Fast Low-Space Algorithms for Subset Sum*

Ce Jin	Nikhil Vyas	Ryan Williams
MIT	MIT	MIT
cejin@mit.edu	nikhilv@mit.edu	rrw@mit.edu

February 10, 2021

Abstract

We consider the canonical Subset Sum problem: given a list of positive integers $a_1, \ldots, a_n$ and a target integer t with $t > a_i$ for all i, determine if there is an $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. The well-known pseudopolynomial-time dynamic programming algorithm [Bellman, 1957] solves Subset Sum in $O(nt)$ time, while requiring $\Omega(t)$ space.

In this paper we present algorithms for Subset Sum with $\widetilde{O}(nt)$ running time and much lower space requirements than Bellman's algorithm, as well as that of prior work. We show that Subset Sum can be solved in $\widetilde{O}(nt)$ time and $O(\log(nt))$ space with access to $O(\log n \log\log n + \log t)$ random bits. This significantly improves upon the $\widetilde{O}(nt^{1+\varepsilon})$-time, $\widetilde{O}(n \log t)$-space algorithm of Bringmann (SODA 2017). We also give an $\widetilde{O}(n^{1+\varepsilon}t)$-time, $O(\log(nt))$-space randomized algorithm, improving upon previous $(nt)^{O(1)}$-time $O(\log(nt))$-space algorithms by Elberfeld, Jakoby, and Tantau (FOCS 2010), and Kane (2010). In addition, we also give a $\mathrm{poly}\log(nt)$-space, $\widetilde{O}(n^2 t)$-time deterministic algorithm.

We also study time-space trade-offs for Subset Sum. For parameter $1 \leq k \leq \min\{n, t\}$, we present a randomized algorithm running in $\widetilde{O}((n+t) \cdot k)$ time and $O((t/k)\,\mathrm{poly}\log(nt))$ space.

As an application of our results, we give an $\widetilde{O}(\min\{n^2/\varepsilon, n/\varepsilon^2\})$-time and $\mathrm{poly}\log(nt)$-space algorithm for "weak" ε-approximations of Subset Sum.

*Supported by NSF CCF-1741615 and NSF CCF-1909429.

1 Introduction

We consider the classical Subset Sum problem in its standard form: given positive integers $a_1, \ldots, a_n$ and a target integer t with $t > a_i$ for all i, determine if there is an $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. In the 1950s, Bellman [Bel57] showed that Subset Sum is in $O(nt)$ time, with a textbook *pseudopolynomial* time algorithm.[1] The algorithm is also a textbook example of *dynamic programming*, requiring $\Omega(t)$ space to store the table. In this paper, we address the question: to what extent can the space complexity of Subset Sum be reduced, without increasing the running time of Bellman's algorithm?[2] Besides the inherent interest in designing algorithms with tiny space overhead, recall that low-space algorithms also imply efficient *parallel algorithms* due to the well-known result that space-s algorithms can be solved in $O(s^2)$ parallel time with $\mathrm{poly}(2^s)$ work/processors [Pap94, Theorem 16.1]. For example, an $O(\log(nt))$-space algorithm can be applied to solve Subset Sum in $O(\log^2(nt))$ parallel time with $\mathrm{poly}(nt)$ work/processors.

Prior Work. Within the last decade, there has been substantial work on improving the space complexity of pseudopolynomial-time algorithms for Subset Sum and related problems. Lokshtanov and Nederlof [LN10] gave an algorithm running in $\widetilde{O}(n^3 t)$ time and $\widetilde{O}(n^2 \log t)$ space, using Fast Fourier Transforms over $\mathbb{C}$. Elberfeld, Jakoby, and Tantau [EJT10] gave a generic meta-theorem for placing problems in LOGSPACE, and their result implies that Subset Sum is in pseudopolynomial time and logarithmic space. However no nice bounds on the running time follow (for example, they need Reingold's algorithm for s-t connectivity [Rei08], which has a rather high-degree polynomial running time). Kane [Kan10] gave a more explicit logspace algorithm; it can be shown that his algorithm as stated solves Subset Sum in deterministic $\widetilde{O}(n^3 t + n^{2.025} t^{1.025})$ time and $O(\log(nt))$ space.[3] Recently, Bringmann [Bri17] showed that, *assuming the Generalized Riemann Hypothesis*, the problem can be solved in randomized $\widetilde{O}(nt)$ time and $\widetilde{O}(n \log t)$ space, and unconditionally in randomized $\widetilde{O}(nt^{1+\varepsilon})$ time and $\widetilde{O}(nt^{\varepsilon})$ space. Although the running time of Bringmann's algorithm is very close to the dynamic programming solution, the space bound is still $\Omega(n)$. (Bringmann also gives an $\widetilde{O}(n + t)$ time algorithm for Subset Sum that uses $\Omega(t)$ space.)

1.1 Our Results

We extend the algebraic and randomized approaches to Subset Sum in novel ways, obtaining randomized algorithms that use very few random bits and essentially preserve Bellman's running time, while reducing the space usage all the way to $O(\log(nt))$. We also obtain deterministic algorithms. Our first main result is the following.

Theorem 1.1. *The Subset Sum problem can be solved by randomized algorithms running in*

1. $\widetilde{O}(nt)$ *time and* $O(\log(nt))$ *space, and read-only random access to* $O(\log n \log\log n)$ *random bits.*
2. $\widetilde{O}(n^{1+\varepsilon} t)$ *time and* $O(\log(nt))$ *space, for any* $\varepsilon > 0$*, with* $O(\log n)$ *random bits.*

Our algorithms are Monte Carlo, in that they always answer NO if there is no solution, and answer YES with probability at least 99% if there is a solution. We also obtain a deterministic algorithm with polylogarithmic space and $\widetilde{O}(n^2 t)$ time.

[1] In this paper, we work in a random-access model of word length $\Theta(\log n + \log t)$, and measure space complexity in total number of bits.

[2] Recall the "space complexity" of an algorithm is the size of the working memory used by it; the input integers $a_1, \ldots, a_n$ are assumed to be stored in a read-only randomly-accessible array, which does not count towards the space complexity.

[3] Kane does not give an explicit running time analysis, but the best known (unconditional) result in number theory on gaps between primes (namely, that there is always a prime in the interval $[N, N + O(N^{0.525})]$ [BHP01]) implies such a running time. See Remark 3.5.

Theorem 1.2. *Subset Sum can be solved deterministically in $\widetilde{O}(n^2 t)$ time and $O(\log t \cdot \log^3 n)$ space.*

Furthermore, we obtain time-space trade-off algorithms for Subset Sum.

Theorem 1.3. *For every parameter $1 \le k \le \min\{n, t\}$, there is a randomized algorithm for Subset Sum with 0.01 one-sided error probability, running in $\widetilde{O}((n+t)\cdot k)$ time and $O((t/k)\operatorname{poly}\log(nt))$ space.*

For $k = 1$, Theorem 1.3 matches the near-linear time algorithm by Bringmann [Bri17]. For larger k, we obtain the first algorithm that uses $o(t)$ space and runs faster than Bellman's dynamic programming.

It is interesting to compare the time-space tradeoff of Theorem 1.3 with known conditional lower bounds on Subset Sum. Note that the product of the time bound T and space bound S in Theorem 1.3 is always at least $\widetilde{\Theta}(nt + t^2)$. Results in fine-grained complexity show that, when $T = S$ (the time bound equals the space bound), an algorithm running in $t^{1-\varepsilon/2}2^{o(n)}$ time and $t^{1-\varepsilon/2}2^{o(n)}$ space for Subset Sum would refute SETH [ABHS19], so in such a case the time-space product must be at least $t^{2-\varepsilon} \cdot 2^{o(n)}$. It is interesting that, while Subset Sum can be solved in essentially nt time and logarithmic space (Theorem 1.1) and also in $\widetilde{O}((n+t)\cdot k)$ time and $\widetilde{O}(t/k)$ space, obtaining $t^{.99}\cdot 2^{o(n)}$ time and $t^{.99}\cdot 2^{o(n)}$ space appears to be impossible!

Remark 1.4. *Our algorithms solve the decision version of Subset Sum. Another well-studied setting is the optimization version, where we want to find the largest $t' \le t$ that can be expressed as a subset sum $t' = \sum_{i\in S} a_i$. We remark that the optimization version reduces to the decision version, with an $O(\log t)$ extra factor in time complexity: perform binary search for t', calling an oracle that determines if there is a subset sum with value $x \in [t', t]$ for our current t'. To implement this oracle, we add $\lfloor \log(t - t' + 1) \rfloor + 1$ numbers $1, 2, 4, 8, \ldots, 2^{\lfloor \log(t-t'+1)\rfloor - 1}, (t - t') - 2^{\lfloor \log(t-t'+1)\rfloor} + 1$ into the input multiset. The subset sums of these extra numbers are exactly $0, 1, 2, \ldots, t-t'$, so we can implement the oracle by running the decision algorithm with target number t.*

Note that a space-s decision algorithm for Subset Sum also implies a search algorithm that finds a subset sum solution (assuming the outputs of the algorithm are written in an append-only storage), with space complexity $O(s + \log t)$ and an extra multiplicative factor of n in the time complexity. By iterating over all $i = 1, \ldots, n$, and solving the instance $a_{i+1}, \ldots, a_n$ with target $t - a_i$ (which can be stored in $O(\log t)$ space), we can determine whether a_i can be included in the subset sum solution for all i, updating the target value accordingly. (For each a_i, we can output whether or not it is included on append-only storage.)

The prior results (compared with ours) are summarized in Figure 1.

An Approximation Algorithm. As an application of our techniques, we also give "weak" approximation algorithms for subset sum with low space usage. We define a *weak ε-approximation algorithm* to be one that, on an instance $A = [a_1, a_2, \ldots, a_n], t$, can distinguish the following two cases:

1. **YES:** There is a subset $S \subseteq [n]$ such that $(1 - \varepsilon/2)t \le \sum_{i\in S} a_i \le t$.
2. **NO:** For all subsets $S \subseteq [n]$ either $\sum_{i\in S} a_i > (1+\varepsilon)t$ or $\sum_{i\in S} a_i < (1-\varepsilon)t$.

Mucha, Węgrzycki, and Włodarczyk [MWW19], who introduced the search version of the above problem, gave a "weak" ε-approximation algorithm in $\widetilde{O}(n + 1/\varepsilon^{5/3})$ time and space. They observed that this weak approximation algorithm implies an approximation algorithm (in the usual sense) for the *Partition problem* (see [BN19] for further improvement on this problem), which is a special case of the Subset Sum problem where the input satisfies $a_1 + \cdots + a_n = 2t$.

We show that our techniques can be applied to give an extremely space-efficient algorithm for weak approximations of Subset Sum:

Theorem 1.5. *There exists a $\widetilde{O}(\min\{n^2/\varepsilon, n/\varepsilon^2\})$-time and $O(\operatorname{poly}\log(nt))$-space algorithm for weak approximation of Subset Sum.*

Reference	Time	Space (bits)	D/R	Caveats
[LN10]	$\widetilde{O}(n^3t)$	$\widetilde{O}(n^2 \log t)$	D	
[EJT10]	$(nt)^{O(1)}$	$O(\log(nt))$	D	
[Kan10]	$\widetilde{O}(n^3t + n^{2.025}t^{1.025})$	$O(\log(nt))$	D	
Corollary 3.2	$\widetilde{O}(n^3t)$	$O(\log(nt))$	D	
Theorem 1.2	$\widetilde{O}(n^2t)$	$O(\log t \cdot \log^3 n)$	D	
[Bri17]	$\widetilde{O}(nt)$	$\widetilde{O}(n \log t)$	R	**Requires GRH**
[Bri17]	$\widetilde{O}(nt^{1+\varepsilon})$	$\widetilde{O}(nt^{\varepsilon})$	R	
Theorem 1.1	$\widetilde{O}(nt)$	$O(\log(nt))$	R	**Access to** $O(\log n \log\log n)$ **random bits**
Theorem 1.1	$\widetilde{O}(n^{1+\varepsilon}t)$	$O(\log(nt))$	R	
[Bri17]	$\widetilde{O}(n+t)$	$\widetilde{O}(t)$	R	
Theorem 1.3	$\widetilde{O}((n+t)\cdot k)$	$\widetilde{O}(t/k)$	R	$1 \le k \le \min\{n, t\}$

Figure 1: Best known (pseudo-polynomial) time-space upper bounds for solving Subset Sum with n elements with a target t. "D" means the algorithm is deterministic, "R" means randomized.

1.2 Overview

All of our algorithms build upon a key idea in Kane's logspace algorithm. Kane skillfully applies the following simple number-theoretic fact (the proof of which we will recall in Section 3):

Lemma 1.6. *Let p be an odd prime and let $a \in [1, 2p-3]$ be an integer.*

- *If $a \neq p-1$ then* $\sum_{x=1}^{p-1} x^a \equiv 0 \pmod{p}$.
- *If $a = p-1$ then* $\sum_{x=1}^{p-1} x^a \equiv p-1 \pmod{p}$.

Given an instance $a_1, \ldots, a_n, t$ of Subset Sum, Kane constructs the "generating function" $E(x) = x^{p-1-t}\prod_{i=1}^{n}(1+x^{a_i}) \pmod{p}$ for a prime $p > nt$. In particular, we compute $E' = \sum_{x=1}^{p-1} E(x) \pmod{p}$. The key idea is that, by Lemma 1.6, subset sums equal to t will contribute $p-1$ in the sum E', while all other subset sums will contribute 0, allowing us to detect subset sums (if p is chosen carefully). The running time for evaluating E' is $\Omega(n^2t)$ because of the large choice of p. Intuitively, Kane needs $p \geq \Omega(nt)$ because the "bad" subset sums (which we want to cancel out) can be as large as $\Omega(nt)$, and modulo p there is no difference between t and $t+p$.

Our algorithms manage to get away with choosing $p \leq \widetilde{O}(t)$, but they accomplish this in rather different ways. We now briefly describe the three main theorems.

The Randomized Logspace Algorithm. Our randomized logspace algorithms (Theorem 1.1) combine ideas from Kane's logspace algorithm and Bringmann's faster $\widetilde{O}(n+t)$-time algorithm, applying several tools from the theory of pseudorandom generators (k-wise δ-dependent hash functions and explicit constructions of expanders) to keep the amount of randomness low.

