Algebraic algorithms for variants of Subset Sum*

Pranjal Dutta[1]** Mahesh Sreekumar Rajasree[2] ***

[1] Chennai Mathematical Institute, India pranjal@cmi.ac.in
[2] Indian Institute of Technology, Kanpur, India mahesr@cse.iitk.ac.in

Abstract. Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the Subset Sum problem (SSUM) is to decide whether there exists $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. Bellman (1957) gave a pseudopolynomial time dynamic programming algorithm which solves the Subset Sum in $O(nt)$ time and $O(t)$ space.
In this work, we present *search* algorithms for variants of the Subset Sum problem. Our algorithms are parameterized by k, which is a given upper bound on the number of realisable sets (i.e. number of solutions, summing exactly t). We show that SSUM with a unique solution is already NP-hard, under randomized reduction. This makes the regime of parametrized algorithms, in terms of k, very interesting.
Subsequently, we present an $\tilde{O}(k \cdot (n+t))$ time deterministic algorithm, which finds the hamming weight of all the realisable sets for a subset sum instance. We also give a poly(knt)-time and $O(\log(knt))$-space deterministic algorithm that finds all the realisable sets for a subset sum instance. Our algorithms use analytic and number-theoretic techniques.

Keywords: subset sum, power series, isolation lemma, hamming weight, interpolation, logspace, Newton's identities

1 Introduction: Variants of Subset Sum

The Subset Sum problem (SSUM) is a well-known NP-complete problem [1, p. 226], where given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the problem is to decide whether there exists $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. In the recent years, provable-secure cryptosystems based on SSUM such as private-key encryption schemes [2], tag-based encryption schemes [3], etc have been proposed. There are numerous improvements made in the algorithms that solve the SSUM problem in both the classical [4,5,6,7,8] and quantum world [9,10,11]. One of the first algorithms was due to Bellman [12] who gave a $O(nt)$ time (*pseudo-polynomial* time) algorithm which requires $\Omega(t)$ space. One can ask for a *search* version of this problem, i.e. to output all the solutions. Since there can be *exponentially* many solutions, it could take $\exp(n)$-time (and space), to output them. This motivates our first problem defined below.

Problem 1 ($k-$ SSSUM). Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the k-solution SSUM ($k-$SSSUM) problem asks to output all $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$ provided with the guarantee that the number of such subsets is at most k.

* The full version is available at this link.
** Supported by Google PhD Fellowship.
*** Supported by Prime Minister's Research Fellowship.

▶ Remark. We denote $1-\mathsf{SSSUM}$ as unique Subset Sum problem (uSSSUM). In stackexchange, a more restricted version was asked where it was assumed that $k = 1$, for *any* realizable t. Here we just want $k = 1$ for some fixed target value t and we do not assume anything for any other value t'.

Now, we consider a different restricted version of the $k - \mathsf{SSSUM}$, where we demand to output only the hamming weights of the k-solutions (we call it $\mathsf{Hamming} - k - \mathsf{SSSUM}$, for definition see Problem 2). By hamming weight of a solution, we mean the number of a_i's in the solution set (which sums up to exactly t). In other words, if $\vec{a} \cdot \vec{v} = t$, where $\vec{a} = (a_1, \ldots, a_n)$ and $\vec{v} \in \{0,1\}^n$, we want $|v|_1$, the ℓ_1-norm of the solution vector.

Problem 2 ($\mathsf{Hamming} - k - \mathsf{SSSUM}$*).* Given an instance of the $k - \mathsf{SSSUM}$, say $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, with the promise that there are at most k-many $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$, $\mathsf{Hamming} - k - \mathsf{SSSUM}$ asks to output all the hamming weights (i.e., $|S|$) of the solutions.

It is obvious that solving $k-\mathsf{SSSUM}$ solves Problem 2. Importantly, the decision problem, namely the HWSSUM is already NP-hard. The HWSSUM problem is : given an instance $(a_1, \ldots, a_n, t, w) \in \mathbb{Z}_{\geq 0}^{n+2}$, decide whether there is a solution to the Subset Sum with hamming weight equal to w. Note that, there is a trivial Cook's reduction from the SSUM to the HWSSUM: SSUM decides 'yes' to the instance $(a_1, \ldots, a_n, t)$ iff at least one of the following HWSSUM instances $(a_1, \ldots, a_n, t, i)$, for $i \in [n]$ decides 'yes'. Therefore, the search-version of HWSSUM, the $\mathsf{Hamming}-k-\mathsf{SSSUM}$ problem, is already an interesting problem and worth investigating.

In this work, we give various deterministic algorithms for Problem 1-2. Our algorithms are algebraic and number theoretic in nature and mainly build upon the previous power series techniques, by Jin and Wu [6] and sparse interpolation [13].

1.1 Main results

In this section, we briefly state our main results. The leitmotif of this paper is to give efficient algorithms for variants of SSUM, with a promise of a bounded number of solutions. Our first theorem gives an efficient pseudo-linear $\tilde{O}(n+t)$ time *deterministic* algorithm for Problem 2, for constant k.

Theorem 1 (Algorithm for hamming weight). *There is a $\tilde{O}(k(n+t))$-time deterministic algorithm for* $\mathsf{Hamming} - k - \mathsf{SSSUM}$.

▶ Remark (Optimality). We emphasize the fact that Theorem 1 is likely to be *near*-optimal for bounded k, due to the following argument. An $O(t^{1-\epsilon})$ time algorithm for $\mathsf{Hamming}-1-\mathsf{SSSUM}$ can be directly used to solve $1-\mathsf{SSSUM}$, as discussed above. By using the *randomized* reduction (Theorem 3), this would give us a randomized $n^{O(1)}t^{1-\epsilon}$-time algorithm for SSUM. But, in [14] the authors showed that SSUM does not have $n^{O(1)}t^{1-\epsilon}$ time algorithm unless the Strong Exponential Time Hypothesis (SETH) is false.

