On the Variants of Subset Sum: Projected and Unbounded

Pranjal Dutta
School of Computing
National University of Singapore (NUS)
Singapore
duttpranjal@gmail.com

Mahesh Sreekumar Rajasree
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
Delhi, India
srmahesh1994@gmail.com

***Abstract*—In this work, we focus on variants of Subset Sum. We first define a new variant called the *Unique Projection Subset Sum* ($\mathsf{u-PSSUM}$) problem: Given $(a_1, \ldots, a_n) \in \mathbb{Z}^n_{\geq 0}$, such that for every t, $\sum_{i\in[n]} c_i a_i = t$, for $c_i \in \{0, 1\}$ has unique solution if exists, $\mathsf{u-PSSUM}$ asks for a vector $\vec{x} = (x_1, \ldots, x_n)$, such that there exists an $S \subseteq [n]$, where $\sum_{i\in S} x_i = t$, and $||\vec{a}-\vec{x}||_p$ is minimized, where $||\cdot||_p$ denotes the ℓ_p norm. We present a deterministic $O(nt)$-time algorithm and a randomized $\widetilde{O}(n + t)$-time algorithm for $\mathsf{u-PSSUM}$, in ℓ_1-norm.**

The second one is already a known variant called *Unbounded Subset Sum* (UBSSUM) problem, which takes a tuple of non-negative integers $(a_1, \ldots, a_n, t)$, as an input, and asks whether there exists non-negative integers $\beta_1, \ldots, \beta_n$, such that $\sum_{i=1}^n \beta_i a_i = t$. We present the first polynomial time reductions from UBSSUM to Closest Vector Problem (CVP) in ℓ_1 and ℓ_∞ norms. We also give new algorithms for two variants of UBSSUM problem.

***Index Terms*—subset sum, unbounded subset sum, lattice, closest vector problem**

I. Introduction

Given a set of non-negative integers $(a_1, \ldots, a_n, t)$, the Subset Sum problem (SSUM) asks whether there exists an $S \subseteq [n]$ such that $\sum_{i\in S} a_i = t$. Such an S is called a *realisable set*, if exists. It is one of the most famous NP-complete problem [1, p. 226] which admits to an $O(nt)$ time algorithm due to Bellman [2]. In this paper, we study a two variants of the subset sum, called UBSSUM and PSSUM.

Definition 1 (Unbounded Subset Sum (UBSSUM))**.** *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}^{n+1}_{\geq 0}$, the* UBSSUM *problem asks whether there exists $\beta_1, \ldots, \beta_n$ such that β_i are non-negative integers and $\sum_{i=1}^n \beta_i a_i = t$.*

UBSSUM is known to be NP-complete [3]. It has a trivial $O(nt)$ time (*pseudo-polynomial* time) dynamic programming algorithm which requires $\Omega(t)$ space [2]. Below, we define two problems (Problem 1-2), which are variants of UBSSUM. We will be working with these variants in this paper.

Problem 1 ($k-$ SUBSSUM)**.** *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}^{n+1}_{\geq 0}$, the $k-$ SUBSSUM problem asks to output all $(\beta_1, \ldots, \beta_n)$ where β_i are non-negative integers and $\sum_{i=1}^n \beta_i a_i = t$ provided the number of such solutions is at most k.*

Problem 2 (Hamming $-$ $k-$ SUBSSUM)**.** *Given a $k-$ SUBSSUM instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}^{n+1}_{\geq 0}$, Hamming $-$ $k-$ SUBSSUM asks to output all the hamming weights of the solutions, i.e., $\sum_{i=1}^n \beta_i$.*

► Remark. We want $\vec{a} \cdot \vec{v} = t$, where $\vec{v} \in \mathbb{Z}^n_{\geq 0}$. Similarly, like in the SSUM case (i.e., $\vec{v} \in \{0, 1\}^n$), we want $|v|_1$, which is exactly the quantity $\sum_i \beta_i$, as above. Thus, this definition can be thought as a natural extension of the hamming weight of the solution, in the unbounded regime.

Next, we define a slightly different-looking variant of Subset Sum problem, called PSSUM_p. We also mention their connection (& difference) in some particular scenarios.

Definition 2 (Projection Subset Sum (PSSUM_p))**.** *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}^{n+1}_{\geq 0}$, the projection subset sum problem asks for a vector $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ such that there exists an $S \subseteq [n]$ where $\sum_{i\in S} x_i = t$ and $||\vec{a}-\vec{x}||_p$ is minimised where $\vec{a} = (a_1, \ldots, a_n)$.*

► Remark. We do not restrict the solution vector $\vec{x}$ to be non-negative. This is to ensure that always exists a solution. Also, note that one can solve the decision version of SSUM by solving the PSSUM_p, i.e., the SSUM is a YES instance iff the solution to PSSUM_p is $\vec{a}$.

Finally we define the PSSUM-variant that we will be working on this paper, named unique-PSSUM_p; this is

a restricted variant on PSSUM_p.

