt{m}$, then apply the algorithm of Lemma 5.4 for $\ell = m/\sqrt{n} \geq m^{1/2}$. This results in a cover of S by $O(\frac{m}{\ell} \log n)$ segments (each of length ℓ), which takes $O((n + \log m)\,\ell) = O(\sqrt{n}m \log m)$ time. Next, compute $\mathcal{S}(S)$ in $O(n\ell \log(n\ell) \log n) = O(\sqrt{n}m \log m \log n)$ time using the algorithm of Lemma 5.2. Since $n = O(\sqrt{m})$, we have the promised running time.

If $|S| \geq 2\sqrt{m}$, then $\mathcal{S}(S) = \mathbb{Z}_m$ [26, Theorem 1.1]. As such, the case where $n = |S| \geq 2\sqrt{m}$ is immediate. □

5.2 The General Case

In this section, we show how to tackle the general case when S is any subset of $\mathbb{Z}_m$ (SS_Mod of Figure 3).

5.2.1 Algorithm Description. Parametrize the input instance via a triple (Γ, μ, τ) as follows: Γ is the input set, μ is its *modulus*, and τ an auxiliary parameter such that Γ only contains elements x for which $\gcd(x, \mu) \mid \tau$. For such an instance (Γ, μ, τ), the algorithm computes the set of all subset sums of Γ modulo μ. The initial instance would then be (S, m, m).

Let q be the smallest prime factor of τ, referred to as *pivot*. Compute the sets Γ/q and $\Gamma \nmid q$ from Γ. Recursively compute the (partial) subset sums $\mathcal{S}(\Gamma/q)$ and $\mathcal{S}(\Gamma \nmid q)$, of the instances $(\Gamma/q, \mu/q, \tau/q)$ and $(\Gamma \nmid q, \mu, \tau/q)$, respectively. Then compute the set of all subset sums $\mathcal{S}(\Gamma) = \{qx \mid x \in \mathcal{S}(\Gamma/q)\} \oplus \mathcal{S}(\Gamma \nmid q)$ by combining them together using Lemma 2.1. At the bottom of the recursion, when $\tau = 1$, for each set compute its subset sums, using the algorithm of Lemma 5.5.

5.2.2 Handling Multiplicities. During the execution of the algorithm there is a natural tree formed by the recursion. Consider an instance (Γ, μ, τ) such that the pivot q divides τ (and μ) with multiplicity r. The top-level recursion would generate instances with sets Γ/q and $\Gamma \nmid q$. In the next level, Γ/q is partitioned into Γ/q^2 and $(\Gamma/q) \nmid q$. On the other side of the recursion, $\Gamma \nmid q$ gets partitioned (naively) into $(\Gamma \nmid q)/q$ (which is an empty set) and $(\Gamma \nmid q) \nmid q = \Gamma \nmid q$. As such, this is a superfluous step and can be skipped. Hence, compressing the r levels of the recursion for this instance results in $r + 1$ instances

$$(\Gamma/q^0) \nmid q,\ (\Gamma/q^1) \nmid q,\ \ldots,\ (\Gamma/q^{r-1}) \nmid q,\ (\Gamma/q^r) \nmid q.$$

The total size of these sets is equal to the size of Γ. In particular, compress this subtree into a single level of recursion with the original call having $r + 1$ children. At each such level of the tree, label the edges by $0, 1, 2, \ldots, r$, based on the multiplicity of the divisor of the resulting (node) instance (i.e., an edge between instance sets Γ and $(\Gamma/q^2) \nmid q$ would be labeled by "2").

5.2.3 Analysis. The recursion tree formed by the execution of the algorithm has a level for each of the $k = O(\log m / \log\log m)$ distinct prime factors of m [46]—assume the root level is the 0th level.