Theorem 2 (Algorithms for finding solutions in low space). *There is a* $\mathsf{poly}(knt)$*-time and* $O(\log(knt))$*-space deterministic algorithm which solves* $k-$ SSSUM*.*

▶ Remark. When considering low space algorithms outputting multiple values, the standard assumption is that the output is written onto a one-way tape which *does not* count into the space complexity; so an algorithm outputting $kn\log n$ bits (like in the above case) could use much less working memory than $kn\log n$; for a reference see McKay and Williams [15].

▶ Comparison with the trivial algorithm. Consider the usual search-to-decision reduction for subset sum: First try to include a_1 in the subset, and if it is feasible then we subtract t by a_1 and add a_1 into the solution, and then continue with a_2, and so on. This procedure finds a single solution, but if we implement it in a recursive way then it can find all the k solutions in $k\cdot n\cdot$ (time complexity for decision version) time; we can think about an n-level binary recursion tree where all the infeasible subtrees are pruned.

Theorem 1 is better than the trivial. Since number of solutions is bounded by k, choosing a prime $p > n+t+k$ suffices in [6], to make the algorithm deterministic. Thus, the time complexity of the decision version is $\tilde{O}((n+t)\log k)$. Hence, from the above, the search complexity is $\tilde{O}(kn(n+t))$ which is *worse* than Theorem 1.

Theorem 2 is better than the trivial. For solving the decision problem in low space, we simply use Kane's $O(\log(nt))$-space $\mathsf{poly}(nt)$-time algorithm [16]. As explained (and improved) in [7], the time complexity is actually $O(n^3t)$ and the extra space usage is $\tilde{O}(n)$ for remembering the recursion stack. Thus the total time complexity is $O(kn^4t)$ and it takes $\tilde{O}(n)+O(\log t)$ space. While Theorem 2 takes $O(\log(knt))$ space and $\mathsf{poly}(knt)$ time. Although our time complexity is worse [3], when $k\le 2^{O((n\log t)^{1-\epsilon})}$, for $\epsilon>0$, our space complexity is *better*.

1.2 Technical overview

All the algorithms presented in this paper consider that the number of solutions is *bounded* by a parameter k. This naturally raises the question whether the SSUM problem is hard, even when the number of solutions is bounded. We will show that this is true even for the case when $k=1$, i.e., uSSSUM is NP-hard under *randomized* reduction.

Theorem 3 (Hardness of uSSSUM). *There exists a randomized reduction which takes a* SSUM *instance* $\mathcal{M}=(a_1,\ldots,a_n,t)\in\mathbb{Z}_{\ge 0}^{n+1}$*, as an input, and produces multiple* SSUM *instances* $\mathcal{SS}_\ell=(b_1,\ldots,b_n,t^{(\ell)})$*, where* $\ell\in[2n^2]$*, such that if*

- $\mathcal{M}$ *is a YES instance of* SSUM $\implies$ $\exists\ell$ *such that* $\mathcal{SS}_\ell$ *is a YES instance of* uSSSUM*;*
- $\mathcal{M}$ *is a NO instance of* SSUM $\implies$ $\forall\ell,\mathcal{SS}_\ell$ *is a NO instance of* uSSSUM*.*

[3] Thm. 2 is *not about* time complexity; as long as it is pseudopolynomial time it's ok.

Proof. The core of the proof is based on the Lemma 1 (Isolation lemma). The reduction is as follows. Let $w_1, \ldots, w_n$ be chosen *uniformly at random* from $[2n]$. We define $b_i = 4n^2 a_i + w_i, \forall i \in [n]$ and the ℓ^{th} SSUM instance as $\mathcal{SS}_\ell = (b_1, \ldots, b_n, t^{(\ell)} = 4n^2 t + \ell)$. Observe that all the new instances are different only in the target values $t^{(\ell)}$.

Suppose $\mathcal{M}$ is a YES instance, i.e., $\exists S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. Then, for $\ell = \sum_{i \in S} w_i$, the $\mathcal{SS}_\ell$ is a YES instance, because

$$\sum_{i \in S} b_i - t^{(\ell)} \;=\; 4n^2 \left(\sum_{i \in S} a_i - t \right) \;-\; \left(\ell - \sum_{i \in S} w_i \right) \;=\; 0 \,.$$

If $\mathcal{M}$ is a NO instance, consider any ℓ and $S \subseteq [n]$. Since $\mathcal{M}$ is a NO instance, $4n^2(\sum_{i \in S} a_i - t)$ is a non-zero multiple of $4n^2$, whereas $|\ell - \sum_{i \in S} w_i| < 4n^2$, which implies that

$$4n^2(\sum_{i \in S} a_i - t) - (\ell - \sum_{i \in S} w_i) \neq 0 \implies \sum_{i \in S} b_i \neq t^{(\ell)} \,.$$

Hence, $\mathcal{SS}_\ell$ is also a NO instance.