The key is to apply an idea from Bringmann's Subset Sum algorithm [Bri17]: we randomly separate the numbers of the Subset Sum instance into different ranges. For each range, we compute corresponding generating functions on the numbers in those ranges, and combine ranges (by multiplication) to obtain the

value of the generating function for the whole set. If we can keep the individual ranges small, we can ensure a prime $p \leq \widetilde{O}(t)$ suffices. To perform the random partitioning with low space, we define a new notion of "efficiently invertible hash functions" and prove they can be constructed with good parameters. Such a hash function h has a load-balancing guarantee similar to a truly random partition, and the additional property that, given a hash value v, we can list all the preimages x satisfying $h(x) = v$ with time complexity that is near-linear in the number of such preimages. Such families of invertible hash functions may be of independent interest.

The Deterministic Algorithm. In the randomized logspace algorithm, we modified Kane's algorithm so it can work modulo a prime $p \leq \widetilde{O}(t)$. This was achieved by mimicking Bringmann's randomized partitioning and color-coding ideas. While this approach yields fast low-space randomized algorithms, it seems one cannot derandomize color-coding deterministically without a large blowup in time.

Instead of randomly partitioning, our deterministic algorithm (Theorem 1.2) uses a very different approach. We apply deterministic approximate counting in low space (approximate logarithm), in the vein of Morris's algorithm [Mor78] for small-space approximate counting, to keep track of the number of elements contributing to a subset sum. This allows us to work modulo a prime $p \leq \widetilde{O}(t)$. Along the way, we define a special polynomial product operation that helps us compute and catalog approximate counts efficiently.

The Time-Space Trade-off. Our time-space trade-off algorithm (Theorem 1.3) uses a batch evaluation idea and additional algebraic tricks. In Kane's framework, we need to evaluate the generating function at all non-zero points $x = 1, 2, \ldots, p-1$. To reduce the running time, we perform the evaluation in k batches, where each batch has $(p-1)/k$ points to evaluate.[4] In one batch, letting the evaluation points be $b_1, b_2, \ldots, b_{(p-1)/k}$, we define a polynomial

$$B(x) := (x - b_1)(x - b_2) \cdots (x - b_{(p-1)/k})$$

of degree $(p-1)/k$. By choosing the batches of evaluation points judiciously (see Lemma 6.1), we can ensure that for every batch, the polynomial $B(x)$ has only two terms, making it possible to perform the $\bmod B(x)$ operation efficiently in low space. We compute the generating function modulo $B(x)$, which agrees with the original polynomial on the evaluation points b_i. Then, we can efficiently recover the values at these points using fast multipoint evaluation [Fid72]. By performing the evaluation in this way, the space complexity of performing polynomial operations becomes roughly p/k. Using Bringmann's algorithm, we can choose $p \leq \widetilde{O}(t)$. The time complexity of one batch is $\widetilde{O}(n + t)$, so the total time is $\widetilde{O}((n + t)k)$.

The Approximation Algorithm. All of our subset sum algorithms can be modified to work even if we want to check if there is a subset with sum in a specified *range*, instead of being a fixed value. This property is used in our "weak" ε-approximation algorithm (Theorem 1.5). Our approximation algorithms also use ideas from previous subset sum approximation algorithms [MWW19, KMPS03] such as rounding elements and separating elements into bins of "small" and "large" elements.

1.3 Other Related Work on Subset Sum

The first $\widetilde{O}(n + t)$-time (randomized) algorithm for Subset Sum was given by Bringmann [Bri17] (see also [JW19] for log-factor improvements). For deterministic algorithms, the best known running time is $\widetilde{O}(n + t\sqrt{n})$ [KX19]. Abboud, Bringmann, Hermelin, and Shabtay [ABHS19] proved that there is no $t^{1-\delta} \cdot 2^{o(n)}$ time algorithm for Subset Sum for any $\delta > 0$, unless the Strong Exponential Time Hypothesis (SETH) fails.

[4] For simplicity, in this discussion we work modulo a prime p. For technical reasons, our actual algorithms have to work over an arbitrary finite field $\mathbb{F}_q$ where q is a prime power. Informally, this is because we need our parameter k to divide $p - 1$; if we allow p to be an arbitrary prime power and apply some analytic number theory, this is essentially possible by adjusting p and k.

Recently, Bringmann and Wellnitz [BW20] showed an $\widetilde{O}(n)$-time algorithm for Subset Sum instances that satisfy certain density conditions.

Bansal, Garg, Nederlof, and Vyas [BGNV18] obtained space-efficient algorithms for Subset Sum with running time exponential in n but polynomial in $\log t$. They showed that Subset Sum (and Knapsack) can be solved in $2^{0.86n} \cdot \mathrm{poly}(n, \log t)$ time and $\mathrm{poly}(n, \log t)$ space (assuming random read-only access to exponentially many random bits). Note that currently the fastest algorithm in this regime uses $2^{n/2} \cdot \mathrm{poly}(n, \log t)$ time and $2^{n/4} \cdot \mathrm{poly}(n, \log t)$ space [HS74, SS81] (the space complexity was recently slightly improved by Nederlof and Węgrzycki [NW20]).

Limaye, Mahajan, and Sreenivasaia [LMS12] proved that Unary Subset Sum (where all numbers are written in unary) is TC^0-complete. This implies that Subset Sum can be solved by constant-depth circuits with a *pseudopolynomial* number of MAJORITY gates.

It is well-known that Subset Sum also has a fully polynomial-time approximation scheme (FPTAS), which outputs a subset with sum in the interval $[(1-\varepsilon)t', t']$, where t' is the maximum subset sum not exceeding t. The most efficient known approximation algorithms are [KMPS03] which runs in time about $\widetilde{O}(\min\{n/\varepsilon, n + 1/\varepsilon^2\})$ and space $\widetilde{O}(n + 1/\varepsilon)$, and [GJL+16] which runs in time $O(n^2/\varepsilon)$ and space $O((\log t)/\varepsilon + \log n)$. Bringmann and Nakos [BN19] showed that the time complexity of approximating Subset Sum cannot be improved to $(n+1/\varepsilon)^{2-\delta}$ for any $\delta > 0$, unless Min-Plus Convolution can be computed in truly subquadratic time.

2 Preliminaries

Let $[n]$ denote $\{1, 2, \ldots, n\}$, and let $\log x$ denote $\log_2 x$. We will use $\widetilde{O}(f)$ (and $\widetilde{\Omega}, \widetilde{\Theta}$) to denote a bound omitting $\mathrm{poly}\log(f)$ factors.

2.1 Finite Fields

We will use finite field arithmetic in our algorithm. Given $k \geq 1$ and prime p, we construct the finite field $\mathbb{F}_q$ of order $q = p^k$ by finding an irreducible polynomial of degree k over $\mathbb{F}_p$, which can be solved by a (Las Vegas) randomized algorithm in time $\mathrm{poly}(\log p, k) \leq \mathrm{poly}\log q$ (e.g., [Sho94]).

For $k = 2$, an irreducible polynomial can be found deterministically in $O(q)$ time and $O(\log q)$ space: it suffices to find a quadratic non-residue modulo p, since $x^2 - a$ can be factored iff a has a square root modulo p. To find a quadratic non-residue, we simply enumerate all $a \in \mathbb{F}_p^*$ and check if a has a square root in $O(p)$ time and $O(\log p)$ space. The total running time is $O(p^2) = O(q)$.

2.2 Expanders

To keep the randomness in our algorithms low, we use "strongly explicit" constructions of expander graphs.

Definition 2.1 (Expander graphs)**.** An n-vertex undirected graph G is an (n, d, λ)-*expander graph* if G is d-regular and $\lambda(G) \leq \lambda$, where $\lambda(G)$ is the second largest eigenvalue (in absolute value) of the normalized adjacency matrix of G (i.e., the adjacency matrix of G divided by d).

For constant $d \in \mathbb{N}$ and $\lambda < 1$, a family of graphs $\{G_n\}_{n\in\mathbb{N}}$ is a (λ, d)-*expander graph family* if for every n, G_n is an (n, d, λ)-expander graph.

Theorem 2.2 (Explicit Expanders [Mar73, GG81])**.** *There exists a (λ, d)-expander graph family $\{G_n\}$ for some constants $d \in \mathbb{N}$ and $\lambda < 1$, along with an algorithm that on inputs $n \in \mathbb{N}, v \in [n], i \in [d]$ outputs the i-th neighbor of v in graph G_n in $\mathrm{poly}\log n$ time and $O(\log n)$ space.*

We also need the following tail bound, showing that a random walk on an expander graph behaves similarly to a sequence of i.i.d. randomly chosen vertices.

Theorem 2.3 (Expander Chernoff Bound, [Gil98]). *Let G be an (n, d, λ)-expander graph on the vertex set $[n]$. Let $f : [n] \to \{0, 1\}$, and $\mu = \mathbb{E}_{v\in[n]} f(v)$. Let $v_1, v_2, \ldots, v_t$ be a random walk on G (where v_1 is uniformly chosen, and v_{i+1} is a random neighbor of v_i for all i). Then for $\delta > 0$,*

$$\Pr_{v_1,\ldots,v_t}\left[\frac{1}{t}\sum_{i=1}^{t} f(v_i) < \mu - \delta\right] \leq e^{-(1-\lambda)\delta^2 t/4}.$$

2.3 Polynomial operations

We will also utilize the classic multipoint evaluation algorithm for (univariate) polynomials.

Theorem 2.4 (Fast multipoint evaluation [Fid72]). [5] *Every univariate polynomial over $\mathbb{F}$ of degree less than n can be evaluated at n points in $O(M(n)\log n)$ time and $O(n \log n \cdot \log |\mathbb{F}|)$ space, where $M(n)$ is the time complexity of polynomial multiplication over $\mathbb{F}$.*

Note that we can perform polynomial multiplication over $\mathbb{F}_q$ using FFT in $\widetilde{O}(n \cdot \operatorname{poly}\log(q))$ time.

3 Kane's Number-Theoretic Approach

In this section, we will review Kane's number-theoretic technique for solving Subset Sum [Kan10]. Kane used this technique to obtain a *deterministic* algorithm for Subset Sum, but it is straightforward to adapt it to the *randomized* setting. We will also improve Kane's deterministic algorithm.

As discussed in the introduction section, Kane utilizes the following simple fact.

Lemma 3.1. *Let $q = p^k$ be a prime power and let a be a positive integer. Let $\mathbb{F}_q^*$ denote the set of non-zero elements in the finite field $\mathbb{F}_q$. Then*

- *If $q - 1$ does not divide a, then $\sum_{x\in\mathbb{F}_q^*} x^a = 0$.*
- *If $q - 1$ divides a, then $\sum_{x\in\mathbb{F}_q^*} x^a = -1$.*

Proof. The multiplicative group $\mathbb{F}_q^*$ is cyclic. Let g be a generator of $\mathbb{F}_q^*$. For $q - 1 \nmid a$, $g^a \neq 1$ and we have

$$\sum_{x\in\mathbb{F}_q^*} x^a = \sum_{i=0}^{q-2} g^{ai} = \frac{1 - g^{a(q-1)}}{1 - g^a} = 0.$$

When $q - 1 \mid a$,

$$\sum_{x\in\mathbb{F}_q^*} x^a = (q - 1) \cdot 1 = -1.$$

The proof is complete. □

Corollary 3.2. *Let $f(x) = \sum_{i=0}^{d} c_i x^i$ be a polynomial of degree at most d, where coefficients c_i are integers. Let $\mathbb{F}_q$ be the finite field of order $q = p^k \geq d + 2$. For $0 \leq t \leq d$, define*

$$r_t := \sum_{x\in\mathbb{F}_q^*} x^{q-1-t} f(x) \in \mathbb{F}_q.$$

Then $r_t = 0$ if and only if c_t is divisible by p.

[5] A good modern reference is [vzGG13, Section 10.1].

Proof. We have

$$r_t = \sum_{i=0}^{d} c_i \sum_{x \in \mathbb{F}_q^*} x^{i+q-1-t} = -c_t,$$

where the last equality follows from Lemma 3.1 and $|i - t| \leq d < q - 1$. Hence r_t equals 0 iff $-c_t$ is an integer multiple of the characteristic of $\mathbb{F}_q$. □

The following lemma generalizes the main technique of Kane's algorithm for Subset Sum [Kan10], extending it to randomized algorithms.

Lemma 3.3 (Coefficient Test Lemma)**.** *Consider a polynomial $f(x) = \sum_{i=0}^{d} c_i x^i$ of degree at most d, with integer coefficients satisfying $|c_i| \leq 2^w$. Suppose there is a T-time S-space algorithm for evaluating $f(a)$ given $a \in \mathbb{F}_q$, where $q \leq O(d + w)$.*

Then, there is an $O((d + w) \cdot (T + w + \log d))$-time and $O(S + \log(dw))$-space algorithm using $\log w - \log\log w + O(1)$ random bits that, given $0 \leq t \leq d$, tests whether $f(x)$ has a non-zero x^t coefficient c_t, with at most 0.01 one-sided probability of error.

Trying all random choices, this yields a deterministic algorithm in $O((d+w) \cdot (T+w+\log d) \cdot w/\log w)$ time and $O(S + \log(dw))$ space.

Proof. We consider two cases, based on whether the degree of f is "high" or "low" compared to the cofficient size.

Case 1: $d > w^2$**.** By the Prime Number Theorem, we can pick $B = O(d)$ such that the interval $[\sqrt{d+2}, \sqrt{B}]$ contains at least $m = 100w/\log\sqrt{d+2}$ primes. We can deterministically find such primes $p_1, \ldots, p_m$ by simple trial division, in $O(\log d)$ space and $O(\sqrt{B} \cdot (\sqrt{B})^{1/2}) \leq O(d^{3/4})$ total time. Let $q_i := p_i^2$. Then the q_i's are prime powers in the interval $[d + 2, B]$.

Randomly pick $i \in [m]$, and let $q := q_i, p := p_i$. We compute $r_t = \sum_{x \in \mathbb{F}_q^*} x^{q-1-t} f(x)$ by running the evaluation algorithm (and the modular exponentiation algorithm for computing x^{q-1-t}) $(q - 1)$ times, in $(q - 1) \cdot (T + O(\log q)) \leq O(d(T + \log d))$ total time. By Corollary 3.2, $r_t = 0$ if and only if p is a prime factor of c_t. As $|c_t| \leq 2^w$, a non-zero c_t has at most $w/\log\sqrt{d+2}$ prime factors from the interval $[\sqrt{d+2}, \sqrt{B}]$, so the probability that p is a prime factor is at most $(w/\log\sqrt{d+2})/m = 1/100$.