Definition 3 (Unique Projection Subset Sum ($\mathsf{u-PSSUM}_p$)). *Assume $(a_1, \ldots, a_n) \in \mathbb{Z}^n_{\geq 0}$, such that for every t, solution for $\sum_{i\in[n]} c_i a_i = t$, for $c_i \in \{0, 1\}$ is unique, if exists. Given such $(a_1, \cdots, a_n) \in \mathbb{Z}^n_{\geq 0}$, and some t, the unique projection subset sum problem asks for a vector $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ such that there exists an $S \subseteq [n]$ where $\sum_{i\in S} x_i = t$ and $||\vec{a} - \vec{x}||_p$ is minimised where $\vec{a} = (a_1, \ldots, a_n)$.*

A. Applications

In recent years, SSUM and its variants have gained significant attractions due to applications in provable-secure cryptosystems [4], [5], and astounding algorithmic improvements both in classical and quantum world [6]–[9]. Unbounded subsetsum (or unbounded knapsack) has also gained attention in the area of complexity and machine learning [10]–[13]. Remarkably, PSSUM problem, is very similar to the *random* SSUM problem over reals, a strikingly important problem in the area of Machine Learning [14], [15], where the question is: *What is the required over-parameterization so that a network of random weights can be pruned to compute/approximate a smaller target network?* Our PSSUM and u − PSSUM problems are more classical, in the theoretical sense. It is interesting to note that even in the classical setting of SSUM, assumption on the uniqueness of every solution *is not known* to help to design any faster algorithm.

B. Our Contributions

In this section, we briefly state our main results. Our results are a mixture of reductions and algorithms for variants of subset-sum problems.

Theorem 1. *There exists a deterministic $O(nt)$-time algorithm and a randomized $\widetilde{O}(n + t)$-time algorithm for $\mathsf{u-PSSUM}_1$.*

Theorem 2. *There exists a deterministic polynomial time reduction from* UBSSUM *to* SSUM.

▶ Remark. The reduction in Theorem 2 is a parsimonious reduction, i.e., there is a one-to-one correspondence between the solutions of the UBSSUM and the solutions of the SSUM instance. Also, any $T(n, t)$ time algorithm that solve SSUM gives an $T(n \log(t), t)$-time algorithm to solve UBSSUM.

Theorem 3. *There exists an efficient reduction from* UBSSUM *to* CVP_∞.

Theorem 4. *There exists an efficient reduction from* UBSSUM *to* CVP_1.

▶ Remark. We emphasis that is the *first* time such direct connections have been shown. It is worth noting that a reduction from UBSSUM to CVP_2 would imply a $2^{O(n)}$ time algorithm for UBSSUM.

Theorem 5. *There is an $\widetilde{O}(k(n+t))$-time deterministic algorithm for* Hamming − k − SUBSSUM.

Theorem 6. *There is a* poly(knt)*-time and $O(\log(knt))$-space deterministic algorithm which solves k − SUBSSUM.*

C. Prior Works

Before going into the proofs of our results, we briefly review the state of the art of the problems (& its variants). Bellman's $O(nt)$ dynamic solution [2] for solving SSUM is one of the first non-trivial algorithms. Koiliaris and Xu gave the fastest deterministic algorithm [16], [17] in time $\tilde{O}(\sqrt{nt})$. Bringmann [18] & Jin and Wu [6] later improved the running time to randomized $\tilde{O}(n+t)$. All these algorithms solve the decision versions and require $\Omega(t)$ space. In terms of space complexity, Jin, Vyas and Williams [7] gave an algorithm $O(\log(nt)$ space, which is a significant improvement over Bringmann's [18] $\widetilde{O}(n \log(t))$ result.

Unlike the SSUM problem, UBSSUM problem has a $O(n + \min_i a_i^2)$-time determinstic algorithm [19]. Recently, Bringmann [18] gave an $\widetilde{O}(t)$ deterministic algorithm for UBSSUM. Klein [20] gave an algorithm that solve UBSSUM in time $O(a_n \log(a_n) \log(F/a_n))$ where F is the Frobenius number and a_n is the largest a_i.

Another variant of SSUM problem is the Subset Product where the sum is replaced with a product. The fastest algorithm that solves Subset Product is due to Dutta and Rajasree [21]. Other works on variants of SSUM includes [22]–[25].

II. Preliminaries and Notations

Notations. $\mathbb{Z}$ and $\mathbb{Q}$ denotes the set of all integers and rational numbers respectively. For any positive integer $n > 0$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Given an n-dimensional vector $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{Q}^n$, the ℓ_p norm of $\vec{v}$ (denoted as $||\vec{v}||_p$) is defined as $||\vec{v}||_p = (\sum_{i=1}^n v_i^p)^{1/p}$.