Lemma 5.6. *Consider running the algorithm on input (S, m, m). Then the values of the moduli at the leaves of the recursion tree are* unique*, and are precisely the divisors of m.*

Proof. Let $m = \prod_{i=1}^{k} q_i^{r_i}$ be the prime factorization of m, where $q_i < q_{i+1}$ for all $1 \leq i < k$. Then every vector $\boldsymbol{x} = (x_1, \ldots, x_k)$, with $0 \leq x_i \leq r_i$, defines a path from the root to a leaf of modulus $m/\prod_{i=1}^{k} q_i^{x_i}$ in the natural way: Starting at the root, at each level of the tree follow the edge labeled x_i. If for two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ there is an $i \in [k]$ such that $x_i \neq y_i$, then the two paths they define will be different (starting at the ith level). And, by the unique factorization of integers, the values of the moduli at the two leaves will also be different. Finally, note that every divisor of m,

$\prod_{i=1}^{k} q_i^{\rho_i}$ with $0 \le \rho_i \le r_i$, occurs as a modulus of a leaf, and can be reached by following the path $(r_1 - \rho_1, \ldots, r_k - \rho_k)$ down the tree. □

THEOREM 5.7. *Let $S \subseteq \mathbb{Z}_m$ be a set of size n, computing the set of all subset sums $\mathcal{S}(S)$ takes $O(\min\{\sqrt{n}m, m^{5/4}\} \log^2 m)$ time.*

PROOF. The algorithm is described in Section 5.2.1 and shown in Figure 3 (SS_MOD). We break down the running time analysis into two parts: the running time at the leaves, and the running time at internal nodes.

Let d be the number of leaves of the recursion tree. Arrange them so the modulus of the ith leaf, μ_i, is the ith largest divisor of m. Note that μ_i is at most m/i for all $i \in [1..d]$. Using Lemma 5.5, the running time is bounded by

$$O\left(\sum_{i=1}^{d} \min\left\{\sqrt{n_i}\,\mu_i, \mu_i^{5/4}\right\} \log n_i \log \mu_i\right) = O\left(\log m \log n \sum_{i=1}^{d} \min\left\{\sqrt{n_i}\,\frac{m}{i}, \left(\frac{m}{i}\right)^{5/4}\right\}\right).$$

Using Cauchy-Schwarz inequality, the first sum of the min is bounded by

$$m \sum_{i=1}^{d} \frac{\sqrt{n_i}}{i} \le m\sqrt{\left(\sum_{i=1}^{d} \left(\sqrt{n_i}\right)^2\right)\left(\sum_{i=1}^{d} \frac{1}{i^2}\right)} = O\left(\sqrt{n}m\right),$$

while the second is bounded by $O(m^{5/4})$. Putting it all together, the total work done at the leaves is $O(\min\{\sqrt{n}m, m^{5/4}\} \log m \log n)$.

Next, consider an internal node of modulus μ, pivot q and $r+1$ children. The algorithm combines these instances by applying r times Lemma 2.1. The total running time necessary for this process is described next. As the moduli of the instances decrease geometrically, pair up the two smallest instances, combine them together, and in turn combine the result with the next (third) smallest instance, and so on. This yields a running time of

$$O\left(\sum_{i=0}^{r} \frac{\mu}{q^i} \log \frac{\mu}{q^i}\right) = O(\mu \log \mu).$$

At the leaf level, by Lemma 5.6, the sum of the moduli $\sum_{i=1}^{d} \mu_i$ is known to be $O(m \log\log m)$ [27, Theorem 323]. As such, the sum of the moduli of all internal nodes is bounded by $O(km \log\log m) = O(m \log m)$, as the sum of each level is bounded by the sum at the leaf level, and there are k levels. As each internal node, with modulus μ, takes $O(\mu \log \mu)$ time and $x \log x$ is a convex function, the total running time spent on all internal nodes is $O(m \log m \log(m \log m)) = O(m \log^2 m)$.

Aggregating everything together, the complete running time of the algorithm is bounded by $O(\min\{\sqrt{n}m, m^{5/4}\} \log^2 m)$, implying the theorem. □

6 RECOVERING THE SOLUTION

Given sets X and Y, a number x is a *witness* for $i \in X \oplus Y$, if $x \in X$ and $i - x \in Y$. A function $w : X \oplus Y \to X$ is a *witness function*, if $w(i)$ is a witness of i.

If one can find a witness function for each $X \oplus Y$ computation of the algorithm, then we can trace back the recursion tree and reconstruct the subset that sums up to t in $O(n)$ time. The problem of finding a witness function quickly can be reduced to the *reconstruction problem* defined next.