We now show that if $\mathcal{M}$ is a YES instance, then one of $\mathcal{SS}_\ell$ is a uSSSUM. Let $\mathcal{F}$ contain all the solutions to the SSUM instance $\mathcal{M}$, i.e. $\mathcal{F} = \{S | S \subseteq [n], \sum_{i \in S} a_i = t\}$. Since w_i's are chosen uniformly at random, Lemma 1 says that there exists a *unique* $S \in \mathcal{F}$, such that $w(S) = \sum_{i \in S} w_i$, is *minimal* with probability at least $1/2$. Let us denote this minimal value $w(S)$ as ℓ^*. Then, $\mathcal{SS}_{\ell^*}$ is uSSSUM because S is the only subset such that $\sum_{i \in S} w_i = \ell^*$. □

Proof idea of Theorem 1. First we sketch the idea for $k = 1$. Suppose, we have a uSSSUM instance such that the hamming weight of the unique solution is w. Choose a prime $q = O(n+k+t)$ and a *primitive root* μ, i.e. $\mathrm{ord}_q(\mu) = q-1$ (for definition, see Definition 2). We can find them efficiently in $\tilde{O}(n + k + t)$ time.

Now, consider the following important polynomial $f(x) = \prod_{i=1}^{n}(1 + \mu \cdot x^{a_i})$. Observe that the coefficient of x^t in f is μ^w. Therefore, by using Lemma 6, we can find μ^w from $f(x)$ and extract w, since $\mathrm{ord}_q(\mu) = q-1 > n \geq w$. This solves Hamming $-$ 1 $-$ SSSUM.

This idea can be extended to general k-SSSUM instance. Observe that, we cannot directly use the above trick, for a single polynomial $f(x)$, since, in this case, the coefficient of x^t is $\sum_{i \leq k} \lambda_i \cdot \mu^{w_i}$, where w_i are the hamming weights of the solution, which occur λ_i times. Eventually, we want to create a polynomial whose roots are of the form μ^{w_i}, so that we can first find the roots μ^{w_i} (over $\mathbb{F}_q$), and from them we can find w_i. To achieve that, we work with k-many polynomials $f_j := \prod_{i=1}^{n}(1 + \mu^j \cdot x^{a_i})$, for $j \in [k]$. Note that the coefficient of x^t in f_j is of the form $\sum_{i \leq k} \lambda_i \cdot \mu^{j w_i}$ (Claim 2). By Newton's Identities (Lemma 3) and Vieta's formulas (Lemma 4), we can now *efficiently* construct a polynomial whose roots are μ^{w_i}. For details, see section 3.

Proof idea of Theorem 2. The above polynomial method *fails* to give a low space algorithm, since Lemma 6 requires $\Omega(t)$ space (eventually it needs to store

all the coefficients mod x^{t+1}). Therefore, our proof idea of Theorem 2 is completely different from that of Theorem 1. Here, we work with a multivariate polynomial $f(x, y_1, \ldots, y_n) = \prod_{i=1}^{n}(1 + y_i x^{a_i})$ over $\mathbb{F}_q$, for a large prime $q = O(nt)$ and its multiple evaluations $f(\alpha, c_1, \ldots, c_n)$, where $(\alpha, c_1, \ldots, c_n) \in \mathbb{F}_q^{n+1}$.

Observe that, the coefficient of x^t in f is a multivariate polynomial $p_t(y_1, \ldots, y_n)$; each of its monomial carries the *necessary information* of a solution, for the instance $(a_1, \ldots, a_n, t)$. More precisely, S is a realisable set of $(a_1, \ldots, a_n, t) \iff \prod_{i \in S} y_i$ is a monomial in p_t. And, the sparsity (number of monomials) of p_t is at most k.

Therefore, it boils down to find the multivariate polynomial p_t. How easy it is to find p_t? Note that we cannot expect to find p_t, just by trivial multiplication as it would take $\tilde{O}(2^n t)$ time! Instead, our algorithm is a *reconstruction* algorithm, which *efficiently* reconstructs p_t, from multiple evaluations points $f(\alpha, c_1, \ldots, c_n)$, for $\alpha \in \mathbb{F}_q^*$. Eventually, we will use sparse interpolation [13] (see Theorem 6), which requires evaluations of the polynomial $p_t(y_1, \ldots, y_n)$ at multiple (polynomially many) points $(c_1, \ldots, c_n) \in \mathbb{F}_q^n$. To find $p_t(c_1, \ldots, c_n)$, we use Kane's identity (Lemma 2) which uses the evaluations $f(\alpha, c_1, \ldots, c_n)$, for $\alpha \in [1, q-1]$. Finding $p_t(c_1, \ldots, c_n)$ can be efficiently done in logspace. The rest (to reconstruct p_t) requires a brief space complexity analysis of [13]. For details, refer to section 4.

1.3 Prior works and their limitations

Before going into the details, we briefly review the state of the art of the problems (& its variants). After Bellman's $O(nt)$ dynamic solution [12], Pisinger [17] first improved it to $O(nt/\log t)$ on word-RAM models. Recently, Koiliaris and Xu gave a deterministic algorithm [18,19] in time $\tilde{O}(\sqrt{n}t)$, which is the best deterministic algorithm so far. Bringmann [5] & Jin and Wu [6] later improved the running time to randomized $\tilde{O}(n + t)$. All these algorithms require $\Omega(t)$ space. Moreover, most of the recent algorithms solve the decision versions. Here we remark that Abboud et al. [14] recently showed that SSUM has no $t^{1-\epsilon}n^{O(1)}$ time algorithm for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) is false. Therefore, the $\tilde{O}(n+t)$ time bound is likely to be *near-optimal*.

In [18] (also see [19, Lemma 2]), the authors gave a deterministic $\tilde{O}(nt)$ algorithm that finds all the hamming weights for all realisable targets less than equal to t. Their algorithm *does not* depend on the number of solutions for a particular target. Compared to this, our Theorem 1 is *faster* when $k = o(n/(\log n)^c)$, for a large constant c. Similarly, with the 'extra' information of k, we give a *faster* deterministic algorithm (which even outputs all the hamming weights of the solutions) compared to $\tilde{O}(\sqrt{n}t)$ decision algorithm in [19,18] (which outputs all the realisable subset sums $\leq t$), when $k = o(\sqrt{n}/(\log n)^c)$, for a large constant c. Here we remark that the $O(nt)$-time dynamic programming algorithm [12] can be easily modified to find all the solutions, but this gives an $O(n(k + t))$-time (and space) algorithm solution.