$\mathbb{F}[x_1, \ldots, x_k]$ denotes the ring of k-variate polynomials over field $\mathbb{F}$ and $\mathbb{F}[[x_1, \ldots, x_k]]$ is the ring of power series in k-variables over $\mathbb{F}$. We will use the shorthand notation $\overline{x}$ to denote the collection of variables

$(x_1, \ldots, x_k)$ for some k. For any non-negative integer vector $\overline{e} \in \mathbb{Z}^k$, $\overline{x}^{\overline{e}}$ denotes $\prod_{i=1}^{k} x_i^{e_i}$. Using these notations, we can write any polynomial $f(\overline{x}) \in \mathbb{Z}[\overline{x}]$ as $f(\overline{x}) = \sum_{\overline{e} \in S} f_{\overline{e}} \cdot \overline{x}^{\overline{e}}$ for some suitable set S. We denote $\text{coef}_{\overline{x}^{\overline{e}}}(f)$, as the coefficient of $\overline{x}^{\overline{e}}$ in the polynomial $f(\overline{x})$ and $\deg_{x_i}(f)$ as the highest degree of x_i in $f(\overline{x})$. Two important power series over $\mathbb{Q}[[x]]$ are:

- $\ln(1+x) = \sum_{k \geq 1} \frac{(-1)^{k-1} x^k}{k}$.
- $\exp(x) = \sum_{k \geq 0} \frac{x^k}{k!}$.

They are inverse to each other and satisfy the basic properties:

$$\exp\left(\ln\left(1 + f(x)\right)\right) = 1 + f(x)$$

$$\ln\left((1 + f(x)) \cdot (1 + g(x))\right) = \ln(1+f(x)) + \ln(1+g(x))$$

for every $f(x), g(x) \in x\mathbb{Q}[x]$ (i.e., constant term is 0). Here is an important lemma to compute $\exp(f(x)) \bmod x^{t+1}$; for details see [26]; for an alternative proof, see [6, Lemma 2].

Lemma 7 ([26]). *Given a polynomial $f(x) \in x\mathbb{F}[x]$ of degree at most t ($t < p$), one can compute a polynomial $g(x) \in \mathbb{F}_p[x]$ in $\widetilde{O}(t)$ time such that $g(x) \equiv \exp(f(x)) \mod \langle x^{t+1}, p \rangle$.*

The following are generalization of the lemmas in [6]. For detailed proofs, we refer to [25].

Lemma 8 (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 $\text{coef}_{x^r}(A(x)) \bmod p$ for all $0 \leq r \leq t$, in $\widetilde{O}((n + (t+1)\log(Wb)))$ randomized time.*

A consequence of the above lemma is the following theorem by setting $W = b = 1$.

Theorem 9 ([6]). *There exists an $\widetilde{O}(n+t)$ time randomized algorithm that solves* SSUM.

Definition 4 ($k-$SSSUM [25]). *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.*

Theorem 10 ([25]). *There is a $\text{poly}(knt)$-time and $O(\log(knt))$-space deterministic algorithm which solves $k-$* SSSUM.

Lemma 11 (Fast multivariate exponentiation [25]). *Let $\overline{x} = (x_1, \ldots, x_k)$ and $f(\overline{x}) = \sum_{i=1}^{t_1} f_i(\overline{x}) \cdot x_1^i \in \mathbb{F}_p[\overline{x}]$ where $f_i(\overline{x}) \in \mathbb{F}_p[x_2, \ldots, x_k]$ such that*

1) *$f(\overline{x}) \bmod \langle x_1, \ldots, x_k \rangle = 0$, i.e., the constant term of $f(\overline{x})$ is 0, and,*
2) *$\deg_{x_j}(f) = t_j$, for positive integers t_j.*

Then, there is an $\widetilde{O}(\prod_{i=1}^{k}(2t_i+1))$ time deterministic algorithm that computes a polynomial $g(\overline{x}) \in \mathbb{F}_p[\overline{x}]$ such that $g(\overline{x}) \equiv \exp(f(\overline{x})) \bmod \langle x_1^{t_1+1}, \ldots, x_k^{t_k+1} \rangle$ over $\mathbb{F}_p$.

Lemma 12 (Fast logarithm computation [25]). *Let $f(\overline{x}) = \prod_{i=1}^{n} \left(1 + \prod_{j=1}^{k} x_j^{a_{ij}}\right)$. Then, there exists an $\widetilde{O}(kn + \prod_{i=1}^{k} t_i)$ time deterministic algorithm that computes $\text{coef}_{\overline{x}^{\overline{e}}}(\ln(f(\overline{x}))) \bmod p$ for all $\overline{e}$, such that $\overline{e} = (e_1, \ldots, e_k)$ with $e_i \leq t_i$.*

Lemma 13 (Kane's Identity [27]). *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 14 (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*

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

Remarks 1. *$E_m(X_1, \ldots, X_n) = 0$ when $m > n$ (for a quick recollection, see wiki).*