Case 2: $d \leq w^2$**.** By the Prime Number Theorem, setting $B = \Theta(d+w)$, the interval $[\max\{d+2, w\}, B]$ contains at least $m = 100w/\log w$ primes. As in the first case, we can deterministically find such primes $p_1, \ldots, p_m$ by trial division, in $O(\log B)$ space and $O(B \cdot \sqrt{B}) \leq O((d + w)w)$ total time.

We randomly pick $i \in [m]$ and let $p := p_i$. We compute $r_t = \sum_{x \in \mathbb{F}_p^*} x^{p-1-t} f(x)$ by running the evaluation algorithm $p - 1$ times, in $(p - 1) \cdot (T + O(\log p)) \leq O((d + w)(T + \log(d + w)))$ total time. By Corollary 3.2, $r_t = 0$ if and only if p is a prime factor of c_t. As $|c_t| \leq 2^w$, a non-zero c_t has at most $w/\log w$ prime factors from the interval $[\max\{d + 2, w\}, B]$, so the probability that p is a prime factor is at most $1/100$.

To make the above tests deterministic, we simply iterate over all $i \in [m]$. Note that we only used finite fields of order $q = p$ or $q = p^2$, the latter of which can be constructed deterministically in $O(q)$ time and $O(\log q)$ space (see Section 2.1). □

Corollary 3.4. *Subset Sum can be solved deterministically in $O(n^3 t \log t/\log n)$ time and $O(\log(nt))$ space.*

Proof. Define $f(x) = \prod_{i=1}^{n}(1 + x^{a_i})$ as the generating function of the Subset Sum instance. It has degree $d = \sum_{i=1}^{n} a_i \leq nt$, and integer coefficients at most 2^n. Given $x \in \mathbb{F}_q$, $f(x)$ can be evaluated in $T = O(n \log t)$ time, $O(\log(nt))$ space. By Lemma 3.3 we have an $O(n^3 t \log t/\log n)$-time $O(\log(nt))$-space deterministic algorithm. □

Remark 3.5. *Our deterministic algorithm is slightly different from Kane's original algorithm (which is essentially the "Case 2" in the proof of Lemma 3.3). Kane's deterministic algorithm needs to find $\widetilde{\Omega}(n)$ many primes $p > nt$ for constructing the finite fields. This could be time-consuming, as it takes square-root time to perform primality test*[6]*. Our algorithm solves this issue by working over $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_p$ (as in "Case 1")*[7]*.*

Here we give an upper bound on the running time of Kane's original algorithm for finding primes. By the prime number theorem, in the interval $[nt, 100nt]$ there exist at least $nt/\log(nt) \geq n/\log n$ primes. On the other hand, by [BHP01], the interval $[N, N + O(N^{0.525})]$ contains a prime, so we can find $\widetilde{\Omega}(n)$ primes from the interval $[nt, nt + n \cdot (nt)^{0.525}]$. Hence, it suffices to search for all the primes from the interval $[nt, nt + \min\{99nt, n \cdot (nt)^{0.525}\}]$, in $O(\min\{nt, n \cdot (nt)^{0.525}\} \cdot \sqrt{nt})$ time and $O(\log(nt))$ space. So the total time complexity of Kane's algorithm is $\widetilde{O}(\min\{nt, n \cdot (nt)^{0.525}\} \cdot \sqrt{nt} + n^3 t) = \widetilde{O}(n^{2.025} t^{1.025} + n^3 t)$, where the first term becomes dominant when $t \gg n^{39}$.

4 Randomized Algorithm

Our randomized algorithm uses *color-coding* (i.e., random partitioning), which plays a central role in Bringmann's algorithm for Subset Sum [Bri17]. By randomly partitioning the input elements into groups, with high probability, the number of *relevant* elements (i.e., elements appearing in a particular subset with sum equal to t) that any group receives is not much larger than the average. Once this property holds, then from each group we only need to compute the possible subset sums achievable by subsets of small size.

However, storing a uniform random partition of the elements will, in principle, require substantial space (at least $\Omega(n)$). We shall modify the random partitioning technique in a way that uses very low space. To do this, we need a way to quickly report the elements from a particular group, without explicitly storing the whole partition in memory. We formalize our requirements in the following definition.

Definition 4.1 (Efficiently Invertible Hash Functions)**.** For $1 \leq m \leq n$, a family $\mathcal{H} = \{h \colon [n] \to [m]\}$ is a family of *efficiently invertible hash functions with parameter $k(n) \geq 1$, seed length $s(n) \geq \log n$, and failure probability δ*, if $|\mathcal{H}| = 2^{s(n)}$ and the following properties hold:

(Load Balancing) Let $S_i = \{x \in [n] : h(x) = i\}$. For every set $S \subseteq [n]$ of size m,

$$\Pr_{h \sim \mathcal{H}} \left[\max_{i \in [m]} |S \cap S_i| \leq k(n) \right] > 1 - \delta.$$

(Efficient Invertibility) Given an $s(n)$-bit seed specifying $h \in \mathcal{H}$ and $i \in [m]$, all elements of S_i can be reported in $O(|S_i| \cdot \operatorname{poly}\log n)$ time and $O(\log n)$ additional space.[8]

The goal of this section is to show that good implementations of efficiently invertible hash functions imply time and space efficient Subset Sum algorithms.

Theorem 4.2. *Suppose for every constant $c \geq 1$, one can implement an efficiently invertible hash family with parameter $k(n) \geq 1$, seed length $s(n) \geq \log n$ and failure probability $1/n^c$.*

Then, given random and read-only access to a string of $s(n)$ random bits, Subset Sum can be solved with constant probability in $\widetilde{O}(nt \cdot \operatorname{poly}(k(n)))$ time and $O(\log(nt))$ space. This implies that Subset Sum can be solved with constant probability in $\widetilde{O}(nt \cdot \operatorname{poly}(k(n)))$ time and $O(s(n) + \log(t))$ space.

[6] We could of course use more time-efficient algorithms such as AKS [AKS04], but it is not clear how to implement them in logspace, rather than poly-log space.

[7] This extension was already observed by Kane in [Kan10, Section 3.1], but the purpose there was not to reduce the time complexity of generating primes.

[8] Note that the output bits do not count towards the space complexity.

Here we state two efficiently invertible hash families of different parameters and seed lengths. Their construction will be described in Section 5.

Theorem 4.3. *For any constants $c \geq 1$ and $\varepsilon > 0$,*

(1) there is an efficiently invertible hash family with parameter $k(n) = O(\log n)$, seed length $s(n) = O(\log n \log\log n)$ and failure probability $1/n^c$. And,

(2) there is an efficiently invertible hash family with parameter $k(n) = O(n^\varepsilon)$, seed length $s(n) = O(\log n)$, and failure probability $1/n^c$.

Note that Theorem 1.1 immediately follows from Theorem 4.2 and Theorem 4.3.

The algorithm given by Theorem 4.2 has a similar overall color-coding structure as in Bringmann's Subset Sum algorithm [Bri17], but it applies a generating function approach akin to Kane's Subset Sum algorithm [Kan10]. In particular, on a Subset Sum instance with target t, the algorithm constructs a polynomial $f(x)$ (depending on the input and the random bits), which always has a zero x^t coefficient for NO-instances, and with 0.99 probability has a non-zero x^t term for YES-instances. Applying the Coefficient Test Lemma (Lemma 3.3), we can use the evaluation of $f(x)$ to solve Subset Sum.

4.1 Description of the algorithm

We now proceed to proving Theorem 4.2. We describe an algorithm `Evaluate` that evaluates such a polynomial $f(x)$ over $\mathbb{F}_q$. To keep the pseudocode clean, we do not specify the low-level implementation of every step. In the next section, we will elaborate on how to implement the pseudocode in small space.

Layer Splitting. We use the layer splitting technique from Bringmann's Subset Sum algorithm [Bri17]. Given the input (multi)set of integers $A = \{a_1, \ldots, a_n\}$ and target t, we define $L_i := A \cap (t/2^i, t/2^{i-1}]$ for $i = 1, 2, \ldots, \lceil \log n \rceil - 1$, and let $L_{\lceil \log n \rceil} = A \backslash (L_1 \cup \cdots \cup L_{\lceil \log n \rceil - 1})$.

In the following analysis, for a given YES-instance, fix an arbitrary solution set $R \subseteq A$ which sums to t. The elements appearing in R are called *relevant*, and the elements in $A \setminus R$ are irrelevant. Observe that the i-th layer L_i can only contain at most 2^i relevant elements.

Two-level random partitioning. For each layer L_i, we apply two levels of random partitioning, also as in Bringmann's algorithm. In the first level, we use our efficiently invertible hash function (specified by a random seed r_1 of length $s(n)$) to divide L_i into $\ell = 2^i$ groups $S_1, \ldots, S_\ell$, so that with high probability each S_j contains at most k relevant elements.

In the second level, we use a pairwise independent hash function (specified by random seed v of length $s' = O(\log n)$) to divide each group S_j into k^2 mini-groups $T_1, \ldots, T_{k^2}$, so that with $\geq 1/2$ probability each relevant element in S_j appears in a *distinct* T_i, isolated from the other relevant elements in S_j.

We repeat the second-level partitioning for $m = O(\log n)$ rounds, increasing the success probability to $1 - 1/\operatorname{poly}(n)$. However, instead of using fresh random bits each round (in which the total seed length would become $ms' = O(\log^2 n)$), we use the standard expander-walk sampling technique: we pseudorandomly generate the seeds $v_1, \ldots, v_m$ by taking a length-m random walk on a strongly explicit expander graph (Theorem 2.2), where each vertex v_i represents a possible s'-bit seed. The random walk is specified by the seed r_2 which only has $s' + O(m) = O(\log n)$ bits.

Function `Evaluate`(x, A, t, r_1, r_2)

Let $L_i := A \cap (t/2^i, t/2^{i-1}]$ for $i = 1, 2, \ldots, \lceil \log n \rceil - 1$
Let $L_{\lceil \log n \rceil} := A \setminus (L_1 \cup \cdots \cup L_{\lceil \log n \rceil - 1})$
$u = 1$
for $i = 1, \ldots, \lceil \log n \rceil$ **do**
    $u = u \cdot$ `PartitionLevel1`(x, L_i, i, r_1, r_2)
return u

Function `PartitionLevel1`(x, L, i, r_1, r_2)

Let $\ell = 2^i$
Partition $L = S_1 \dot\cup \cdots \dot\cup S_\ell$ using the efficiently invertible hash function h with parameter $k(n)$, specified by seed r_1
$u = 1$
for $j = 1, \ldots, \ell$ **do**
    $u = u \cdot$ `PartitionLevel2`(x, S_j, k, r_1, r_2)
return u

Function `PartitionLevel2`(x, S, k, r_1, r_2)

$u = 0, m = c' \log n$ (where c' is a sufficiently large constant)
Let $v_1, \ldots, v_m \in \{0,1\}^{O(\log n)}$ be the seeds extracted from r_2 using expander walk sampling.
for $j = 1, \ldots, m$ **do**
    Partition $S = T_1 \dot\cup \cdots \dot\cup T_{k^2}$ using the pairwise-independent hash function specified by seed v_j
    $w = \prod_{i=1}^{k^2} \left(1 + \sum_{a \in T_i} x^a\right)$
    $u = u + w$
return u

Analysis. Now we turn to analyzing the algorithm. It is easy to see that the output values of all three algorithms `Evaluate`, `PartitionLevel1`, and `PartitionLevel2` can be expressed as *polynomials* in the variable x, with coefficients determined by the parameters (except x) passed to the function calls. In the following we will state and prove the key properties satisfied by the polynomials output by these algorithms.

In our analysis, the "x^σ term" of a polynomial will be construed as the integer coefficient of x^σ over $\mathbb{Z}$, although in our actual implementation we will perform all arithmetic operations (additions and multiplications) in $\mathbb{F}_q$ (as in the Coefficient Test of Lemma 3.3).

Lemma 4.4. *The output of* `PartitionLevel2`(x, S, k, r_1, r_2) *is a polynomial $P(x)$ of degree at most $k^2 \cdot \max_{a \in S} a$, with non-negative coefficients bounded by $P(1) \le 2^{O(k^2 \log n)}$.*

Suppose S contains at most k relevant elements $R_S \subseteq S$, and let $\sigma = \sum_{a \in R_S} a$. Then $P(x)$ contains a positive x^σ term with $1 - \frac{1}{n^{c_2}}$ probability for any desired constant $c_2 \ge 1$.

Proof. By construction, the output polynomial $P(x)$ has degree at most $k^2 \cdot \max_{a \in S} a$ and non-negative coefficients bounded by $P(1) \le m \cdot (n+1)^{k^2} \le 2^{O(k^2 \log n)}$.

Let $h'_v\colon [|S|] \to [k^2]$ be the pairwise independent hash function specified by the seed v of length $s' \le O(\log|S| + \log k^2) \le O(\log n)$ bits (such a pairwise independent family can be easily evaluated in logspace and essentially linear time, cf. [Vad12]). For randomly chosen v, each pair of distinct relevant items are hashed into the same mini-group T_i with probability $1/k^2$. By a union bound over $k(k-1)/2$ pairs of relevant items, the probability that all relevant items are isolated (end up in distinct mini-groups) is at least $1 - \frac{k(k-1)/2}{k^2} \ge 1/2$.

The seeds $v_1, \ldots, v_m$ are generated by taking a length-m random walk on a strongly explicit $(2^{s'}, d, \gamma)$-expander graph for some constants $d \in \mathbb{N}, \lambda < 1$ (Theorem 2.2). By the Expander Chernoff Bound (Theorem 2.3), over the choice of random seeds $v_1, \ldots, v_m$,

$$\Pr\left[\frac{1}{m}\sum_{j=1}^{m}[v_j \text{ isolates all relevant items}] < 1/2 - 1/4\right] \le e^{-(1-\lambda)(1/4)^2 m/4}.$$

Choosing $m := c' \log n$ for a sufficiently large constant c', the above probability is at most n^{-c_2} for any desired constant $c_2 \ge 1$. Hence with probability at least $1 - n^{-c_2}$, there is a $j \in [m]$ such that the seed v_j isolates all relevant items into different mini-groups $T_1, \ldots, T_{k^2}$. In such a case, the product

$$\prod_{i=1}^{k^2}\left(1 + \sum_{a \in T_i} x^a\right)$$

must have a positive x^σ term. Hence, the sum u also has a positive x^σ term. □

Lemma 4.5. *The output of* $\texttt{PartitionLevel1}(x, L, i, r_1, r_2)$ *is a polynomial* $Q(x)$ *of degree at most* $2k^2t$*, with non-negative coefficients bounded by* $Q(1) \le 2^{O(2^i k^2 \log n)}$.