On the other hand, there have been quite some work on solving SSUM in LOGSPACE. Lokshtanov and Nederlof [20], and Kane[16] (2010) gave $O(\log nt)$

space $\mathsf{poly}(nt)$-time deterministic algorithm, which have been very recently improved to $\tilde{O}(n^2t)$-time and $\mathsf{poly}\log(nt)$ space. On the other hand, Bringmann [5] gave a $nt^{1+\epsilon}$ time, $O(n\log t)$ space *randomized* algorithm, which have been improved to $O(\log n \log\log n + \log t)$ space in [7]. Again, most of the algorithms are decision algorithms and do not output the solution set. In contrast to this, our algorithm in Theorem 2 uses only $O(\log(knt))$ space and outputs all the solution sets, which is near-optimal.

Finally, we remark that in the proof of Theorem 1, we extend analytic tools from [6] to our advantage (see Lemma 6), yet our algorithm for Theorem 1 is *deterministic* (unlike in [6]).

2 Preliminaries and Notations

Notations. $\mathbb{Z}$ and $\mathbb{Q}$ denotes the set of all integers and rationals, respectively. For any integer $n > 0$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$, while $2^{[n]}$ denotes the set of all subsets of $[n]$. log denotes $\log_2$. We also denote $\tilde{O}(g)$ to be $g \cdot \mathsf{poly}(\log g)$.

Sparsity of a polynomial $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ over a field $\mathbb{F}$, denotes the number of nonzero terms in f.

A weight function $w : [n] \longrightarrow [m]$, can be naturally extended to a set $S \in 2^{[n]}$, by defining $w(S) := \sum_{i\in S} w(i)$.

Definition 1 (Subset Sum problem (SSUM)). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the subset sum problem is to decide whether t is a realisable target with respect to $(a_1, \ldots, a_n)$, i.e., there exists $S \subseteq [n]$ such that $\sum_{i\in S} a_i = t$. Here, n is called the* size*, t is the* target *and any $S \subseteq [n]$ such that $\sum_{i\in S} a_i = t$ is a* realisable set *of the subset sum instance.*

Assumptions. Throughout the paper, we assume that $t \geq \max a_i$ for simplicity. Also, we work in the Turing model where basic operations like addition and multiplication over $\mathbb{F}_p$ are not unit-cost unlike Word Ram model considered in [6], for simplicity; in the word RAM model our results will give slightly better result shaving one $\log p$ factor.

Lemma 1 ([21, Isolation Lemma]). *Let n and N be positive integers, and let $\mathcal{F}$ be an arbitrary family of subsets of $[n]$. Suppose $w(x)$ is an integer weight given to each element $x \in [n]$ uniformly and independently at random from $[N]$. The weight of $S \in \mathcal{F}$ is defined as $w(S) = \sum_{x\in S} w(x)$. Then, with probability at least $1 - n/N$, there is a unique set $S' \in \mathcal{F}$ that has the minimum weight among all sets of $\mathcal{F}$.*

Lemma 2 (Kane's Identity [16]). *Let $f(x) = \sum_{i=0}^{d} c_i x^i$ be a polynomial of degree at most d with coefficients c_i being integers. Let $\mathbb{F}_q$ be the finite field of order $q = p^k > d + 2$. For $0 \leq t \leq d$, define*

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

Then, $r_t = 0 \iff c_t$ is divisible by p.

Lemma 3 (Newton's Identities). *Let $X_1, \ldots, X_n$ be $n \geq 1$ variables. Let $P_m(X_1, \ldots, X_n) = \sum_{i=1}^{n} X_i^m$, be the m-th power sum and $E_m(X_1, \ldots, X_n)$ be the m-th elementary symmetric polynomials i.e. $E_m(x_1, \ldots, x_n) = \sum_{1 \leq j_1 \leq \ldots \leq j_m \leq n} X_{j_1} \cdots X_{j_m}$, then*

$$m \cdot E_m(X_1, \ldots, X_n) \;=\; \sum_{i=1}^{m} (-1)^{i-1} E_{m-i}(X_1, \ldots, X_n) \cdot P_i(X_1, \ldots, X_n) \; .$$

Lemma 4 (Vieta's formulas). *Let $f(x) = \prod_{i=1}^{n}(x - a_i)$ be a monic polynomial of degree n. Then, $f(x) = \sum_{i=0}^{n} c_i x^i$ where $c_{n-i} = (-1)^i E_i(a_1, \ldots, a_n), \forall 1 \leq i \leq n$ and $c_n = 1$.*

Lemma 5 (Polynomial division with remainder [22, Theorem 9.6]). *Given a d-degree polynomial f and a linear polynomial g over a finite field $\mathbb{F}_p$, there exists a deterministic algorithm that finds the quotient and remainder of f divided by g in $\tilde{O}(d \log p)$-time.*

Definition 2 (Order of a number $\bmod p$**).** *The order of a ($\bmod p$), denoted as $ord_p(a)$ is defined to be the* smallest *positive integer m such that $a^m \equiv 1 \mod p$.*

Note that when p is prime, $\mathrm{ord}_p(a)$ is clearly finite since $a^{p-1} \equiv 1 \mod p$, from Fermat's Little Theorem. Emil Artin (1927, see [23]) conjectured that for any non-square $a \in \mathbb{Z} \backslash \{-1\}$, there exist infinitely many primes p such that a is a *primitive root* modulo p, i.e. $\mathrm{ord}_p(a) = p - 1$. There has been impressive amount of work done to understand behaviour and distribution of $\mathrm{ord}_p(a)$ [24,25,26]. In particular, we have the following.