Lemma 15 (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 16 (Polynomial division with remainder [28, 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.*

Lemma 17 (Polynomial inversion [28, Theorem 9.4]). *There exists an $\widetilde{O}(t \log(q))$ time algorithm that takes as input $f(x) \in \mathbb{F}_q[x]$ and $t \in \mathbb{N}$ and outputs $g \in \mathbb{F}_q[x]$ such that $f \cdot g \equiv 1 \mod x^t$.*

Theorem 18 ([29]). *For $n \geq 25$, there is a prime in the interval $[n, \frac{6}{5} \cdot n]$.*

Theorem 19 ([30]). *There exists a $\tilde{O}(p^{1/4+\epsilon})$ time algorithm to determinstically find a primitive root over $\mathbb{F}_p$.*

Definition 5 (Lattice). *A lattice $\mathcal{L}(B)$ where $B = [b_1, \ldots, b_n] \in \mathbb{Q}^{m \times n}$ is the set of all integer combination of column vectors of B, i.e.,*

$$\mathcal{L}(B) = \{B\vec{z} \mid \forall \vec{z} \in \mathbb{Z}^n\}$$

Definition 6 (Closest Vector Problem (CVP_p)). *Given $B \in \mathbb{Q}^{m \times n}, \vec{t} \in \mathbb{Q}^m$, the closest vector problem asks for a closest lattice vector $\vec{v} \in \mathcal{L}(B)$ to $\vec{t}$, i.e.,*

$$||\vec{v} - \vec{t}||_p \leq ||\vec{w} - \vec{t}||_p \ , \forall \vec{w} \in \mathcal{L}(B)$$

III. Proof of Theorem 1

Both the algorithms work on a common idea which we present below. The randomization helps to get a slightly better result, which we will discuss towards the end of the proof.

Proof. Let the input be $a_1, \ldots, a_n, t$. Without loss of generality, we can assume that $a_i \leq a_{i+1}, \forall i \in [n-1]$. If $t < a_1$, then it is easy to see that $(t, a_2, \ldots, a_n)$ is a solution. So, it is enough to handle the case when $t \geq a_1$. Let t' be the *closest* positive integer such that there exists $S \subseteq [n]$, where $\sum_{i \in S} a_i = t'$, i.e., t' is the closest integer to t that is also a subset sum of $a_1, \ldots, a_n$. Let $j \in S$, be any index in S. It is not apriori clear how to find such an index j. We will discuss this in the time-complexity analysis of the algorithm. Assuming one can efficiently find an index j, we define:

$$x_i \ := \ a_i, \forall \, i \ \in \ [n] \ \setminus \ \{j\} \ ,$$

$$x_j \ := \ a_j \ + \ t \ - \ t' \ .$$

Claim 1. *$\vec{x} = (x_1, \ldots, x_n)$ is a solution to PSSUM_1. Moreover, t' is always bounded by $2t$.*

Proof of Claim 1. The proof is divided into two parts.

First part. Suppose there exists a *better* solution $\vec{x'} = (x'_1, x'_2, \ldots, x'_n)$, i.e., $||\vec{a} - \vec{x}'||_1 < ||\vec{a} - \vec{x}||_1 = |t - t'|$, and there exists $S' \subseteq [n]$, such that $\sum_{i \in S'} x'_i = t$. Then, we get $\sum_{i \in S'} a_i - t = \sum_{i \in S'} (a_i - x'_i) \leq \sum_{i \in S'} |a_i - x'_i| \leq \sum_{i=1}^n |a_i - x'_i| < |t - t'|$.

Similarly, $t - \sum_{i \in S'} a_i < |t - t'|$. This implies that $|\sum_{i \in S'} a_i - t| < |t - t'|$, and this is a contradiction, because t' is the *closest* to t, and thus, any subset sum of $a_1, \ldots, a_n$ (even with respect to S') must be farther from t.

Second part. To show that t' is always bounded by $2t$, we argue in cases.

1) If $t \geq \sum_{i=1}^n a_i$, then clearly $t' = \sum_{i=1}^n a_i \leq 2t$.
2) If $t < \sum_{i=1}^n a_i$, then t must lie between two integers $s_1 = \sum_{i \in S_1} a_i$ and $s_2 = \sum_{i \in S_2} a_i$, where $S_1, S_2 \subseteq [n]$, i.e., $s_1 \leq t \leq s_2$. Let us further assume that s_1 and s_2 are the closest to t.

   Case 1. If t is closer to s_1, then $t' = s_1 \leq t \leq 2t$.

   Case 2. If t is closer to s_2, then $s_2 - t < t - s_1 \implies s_2 < 2t - s_1 < 2t$, because s_2 is a positive integer.