In the reconstruction problem, there are hidden sets $S_1, \ldots, S_n \subseteq [m]$, and we have two oracles SIZE and SUM that take as input a query set Q.

- SIZE(Q) returns the size of each intersection:

$$\left(|S_1 \cap Q|, |S_2 \cap Q|, \ldots, |S_n \cap Q|\right).$$

- $\textsc{Sum}(Q)$ returns the sum of elements in each intersection:

$$\left(\sum_{s\in S_1\cap Q} s, \sum_{s\in S_2\cap Q} s, \ldots, \sum_{s\in S_n\cap Q} s\right).$$

The reconstruction problem asks to find n values $x_1, \ldots, x_n$ such that for all i, if S_i is non-empty, $x_i \in S_i$. Let f be the running time of calling the oracles and assume $f = \Omega(m + n)$, then is it known that one can find $x_1, \ldots, x_n$ in $O(f \log n \operatorname{polylog} m)$ time [3].

If $X, Y \subseteq [u]$, then finding the witness of $X \oplus Y$ is just a reconstruction problem. Here, the hidden sets are $W_0, \ldots, W_{2u} \subseteq [2u]$, where $W_i = \{x \mid x + y = i \text{ and } x \in X, y \in Y\}$ is the set of witnesses of i. Next, define the polynomials $\chi_Q(x) = \sum_{i\in Q} x^i$ and $I_Q(x) = \sum_{i\in Q} ix^i$. The coefficient for x^i in $\chi_Q\chi_Y$ is $|W_i \cap Q|$ and in $I_Q\chi_Y$ is $\sum_{s\in W_i\cap Q} s$, which are precisely the ith coordinate of $\textsc{Size}(Q)$ and $\textsc{Sum}(Q)$, respectively. Hence, the oracles can be implemented using polynomial multiplication, in $\tilde{O}(u)$ time per call. This yields an $\tilde{O}(u)$ time deterministic algorithm to compute $X \oplus Y$ *with* its witness function. The exact same argument also shows we can compute $X \oplus Y$, where $X \times Y \subseteq [u] \times [v]$ with a witness function in $\tilde{O}(uv)$ time.

Hence, with a polylogarithmic slowdown, we can find a witness function every time we perform a $\oplus$ operation, thus effectively maintaining which subsets sum up to which sum.

7 APPLICATIONS AND EXTENSIONS

Since every algorithm that uses subset sum as a subroutine can benefit from the new algorithm, we only highlight certain selected applications and some interesting extensions. Most of these applications are derived directly from Theorem 3.1. Finally, we highlight an additional result with applications in error correction codes.

7.1 Bottleneck Graph Partition

Let $G = (V, E)$ be a graph with n vertices m edges and let $w : E \to \mathbb{R}^+$ be a weight function on the edges. The *bottleneck graph partition* problem is to split the vertices into two equal-sized sets such that the value of the bottleneck (maximum-weight) edge, over all edges across the cut, is minimized. This is the simplest example of a graph partition problem with cardinality constraints. The standard divide-and-conquer algorithm reduces this problem to solving $O(\log n)$ subset sum problems: Pick a weight, delete all edges with smaller weight, and decide if there exists an arrangement of components that satisfies the size requirement [28]. The integers being summed are the various sizes of the components, the target value is $n/2$, and the sum of all inputs is n. Previously, using the $O(\sigma^{3/2})$ algorithm by Klinz and Woeginger, the best known running time was $O(m + n^{3/2} \log n)$ [34]. Using Theorem 3.1, this is improved to $O(m) + \tilde{O}(n)$ time.

7.2 Counting and Power Index

Here, we show that the standard divide-and-conquer algorithm can also answer the counting version of all subset sums. Namely, computing the function $N_{u,S}(x)$: the number of subsets of S that sum up to x, where $x \le u$.