Let $R_L \subseteq L$ *be the subset of relevant elements, and* $\sigma = \sum_{a \in R_L} a$*. Then* $Q(x)$ *contains a positive* x^σ *term with* $1 - \frac{1}{n^{c_3}}$ *probability for any desired constant* $c_3 \ge 1$.

Proof. By Lemma 4.4, the degree of every return value of $\texttt{PartitionLevel2}(x, S_j, k, r_1, r_2)$ as a polynomial is at most $k^2 \cdot \max_{a \in S_j} a$. Hence, the degree of $Q(x)$ as a polynomial is at most $\ell \cdot k^2 \cdot \max_{a \in L_i} a \le \ell \cdot k^2 \cdot t/2^{i-1} = 2k^2 t$. And $Q(1)$ is the product of $\ell = 2^i$ many $P(1)$'s, which is at most $2^{O(2^i k^2 \log n)}$.

Recall that $|R_L| \le \ell = 2^i$. Let $R_j := R_L \cap S_j$ and $\sigma_j := \sum_{a \in R_j} a$. By the properties of efficiently invertible hash families, we have $|R_j| \le k$ for every $j \in [\ell]$ with probability $1 - n^{-c}$.

Thus by Lemma 4.4 and the union bound, with probability $1 - n^{-c_3}$ (for any desired constant $c_3 \ge 1$), for every $j \in [\ell]$, the output polynomial $P(x)$ of $\texttt{PartitionLevel2}(x, S_j, k, r_1, r_2)$ has a positive x^{σ_j} term. In such a case, $Q(x)$ has a positive $x^{\sigma_1 + \cdots + \sigma_\ell} = x^\sigma$ term. □

Lemma 4.6. *The output of* $\texttt{Evaluate}(x, A, r_1, r_2)$ *is a polynomial* $S(x)$ *of degree at most* $2k^2t \cdot \lceil \log n \rceil$*, with non-negative coefficients bounded by* $S(1) \le 2^{O(\min\{n,t\} \cdot k^2 \log n)}$.

Let $R \subseteq A$ *be the subset of relevant elements, and* $t = \sum_{a \in R} a$*. Then* $S(x)$ *contains a positive* x^t *term with* $1 - \frac{1}{n^{c_4}}$ *probability, for any desired constant* $c_4 \ge 1$.

Proof. By Lemma 4.5, each call to $\texttt{PartitionLevel1}(x, L_i, i, r_1, r_2)$ returns a polynomial $Q(x)$ of degree at most $2k^2t$ with non-negative coefficients, and $Q(1) \le 2^{O(2^i k^2 \log n)}$. Hence, the degree of the product polynomial $S(x)$ is at most $2k^2t \cdot \lceil \log n \rceil$. And,

$$S(1) \le 2^{O(k^2 \log n) \cdot (2^1 + 2^2 + \cdots + 2^{i'})} \le 2^{O(2^{i'} k^2 \log n)},$$

where $i' \le \lceil \log n \rceil$ and $t/2^{i'-1} \ge 1$ (for non-empty $L_1, \ldots, L_{i'}$).

Let $R_i := R \cap L_i$ and $\sigma_i = \sum_{a \in R_i} a$. By Lemma 4.5 and the union bound, with probability at least $1 - n^{-c_4}$ (for any desired constant $c_4 \geq 1$), for every $i = 1, \ldots, \lceil \log n \rceil$, the output polynomial $Q(x)$ of $\texttt{PartitionLevel1}(x, L_i, i, r_1, r_2)$ contains a positive x^{σ_i} term. In such a case, the final product $S(x)$ has a positive $x^{\sigma_1 + \cdots + \sigma_{\lceil \log n \rceil}} = x^t$ term. □

4.2 Implementation

Now we describe in more detail how to implement `Evaluate`, `PartitionLevel1`, and `PartitionLevel2` in logarithmic space (assuming read-only access to random seeds r_1, r_2), and analyze the time complexity.

Lemma 4.7. *Assuming read-only access to random seeds r_1, r_2, the procedure* `Evaluate` *(where arithmetic operations are over $\mathbb{F}_q$ with $q = \Omega(t)$) runs in $\widetilde{O}(n \cdot \mathrm{poly}(k, \log q))$ time and $O(\log(nq))$ working space.*

Proof. Recall $A = \{a_1, \ldots, a_n\}$ is the set of input integers. In $\texttt{Evaluate}(x, A, t, r_1, r_2)$, we do not have enough space to collect all elements of L_i and pass them to `PartitionLevel1`. Instead, in `PartitionLevel1`, we partition the set A into ℓ groups $S'_1, \ldots, S'_\ell$ using an efficiently invertible hash function $h \colon [n] \to [\ell]$, specified by seed r_1 (on different layers, the value of $\ell = 2^i$ is different, but we will use the same seed r_1 to generate $h \colon [n] \to [\ell]$). Next, we define the partition of $L_i = S_1 \dot\cup \cdots \dot\cup S_\ell$ by $S_j := S'_j \cap L$. By the efficient invertibility property, assuming read-only access to the seed r_1, we can iterate over the elements of S_j in $O(|S'_j| \cdot \mathrm{poly}\log(n))$ time and $O(\log n)$ space, by iterating over $|S'_j|$ and ignoring the elements that do not belong to L_i. Hence, iterating over $S_1, \ldots, S_\ell$ takes $\widetilde{O}(n)$ time in total.

In $\texttt{PartitionLevel2}(x, S, k, r_1, r_2)$, we iterate over $j = 1, \ldots, m$, and compute each v_j in $O(\log n)$ space by following the expander walk. In the loop body, for every T_i, we compute $\left(1 + \sum_{a \in T_i} x^a\right)$ in $O(\log n + \log q)$ space by iterating over S and ignoring the elements that do not belong to T_i. In total, we iterate over S for mk^2 passes during the execution of $\texttt{PartitionLevel2}(x, S, k, r_1, r_2)$. In `PartitionLevel1`, the procedure `PartitionLevel2` is called once for each S_j, so the total time is $\widetilde{O}(n \cdot mk^2 \cdot \mathrm{poly}\log(q))$. Therefore, over all $\lceil \log n \rceil$ layers, the total time complexity of `Evaluate` is $\widetilde{O}(n \cdot \mathrm{poly}(k \log q))$, and the space complexity is $O(\log(nq))$. □

We complete this section by proving Theorem 4.2, which demonstrates how to solve Subset Sum efficiently from an efficiently invertible hash family by computing `Evaluate` on many values of x.

Proof of Theorem 4.2. By Lemma 4.6, the output of `Evaluate` is a polynomial of degree $d = O(t \cdot \mathrm{poly}(k, \log n))$, with non-negative coefficients bounded by 2^w where $w = O(\min\{n, t\} \cdot \mathrm{poly}(k, \log n))$.

Applying the Coefficient Test Lemma (Lemma 3.3) to the efficient `Evaluate` algorithm of Lemma 4.7 with running time $T = \widetilde{O}(n \cdot \mathrm{poly}(k, \log q))$ where $q \leq O(d + w)$, we can test whether the output of `Evaluate` as a polynomial contains a positive x^t term in $\widetilde{O}((d + w)(T + w)) \leq \widetilde{O}(nt \cdot \mathrm{poly}(k))$ time and $O(\log(nt))$ space. □

To complete our randomized algorithm for Subset Sum, it remains to provide the efficiently invertible hash families claimed in Theorem 4.3, which we do next.

5 Construction of Efficiently Invertible Hash Families

Now we show how to construct the efficiently invertible hash families (Definition 4.1) used by the function `PartitionLevel1` in our algorithm from Section 4. Our construction has a similar structure to the hash family constructed by Celis, Reingold, Segev, and Wieder [CRSW13], which achieves the load-balancing property of our hash functions. By making several modifications to their construction, the hash family can be made efficiently invertible.

Theorem 5.1. *For any $c > 0$, and integers $1 \le m \le n$ which are both powers of 2, there is a family $\mathcal{H}_{n,m}$ of* bijections $h\colon [n] \to [m] \times [n/m]$ *satisfying the following conditions.*

- *Each function $h \in \mathcal{H}_{n,m}$ can be described by an $O(\log n \log\log n)$-bit seed.*
- *Given the seed description of $h \in \mathcal{H}_{n,m}$, $h(x)$ can be computed in $O((\log\log n)^2)$ time and $O(\log n)$ space for any $x \in [n]$, and $h^{-1}(i,j)$ can be computed in $O((\log\log n)^2)$ time and $O(\log n)$ space for any $i \in [m], j \in [n/m]$.*
- *For $i \in [m]$, let $S_i := \{x \in [n] : h(x) = (i,j)$ for some $j \in [n/m]\}$. There is a constant $\gamma > 0$ such that for every set $S \subseteq [n]$ of size m,*

$$\Pr_{h \in \mathcal{H}_{n,m}}\left[\max_{i \in [m]} |S \cap S_i| \le \gamma \log n\right] > 1 - \frac{1}{n^c}.$$

Remark 5.2. *The original construction of [CRSW13] achieves the optimal load-balancing parameter $\frac{\gamma \log n}{\log\log n}$, rather than $\gamma \log n$. Such a bound is also achievable for us; for simplicity we state a weaker version here, which is sufficient for our purpose.*

We observe that Theorem 5.1 immediately implies an efficiently invertible hash family with the desired parameters claimed in Theorem 4.3(1). In particular, to invert a hash value i, we simply iterate over $j \in [n/m]$ and output the unique $x \in [n]$ such that $h(x) = (i,j)$.

Corollary 5.3. *For any constant $c \ge 1$, there is an efficiently invertible hash family with parameter $k(n) = O(\log n)$, seed length $s(n) = O(\log n \log\log n)$ and failure probability $1/n^c$.*

We now describe the construction of $\mathcal{H}_{n,m}$. The analysis of correctness is basically the same as [CRSW13]; for completeness, we include this analysis in Appendix A.

We begin with a construction of an almost k-wise independent hash family.

Definition 5.4. A family $\mathcal{F}$ of functions $f\colon [u] \to [v]$ is k-wise δ-dependent if for any $k' \le k$ distinct elements $x_1, \ldots, x_{k'} \in [u]$, the statistical distance between the distribution $(f(x_1), \ldots, f(x_{k'}))$ where f is uniformly randomly chosen from $\mathcal{F}$ and the uniform distribution over $[v]^{k'}$ is at most δ.

Lemma 5.5 ([AGHP90, MRRR14])**.** *Let $k \cdot \ell \le O(\log n)$, $w \le O(\log n)$, and $\delta = 1/\operatorname{poly}(n)$. There is a k-wise δ-dependent family $\mathcal{H}$ of functions from $\{0,1\}^w$ to $\{0,1\}^\ell$, where each $h \in \mathcal{H}$ can be specified by an $O(\log n)$-bit seed, and each h can be evaluated in $\operatorname{poly}\log n$ time.*

Our construction of the family of bijections in Theorem 5.1 has a d-level structure, where $d = O(\log\log n)$. First we assign some parameters:

- $m_0 = m, m_i = m_{i-1}/2^{\ell_i}$ for every $i \in [d]$;
- $\ell_i = \lfloor (\log m_{i-1})/4 \rfloor$ for $i \in [d-1]$, and $\ell_d = \log m - \sum_{i=1}^{d-1} \ell_i$;
- $k_i \ell_i = \Theta(\log n)$, and k_i is even for every $i \in [d-1]$;
- $k_d = \Theta(\log n / \log\log n)$;
- $\delta = 1/\operatorname{poly}(n)$.

The constant factors $d, k_i, \log(1/\delta)$ all depend on the constant c, and are specified further in the analysis of Appendix A.

For every $i \in [d]$, let

$$n_i := m_i \cdot (n/m).$$

For each $i \in [d]$, independently sample a $g_i \colon [n_i] \to [2^{l_i}]$ from a k_i-wise δ-dependent family using a $O(\log n)$-bit random seed (Lemma 5.5). The total seed length is thus $d \cdot O(\log n) \leq O(\log n \log\log n)$. For $i \in [d]$, we define a bijection $f_i \colon [2^{l_i}] \times [n_i] \to [n_{i-1}]$ by

$$f_i(b, u) := (b \oplus g_i(u)) \circ u,$$

where $\circ$ stands for concatenation. Note that $f_i^{-1}(\cdot)$ can be computed with one evaluation of $g_i(\cdot)$.

Now we define the bijections $h \colon [n] \to [m] \times [n/m]$. Given input $x_0 \in [n]$, let

$$(b_i, x_i) = f_i^{-1}(x_{i-1})$$

for $i = 1, 2, \ldots, d$. Then we define

$$h(x_0) := (b_1 \circ \cdots \circ b_d, x_d).$$

To compute the inverse $x_0 = h^{-1}(b_1 \circ \cdots \circ b_d, x_d)$, we can simply compute $x_{i-1} = f_i(b_i, x_i)$ for each $i = d, d-1, \ldots, 1$, and eventually find x_0. This also shows that h is a bijection.

Remark 5.6. *The above construction of the bijection $h \colon [n] \to [m] \times [n/m]$ can be naturally viewed as a depth-d tree structure, similar to [CRSW13]. The root node (at level 0) represents an array containing all elements of $[n]$ in ascending order. For $0 \leq i \leq d$, the i-th level of the tree has $2^{\ell_1+\ell_2+\cdots+\ell_i} = n/n_i$ nodes, each representing a length-n_i array. A node B at level $(i-1)$ has 2^{ℓ_i} children $B_1, \ldots, B_{2^{\ell_i}}$, which form a partition of the elements in array B: the u-th element of array B_b equals the $f_i(b, u)$-th element of array B. At level d, there are m leaf nodes $S_1, \ldots S_m$, forming a partition of $[n]$, where the u-th element of array S_b is $h^{-1}(b, u)$.*

By reducing the depth d to a constant in the construction, we can achieve $O(\log n)$ seed length, but with a worse load-balancing parameter $k(n) = n^{\varepsilon}$, for any $\varepsilon > 0$.