□

From Claim 1, the time complexity of the above algorithm is at most as the time complexity of finding *all* possible subset sums achievable by $a_1, \ldots, a_n$, up to $2t$. Now, we describe both the algorithms.

a) Deterministic Algorithm: Using Bellman's dynamic algorithm [2], this would take only $O(nt)$ time and $O(t)$ space. This includes finding the index j. Formally, we buildup the table M, such that $M[j, t']$ is 1 iff the subset $S \subseteq [j]$ is a solution for $\sum_{i \in S} a_i = t'$. From our analysis, $j \leq n$ and $t' \leq 2t$, and hence the algorithm follows. The proof works because dynamically it computes all possible answers (0 or 1) till $2t$.

b) Randomized Algorithm: By carefully analyzing the algorithm in Theorem 9 which uses Lemma 11 and Lemma 12, we can show that there exists $\widetilde{O}(n + t)$ time algorithm to find the exact value of t'. Consider the univariate polynomial $f(x) = \prod_{i=1}^n (1 + x^{a_i})$ and a randomly chosen prime $p \in [n + 2t + 1, (n + 2t)^3]$. Using simple counting technique, one call show that p does not divide t' with high probability. Using Lemma 12 (where $k = 1$), we can compute the polynomial $F(x) = \ln(f(x)) \mod \langle x^{2t+1}, p \rangle$ in time $\widetilde{O}(n + t)$ time. We then invoke Lemma 11 (with $k = 1$) to compute $g(x) = \exp(F(x)) = \exp(\ln(f(x))) = f(x) \mod \langle x^{2t+1}, p \rangle$ in time $\widetilde{O}(t)$. All that remains now is to scan through all the non-zero coefficients in $g(x)$ to find t' which is closest to t. This is just a linear search, hence it takes $O(t)$ time.

Once we have found the t', we need to find an index j. This part is bit tricky, but still we will use some nice algebraic structure to exploit the uniqueness of solutions. Consider, the polynomial

$$f_1(x) = \prod_{i=1}^{\lfloor n/2 \rfloor} (1 + \mu \cdot x^{a_i}) \prod_{i=\lfloor n/2 \rfloor + 1}^{n} (1 + x^{a_i}) \bmod p \, ,$$

where p is chosen as before, and μ is the primitive root i.e., $\text{ord}_p(\mu) = p-1$. We can find it determinstically in $\tilde{O}(n+t)^{1/2}$-time [30]. Note that, if any index from $[1, \lfloor n/2 \rfloor]$ is not in the solution set, the coefficient of $x^{t'}$ in f_1 remains 1; otherwise it becomes μ^ℓ for some $n \geq \ell \geq 1$. Note that $\mu^\ell \neq 1 \bmod p$, by assumption since $\text{ord}_p(\mu) = p-1 > n$. Therefore by checking the coefficient of $x^{t'}$, we can find decide where one index comes from first half or second half. This step takes $\tilde{O}(n+t)$-time. We keep on doing a binary search until we find a single index j. The total time complexity of the algorithm remains $\tilde{O}((n+t)\log n) = \tilde{O}(n+t)$. This finishes the analysis.

□

IV. Efficient reduction from UBSSUM to SSUM, CVP_∞ and CVP_1

We first present a deterministic polynomial time reduction from UBSSUM to SSUM.

Proof of Theorem 2. Let $(a_1, \ldots, a_n, t) \in \mathbb{Z}^{n+1}_{\geq 0}$ be an instance of UBSSUM. The reduction generates the following SSUM instance $(\underbrace{a_1, 2a_1, 4a_1, \ldots, 2^\gamma a_1}_{\gamma+1 \text{ entries}}, \underbrace{a_2, 2a_2, \ldots, 2^\gamma a_2}_{\gamma+1 \text{ entries}}, \ldots, \underbrace{a_n, 2a_n, \ldots, 2^\gamma a_n}_{\gamma+1 \text{ entries}}, t)$ of size $n(\gamma+1)$ where $\gamma = \lfloor \log(t) \rfloor$.

Let $(\beta_1, \ldots, \beta_n)$ be a solution to the UBSSUM instance, i.e., $\sum_{i=1}^n \beta_i a_i = t$. Since, β_i, a_i, t are all non-negative integers, we have $\beta_i \leq t, \forall i \in [n]$. Therefore, β is at most $(\gamma+1)$-bit integer. Let $\beta_i^{(j)}$ be the j^{th} bit of β_i, then we have

$$t = \sum_{i=1}^n \beta_i a_i = \sum_{i=1}^n \left(\sum_{j=0}^{\gamma} \beta_i^{(j)} 2^j \right) \cdot a_i$$
$$= \sum_{i=1}^n \sum_{j=0}^{\gamma} \beta_i^{(j)} \cdot (2^j a_i)$$

which implies that the SSUM instance also has a solution. Similarly, we can show the reverse direction, i.e., if SSUM instance has a solution, then UBSSUM is also has a solution. This concludes the proof. □

We now focus on two reductions from UBSSUM to CVP problem.