Theorem 4 ([27]). *There exists a $\tilde{O}(p^{1/4+\epsilon})$ time algorithm to determinstically find a primitive root over $\mathbb{F}_p$.*

Theorem 5 ([28]). *For $n \geq 25$, there is a prime in the interval $[n, 6/5 \cdot n]$.*

Here is the most important lemma, which is an extension of [6, Lemma 4], where the authors considered the simplest form. In this paper, we need the extensions for the 'robust' usage of this lemma (in section 3).

Lemma 6 (Coefficient Extraction Lemma). *Let $A(x) = \prod_{i \in [n]}(1 + W^b \cdot x^{a_i})$, for any non-negative integers a_i, b and $W \in \mathbb{Z}$. Then, for a prime $p > t$, one can compute $\mathrm{coef}_{x^r}(A(x)) \bmod p$ for all $0 \leq r \leq t$, in time $\tilde{O}((n + t \log(Wb)) \log p)$.*

3 Proof of Theorem 1

We present an $\tilde{O}(k(n + t))$-time deterministic algorithm for outputting all the hamming weight of the solutions, given a $\mathsf{Hamming} - k - \mathsf{SSSUM}$ instance i.e. there are only at most k-many solutions to the SSUM instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$.

Proof of Theorem 1. We start with some notations that we will use throughout the proof.

▶ Basic notations. Assume that the SSUM instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$ has *exactly* m ($m \leq k$) many solutions, and they have ℓ many *distinct* hamming weights $w_1, \ldots, w_\ell$; since two solutions can have same hamming weight, $\ell \leq m$. Moreover, assume that there are λ_i many solutions which appear with hamming weight w_i, for $i \in [\ell]$. Thus, $\sum_{i\in[\ell]} \lambda_i = m \leq k$.

▶ Choosing prime q and a primitive root μ. We will work with a fixed q in this proof, where $q > n + k + t := M$ (we will mention why such a requirement later). We can find a prime q in $\tilde{O}(n+k+t)$ time, since we can go over every element in the interval $[M, 6/5 \cdot M]$, in which we know a prime exists (Theorem 5) and primality testing is efficient [29]. Once we find q, we choose μ such that μ is a *primitive root* over $\mathbb{F}_q$, i.e. $\mathrm{ord}_q(\mu) = q-1$. This μ can be found in $\tilde{O}((n+k+t)^{1/4+\epsilon})$ time using Theorem 4. Thus, the total time complexity of this step is $\tilde{O}(n + k + t)$.

▶ The polynomials. Define the k-many univariate polynomials as follows:

$$f_j(x) \;:=\; \prod_{i\in[n]} \left(1 + \mu^j x^{a_i}\right) , \forall\, j \;\in\; [k] \,.$$

We remark that we do not know ℓ apriori, but we can find m efficiently.

Claim 1 (Finding the exact number of solutions). Given a Hamming$-k-$SSSUM instance, one can find the exact number of solutions, m, deterministically, in $\tilde{O}((n + t)\log(q))$ time.

Proof. Use [6] (see Lemma 6, for the general statement) which gives a deterministic algorithm to find the coefficient of x^t of $\prod_{i\in[n]} (1 + x^{a_i})$ over $\mathbb{F}_q$; this takes time $\tilde{O}((n + t)\log(q))$. □

Since we know the exact value of m, we will just work with f_j for $j \in [m]$, which suffices for our algorithmic purpose. Here is an important claim about coefficients of x^t in f_j's.

Claim 2. $C_j = \mathrm{coef}_{x^t}(f_j(x)) \;=\; \sum_{i\in[\ell]} \lambda_i \cdot \mu^{jw_i}$, for each $j \in [m]$.

Proof. If $S \subseteq [n]$ is a solution to the instance with hamming weight, say w, then this will contribute μ^{jw} to the coefficient of x^t of $f_j(x)$. Since, there are ℓ many weights $w_1, \ldots, w_\ell$ with multiplicity $\lambda_1, \ldots, \lambda_\ell$, the claim easily follows. □

Using Lemma 6, we can find $C_j \mod q$ for each $j \in [m]$ in $\tilde{O}((n+t\log(\mu j))\log q)$ time, owing total $\tilde{O}(k(n+t))$, since $q = O(n+k+t)$, $\mu \leq q-1$, and $\sum_{j\in[m]} \log j = \log(m!) \leq \log(k!) = \tilde{O}(k)$.

Using the Newton's Identities (Lemma 3), we have the following relations, for $j \in [m]$:

$$E_j\left(\mu^{w_1}, \ldots, \mu^{w_\ell}\right) \;\equiv\; j^{-1} \cdot \left(\sum_{i=1}^{j} (-1)^{i-1}\, E_{j-i}\left(\mu^{w_1}, \ldots, \mu^{w_k}\right) \cdot P_i\left(\mu^{w_1}, \ldots, \mu^{w_\ell}\right) \right) \mod q \,. \tag{1}$$