Corollary 5.7. *For any constants $c \geq 1$ and $\varepsilon > 0$, there is an efficiently invertible hash family with parameter $k(n) = O(n^{\varepsilon})$, seed length $s(n) = O(\log n)$, and failure probability $1/n^c$.*

The proof sketch is deferred to Appendix A.

6 Time-Space Tradeoffs for Subset Sum

In this section we present an algorithm achieving a time-space tradeoff for Subset Sum. Our algorithm uses several algebraic and number-theoretic ideas in order to trade more space for a faster running time.

Reminder of Theorem 1.3 *For any parameter $1 \leq k \leq \min\{n, t\}$, there is a randomized algorithm for Subset Sum with 0.01 one-sided error probability, running in $\widetilde{O}((n+t) \cdot k)$ time and $O((t/k) \operatorname{poly}\log(nt))$ space.*

Our algorithm also uses Bringmann's framework combined with Kane's number-theoretic technique, as described in Section 4. Recall that in the algorithm of Section 4, assuming we use the hash family from Theorem 4.3(1) with parameter $O(\log n)$, we evaluated a particular generating function (a polynomial of

degree at most $t \operatorname{poly}\log n$, by Lemma 4.6) at all points $b \in \mathbb{F}_q^*$, where $q = t \operatorname{poly}\log n$ was a randomly chosen prime power (as described in Lemma 3.3).

Here, our new idea is to perform the evaluation in S batches, where each batch has $(q-1)/S$ points to evaluate. In one batch, letting the evaluation points be $b_1, b_2, \ldots, b_{(q-1)/S}$, we define a polynomial

$$B(x) := (x - b_1)(x - b_2) \cdots (x - b_{(q-1)/S})$$

of degree $(q-1)/S$. Then we run the algorithm of Section 4, with the following key modifications (the first were used in Bringmann's $\widetilde{O}(n+t)$-time $\widetilde{O}(t)$-space algorithm [Bri17], while the second one is new and crucial to the space improvement).

(1) Instead of plugging in a specific value for x, we treat x as a formal variable, and the intermediate results during computation are all expanded as a polynomial in x. We use FFT for polynomial multiplication.

(2) All polynomials are computed modulo $B(x)$. Since $B(x)$ has degree $(q-1)/S$, the polynomial operations now only take $O((q/S)\log q)$ space.

Finally we obtain the generating function modulo $B(x)$, which agrees with the original polynomial on the evaluation points b_i. We can evaluate at these points using Theorem 2.4 in $(q/S)\operatorname{poly}\log q \le (t/S)\operatorname{poly}\log(nt)$ time and space (recall that $q = t \operatorname{poly}\log n$).

In order to run the above algorithm efficiently in low space, we have to make some adjustments to the lower-level implementation, which we elaborate in the following.

- In `PartitionLevel1`, we need to compute the product of $\ell = 2^i$ many polynomials of degree $\min\{d, (q-1)/S\}$ modulo $B(x)$, where $d = O((\log^2 n)\cdot 2t/\ell)$ (see Lemma 4.4). When $d < (q-1)/S$ is small, it might be slow to multiply them one by one. Instead, we divide them into groups, each having $\Theta(q/(Sd))$ polynomials. The polynomials in one group have total degree $O(q/S)$, and their product can be computed in $(q/S)\operatorname{poly}\log q$ time and space, by multiplying them in a natural binary-tree structure. There are $O\big(\ell/(q/(Sd))\big) \le S \operatorname{poly}\log n$ groups. We multiply the product of each group one by one modulo $B(x)$, in total time

  $$(S \operatorname{poly}\log n) \cdot ((q/S)\operatorname{poly}\log q) \le q \operatorname{poly}\log(nq),$$

  and $(q/S)\operatorname{poly}\log q$ space.

  When $d \ge (q-1)/S$, we can simply multiply them one by one, in $q\operatorname{poly}\log(nq)$ total time and $(q/S)\operatorname{poly}\log q$ space.

- In `PartitionLevel2`, we need to compute the polynomial $(1 + \sum_{a \in T_i} x^a)$ modulo $B(x)$. When $a \gg (q-1)/S$, this is not easy to compute efficiently in $(q/S)\operatorname{poly}\log q$ space. To resolve this issue, we will carefully pick the evaluation points $b_1, \ldots, b_{(q-1)/S}$ so that $B(x)$ *only has two terms*, i.e., it has the form $B(x) = x^{(q-1)/S} - h$. Then, $x^a \bmod B(x)$ is always a monomial, which can be easily computed in $\operatorname{poly}\log(aq)$ time.

Hence, one batch of evaluation can be performed in $\widetilde{O}(n+t)$ time and $O((t/S)\operatorname{poly}\log(nt))$ space. The total time complexity is $\widetilde{O}((n+t)S)$.

Now we show how to pick the evaluation points $b_1, \ldots, b_{(q-1)/S}$ so that $B(x)$ always has only two terms. We assume S is a divisor of $q-1$. We use the following algebraic lemma:

Lemma 6.1. *Let S divide $q-1$. The set $\mathbb{F}_q^*$ can be partitioned disjointly into S sets $P_0, \ldots, P_{S-1}$ such that for all $j = 0, \ldots, S-1$ there is a polynomial $B_j(x)$ of two terms that vanishes only on P_j.*

Proof. Take a generator g of $\mathbb{F}_q^*$.[9] For $j = 0, \ldots, S-1$, define the set of points

$$P_j := \{g^{aS+j} \mid a = 0, \ldots, (q-1)/S - 1\},$$

and the polynomial

$$B_j(x) := x^{(q-1)/S} - g^{j(q-1)/S}.$$

Note that $|P_j| = (q-1)/S$ and $P_0 \dot\cup P_1 \dot\cup \cdots \dot\cup P_{S-1}$ is a partition of $\mathbb{F}_q^*$. Now we want to show that, for all $0 \le j \le S-1$, $B_j(x) = \prod_{b \in P_j}(x-b)$.

For every $b = g^{aS} \in P_0$, $B_0(b) = g^{a(q-1)} - 1 = 0$. Since $|P_0| = (q-1)/S$, these are all the roots of $B_0(x)$, and we have $B_0 = \prod_{b \in P_0}(x-b)$. So the claim holds for $j = 0$.

For $j \neq 0$, note that

$$\begin{aligned}
\prod_{b \in P_j}(x-b) &= \prod_a (x - g^{aS+j}) \\
&= g^{j(q-1)/S} \prod_a (x/g^j - g^{aS}) \\
&= g^{j(q-1)/S} B_0(x/g^j) \\
&= g^{j(q-1)/S}((x/g^j)^{(q-1)/S} - 1) \\
&= B_j(x).
\end{aligned}$$

□

Hence, when S divides $q-1$, we can simply use $P_0, P_1, \ldots, P_{S-1}$ as the batches.

The following lemma ensures that, for any given parameter $1 \le k \le \min\{n, t\}$, we can find a prime power $q = \widetilde{\Theta}(t)$ such that $q-1$ has a divisor $S = \widetilde{\Theta}(k)$.

Lemma 6.2. *For all sufficiently large R and $4 \le K \le R/4$, there are at least $\Omega(R/\log^2 R)$ prime powers $q \in (R/2, R]$ such that $q-1$ has an integer divisor in the interval $[K/2, 2K \cdot \log^{15} K]$.*

We provide a proof of this lemma in Appendix B. It relies on the Bombieri-Vinogradov Theorem [Bom65, Vin65] from analytic number theory.

To finish the proof of Theorem 1.3, we apply Corollary 3.2 in a similar way as we did in proving the Coefficient Test Lemma. By Lemma 4.6, there is some $w = \widetilde{\Theta}(t \cdot \operatorname{poly}\log n)$ such that the generating function has degree at most w and integer coefficients of magnitude at most 2^w. For a given parameter $1 \le k \le \min\{n, t\}$, we apply the Lemma 6.2 with some $R = \widetilde{\Theta}(w)$ so that there are at least $100w$ prime powers q in the interval $[w+2, R]$, such that $q-1$ has a divisor $S \in [K/2, \widetilde{O}(K)]$, where $K = \widetilde{\Theta}(k \operatorname{poly}\log(nt))$. To find these prime powers q, we iterate over the interval $[w+2, R]$ and use AKS primality test, and then iterate over the interval $[K/2, \widetilde{O}(K)]$ to find a divisor S of $q-1$, using $\widetilde{O}(RK) \le kt \operatorname{poly}\log(nt)$ time and $\operatorname{poly}\log(nt)$ space. Then, we pick a random q from them, and run the evaluation algorithm described above in $\widetilde{O}((n+t)S)$ total time and $(t/S)\operatorname{poly}\log(nt)$ space.

7 Deterministic Algorithm

In this section, we present a faster low-space deterministic algorithm for Subset Sum.

Reminder of Theorem 1.2 *Subset Sum can be solved deterministically in $\widetilde{O}(n^2 t)$ time and $O(\log t \cdot \log^3 n)$ space.*

[9] To check whether some $h \in \mathbb{F}_q^*$ is a generator, we can simply enumerate all factors r of $q-1$ and check if $h^r = 1$. This takes at most $O(\sqrt{q}\operatorname{poly}\log(q))$ time. There are $\phi(q-1) \ge \Omega(q^{0.99})$ many generators in $\mathbb{F}_q^*$ (see e.g., [HW75, Theorem 327]), so we can find a generator with a (Las Vegas) randomized algorithm in $o(q)$ time.

Consider a subset sum instance $A = (a_1, a_2, \ldots, a_n)$ with target sum t. In Section 4, we modified Kane's algorithm so that it works modulo a prime $p \leq \widetilde{O}(t)$, rather than needing $p \geq \Omega(nt)$. We achieved that by first randomly splitting A into various sets L_i (as Bringmann does), whereby in each L_i we could estimate a nice upper bound on the number of elements that may contribute towards a subset sum of value t. To ensure that we only had the contribution of these sums and no larger sums, we used color-coding (and efficient hash functions to keep the space and the randomness low). As mentioned in the introduction, this approach gives fast low-space randomized algorithms, but it seems very difficult to derandomize color-coding quickly.

Here, to obtain a good deterministic algorithm, we give an alternative approach. As before, we split A into L_i lists. From there, we try to deterministically approximate the *number of elements*, by only keeping track of the approximate logarithm of the number of elements, in a similar spirit to Morris's algorithm [Mor78] for small-space approximate counting. We start by defining a special polynomial product operation that will help us approximately count.

Definition 7.1 (Product for Approximate Counting)**.** Let $\varepsilon \in (0, 1)$. Let $q_1(y) = 1 + \sum_{i=1}^{d_1} v_i y^i$ and $q_2(y) = 1 + \sum_{j=1}^{d_2} w_j y^j$ be two polynomials with coefficients v_i, w_i from a ring R. Define

$$q_1(y) \star q_2(y) := 1 + \sum_{i=1}^{d_1} v_i y^i + \sum_{j=1}^{d_2} w_j y^j + \sum_{1 \leq i \leq d_1, 1 \leq j \leq d_2} v_i w_j y^{u(i,j)},$$

where $u(i, j)$ is the integer such that

$$(1+\varepsilon)^{u(i,j)} \geq (1+\varepsilon)^i + (1+\varepsilon)^j > (1+\varepsilon)^{u(i,j)-1}.$$

Note the operation $\star$ implicitly depends on the ε chosen. Intuitively, the $\star$ operation uses the exponents of polynomials to approximately count. The exponent represents $\log_{1+\varepsilon}(\text{count})$ approximately. If we had $(1+\varepsilon)^{u(i,j)} = (1+\varepsilon)^i + (1+\varepsilon)^j$, where $i = \log_{1+\varepsilon}(\text{count}_1)$ and $j = \log_{1+\varepsilon}(\text{count}_2)$, then we would in fact have $(1+\varepsilon)^{u(i,j)} = \text{count}_1 + \text{count}_2$, i.e., our counting would be exact. As we only have $(1+\varepsilon)^{u(i,j)} \geq (1+\varepsilon)^i + (1+\varepsilon)^j > (1+\varepsilon)^{u(i,j)-1}$, we potentially lose a multiplicative factor of $(1+\varepsilon)$ every time we apply the $\star$ operation. Note that the approximation factor improves, as we decrease ε.

Now we give the pseudocode of our deterministic algorithm.

Function `Evaluate2`(x, A, t)
1: Let $L_i := A \cap (t/2^i, t/2^{i-1}]$ for all $i = 1, 2, \ldots, \lceil \log n \rceil - 1$.
2: Let $L_{\lceil \log n \rceil} := A \backslash (L_1 \cup \cdots \cup L_{\lceil \log n \rceil - 1})$.
3: Set $u := 1$.
4: **for** $i = 1, \ldots, \lceil \log(n) \rceil$ **do**
5: $\quad u := u \cdot$ `ApproxCount`$(x, L_i, 2^i)$.
6: **return** u

Function `ApproxCount`(x, L, z)
1: Let $L = \{b_0, b_1, \ldots, b_{m-1}\}$.
2: Let $F(i, i+1, x) := 1 + yx^{b_i}$ where y is a formal variable.
3: Let $\varepsilon = 1/\log_2(m)$, and define $F(i, j, x)$ (a univariate polynomial in variable y) recursively:
$\quad F(i, j, x) := F(i, (i+j)/2, x) \star F((i+j)/2, j, x)$.
4: Let $v =$ sum of the coefficients of y^k in $F(0, m, x)$ for $0 \leq k \leq 1 + \log_{1+\varepsilon}(z) + \log m$.
5: **return** v

For fixed x and L, $F(i,j,x)$ is a univariate polynomial in variable y with coefficients depending on x and L.

We will use x^S for $S \subseteq \{0,1,\ldots,m-1\}$ and $L = \{b_0, b_1, \ldots, b_{m-1}\}$ to denote $x^{\sum_{i\in S} b_i}$. We omit L from the notation as it will be clear from the context.

The key to our analysis is the following lemma regarding $F(i,j,x)$, which proves that $F(i,j,x)$ can be used to approximately count. In particular, $F(i,j,x)$ contains terms of the form $y^k x^S$; we will ensure that the exponent k approximates $\log_{1+\varepsilon}|S|$.