Proof of Theorem 3. Suppose $a_1, \ldots, a_n, b$ is the input to the UBSSUM problem. Let $\lambda = (b+1)$. Consider the CVP_∞ instance as follows.

$$B = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & 1 \\ \lambda a_1 & \lambda a_2 & \ldots & \lambda a_n \end{bmatrix}, t = \begin{bmatrix} b \\ b \\ \vdots \\ b \\ \lambda b \end{bmatrix}$$

and the bound is b. If the UBSSUM instance is YES, this implies that $\exists x \in \mathbb{Z}^n_{\geq 0}$ such that $ax = b$. Also, we have $||t - Bx||_\infty \leq b$.

Assume that CVP_∞ is a YES instance. This means there exists $x \in \mathbb{Z}^n$ such that $||t - Bx||_\infty \leq b$. Since λ is large, we have $(t - Bx)_{n+1} = 0$. We now show that $x \in \mathbb{Z}^n_{\geq 0}$. Suppose, $x_i < 0 \implies (t - Bx)_i > b$ which is a contradiction. □

Proof of Theorem 4. Suppose $a_1, \ldots, a_n, b$ is the input to the UBSSUM problem. Let $\lambda = b+1$. Consider the following CVP_1 instance.

$$B = \begin{bmatrix} -a_1 & 0 & \ldots & 0 \\ 0 & -a_2 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & -a_n \\ \lambda a_1 & \lambda a_2 & \ldots & \lambda a_n \end{bmatrix}, t = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \lambda b \end{bmatrix}$$

where the bound is b. If the UBSSUM instance is YES, this implies that $\exists x \in \mathbb{Z}^n_{\geq 0}$ such that $ax = b$. This implies that $||t - Bx||_1 = b$, hence the CVP_1 instance is YES.

Assume that CVP_1 is a YES instance. This means that there exists $x \in \mathbb{Z}^n$ such that $||t - Bx|| \leq b$. Since $\lambda = b+1$ this implies that $(t - Bx)_{n+1}$ is a multiple of $b+1$ and is smaller than b. Therefore, $(t - Bx)_{n+1} = 0 \implies \sum_{i=1}^n a_i x_i = b$. But, since $||t - Bx||_1 \leq b$, we have

$$b \geq \sum_{i=1}^n |a_i x_i| = \sum_{i=1}^n a_i |x_i| \geq \sum_{i=1}^n a_i x_i = b$$

Therefore, all the inequalities in the above equations are equality and a necessary condition for this is $|x_i| = x_i, \forall i \in [n] \implies x_i \leq 0, \forall i \in [n]$. □

► Remark. The best-known algorithm for CVP_1 and CVP_∞ runs in $n^{O(n)}$-time [31], which is not better than the n^n-time algorithm based on solving Integer Linear Programming (ILP) [32]. However, this is the *first* time such reductions have been shown. And moreover, CVP_p is *believed* to have a 2^n-time algorithm for any $1 \leq p \leq \infty$, in which case these reductions will give a better algorithm than the existing algorithm.

V. Algorithms for Unbounded Subset Sum UBSSUM

Proof of Theorem 5. The algorithm starts with picking a prime q and a primitive root $\mu \in \mathbb{F}_q$.

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 by scanning through the interval $[M, 6/5 \cdot M]$, in which we know a prime exists (Theorem 18) and primality testing is efficient [33]. Once we find q, we choose μ such that μ is a *primitive root* over $\mathbb{F}_q$, i.e., $\text{ord}_q(\mu) = q-1$. This μ can be found in $\tilde{O}((n+k+t)^{1/4+\epsilon})$ time using Theorem 19. Thus, the total time complexity of this step is $\tilde{O}(n+k+t)$.

Finding the exact number of solutions m. We will show how to find the exact number of solutions $m (m \leq k)$ for the input $k-$ SUBSSUM instance. To do that, define the polynomial f_0:

$$f_0(x) := \prod_{i=1}^{n} \left(\frac{1}{1-x^{a_j}}\right) = \prod_{i=1}^{n} \left(1 + x^{a_i} + x^{2a_i} + \ldots\right)$$

In the above, we used the *inverse identity* $1/(1-x) = \sum_{i \geq 0} x^i$. By expanding the above, it is easy to see that $\text{coef}_{x^t}(f_0(x)) = m$, where m is the *exact* number of solutions to the $k-$ SUBSSUM. Let us define the polynomial $h_0(x) = \prod_{i=1}^{n}(1-x^{a_i})$. Note that, we can compute $h_0(x) = f_0(x)^{-1} \bmod x^{t+1}$, over $\mathbb{F}_q$ efficiently in $\widetilde{O}(k+t)$ time by following the same steps used in Theorem 9 (refer to the end of Section III). After computing $h_0(x)$, we can find its inverse $f(x)$ in $\widetilde{O}(t \log(q))$ time using Lemma 17.