In the above, by $E_j(\mu^{w_1}, \ldots, \mu^{w_\ell})$, we mean $E_j(\underbrace{\mu^{w_1}, \ldots, \mu^{w_1}}_{\lambda_1 \text{ times}}, \underbrace{\mu^{w_2}, \ldots, \mu^{w_2}}_{\lambda_2 \text{ times}}, \ldots, \underbrace{\mu^{w_\ell}, \ldots, \mu^{w_\ell}}_{\lambda_\ell \text{ times}})$, and similar for P_j. Since $q > k$, $j^{-1} \bmod q$ exists, and thus the above relations are valid. Here is another important and obvious observation, just from the definition of P_j's:

Observation 1 *For $j \in [k]$, $C_j \equiv P_j(\mu^{w_1}, \ldots, \mu^{w_\ell}) \bmod q$.*

Note that we know $E_0 = 1$ and P_j's (and $j^{-1} \bmod q$) are already computed. To compute E_j, we need to know $E_1, \ldots, E_{j-1}$ and additionally we need $O(j)$ many additions and multiplications. Suppose, $T(j)$ is the time to compute $E_1, \ldots, E_j$. Then, the trivial complexity is $T(m) \leq \tilde{O}(k^2 \log q) + \tilde{O}(k(n+t))$. But one can do better than $\tilde{O}(k^2 \log q)$ and make it $\tilde{O}(k \log q)$ (i.e solve the recurrence, using FFT), owing the total complexity to $T(m) \leq \tilde{O}(k(n+t))$ (since $q = O(n+k+t)$).

Once, we have computed E_j, for $j \in [m]$, define a new polynomial

$$g(x) := \sum_{j=0}^{m} (-1)^j \cdot E_j(\mu^{w_1}, \ldots, \mu^{w_\ell}) \cdot x^j .$$

Using Lemma 4, it is immediate that $g(x) = \prod_{i=1}^{\ell} (x - \mu^{w_i})^{\lambda_i}$. Further, by definition, $\deg(g) = m$. From g, now we want to extract the roots, namely $\mu^{w_1}, \ldots, \mu^{w_\ell}$ over $\mathbb{F}_q$. We do this, by checking whether $(x - \mu^i)$ divides g, for $i \in [n]$ (since $w_i \leq n$). Using Lemma 5, a single division with remainder takes $\tilde{O}(k \log q)$, therefore, the total time to find all the w_i is $\tilde{O}(nk \log q) = \tilde{O}(nk)$.

Here, we *remark* that we do not use the determinstic root finding or factoring algorithms (for e.g. [30,31]), since it takes $\tilde{O}(mq^{1/2}) = \tilde{O}(k \cdot (k+t)^{1/2})$ time, which could be larger than $\tilde{O}(k(n+t))$.

▶ Reason for choosing q and μ. In the hindsight, there are three important properties of the prime q that will suffice to successfully output the w_i's using the above described steps:

1. Since, Lemma 6 *requires* to compute the inverses of numbers upto t, hence, we would want $q > t$.
2. While computing $E_j(\mu^{w_1}, \ldots, \mu^{w_k})$ using Lemma 3 in the above, one should be able to compute the inverse of all j's less than equal to m. So, we want $q > m$,.
3. To obtain w_i from $\mu^{w_i} \bmod q$, we want $\mathrm{ord}_q(\mu) > n$ (for definition see Definition 2). Since, $w_i \leq n$, this would ensure that we have found the correct w_i.

Here, we remark that we do not need to concern ourselves about the 'largeness' of the coefficients of C_j and make it nonzero $\bmod q$, as required in [6]. For the first two points, it suffices to choose $q > k + t$. Since μ is a primitive root over $\mathbb{F}_q$, this guarantees that $\mathrm{ord}_q(\mu) = q - 1 > n$ and thus we will find w_i from μ^{w_i} correctly.

▶ Total time complexity. The time complexity to find the correct m, q and μ is $\tilde{O}(n + k + t)$. Finding the coefficients of g takes $\tilde{O}(k(n+t))$ time and then

finding w_i from g takes $\tilde{O}(nk \log q)$ time. Thus, the total time complexity remains $\tilde{O}(k(n+t))$. □

Remark 1. The above algorithm can be extended to find the multiplicities λ_i's in $\tilde{O}(k(n+t)+k^{3/2})$ time by finding the largest λ_i, by binary search, such that $(x-\mu^{w_i})^{\lambda_i}$ divides $g(x)$. Finding each λ_i takes $\tilde{O}(m \log q \log(\lambda_i))$ time over $\mathbb{F}_q$, for the same q as above, since the polynomial division takes $\tilde{O}(m \log q)$ time and binary search introduces a multiplicative $O(\log(\lambda_i))$ term. Since, $\sum_{i\in[\ell]} \log(\lambda_i) = \log\left(\prod_{i\in[\ell]} \lambda_i\right)$, using AM-GM, $\prod_{i\in[\ell]} \lambda_i \leq (m/\ell)^\ell$, which is maximized at $\ell = \sqrt{m} \leq \sqrt{k}$, implying $\sum_{i\in[\ell]} \log(\lambda_i) \leq O(\sqrt{k} \log k)$. Since, $m \leq k$, this explains the additive $k^{3/2}$ term in the complexity.

4 Proof of Theorem 2

In this section, we will present a low space algorithm for finding all the realisable sets for $k-\mathsf{SSSUM}$. Our low space algorithms build upon a fundamental number-theoretic identity [16], and efficient sparse multivariate polynomial reconstruction [13].

Proof of Theorem 2. Here are some notations that we will follow throughout the proof.

▶ Basic notations. Let us assume that there are exactly m ($m \leq k$) many realisable sets $S_1, \ldots, S_m$, each $S_i \subseteq [n]$. We remark that for our algorithm we do not need to apriori calculate m.