Lemma 7.2. *$F(i,j,x)$ has the following properties:*

1. *$F(i,j,x) = 1 + \sum_{k=1}^{p} \sum_{S\in\mathcal{S}_{i,j,k}} y^k x^S$ where $p \le 1 + \log_{1+\varepsilon}(m) + \log_2(m)$, and for all $S \in \mathcal{S}_{i,j,k}$ we have $\emptyset \neq S \subseteq [i,j)$.*
2. *For every S, i, j such that $\emptyset \neq S \subseteq [i,j)$ there exists a unique integer k such that $S \in \mathcal{S}_{i,j,k}$. Furthermore, $\log_{1+\varepsilon}(|S|) \le k \le 1 + \log_{1+\varepsilon}(|S|) + \log_2(j-i)$.*

Proof. We prove all properties simultaneously by induction on $j-i$. For the base case of $j = i+1$, observe that

1. For $F(i,i+1,x) := 1 + yx^{b_i}$ we have $1 = p \le 1 + \log_{1+\varepsilon}(m) + \log_2(m)$ and $\mathcal{S}_{i,i+1,1} = \{\{i\}\}$.
2. Only the set $S = \{i\}$ satisfies $\emptyset \neq S \subseteq [i,i+1)$, and $\{i\} \in \mathcal{S}_{i,i+1,1} = \{\{i\}\}$, where $0 = \log_{1+\varepsilon}(|S|) \le k = 1 \le 1 + \log_{1+\varepsilon}(|S|) + \log_2(j-i) = 1$.

Let us now move to proving the induction hypothesis. Recall $F(i,j,x)$ is defined to be $F(i,j',x) \star F(j',j,x)$ for $j' = (i+j)/2$. By induction we have that $F(i,j',x) = 1 + \sum_{k_1=1}^{p} \sum_{S_1\in\mathcal{S}_{i,j',k_1}} y^{k_1} x^{S_1}$ and $F(j',j,x) = 1 + \sum_{k_2=1}^{p} \sum_{S_2\in\mathcal{S}_{j',j,k_2}} y^{k_2} x^{S_2}$. Hence we have

$$
\begin{aligned}
&F(i,j,x)\\
&= F(i,j',x) \star F(j',j,x)\\
&= \left(1 + \sum_{k_1=1}^{p} \sum_{S_1\in\mathcal{S}_{i,j',k_1}} y^{k_1} x^{S_1}\right) \star \left(1 + \sum_{k_2=1}^{p} \sum_{S_2\in\mathcal{S}_{j',j,k_2}} y^{k_2} x^{S_2}\right)\\
&= 1 + \sum_{k_1=1}^{p} \sum_{S_1\in\mathcal{S}_{i,j',k_1}} y^{k_1} x^{S_1} + \sum_{k_2=1}^{p} \sum_{S_2\in\mathcal{S}_{j',j,k_2}} y^{k_2} x^{S_2}\\
&\quad + \sum_{k_1=1}^{p}\sum_{k_2=1}^{p} \sum_{S_1\in\mathcal{S}_{i,j',k_1}} \sum_{S_2\in\mathcal{S}_{j',j,k_2}} y^{u(k_1,k_2)} x^{S_1\cup S_2},
\end{aligned}
$$

where the last equality follows because $S_1 \cap S_2 = \emptyset$ (as $S_1 \subseteq [i,j')$ and $S_2 \subseteq [j',j)$).

Consider a nonempty subset $S \subseteq [i,j)$ and let $S_1 = S \cap [i,j')$ and $S_2 = S \cap [j',j)$. We will prove the existence of a unique monomial $y^k x^S$ which occurs with coefficient 1 and $\log_{1+\varepsilon}(|S|) \le k \le 1 + \log_{1+\varepsilon}(|S|) + \log_2(j-i)$ is satisfied.

Our analysis has three separate cases:

Case 1. $S_1 = \emptyset$ and $S = S_2 \neq \emptyset$. By induction (Property 2) there exists a unique k_2 such that $y^{k_2}x^{S_2} = y^{k_2}x^S$ is a monomial in $F(j', j, x)$. Furthermore by induction (Property 1) this monomial occurs with coefficient 1. By Definition 7.1 this monomial gets carried over to $F(i, j, x)$ with coefficient 1. As for all k' every set in $\mathcal{S}_{i,j',k'}$ is non-empty (Property 1) there is no other occurrence of the monomial $y^{k_2}x^S$ in $F(i, j, x)$. Hence we have $k = k_2$.

By induction (Property 2), we have $\log_{1+\varepsilon}(|S_2|) \leq k = k_2 \leq 1 + \log_{1+\varepsilon}(|S_2|) + \log_2((j-i)/2) < 1 + \log_{1+\varepsilon}(|S|) + \log_2(j-i)$.

Case 2. $S_2 = \emptyset$ and $S = S_1 \neq \emptyset$. This is entirely symmetric to the case above.

Case 3. $S_1, S_2 \neq \emptyset$. By induction there exist unique k_1, k_2 such that $y^{k_1}x^{S_1}$ is a monomial in $F(i, j', x)$ and $y^{k_2}x^{S_2}$ is a monomial in $F(j', j, x)$. Furthermore by induction both of these monomials have coefficient 1. By Definition 7.1 the only term containing x^S will be $y^{u(k_1,k_2)}x^S = y^k x^S$ occurring with coefficient 1. Hence we have $k = u(k_1, k_2)$.

By induction (Property 2), $\log_{1+\varepsilon}(|S_1|) \leq k_1$ and $\log_{1+\varepsilon}(|S_2|) \leq k_2$, which imply that

$$\begin{aligned}(1+\varepsilon)^{u(k_1,k_2)} &\geq (1+\varepsilon)^{k_1} + (1+\varepsilon)^{k_2} \\ &\geq |S_1| + |S_2| \\ &= |S|,\end{aligned}$$

or equivalently, $k = u(k_1, k_2) \geq \log_{1+\varepsilon}(|S|)$.

By induction (Property 2) we have that

$$\begin{aligned}k_1 &\leq 1 + \log_{1+\varepsilon}(|S_1|) + \log_2((j-i)/2) \\ &= \log_{1+\varepsilon}(|S_1|) + \log_2(j-i).\end{aligned}$$

Similarly we have $k_2 \leq \log_{1+\varepsilon}(|S_2|) + \log_2(j-i)$.

Hence we have

$$\begin{aligned}&(1+\varepsilon)^{k_1} + (1+\varepsilon)^{k_2} \\ &\leq |S_1|(1+\varepsilon)^{\log_2(j-i)} + |S_2|(1+\varepsilon)^{\log_2(j-i)} \\ &= |S|(1+\varepsilon)^{\log_2(j-i)} \\ &= (1+\varepsilon)^{\log_{1+\varepsilon}(|S|)+\log_2(j-i)}.\end{aligned}$$

Hence by Definition 7.1,

$$\begin{aligned}k = u(k_1, k_2) &< 1 + \log_{1+\varepsilon}((1+\varepsilon)^{k_1} + (1+\varepsilon)^{k_2}) \\ &\leq 1 + \log_{1+\varepsilon}(|S|) + \log_2(j-i),\end{aligned}$$

which completes the proof by induction. Finally, since $|S| \leq m$ and $j - i \leq m$, we have that $p \leq 1 + \log_{1+\varepsilon}(m) + \log_2(m)$. □

Using Lemma 7.2, we can infer useful properties of the polynomial returned by `ApproxCount`:

Lemma 7.3. `ApproxCount`(x, L, z) *where* $|L| = m$ *returns* $\sum_{S\in\mathcal{S}} x^S$ *such that*

1. *for every set* $S \subseteq L$ *with* $|S| \leq z$, $S \in \mathcal{S}$.
2. *for all* $S \in \mathcal{S}$, $|S| \leq O(z)$.

Proof. Let us first prove Property 1. By Lemma 7.2, for every set $S \subseteq L$ with $|S| \leq z$ there is a unique k such that $x^S y^k$ is a monomial in $F(0, m, x)$ and that for this unique k, $x^S y^k$ occurs with coefficient 1. Furthermore, $k \leq 1 + \log_{1+\varepsilon}(|S|) + \log m \leq 1 + \log_{1+\varepsilon}(z) + \log m$. Since we are adding the coefficients of y^k for all such k, we get that the monomial x^S occurs in the polynomial returned by $\texttt{ApproxCount}(x, L, z)$ with coefficient 1.

By Lemma 7.2, if $x^S y^k$ is a monomial in $F(0, m, x)$, then $\log_{1+\varepsilon}(|S|) \leq k$ or equivalently $|S| \leq (1+\varepsilon)^k$. Since we are restricting $k \leq 1 + \log_{1+\varepsilon}(z) + \log m$, it follows that $|S| \leq z(1+\varepsilon)^{1+\log m} \leq z m^{O(\varepsilon)} \leq O(z)$, where the last inequality follows from $\varepsilon = 1/\log_2(m)$. □

We now use Lemma 7.3 to argue about the polynomial returned by `Evaluate2`.

Lemma 7.4. $\texttt{Evaluate2}(x, A, t)$ *returns* $\sum_{S \in \mathcal{S}} x^S$ *such that*

1. *for every set* $S \subseteq [n]$ *with* $\sum_{i \in S} a_i \leq t$, $S \in \mathcal{S}$.
2. *for all* $S \in \mathcal{S}$, $\sum_{i \in S} a_i \leq O(t \log n)$.

Proof. Consider a set $S \subseteq [n]$ with $\sum_{i \in S} a_i \leq t$. Let $S_i = S \cap L_i$. By the definition of L_i, $|S_i| \leq 2^i$. By Lemma 7.3, $\texttt{ApproxCount}(x, L_i, 2^i)$ has a term x^{S_i} with coefficient 1. As $\texttt{Evaluate2}(x, A, t) = \prod_i \texttt{ApproxCount}(x, L_i, 2^i)$ hence $\texttt{Evaluate2}(x, A, t)$ will have the term $\prod_i x^{S_i} = x^S$ with coefficient 1. This proves property 1.

By Lemma 7.3, $\texttt{ApproxCount}(x, L_i, 2^i)$ only has monomials of the form x^{S_i} where $S_i \subseteq L_i$ such that

$$\begin{aligned}\sum_{j \in S_i} a_j &\leq (\max_{j \in S_i} a_j) \cdot |S_i| \\ &\leq (t/2^{i-1}) \cdot O(2^i) \\ &\leq O(t).\end{aligned}$$

Any monomial in $\texttt{Evaluate2}(x, L, z)$ will have the form $\prod_i x^{S_i} = x^S$ where x^{S_i} is a monomial in $\texttt{ApproxCount}(x, L_i, 2^i)$ and $S = \cup_i S_i$. Property 2 follows from

$$\begin{aligned}\sum_{j \in S} a_j &= \sum_{i=1}^{\lceil \log n \rceil} \sum_{j \in S_i} a_j \\ &\leq \sum_{i=1}^{\lceil \log n \rceil} O(t) \\ &\leq O(t \log n).\end{aligned}$$

□

Corollary 7.5. *Let the output of* $\texttt{Evaluate2}(x, A, t)$ *is a polynomial* $P(x)$ *where* $A = [a_1, a_2, \ldots, a_n]$. *Then:*

1. $P(x)$ *is a polynomial of degree at most* $d = O(t \log n)$, *with non-negative coefficients that are bounded above by* $2^{\min\{n, d \log n\}}$.
2. $P(x)$ *contains the monomial* x^t *iff there exists a* $R \subseteq [n]$ *be such that* $t = \sum_{i \in R} a_i$.

Proof. Let us first prove Property 1. By Lemma 7.4 the degree is bounded by $\max_{S \in \mathcal{S}} \sum_{i \in S} a_i \leq O(t \log n)$. By Lemma 7.4, $\texttt{Evaluate2}(x, A, t)$ is a sum of monomials of the form x^S where $S \subseteq [n]$ and the coefficient of x^S is either 0 or 1. Hence for any $a \in \mathbb{Z}_{\geq 0}$ the coefficient of x^a is non-negative and bounded above by the number of subsets with sum equal to a which is always at most $\binom{n}{a} \leq \min\{2^n, n^a\}$ as all $a_i \in \mathbb{Z}_{>0}$. Using $a \leq d$ the bound on the coefficients follows.

As all monomials are of the form x^S for $S \subseteq [n]$ we can only have the monomial x^t if there exists a set $R \subseteq [n]$ be such that $t = \sum_{i \in R} a_i$. Conversely, if there exists a set $R \subseteq [n]$ be such that $t = \sum_{a \in R} a$ then $P(x)$ contains the monomial x^t by Property 2 of Lemma 7.4. □

7.1 Implementation

Now we describe in more detail how to implement the procedures `Evaluate2` and `ApproxCount` with low time and space.

Lemma 7.6. *The procedure* `Evaluate2` *(where arithmetic operations are over $\mathbb{F}_q$ with $q = \Omega(t)$) can be implemented in $O(n \cdot \operatorname{poly}\log(qn))$ time and $O(\log q \cdot \log^3 n)$ working space.*

Proof. Recall $A = \{a_1, \ldots, a_n\}$ is the set of input integers. In $\texttt{Evaluate2}(x, A, t)$, we do not have enough space to collect all elements of L_i and pass them to `ApproxCount`. Instead, we will pass the list A (i.e., our input). To correct this, in $\texttt{ApproxCount}(x, L, z)$ when we encounter an $a_i \notin L$ we just ignore it by setting $F(i, i+1, x) = 1$.

By Property 1 of Lemma 7.2, any polynomial of the form $F(i, j, x)$ computed in $\texttt{ApproxCount}(x, L, z)$ has degree at most $1 + \log_{1+\varepsilon}(m) + \log_2(m)$, where $\varepsilon = 1/\log_2(m)$. Thus, the space needed to store a single polynomial is

$$O(\log q \cdot (1 + \log_{1+\varepsilon}(m) + \log_2(m))) \leq O(\log q \cdot \log^2 m).$$

To compute $F(0, m, x)$ in $\texttt{ApproxCount}(x, L, z)$, we perform the recursion in a depth-first way, with recursion depth at most $\log n$. At any point in time, we will have only stored (at most) one polynomial at every level of the recursion tree, so the total space usage of $\texttt{ApproxCount}(x, L, z)$ is $O(\log q \cdot \log^3 n)$.