Defining more polynomials. Once, we know m, we define m many polynomials f_j, for $j \in [m]$, as follows.

$$f_j(x) := \prod_{i=1}^{n} \left(\frac{1}{1-\mu^j x^{a_j}}\right) = \left(\prod_{i=1}^{n} \left(1-\mu^j x^{a_i}\right)\right)^{-1} =: (h_j(x))^{-1}$$

Let us assume that there are ℓ many *distinct* hamming weights $w_1, \ldots, w_\ell$ where $\ell \leq m$. Moreover, assume that there are λ_i many solutions which appear with hamming weight w_i, for $i \in [\ell]$. Thus, $\sum_{i \in [\ell]} \lambda_i = m \leq k$. It is not hard to observe that $\text{coef}_{x^t}(f_j(x)) = \sum_{i \in [\ell]} \lambda_i \cdot \mu^{jw_i}$. To find the coefficients of $f_j(x)$, we first compute all the coefficients of $h_j(x)$ in $\widetilde{O}(k(n+t))$ time by following the same steps used in Theorem 9 (refer to the end of Section III). Then, we can find the inverse of $h_j(x)$ using Lemma 17, which can again be done in $\widetilde{O}(n+t)$ time. Since, $j \in [m]$, the total time to compute all $f_j(x)$ is $\widetilde{O}(k(n+t))$. Once we have computed the coefficients of $f_j(x)$, we need to extract $w_1, \ldots, w_\ell$.

Extracting the weights $w_1, \ldots, w_\ell$. Let $C_j = \text{coef}_{x^t}(f_j(x))$. Using Lemma 8, we can find $C_j \bmod q$ for each $j \in [m]$ in $\tilde{O}((n+t\log(\mu j)))$ 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 14), we have the following relations, for $j \in [m]$:

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

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. By the choice of q, we have $q > k$, therefore, $j^{-1} \bmod q$ exists, and thus the above relations are valid. From definition of P_j and C_j, we have

$$P_j\left(\mu^{w_1}, \ldots, \mu^{w_\ell}\right) = \sum_{i=1}^{\ell} \lambda_j \mu^{w_j} \equiv C_j \mod q$$

for all $j \in [m]$. 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 $T(m) \leq \tilde{O}(k^2) + \tilde{O}(k(n+t))$. But one can do better than $\tilde{O}(k^2)$ and make it $\tilde{O}(k)$ (i.e solve the recurrence, using FFT), owing the total complexity to $T(m) \leq \tilde{O}(k(n+t))$ (since $q = O(n+k+t)$). For details, see Appendix A.

We define a new polynomial $g(x)$ using E_j, for $j \in [m]$ that were computed in the above.

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

Using Lemma 15, it is immediate that $g(x) = \prod_{i=1}^{\ell}(x - \mu^{w_i})^{\lambda_i}$. In other words, μ^{w_i} are the roots of g. Moreover, the degree of g is m. From g, now we want to extract the

roots, namely $\mu^{w_1}, \ldots, \mu^{w_\ell}$ over $\mathbb{F}_q$ by checking whether $(x - \mu^i)$ divides g, for $i \in [n]$ (since $w_i \leq n$). Using Lemma 16, a single division with remainder takes $\tilde{O}(k)$, therefore, the total time to find all the w_i is $n \times \tilde{O}(k) = \tilde{O}(nk)$.

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 8 *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 14 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 $\text{ord}_q(\mu) > n$. 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 $\text{ord}_q(\mu) = q - 1 > n$ and thus we will find w_i from μ^{w_i} correctly.

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

□

Proof of Theorem 6. The algorithm first reduces UBSSUM to SSUM using Theorem 2 which preserves the number of solutions but the size of the SSUM instance is now $n \log t$. Then, it runs Theorem 10 on the SSUM instance to find all its solutions. From the solutions of the SSUM instance, it constructs all the solutions of the UBSSUM instance. This gives $\text{poly}(knt)$-time and $O(\log(knt \log t)) = O(\log(knt))$-space algorithm. □

ACKNOWLEDGMENT

The authors would like to thank Dr. Purushottam Kar (Dept. of CSE, IIT Kanpur) for introducing them to PSSUM. First author is supported by the project "Foundation of Lattice-based Cryptography", funded by NUS-NCS Joint Laboratory for Cyber Security.