For two functions $f, g : X \to Y$, define $f \odot g : X \to Y$ to be

$$(f \odot g)(x) = \sum_{t\in X} f(x)g(x - t).$$

Corollary 7.1. *Given two functions $f, g : [u] \to [b]$, one can compute $f \odot g$ in $O(u \log u \log b)$ time.*

Proof. This is an immediate extension of Lemma 2.1 using the fact that multiplication of two degree u polynomials, with coefficients at most b, takes $O(u \log u \log b)$ time [47]. □

Theorem 7.2. *Let S be a set of n positive integers. One can compute the function $N_{u,S}$ in $O(nu \log u \log n)$ time.*

Proof. Partition S into two (roughly) equally sized sets S_1 and S_2. Compute N_{u,S_1} and N_{u,S_2} recursively and combine them into $N_{u,S} = N_{u,S_1} \odot N_{u,S_1}$ using Corollary 7.1, in $O(u \log u \log 2^n) = O(nu \log u)$ time. The final running time is then given by Observation 1. □

7.2.1 Power Indices. The *Banzhaf index* of a set S of n voters with cutoff u can be recovered from $N_{u,S}$ in linear time. Theorem 7.2 yields an algorithm for computing the Banzhaf index in $\tilde{O}(nu)$ time. Previous dynamic programming algorithms take $O(nu)$ arithmetic operations, which translates to $O(n^2 u)$ running time [54]. Similar speed-ups (of roughly a factor n) can be obtained for the *Shapley-Shubik index*.

7.3 Covering $\mathbb{Z}_m$ by Segments

The results of Section 5, along with the analysis of the recursion tree, conclude the following corollary on covering $\mathbb{Z}_m$ with a small number of segments. In a study on error correction codes, Chen et al. [12] showed that one can cover $\mathbb{Z}_m$ with $m^{1+o(1)}/\sqrt{\ell}$ segments of length ℓ. In this work this is improved to $m^{1+o(1)}/\ell$, an improvement factor of $\sqrt{\ell}$.

For an integer x, let $\sigma_0(x)$ denote the *number of divisors* of x and $\sigma_1(x)$ the *sum of its divisors*. It is known that $\sigma_1(x) = O(m \log\log m) = m^{1+o(1)}$ and $\sigma_0(x) = m^{o(1)}$ [27].

Corollary 7.3. *There exists a constant c, for all $m \geq 3$ and ℓ such that $2^{\frac{c \ln m}{\ln\ln m}} \leq \ell \leq m$, one can cover $\mathbb{Z}_m$ with $O((\sigma_1(m) \ln m)/\ell) + \sigma_0(m) = m^{1+o(1)}/\ell$ segments of length ℓ. Furthermore, such a cover can be computed in $O(m\ell)$ time.*

Proof. Let $S_{m/d} = \{x/(m/d) \mid x \in \mathbb{Z}_m \text{ and } \gcd(x, m) = m/d\}$, for all $d \mid m$. Note that $S_{m/d} = \mathbb{Z}_d^*$, hence by Lemma 5.4, each $S_{m/d}$ has a cover of $O((d \ln \ell)/\ell) = O((d \ln m)/\ell)$ segments. Next, "lift" the segments of each set $S_{m/d}$ back up to $\mathbb{Z}_m$ (by multiplying by m/d), forming a cover of $\mathbb{Z}_m$. The number of segments in the final cover is bounded by

$$\sum_{\substack{d \mid m \\ \ell \leq d}} O\left(\frac{d}{\ell} \ln m\right) + \sum_{\substack{d \mid m \\ \ell > d}} 1 \leq O\left(\frac{\sigma_1(m) \ln m}{\ell}\right) + \sigma_0(m).$$

The time to cover each $S_{m/d}$, by Lemma 5.4, is $O((n + \log m)\,\ell) = O((\varphi(d) + \log d)\ell)$, since there are $\varphi(d)$ elements in $S_{m/d}$, and $S_{m/d} \subseteq \mathbb{Z}_d$. Also, $\varphi(d)$ dominates $\log d$, as $O(\varphi(d)) = \Omega(d/\log\log d)$ [27, Theorem 328]; therefore, the running time simplifies to $O(\varphi(d)\ell)$. Summing over all $S_{m/d}$, we have

$$\sum_{d \mid m} O(\varphi(d)\ell) = O\left(\ell \sum_{d \mid m} \varphi(d)\right) = O(m\ell)\,,$$

since $\sum_{d \mid m} \varphi(d) = m$ [27, Sec 16.2], implying the corollary. □

If $\ell < 2^{\frac{c \ln m}{\ln\ln m}}$, then $\ell = m^{o(1)}$. The corollary above then shows that for all ℓ, there is a cover of $\mathbb{Z}_m$ with $m^{1+o(1)}/\ell$ segments of length ℓ. Our analysis only employed elementary number theory; more involved techniques, such as the ones found in sieve theory, might yield better bounds.