In $\texttt{ApproxCount}(x, L, z)$, we compute the approximate counting product $O(m)$ times, where each multiplication takes $\operatorname{poly}(\log q, \log m, \varepsilon^{-1})$ time. So the running time of $\texttt{ApproxCount}(x, L, z)$ is $O(n \cdot \operatorname{poly}\log(qn))$. It follows that the total running time and space requirements of $\texttt{Evaluate2}(x, A, t)$ are $O(n \cdot \operatorname{poly}\log(qn))$ and $O(\log q \cdot \log^3 n)$, respectively. □

Finally, we can complete the proof of Theorem 1.2 and give the final deterministic algorithm.

Proof of Theorem 1.2. By Lemma 3.3, we have a deterministic algorithm for checking if the monomial x^t has a nonzero coefficient in $P(x) = \texttt{Evaluate2}(x, A, t)$, in time $\widetilde{O}((d+w)(T+w)w)$, where:

- d denotes the degree of $P(x)$ and $d \leq O(t \log n)$ by Corollary 7.5,
- 2^w denotes the largest coefficient of $P(x)$ and $w \leq \min\{n, d\log(n)\}$ by Corollary 7.5, and
- T denotes the time to calculate $P(x)$ for a given $x \in \mathbb{F}_q$ for $q \leq O(d+w) = \widetilde{O}(t \cdot \operatorname{poly}\log n)$.

By Lemma 7.6, we have $T \leq O(n \cdot \operatorname{poly}\log(qn)) \leq \widetilde{O}(n \cdot \operatorname{poly}\log(nt))$. Plugging in the upper bounds for d, T, w, the total running time is

$$\begin{aligned}\widetilde{O}((d+w)(T+w)w) &\leq \widetilde{O}((d\log n) \cdot (n \operatorname{poly}\log(nt)) \cdot n) \\ &\leq \widetilde{O}(n^2 t).\end{aligned}$$

By Lemma 3.3 the space of the deterministic algorithm is $O(S + \log(d+w))$ where S is the space required to calculate $P(x)$ for a given $x \in \mathbb{F}_q$ for $q \leq O(d+w) \leq \widetilde{O}(t \operatorname{poly}\log n)$. By Lemma 7.6, $S = O(\log q \cdot \log^3 n)$. Hence, assuming $t \geq \log(n)$, we have $\log q = O(\log t)$ and the overall space complexity is $O(\log t \cdot \log^3 n)$. In the case of $t < \log n$, we simply use the deterministic $O(nt)$-time $O(t + \log n)$-space dynamic programming algorithm instead. □

8 Approximation Algorithms

In this section, we present a fast low-space randomized algorithm for the following **Weak Subset Sum Approximation Problem** (a.k.a. WSSAP):

Definition 8.1 (WSSAP)**.** Given a list of positive integers $A = [a_1, a_2, \ldots, a_n]$, target t, $0 < \varepsilon < 1$ with the promise that they fall into one of the following two cases:

- **YES:** There exists a subset $S \subseteq [n]$ such that $(1-\varepsilon/2)t \leq \sum_{i\in S} a_i \leq t$.
- **NO:** For all subsets $S \subseteq [n]$ either $\sum_{i\in S} a_i > (1+\varepsilon)t$ or $\sum_{i\in S} a_i < (1-\varepsilon)t$.

decide whether it is a YES instance or a NO instance.

The search version of the above definition was introduced by Mucha, Węgrzycki, and Włodarczyk [MWW19] as a "weak" notion of approximation. They gave a $\widetilde{O}(n + 1/\varepsilon^{5/3})$ time and space algorithm.

Note the usual decision notion of "approximate subset sum" distinguishes between the two cases of
(1) there is an $S \subseteq [n]$ such that $\sum_{i\in S} a_i \in [(1-\varepsilon/2)t, t]$, and
(2) for all $S \subseteq [n]$, $\sum_{i\in S} a_i < (1-\varepsilon)t$ or $\sum_{i\in S} a_i > t$.
This is, in principle, a harder problem.

Reminder of Theorem 1.5 *There is a $\widetilde{O}(\min\{n^2/\varepsilon, n/\varepsilon^2\})$-time and $O(\operatorname{poly}\log(n,t))$-space algorithm for WSSAP.*

Proof. Our algorithm runs two different algorithms, and takes the output of the one that stops first. Algorithm 1 will use $\widetilde{O}(n^2/\varepsilon)$ time and $\operatorname{poly}\log(nt)$ space; Algorithm 2 will use $\widetilde{O}(n/\varepsilon^2)$ and $\operatorname{poly}\log(nt)$ space.

Algorithm 1: Define $b_i = \lfloor \frac{a_i}{N} \rfloor$ where $N = \varepsilon t/(2n)$. First we will prove that (A,t) is a YES instance if and only if there is a subset $S \subseteq [n]$ such that $N\sum_{i\in S} b_i \in [t-\varepsilon t, t]$.

Assume (A,t) is a YES instance. We have $a_i - N \leq Nb_i \leq a_i$; hence for set $S \subseteq [n]$ such that $\sum_{i\in S} a_i \in [t(1-\varepsilon/2), t]$, we have $N\sum_{i\in S} b_i \in \left[\sum_{i\in S} a_i - Nn, \sum_{i\in S} a_i\right] \subseteq [t(1-\varepsilon/2) - Nn, t] = [t-\varepsilon t, t]$.

On the other hand, suppose there is an $S \subseteq [n]$ such that $N\sum_{i\in S} b_i \in [t-\varepsilon t, t]$. Then as $Nb_i \leq a_i \leq Nb_i + N$ we have that $\sum_{i\in S} a_i \in [t-\varepsilon t, t+\varepsilon t/2]$, which implies that the original instance was a YES instance.

Therefore, (A,t) is a YES instance if and only if there is a set S such that $N\sum_{i\in S} b_i \in [t-\varepsilon t, t]$, i.e., $\sum_{i\in S} b_i \in [2n(1-\varepsilon)/\varepsilon, 2n/\varepsilon] = [t'(1-\varepsilon), t']$ for $t' = 2n/\varepsilon$.

So we have reduced the original problem to a subset sum instance on a list $B = [b_1, b_2, \ldots, b_n]$ where we want to know if there is a subset with sum in the range $[t'(1-\varepsilon), t']$. Our algorithm (Theorem 1.1) can also handle this modification, as all we need to change is that we need to detect if there is a monomial of the form x^d for some $d \in [t'(1-\varepsilon), t']$ instead of $d = t'$. This change can be handled in Corollary 3.2 by multiplying with

$$\sum_{i=t'(1-\varepsilon)}^{t'} x^{q-1-i} = x^{q-1-t'}(1-x^{\varepsilon t'+1})/(1-x),$$

instead of $x^{q-1-t'}$. Alternatively, one could also use the reduction described in Remark 1.4. By Lemma 3.3, Algorithm 1 runs in $\widetilde{O}(nt') = \widetilde{O}(n^2/\varepsilon)$ time and $\operatorname{poly}\log(nt)$ space.

Now we describe Algorithm 2.

Algorithm 2: Let $S_{big} = \{i \mid a_i > \varepsilon t\}$, $A_{big} = \{a_i \mid a_i > \varepsilon t\}$ and $S_{small} = \{i \mid a_i \leq \varepsilon t\}$, $A_{small} = \{a_i \mid a_i \leq \varepsilon t\}$. Let $h = \sum_{i\in S_{small}} a_i$.

If there is a subset of A_{big} with sum in $[(1-\varepsilon)t-h,(1+\varepsilon)t]$, then we can add elements from A_{small} to the set, until our sum is in the range $[(1-\varepsilon)t,(1+\varepsilon)t]$. Hence the input must be a YES instance. On the other hand, if some subset of A has a sum in the range $[(1-\varepsilon/2)t,t]$, the restriction of this subset on A_{big} has a sum in the range $[(1-\varepsilon/2)t-h,t]$. Therefore, deciding if (A,t) is a YES-instance is equivalent to deciding if there exists a subset of A_{big} with sum in $[(1-\varepsilon)t-h,(1+\varepsilon)t]$.

As all elements in A_{big} have values greater than εt, the number of elements in a subset of A_{big} with sum in $[(1-\varepsilon)t-h,(1+\varepsilon)t]$ is at most $(1+\varepsilon)t/(\varepsilon t)\le 2/\varepsilon$.

Define $b_i=\lfloor\frac{a_i}{N}\rfloor$ where $N=\varepsilon^2 t/8$.

We claim that (A,t) is a YES instance if and only if there exists a set $S\subseteq S_{big}$ such that $N(\sum_{i\in S}b_i)\in[t(1-\varepsilon)-h,(1+\varepsilon/2)t]$.

We have $a_i-N\le Nb_i\le a_i$. If (A,t) is a YES instance, then there is a subset $S\subseteq S_{big}$ with $\sum_{i\in S}a_i\in[(1-\varepsilon/2)t-h,t]$ which implies that $N\sum_{i\in S}b_i\in\left[\sum_{i\in S}a_i-N|S|,\sum_{i\in S}a_i\right]\subseteq[(1-\varepsilon/2)t-h-(\varepsilon^2t/8)(2/\varepsilon),t]\subseteq[t(1-\varepsilon)-h,t]$.

For the other direction, suppose there exists a set $S\subseteq S_{big}$ such that $N\sum_{i\in S}b_i\in[t(1-\varepsilon)-h,t(1+\varepsilon/2)]$. For all $i\in S_{big}$, $b_i\ge(a_i-N)/N\ge(\varepsilon t-\varepsilon^2t/8)/(\varepsilon^2t/8)\ge 6/\varepsilon$, and hence $|S|\le\big(t(1+\varepsilon/2)/N\big)/(6/\varepsilon)=2(2+\varepsilon)/(3\varepsilon)\le 2/\varepsilon$. Then as $Nb_i\le a_i\le Nb_i+N$ we have that $\sum_{i\in S}a_i\in[t(1-\varepsilon)-h,t(1+\varepsilon/2)+N|S|]\subseteq[t(1-\varepsilon)-h,t(1+\varepsilon)]$, which implies that the original instance (A,t) was a YES instance. This completes the proof of our claim above.

We have now reduced the original problem to a list $B_{big}=\{b_i\mid a_i\ge\varepsilon t\}$ where we want to know if there is a subset with sum in the range $[t'(1-\varepsilon/2)-h/N,t'(1+\varepsilon/2)]$ for $t'=t/N=8/\varepsilon^2$. As in Algorithm 1, we can solve this in time $\widetilde{O}(nt'(1+\varepsilon/2))\le\widetilde{O}(n/\varepsilon^2)$ and $\operatorname{poly}\log(nt)$ space.

Combining Algorithms 1 and 2, we obtain an algorithm running in $\widetilde{O}(\min\{n^2/\varepsilon,n/\varepsilon^2\})$ time and $O(\operatorname{poly}\log(nt))$ space. □

9 Conclusion

In this paper, we have given novel Subset Sum algorithms with about the same running time as Bellman's classic $O(nt)$ time algorithm, but with radically lower space complexity. We have also provided algorithms giving a general time-space tradeoff for Subset Sum. The algorithms apply several interesting number-theoretic and algebraic tricks; we believe these tricks ought to have further applications.

We conclude with some open problems. First, the fastest known pseudopolynomial algorithm for Subset Sum runs in $\widetilde{O}(n+t)$-time [Bri17, JW19]. The fastest known $O(\operatorname{poly}\log(nt))$-space algorithm is given in this work. Is there an algorithm running in $\widetilde{O}(n+t)$ time and $\operatorname{poly}(n,\log t)$ space? Perhaps some kind of conditional lower bound is possible, but this may require a new kind of fine-grained hypothesis.

Second, can the time-space tradeoff in our algorithm (Theorem 1.3) be improved to work all the way to $\operatorname{poly}\log(nt)$ space? Currently it only works down to $\widetilde{O}(t/\min\{n,t\})$ space.

Finally, another interesting open problem is whether our efficiently invertible hash families have further applications. They should be particularly useful for constructing randomized algorithms using low space.

Acknowledgements

Ce Jin would like to thank Jun Su and Ruixiang Zhang for pointing him to the Bombieri-Vinogradov theorem.

References

[ABHS19] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019. doi:10.1137/1.9781611975482.3. 2, 4

[AGHP90] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. In *Proceedings of the 31st Symposium on Foundations of Computer Science (FOCS)*, pages 544–553, 1990. doi:10.1109/FSCS.1990.89575. 13

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, pages 781–793, 2004. doi:10.4007/annals.2004.160.781. 8

[Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. 1

[BGNV18] Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum, k-sum, and related problems. *SIAM Journal on Computing*, 47(5):1755–1777, 2018. doi:10.1137/17M1158203. 5

[BHP01] Roger C. Baker, Glyn Harman, and János Pintz. The difference between consecutive primes, II. *Proceedings of the London Mathematical Society*, 83(3):532–562, 2001. doi:10.1112/plms/83.3.532. 1, 8

[BN19] Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. *CoRR*, abs/1912.12529, 2019. To appear in SODA 2021. URL: http://arxiv.org/abs/1912.12529. 2, 5

[Bom65] E. Bombieri. On the large sieve. *Mathematika*, 12(2):201–225, 1965. doi:10.1112/S0025579300005313. 16, 28

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994. doi:10.1109/SFCS.1994.365687. 26

[Bri17] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017. doi:10.1137/1.9781611974782.69. 1, 2, 3, 4, 8, 9, 15, 23

[BW20] Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. *CoRR*, abs/2010.09096, 2020. To appear in SODA 2021. URL: http://arxiv.org/abs/2010.09096. 5

[CRSW13] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. *SIAM Journal on Computing*, 42(3):1030–1050, 2013. doi:10.1137/120871626. 12, 13, 14, 26, 27

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Scienc (FOCS)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21. 1, 3

[Fid72] Charles M. Fiduccia. Polynomial evaluation via the division algorithm the fast fourier transform revisited. In *Proceedings of the 4th ACM Symposium on Theory of Computing (STOC)*, pages 88–93, 1972. doi:10.1145/800152.804900. 4, 6

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981. doi:10.1016/0022-0000(81)90040-4. 5

[Gil98] David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998. doi:10.1137/S0097539794268765. 6

[GJL+16] Anna Gál, Jing-Tang Jang, Nutan Limaye, Meena Mahajan, and Karteek Sreenivasaiah. Space-efficient approximations for subset sum. *ACM Trans. Comput. Theory*, 8(4):16:1–16:28, 2016. doi:10.1145/2894843. 5

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823. 5

[HW75] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, fourth edition, 1975. 16

[JW19] Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA)*, pages 17:1–17:6, 2019. doi:10.4230/OASIcs.SOSA.2019.17. 4, 23

[Kan10] Daniel M. Kane. Unary subset-sum is in logspace. *CoRR*, abs/1012.1336, 2010. URL: http://arxiv.org/abs/1012.1336. 1, 3, 6, 7, 8, 9

[KMPS03] Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences*, 66(2):349–370, 2003. doi:10.1016/S0022-0000(03)00006-0. 4, 5

[KX19] Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863. 4

[LMS12] Nutan Limaye, Meena Mahajan, and Karteek Sreenivasaiah. The complexity of unary subset sum. In *Proceedings of the 18th International Computing and Combinatorics Conference (COCOON)*, pages 458–469, 2012. doi:10.1007/978-3-642-32241-9_39. 5

[LN10] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 321–330, 2010. doi:10.1145/1806689.1806735. 1, 3

[Mar73] Grigory A. Margulis. Explicit construction of concentrators. *Problems of Information Transmission*, 9(4):325–332, 1973. 5

[Mor78] Robert H. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627. 4, 17

[MRRR14] Raghu Meka, Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Fast pseudorandomness for independence and load balancing. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 859–870, 2014. doi:10.1007/978-3-662-43948-7_71. 13

[MWW19] Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–88, 2019. doi:10.1137/1.9781611975482.5. 2, 4, 22

[NW20] Jesper Nederlof and Karol Węgrzycki. Improving schroeppel and shamir's algorithm for subset sum via orthogonal vectors. *CoRR*, abs/2010.08576, 2020. URL: http://arxiv.org/abs/2010.08576. 5

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. 1

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291. 1

[Sho94] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994. doi:10.1006/jsco.1994.1025. 5

[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. doi:10.1137/0210033. 5

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012. doi:10.1561/0400000010. 11

[Vin65] A. I. Vinogradov. The density hypothesis for Dirichet L-series. *Izv. Akad. Nauk SSSR Ser. Mat.*, 29:903–934, 1965. 16, 28

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge university press, 2013. 6

A Proof of the load-balancing property

We will utilize a tail bound for $2k$-wise δ-dependent random variables.