▶ The multivariate polynomial. For our purpose, we will be working with the following $(n+1)$-variate polynomial:

$$f(x, y_1, \ldots, y_n) := \prod_{i\in[n]} (1 + y_i x^{a_i}) .$$

Since, we have a $k-\mathsf{SSSUM}$ instance $(a_1, \ldots, a_n, t)$, $\mathrm{coef}_{x^t}(f)$ has the following properties.

1. It is an n-variate polynomial $p_t(y_1, \ldots, y_n)$ with sparsity *exactly* m.
2. p_t is a multilinear polynomial in $y_1, \ldots, y_n$, i.e. individual degree of y_i is at most 1.
3. The total degree of p_t is at most n.
4. if $S \subseteq [n]$ is a realisable set, then $\mathbf{y}_S := \prod_{i\in S} y_i$, is a monomial in p_t.

In particular, the following is an immediate but important observation.

Observation 2 $p_t(y_1, \ldots, y_n) = \sum_{i\in[m]} \mathbf{y}_{S_i}$.

Therefore, it suffices to know the polynomial p_t. However, we cannot treat y_i as new variables and try to find the coefficient of x^t since the trivial multiplication algorithm (involving $n+1$ variables) takes $\exp(n)$-time. This is because,

$f(x, y_1, \ldots, y_n) \mod x^{t+1}$ can have $2^n \cdot t$ many monomials as coefficient of x^i, for any $i \leq t$ can have 2^n many multilinear monomials.

However, if we substitute $y_i = c_i \in \mathbb{F}_q$, for some prime q, we claim that we can figure out the value $p_t(c_1, \ldots, c_n)$ from the coefficient of x^t in $f(x, c_1, \ldots, c_n)$ efficiently (see Claim 3). Once we have figured out, we can simply interpolate using the following theorem to reconstruct the polynomial p_t. Before going into the technical details, we state the sparse interpolation theorem below; for simplicity we consider multilinearity (though [13] holds for general polynomials as well).

Theorem 6 ([13]). *Given a black box access to a multilinear polynomial $g(x_1, \ldots, x_n)$ of degree d and sparsity at most s over a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq (nd)^6$, there is a $\mathsf{poly}(snd)$-time and $O(\log(snd))$-space algorithm that outputs all the monomials of g.*

Remark. We represent one monomial in terms of indices (to make it consistent with the notion of realisable set), i.e. for a monomial $x_1x_5x_9$, the corresponding indices set is $\{1, 5, 9\}$. Also, we do not include the indices in the space complexity, as mentioned earlier.

▶ Brief analysis on the space complexity of [13]. Klivans and Spielman [13], did not explicitly mention the space complexity. However, it is not hard to show that the required space is indeed $O(\log(snd))$. [13] shows that substituting $x_i = y^{k^{i-1} \bmod p}$, for some $k \in [2s^2n]$ and $p > 2s^2n$, makes the exponents of the new univaraite polynomial (in y) *distinct* (see [13, Lemma 3]); the algorithm actually tries for all k and find the correct k. Note that the degree becomes $O(s^2nd)$. Then, it tries to first find out the coefficients by simple univariate interpolation [13, Section 6.3] . Since we have blackbox access to $g(a_1, \ldots, a_n)$, finding out a single coefficient, by univariate interpolation (which basically sets up linear equations and solve) takes $O(\log(snd))$ space and $\mathsf{poly}(snd)$ time only. In the last step, to find one coefficient, we can use the standard univariate interpolation algorithm which uses the Vandermonde matrices and one entry of the inverse of the Vandermonde is log-space computable [4].

At this stage, we know the coefficients (one by one), but we do not know which monomials the coefficients belong. However, it suffices to substitute $x_i = 2y^{k^{i-1} \bmod p}$. Using this, we can find the the correct value of the first exponent in the monomial. For eg. if after the correct substitution, y^{10} appears with coefficient say 5, next step, when we change just x_1, if it does not affect the coefficient 5, y_1 is not there in the monomial corresponding to the monomial which has coefficient 5, otherwise it is there (here we also use that it is multilinear and hence the change in the coefficient must be reflected). This step again requires univariate interpolation, and one has to repeat this experiment wrt each variable to know the monomial exactly corresponding to the coefficient we are

[4] In fact Vandermonde determinant and inverse computations are in $\mathsf{TC}^0 \subset \mathsf{LOGSPACE}$, see [32].

working with. We can reuse the space for interpolation and after one round of checking with every variable, it outputs one exponent at this stage. This requires $O(\log(snd)$-space and $\mathsf{poly}(snd)$ time.

With a more careful analysis, one can further improve the field requirement to $|\mathbb{F}| \geq (nd)^6$ only (and not dependent on s); for details see [13, Thm. 5 & 11].

Now we come back to our subset sum problem. Since we want to reconstruct an n-variate m sparse polynomial p_t which has degree at most n, it suffices to work with $|\mathbb{F}| \geq n^{12}$. However, we also want to use Kane's identity (Lemma 2), which requires $q > \deg(f(x, c_1, \ldots, c_n)) + 2$, and $\deg(f(x, c_1, \ldots, c_n)) \leq nt$. Denote $M := \max(nt+3, n^{12})$. Thus, it suffices to we work with $\mathbb{F} = \mathbb{F}_q$ where $q \in [M, (6/5) \cdot M]$, such prime exists (Theorem 5) and easy to find deterministically in $\mathsf{poly}(nt)$ time and $O(\log(nt))$ space using [29]. In particular, we will substitute $y_i = c_i \in [0, q-1]$.

Claim 3. Fix $c_i \in [0, q-1]$, where $q \in [M, (6/5) \cdot M]$. Then, there is a $\mathsf{poly}(nt)$-time and $O(\log(nt))$ space algorithm which computes $p_t(c_1, \ldots, c_n)$ over $\mathbb{F}_q$.