ACKNOWLEDGMENTS

We would like to thank Sariel Har-Peled for his invaluable help in the editing of this article as well as for various suggestions on improving the presentation of the results. We would also like to thank Jeff Erickson and Kent Quanrud for their insightful comments and feedback. We would like to thank Igor Shparlinski and Arne Winterhof for pointing out a problem in a proof, and George Shakan for pointing out the paper [12]. Finally, we also like to thank the anonymous reviewers for their helpful and meaningful comments on the different versions of this manuscript.

REFERENCES

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. 2019. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. Society for Industrial and Applied Mathematics, 41–57. Retrieved from: http://dl.acm.org/citation.cfm?id=3310435.3310438.

[2] N. Alon. 1987. Subset sums. *J. Numb. Theory* 27, 2 (1987), 196–205. DOI: https://doi.org/10.1016/0022-314X(87)90061-8

[3] Yonatan Aumann, Moshe Lewenstein, Noa Lewenstein, and Dekel Tsur. 2011. Finding witnesses by peeling. *ACM Trans. Alg.* 7, 2, Article 24 (Mar. 2011), 15 pages. DOI: https://doi.org/10.1145/1921659.1921670

[4] Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. 2019. Fast modular subset sum using linear sketching. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. Society for Industrial and Applied Mathematics, 58–69. Retrieved from: http://dl.acm.org/citation.cfm?id=3310435.3310439.

[5] Richard Bellman. 1956. Notes on the theory of dynamic programming IV—Maximization over discrete sets. *Naval Res. Logist. Quart.* 3, 1-2 (1956), 67–70. DOI: https://doi.org/10.1002/nav.3800030107

[6] Richard E. Blahut. 1985. *Fast Algorithms for Digital Signal Processing* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

[7] Béla Bollobás, Svante Janson, and Oliver Riordan. 2010. On covering by translates of a set. *Rand. Struct. Alg.* 38, 1–2 (2010), 33–67. DOI: https://doi.org/10.1002/rsa.20346

[8] Karl Bringmann. 2017. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*. Society for Industrial and Applied Mathematics, 1073–1084. Retrieved from: http://dl.acm.org/citation.cfm?id=3039686.3039755.

[9] Leizhen Cai, Siu Man Chan, and Siu On Chan. 2006. *Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems*. Springer Berlin Heidelberg, 239–250. DOI: https://doi.org/10.1007/11847250_22

[10] Yair Caro and Raphael Yuster. 1998. The characterization of zero-sum (mod 2) bipartite Ramsey numbers. *J. Graph Theory* 29, 3 (1998), 151–166.

[11] Mark Chaimovich. 1999. New algorithm for dense subset-sum problem. *Astérisque* 258 (1999), 363–373.

[12] Zhixiong Chen, Igor E. Shparlinski, and Arne Winterhof. 2014. Covering sets for limited-magnitude errors. *IEEE Trans. Inform. Theory* 60, 9 (Sep. 2014), 5315–5321. DOI: https://doi.org/10.1109/TIT.2014.2338078

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, and Clifford Stein. 2014. *Introduction to Algorithms* (3rd ed.). The MIT Press, Cambridge, MA.

[14] George B. Dantzig. 1957. Discrete-variable extremum problems. *Op. Res.* 5, 2 (1957), 266–277. Retrieved from: http://www.jstor.org/stable/167356.

[15] Jonathan L. Dautrich, Jr. and Chinya V. Ravishankar. 2013. Compromising privacy in precise query protocols. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. ACM, 155–166.