Lemma A.1 ([CRSW13, Lemma 2.2], [BR94, Lemma 2.2])**.** *Let $X_1, \ldots, X_n \in \{0,1\}$ be $2k$-wise δ-dependent random variables, for some $k \in \mathbb{N}$ and $0 \leq \delta < 1$, and let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then, for any $t > 0$ it holds that*

$$\Pr[|X - \mu| > t] \leq 2\left(\frac{2nk}{t^2}\right)^k + \delta\left(\frac{n}{t}\right)^{2k}.$$

Now we analyze the load-balancing guarantee of this construction as in [CRSW13]. We follow the discussion in Remark 5.6 and view the evaluation of $h(x)$ as tracing the tree path along which x moves from root to leaf. For $i \in [d]$, recall that $g_i \colon [n_i] \to [2^{\ell_i}]$ is sampled from a k_i-wise δ-dependent family, and bijection $f_i \colon [2^{\ell_i}] \times [n_i] \to [n_{i-1}]$ is defined by

$$f_i(b, u) = (b \oplus g_i(u)) \circ u.$$

For a node B at level $(i-1)$, the u-th element of its b-th child equals the $f_i(b,u)$-th element of array B. That is, the s-th element of array B is assigned to its $h_i(s)$-th child, where $h_i \colon [n_{i-1}] \to [2^{\ell_i}]$ is defined by

$$s = b_s \circ u_s,\ h_i(s) := b_s \oplus g_i(u_s).$$

Lemma A.2 (Similar to Lemma 3.2 in [CRSW13])**.** *For any $i \in \{0, 1, \ldots, d-2\}$, $\alpha \geq \Omega(1/\log\log n)$, $0 < \alpha_i < 1$, and set $S_i \subseteq [n_i]$ of size at most $(1+\alpha_i)m_i$,*

$$\Pr_{h_i}\left[\max_{y \in \{0,1\}^{\ell_{i+1}}} |h_{i+1}^{-1}(y) \cap S_i| \leq (1+\alpha)(1+\alpha_i)m_{i+1}\right]$$

is at least $1 - \frac{1}{n^{c+1}}$.

Proof. Fix $y \in \{0,1\}^{\ell_{i+1}}$, let $X = |h_{i+1}^{-1}(y) \cap S_i|$. Without loss of generality, we assume $|S_i| \geq \lfloor (1+\alpha_i)m_i \rfloor$ (otherwise we could add dummy elements).

Each element from S_i can be expressed as $b_s \circ u_s$, where $b_s \in [2^{\ell_{i+1}}]$, $u_s \in [n_{i+1}]$, and $h_{i+1}(b_s \circ u_s) = b_s \oplus g_{i+1}(u_s)$. Group S_i's elements according to u_s. Then each group has at most one element such that $h(b_s \circ u_s) = y$. Assign each group a random variable from $\{0,1\}$, indicating whether it contains an element being hashed to y. Then X equals the sum of these k_{i+1}-wise δ-dependent (since g_{i+1} is) random variables. And $\mathbb{E}[X] = |S_i|/2^{\ell_{i+1}}$. Then by the tail lemma (Lemma A.1) we have

$$\begin{aligned}\Pr[X > (1+\alpha)\mu] &\leq 2\left(\frac{|S_i|k_{i+1}}{(\alpha\mu)^2}\right)^{k_{i+1}/2} + \delta\left(\frac{|S_i|}{\alpha\mu}\right)^{k_{i+1}} \\ &= 2\left(\frac{2^{2\ell_{i+1}}k_{i+1}}{\alpha^2|S_i|}\right)^{k_{i+1}/2} + \delta\left(\frac{2^{\ell_{i+1}}}{\alpha}\right)^{k_{i+1}}.\end{aligned}$$

Since $|S_i| \geq m_i \geq 2^{4\ell_{i+1}}$ and $\alpha = \Omega(1/\log\log n)$, the first summand

$$2\left(\frac{2^{2\ell_{i+1}}k_{i+1}}{\alpha^2|S_i|}\right)^{k_{i+1}/2} \leq 2\left(\frac{k_{i+1}}{\alpha^2 2^{2\ell_{i+1}}}\right)^{k_{i+1}/2} \leq \frac{1}{n^{c+2}}, \tag{1}$$

where the last inequality follows from the choice of k_{i+1} and ℓ_{i+1} such that $k_{i+1}\ell_{i+1} = \Omega(\log n)$. This also enables us to upper bound the second summand, noting that for an appropriate choice of $\delta = \mathrm{poly}(1/n)$ it holds that

$$\delta\left(\frac{2^{\ell_{i+1}}}{\alpha}\right) \leq \frac{1}{2n^{c+2}}. \tag{2}$$

Therefore, by combining Equations (1) and (2), and recalling that $m_{i+1} = m_i/2^{\ell_{i+1}}$ we obtain

$$\begin{aligned}\Pr[X > (1+\alpha)(1+\alpha_i)m_{i+1}] &= \Pr[X > (1+\alpha)(1+\alpha_i)\frac{m_i}{2^{\ell_{i+1}}}] \\ &\leq \Pr[X > (1+\alpha)\mu] \\ &\leq \frac{1}{n^{c+2}}.\end{aligned}$$

The lemma now follows by a union bound over all $y \in \{0,1\}^{\ell_{i+1}}$; note there are at most n such values. □

The rest of the proof follows in almost the same way as in [CRSW13].[10]

Proof of Theorem 5.1. Fix a set $S \subseteq [n]$ of size m, and let $\alpha = \Omega(1/\log\log n)$. We inductively argue that for every level $i \in \{0,1,\ldots,d-1\}$, with probability at least $1 - i/n^{c+1}$ the maximal load in level i is at most $(1+\alpha)^i m_i$ elements per bin. For $i = 0$ this follows from definition. For inductive step, we assume the claim holds for level i. Now we apply Lemma A.2 for each bin in level i with $(1+\alpha_i) = (1+\alpha)^i$. By union bound, with probability at least $1 - (i/n^{c+1} + 1/n^{c+1})$, the maximal load in level $i+1$ is at most $(1+\alpha)^{i+1}m_{i+1}$, which shows the inductive step. In particular, this guarantees that with probability at least $1-(d-1)/n^{c+1}$, the maximal load in level $d-1$ is $(1+\alpha)^{d-1}m_{d-1} \leq 2m_{d-1}$, for some appropriate choice of $d = O(\log\log n)$.

In order to bound the number m_{d-1}, we note that for every $i \in [d-1]$ it holds that $\ell_i \geq (\log m_{i-1})/4 - 1$, so $m_i = m_{i-1}/2^{\ell_i} \leq 2m_{i-1}^{3/4}$. By induction, we have

$$m_i \leq 2^{\sum_{j=0}^{i-1}(3/4)^j} n^{(3/4)^i} \leq 16n^{(3/4)^i}.$$

[10] Since we only need to prove a load-balancing parameter of $O(\log n)$, we omit the last step of the proof from [CRSW13] which aims for proving a stronger $O(\log n/\log\log n)$ bound.

Thus for an appropriate choice of $d = O(\log \log n)$ it holds that $m_{d-1} \leq \log n$. The proof directly follows. □

Proof of Corollary 5.7. (Sketch) Let $\varepsilon > 0$ be given. We simply modify the previous construction by reducing the depth d to a (sufficiently large) constant. Then, as in the proof of Theorem 5.1, for an appropriate choice of $d = O(1)$, we have $m_{d-1} \leq n^{\varepsilon}$. The rest of the proof is the same as in Theorem 5.1. □

B Proof of Lemma 6.2

We use the Bombieri-Vinogradov theorem [Bom65, Vin65] from analytic number theory.

Theorem B.1 (Bombieri-Vinogradov)**.** *Fixing $A > 0$, there is a constant $C > 0$ such that for all $x \geq 2$ and $Q \in [x^{1/2} \log^{-A} x, x^{1/2}]$,*

$$\sum_{q \leq Q} \max_{y \leq x} \max_{\substack{1 \leq a \leq q, \\ \gcd(a,q)=1}} \left| \psi(y; q, a) - \frac{y}{\phi(q)} \right| \leq C x^{1/2} Q \log^5 x.$$

Here $\phi(q)$ is the Euler totient function[11]*, and*

$$\psi(y; q, a) := \sum_{\substack{n \leq y \\ n \equiv a \pmod q}} \Lambda(n),$$

where $\Lambda(n)$ denotes the von Mangoldt function[12]*.*

Lemma B.2. *For sufficiently large R and $4 \leq S \leq R^{1/2} \log^{-7} R$, there exists $\Omega(R/\log^2 R)$ many prime powers $r = p^k \in (R/2, R]$ such that $r - 1$ has an integer divisor in interval $[S/2, S]$.*

Proof. Applying the Bombieri-Vinogradov theorem with $A := 7, x := R, Q := R^{1/2} \log^{-7} R, a := 1$, we have

$$\sum_{q \leq R^{1/2} \log^{-7} R} \max_{y \leq R} \left| \psi(y; q, 1) - \frac{y}{\phi(q)} \right| \leq CR/\log^2 R.$$

As $S \leq R^{1/2} \log^{-7} R$, we can restrict q to the primes in the interval $[S/2, S]$. By the prime number theorem and the Bertrand-Chebyshev theorem, the number of primes in $[S/2, S]$ is at least $C' \cdot S/\log S$ for some positive constant $C' > 0$. Therefore there are at least $\frac{2C'}{3} \cdot S/\log S$ primes $q \in [S/2, S]$ such that

$$\max_{y \leq R} \left| \psi(y; q, 1) - \frac{y}{\phi(q)} \right| \leq \frac{CR/\log^2 R}{\frac{C'}{3} \cdot S/\log S} < \frac{3CR/\log R}{C'S}.$$

Setting $y = R/2$ and $y = R$, we get

$$\begin{aligned} \psi(R; q, 1) - \psi(R/2; q, 1) &= \frac{R/2}{\phi(q)} + \left(\psi(R; q, 1) - \frac{R}{\phi(q)} \right) - \left(\psi(R/2; q, 1) - \frac{R/2}{\phi(q)} \right) \\ &\geq \frac{R/2}{S} - 2 \cdot \frac{3CR/\log R}{C'S} \\ &\geq \frac{R}{4S}, \end{aligned}$$

[11] $\phi(q)$ is the number of integers $1 \leq a \leq q$ that are coprime with q.

[12] $\Lambda(n) = \ln p$ if $n = p^k$ for a prime p and a positive integer k; otherwise $\Lambda(n) = 0$

for sufficiently large R. Hence there are at least $R/(4S)$ many $r \in (R/2, R]$ such that $r \equiv 1 \pmod{q}$ and $\Lambda(r) > 0$, i.e., r is a prime power.

Since the number of choices for prime q is at least $\frac{2C'}{3} \cdot S/\log S$, there exist at least $(\frac{2C'}{3} \cdot S/\log S) \cdot R/(4S) = (C'/6) \cdot R/\log S$ pairs of such (q, r). As each $r - 1$ has at most $\log(r-1)$ prime factors, the the number of distinct r is at least $\Omega(R/(\log S \log R))$. □

Corollary B.3. *For sufficiently large R and $4 \le S \le R/4$, there exists $\Omega(R/\log^2 R)$ many prime powers $r = p^k \in (R/2, R]$ such that $r - 1$ has an integer divisor in interval $[S/2, 2S \cdot \log^{15} S]$.*

Proof. Let $S_0 := R^{1/2} \log^{-7} R$. The case $S \in [4, S_0]$ follows from Lemma B.2. Now we assume $S \in (S_0, R/8]$. Note that $r - 1$ has a divisor in $(S/2, 2S \cdot \log^{15} S]$ iff it has a divisor in interval

$$\left[\frac{r-1}{2S \cdot \log^{15} S}, \frac{r-1}{S/2}\right].$$

For $r \in (R/2, R]$ we have $(r-1)/(S/2) \ge R/S$ and $(r-1)/(2S \cdot \log^{15} S) \le R/(2S \cdot \log^{15} S)$, so it suffices to let $r - 1$ have a divisor in interval

$$I := \left[\frac{R}{2S \log^{15} S}, \frac{R}{S}\right].$$

Note that $R/(2S) \ge 4$, and

$$\frac{R}{2S \log^{15} S} < \frac{R}{2S_0 \log^{15} S_0} < S_0/2$$

for sufficiently large R. So there exists $4 \le S' \le S_0$ such that $[S'/2, S'] \subseteq I$. We then apply Lemma B.2 with S' in place of S. □