Proof. Note that, we can evaluate each $1+c_i x^{a_i}$, at some $x = \alpha \in \mathbb{F}_q$, in $\tilde{O}(\log nt)$ time and $O(\log(nt))$ space. Multiplying n of them takes $\tilde{O}(n \log(nt))$-time and $O(\log(nt))$ space.

Once we have computed $f(\alpha, c_1, \ldots, c_n)$ over $\mathbb{F}_q$, using Kane's identity (Lemma 2), we can compute $p_t(c_1, \ldots, c_n)$, since

$$p_t(c_1, \ldots, c_n) \;=\; - \sum_{\alpha \in \mathbb{F}_q^*} \alpha^{q-1-t} f(\alpha, c_1, \ldots, c_n) \; .$$

As each evaluation $f(\alpha, c_1, \ldots, c_n)$ takes $\tilde{O}(n \log(nt))$ time, and we need $q-1$ many additions, multiplications and modular exponentiations, total time to compute is $\mathsf{poly}(nt)$. The required space still remains $O(\log(nt))$. □

Once, we have calculated $p_t(c_1, \ldots, c_n)$ efficiently, now we try different values of $(c_1, \ldots, c_n)$ to reconstruct p_t using Theorem 6. Since, p_t is a n-variate at most k sparse polynomial with degree at most n, it still takes $\mathsf{poly}(knt)$ time and $O(\log(knt))$ space. This finishes the proof. □

5 Conclusion

This work introduces some interesting search versions of variants of SSUM problem and gives efficient algorithms for each of them. This opens a variety of questions which require further rigorous investigations.

1. Can we improve the time complexity of Theorem 2? Because of using Theorem 6, the complexity for interpolation is already cubic. Whether some other algebraic (non-algebraic) techniques can improve the time complexity, while keeping it low space, is not at all clear.
2. Can we use these algebraic-number-theoretic techniques, to give a *deterministic* $\tilde{O}(n+t)$ time algorithm for decision version of SSUM?
3. Can we improve Remark 1 to find both the hamming weights w_i as well as the multiplicities λ_i, in $\tilde{O}(k(n+t))$ time?

References

1. Harry R Lewis. Computers and intractability. a guide to the theory of np-completeness, 1983. 1
2. Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *Theory of Cryptography Conference*, pages 382–400. Springer, 2010. 1
3. Sebastian Faust, Daniel Masny, and Daniele Venturi. Chosen-ciphertext security from subset sum. In *Public-Key Cryptography–PKC 2016*, pages 35–46. Springer, 2016. 1
4. Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1777–1796. SIAM, 2021. 1
5. Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017. 1, 5, 6
6. Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. *arXiv preprint arXiv:1807.11597*, 2018. 1, 2, 3, 5, 6, 7, 8, 9
7. Ce Jin, Nikhil Vyas, and Ryan Williams. Fast low-space algorithms for subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1776. SIAM, 2021. 1, 3, 6
8. Andre Esser and Alexander May. Low weight discrete logarithm and subset sum in 20. 65n with polynomial memory. *memory*, 1:2, 2020. 1
9. Daniel J Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *International Workshop on Post-Quantum Cryptography*, pages 16–33. Springer, 2013. 1
10. Alexander Helm and Alexander May. Subset sum quantumly in 1.17ˆ n. In *13th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 1
11. Yang Li and Hongbo Li. Improved quantum algorithm for the random subset sum problem. *arXiv preprint arXiv:1912.09264*, 2019. 1
12. Richard E. Bellman. Dynamic programming, 1957. 1, 5
13. Adam R Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 216–223, 2001. 2, 5, 10, 11, 12
14. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019. 2, 5
15. Dylan M McKay and Richard Ryan Williams. Quadratic time-space lower bounds for computing natural functions with a random oracle. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 3
16. Daniel M Kane. Unary subset-sum is in logspace. *arXiv preprint arXiv:1012.1336*, 2010. 3, 5, 6, 10
17. David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999. 5

18. Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms (TALG)*, 15(3):1–20, 2019. 5
19. Konstantinos Koiliaris and Chao Xu. Subset sum made simple. *arXiv preprint arXiv:1807.08248*, 2018. 5
20. Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 321–330, 2010. 5
21. Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354, 1987. 6
22. Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013. 7
23. Pieter Moree. Artin's primitive root conjecture–a survey. *Integers*, 12(6):1305–1416, 2012. 7
24. Rajiv Gupta and M Ram Murty. A remark on artin's conjecture. *Inventiones mathematicae*, 78(1):127–130, 1984. 7
25. Pál Erdös and M Ram Murty. On the order of a (mod p). In *CRM Proceedings and Lecture Notes*, volume 19, pages 87–97, 1999. 7
26. Koji Chinen and Leo Murata. On a distribution property of the residual order of a (modp). *Journal of Number Theory*, 105(1):60–81, 2004. 7
27. Igor Shparlinski. On finding primitive roots in finite fields. *Theoretical computer science*, 157(2):273–275, 1996. 7
28. Jitsuro Nagura. On the interval containing at least one prime number. *Proceedings of the Japan Academy*, 28(4):177–181, 1952. 7
29. Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004. 8, 12
30. Victor Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Information Processing Letters*, 33(5):261–267, 1990. 9
31. Jean Bourgain, Sergei Konyagin, and Igor Shparlinski. Character sums and deterministic polynomial root finding in finite fields. *Mathematics of Computation*, 84(296):2969–2977, 2015. 9
32. Alexis Maciel and Denis Therien. Threshold circuits of small majority-depth. *Information and Computation*, 146(1):55–83, 1998. 11