[16] Josep Diaz, Fabrizio Grandoni, and Alberto Marchetti Spaccamela. 2009. Balanced cut approximation in random geometric graphs. *Theoret. Comput. Sci.* 410, 27 (2009), 2725–2731. DOI: https://doi.org/10.1016/j.tcs.2009.03.037

[17] David Eppstein. 1997. Minimum range balanced cuts via dynamic subset sums. *J. Alg.* 23, 2 (1997), 375–385. DOI: https://doi.org/10.1006/jagm.1996.0841

[18] Paul Erdös, Abraham Ginzburg, and Abraham Ziv. 1961. Theorem in the additive number theory. *Bull. Res. Council Israel, Sect. F.* 10 (1961), 41–43.

[19] Bruce Faaland. 1973. Solution of the value-independent knapsack problem by partitioning. *Op. Res.* 21, 1 (1973), 332–337.

[20] M. L. Fredman and D. E. Willard. 1990. BLASTING through the information theoretic barrier with FUSION TREES. In *Proceedings of the 22nd ACM Symposium on Theory of Computing (STOC'90)*. ACM, 1–7. DOI: https://doi.org/10.1145/100216.100217

[21] Zvi Galil and Oded Margalit. 1991. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.* 20, 6 (Dec. 1991), 1157–1189. DOI: https://doi.org/10.1137/0220072

[22] Zvi Galil and Oded Margalit. 1991. An almost linear-time algorithm for the dense subset-sum problem. In *Automata, Languages and Programming*, Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo (Eds.). Springer Berlin Heidelberg, 719–727.

[23] Celia A. Glass and Hans Kellerer. 2007. Parallel machine scheduling with job assignment restrictions. *Nav. Res. Logist.* 54, 3 (2007), 250–257. DOI: https://doi.org/10.1002/nav.20202

[24] C. Guéret and C. Prins. 1999. A new lower bound for the open-shop problem. *Ann. Op. Res.* 92, 0 (1999), 165–183. DOI: https://doi.org/10.1023/A:1018930613891

[25] Venkatesan Guruswami, Yury Makarychev, Prasad Raghavendra, David Steurer, and Yuan Zhou. 2011. Finding almost-perfect graph bisections. In *Proceedings of the Conference on Innovations in Computer Science*. 321–337. Retrieved from: http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/11.html.

[26] Y. O. Hamidoune, A. S. Lladó, and O. Serra. 2008. On complete subsets of the cyclic group. *J. Combin. Theory, Series A* 115, 7 (2008), 1279–1285.

[27] Godfrey Harold Hardy and Edward Maitland Wright. 1979. *An Introduction to the Theory of Numbers* (5th ed.). Clarendon Press, Oxford. Retrieved from: http://opac.inria.fr/record=b1099316.

[28] Dorit S. Hochbaum and Anu Pathria. 1996. The bottleneck graph partition problem. *Networks* 28, 4 (1996), 221–225. DOI: https://doi.org/10.1002/(SICI)1097-0037(199612)28:4⟨221::AID-NET6⟩3.0.CO;2-N

[29] Ellis Horowitz and Sartaj Sahni. 1974. Computing partitions with applications to the knapsack problem. *J. ACM* 21, 2 (Apr. 1974), 277–292. DOI: https://doi.org/10.1145/321812.321823

[30] Russell Impagliazzo and Ramamohan Paturi. 2001. On the complexity of k-SAT. *J. Comput. Syst. Sci.* 62, 2 (2001), 367–375. DOI: https://doi.org/10.1006/jcss.2000.1727

[31] Ce Jin and Hongxun Wu. 2018. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA'19), (OpenAccess Series in Informatics (OASIcs))*, Jeremy T. Fineman and Michael Mitzenmacher (Eds.), Vol. 69. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 17:1–17:6. DOI: https://doi.org/10.4230/OASIcs.SOSA.2019.17

[32] Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger (Eds.). Springer US, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9

[33] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems*. Springer.

[34] Bettina Klinz and Gerhard J. Woeginger. 1999. A note on the bottleneck graph partition problem. *Networks* 33, 3 (1999), 189–191. DOI: https://doi.org/10.1002/(SICI)1097-0037(199905)33:3⟨189::AID-NET5⟩3.0.CO;2-2

[35] Konstantinos Koiliaris and Chao Xu. 2017. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1062–1072.

[36] Konstantinos Koiliaris and Chao Xu. 2018. Subset sum made simple. *CoRR* abs/1807.08248 (2018). Retrieved from: http://arxiv.org/abs/1807.08248.

[37] Eugene L. Lawler. 1979. Fast approximation algorithms for knapsack problems. *Math. Op. Res.* 4, 4 (1979), 339–356. DOI: https://doi.org/10.1287/moor.4.4.339

[38] Daniel Lokshtanov and Jesper Nederlof. 2010. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*. ACM, 321–330.

[39] Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. 2019. A subquadratic approximation scheme for partition. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. Society for Industrial and Applied Mathematics, 70–88. Retrieved from: http://dl.acm.org/citation.cfm?id=3310435.3310440.

[40] John E. Olson. 1968. An addition theorem modulo p. *J. Combin. Theory* 5, 1 (1968), 45–52.

[41] John E. Olson. 1975. Sums of sets of group elements. *Acta Arith.* 28, 2 (1975), 147–156.

[42] U. Pferschy. 1999. Dynamic programming revisited: Improving knapsack algorithms. *Computing* 63, 4 (1999), 419–430. DOI: https://doi.org/10.1007/s006070050042

[43] David Pisinger. 1999. Linear time algorithms for knapsack problems with bounded weights. *J. Alg.* 33, 1 (1999), 1–14.

[44] David Pisinger. 2003. Dynamic programming on the word RAM. *Algorithmica* 35, 2 (2003), 128–145. DOI: https://doi.org/10.1007/s00453-002-0989-y

[45] Xiangtong Qi. 2008. Coordinated logistics scheduling for in-house production and outsourcing. *IEEE Trans. Auto. Sci. Eng.* 5, 1 (Jan. 2008), 188–192. DOI: https://doi.org/10.1109/TASE.2006.887159

[46] Guy Robin. 1983. Estimation de la fonction de Tchebychef θ sur le k-ième nombre premier et grandes valeurs de la fonction $\omega(n)$ nombre de diviseurs premiers de n. *Acta Arith.* 42, 4 (1983), 367–389.

[47] Arnold Schönhage. 1982. *Asymptotically Fast Algorithms for the Numerical Muitiplication and Division of Polynomials with Complex Coefficients*. Springer Berlin Heidelberg, 3–15. DOI: https://doi.org/10.1007/3-540-11607-9_1

[48] Oliver Serang. 2014. The probabilistic convolution tree: Efficient exact Bayesian inference for faster LC-MS/MS protein inference. *PLoS ONE* 9, 3 (03 2014), e91507. DOI: https://doi.org/10.1371/journal.pone.0091507

[49] Oliver Serang. 2015. A fast numerical method for max-convolution and the application to efficient max-product inference in Bayesian networks. *J. Computat. Bio.* 22, 8 (2015), 770–783. DOI: https://doi.org/10.1089/cmb.2015.0013

[50] D. Suryanarayana. 1974. On $\Delta(x, n) = \phi(x, n) - x\phi(n)/n$. *Proc. Amer. Math. Soc.* 44, 1 (Jan. 1974), 17–21. DOI: https://doi.org/10.1090/S0002-9939-1974-0332636-5

[51] Endre Szemerédi. 2016. Structural approach to subset sum problems. *Found. Computat. Math.* 16, 6 (01 Dec. 2016), 1737–1749. DOI: https://doi.org/10.1007/s10208-016-9326-8

[52] E. Szemerédi. 1970. On a conjecture of Erdös and Heilbronn. *Acta Arith.* 17, 3 (1970), 227–229. Retrieved from: http://eudml.org/doc/204952.

[53] Quoc Trung Tran, Chee-Yong Chan, and Guoping Wang. 2011. Evaluation of set-based queries with aggregation constraints. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM'11.* 1495–1504. DOI: https://doi.org/10.1145/2063576.2063791

[54] Takeaki Uno. 2012. Efficient computation of power indices for weighted majority games. In *Algorithms and Computation*, Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee (Eds.). Lecture Notes in Computer Science, Vol. 7676. Springer Berlin Heidelberg, 679–689.

[55] Van Vu. 2008. *A Structural Approach to Subset-Sum Problems.* Springer Berlin Heidelberg, 525–545. DOI: https://doi.org/10.1007/978-3-540-85221-6_19

Received February 2018; revised December 2018; accepted May 2019