REFERENCES

[1] H. R. Lewis, "Computers and intractability. a guide to the theory of NP-completeness," 1983.
[2] R. E. Bellman, "Dynamic programming," 1957.
[3] D. S. Johnson, "The np-completeness column: an ongoing guide," *Journal of Algorithms*, vol. 6, no. 3, pp. 434–451, 1985.
[4] V. Lyubashevsky, A. Palacio, and G. Segev, "Public-key cryptographic primitives provably as secure as subset sum," in *TCC 2010*, 2010, pp. 382–400.
[5] S. Faust, D. Masny, and D. Venturi, "Chosen-ciphertext security from subset sum," in *Public-Key Cryptography–PKC 2016*. Springer, 2016, pp. 35–46.
[6] C. Jin and H. Wu, "A simple near-linear pseudopolynomial time randomized algorithm for subset sum," *SOSA 2019*, 2019.
[7] C. Jin, N. Vyas, and R. Williams, "Fast low-space algorithms for subset sum," in *SODA 2021*, 2021, pp. 1757–1776.
[8] A. Esser and A. May, "Low Weight Discrete Logarithm and Subset Sum in $2^{0.65n}$ with Polynomial Memory," *memory*, vol. 1, p. 2, 2020.
[9] D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer, "Quantum algorithms for the subset-sum problem," in *IWPQC 2013*, 2013, pp. 16–33.
[10] L. Tran-Thanh, A. Chapman, E. M. De Cote, A. Rogers, and N. R. Jennings, "Epsilon–first policies for budget–limited multi-armed bandits," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010, pp. 1211–1216.
[11] L. Tran-Thanh, A. Chapman, A. Rogers, and N. Jennings, "Knapsack based optimal policies for budget–limited multi–armed bandits," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 1134–1140.
[12] W. Ding, T. Qin, X.-D. Zhang, and T.-Y. Liu, "Multi-armed bandit with budget constraint and variable costs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, no. 1, 2013, pp. 232–238.
[13] L. Tran-Thanh, Y. Xia, T. Qin, and N. R. Jennings, "Efficient algorithms with performance guarantees for the stochastic multiple-choice knapsack problem," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 403–409.
[14] G. S. Lueker, "Exponentially small bounds on the expected optimum of the partition and subset sum problems," *Random Structures & Algorithms*, vol. 12, no. 1, pp. 51–62, 1998.
[15] A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. Papailiopoulos, "Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2599–2610, 2020.
[16] K. Koiliaris and C. Xu, "Faster pseudopolynomial time algorithms for subset sum," *ACM Transactions on Algorithms (TALG)*, vol. 15, no. 3, pp. 1–20, 2019.
[17] Koiliaris, Konstantinos and Xu, Chao, "Subset sum made simple," *arXiv preprint arXiv:1807.08248*, 2018.
[18] K. Bringmann, "A near-linear pseudopolynomial time algorithm for subset sum," in *SODA 2017*, 2017, pp. 1073–1084.
[19] P. Hansen and J. Ryan, "Testing integer knapsacks for feasibility," *European journal of operational research*, vol. 88, no. 3, pp. 578–582, 1996.
[20] K.-M. Klein, "On the fine-grained complexity of the unbounded subsetsum and the frobenius problem," in *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022, pp. 3567–3582.
[21] P. Dutta and M. S. Rajasree, "Efficient reductions and algorithms for subset product," in *Conference on Algorithms and Discrete Applied Mathematics*. Springer, 2023, pp. 3–14.
[22] K. Potępa, "Faster deterministic modular subset sum," *arXiv preprint arXiv:2012.06062*, 2020.

[23] K. Axiotis, A. Backurs, K. Bringmann, C. Jin, V. Nakos, C. Tzamos, and H. Wu, "Fast and simple modular subset sum," in *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 2021, pp. 57–67.
[24] C. Jin and Y. Xu, "Removing additive structure in 3sum-based reductions," *arXiv preprint arXiv:2211.07048*, 2022.
[25] Pranjal Dutta and Mahesh Sreekumar Rajasree, "Algebraic Algorithms for Variants of Subset Sum," in *Conference on Algorithms and Discrete Applied Mathematics*. Springer, 2022, pp. 237–251.
[26] R. P. Brent, "Multiple-precision zero-finding methods and the complexity of elementary function evaluation," in *Analytic computational complexity*. Elsevier, 1976, pp. 151–176.
[27] D. M. Kane, "Unary subset-sum is in logspace," *arXiv preprint arXiv:1012.1336*, 2010.
[28] J. Von Zur Gathen and J. Gerhard, *Modern computer algebra*. Cambridge university press, 2013.
[29] J. Nagura, "On the interval containing at least one prime number," *Proceedings of the Japan Academy*, vol. 28, no. 4, pp. 177–181, 1952.
[30] I. Shparlinski, "On finding primitive roots in finite fields," *Theoretical computer science*, vol. 157, no. 2, pp. 273–275, 1996.
[31] J. Blömer and S. Naewe, "Solving the closest vector problem with respect to l_p norms," *arXiv preprint arXiv:1104.3720*, 2011.
[32] R. Kannan, "Minkowski's convex body theorem and integer programming," *Mathematics of operations research*, vol. 12, no. 3, pp. 415–440, 1987.
[33] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in p," *Annals of mathematics*, pp. 781–793, 2004.

APPENDIX

A. Solving linear recurrence: Tool for Section V

In this section, we present a brief outline of how to speed up the computation of E_i, for $i \in [m]$, in Section V using FFT, instead of a sequential approach. Equation (1) gives the following relation:

$$E_j \equiv j^{-1} \cdot \left(\sum_{i \in [j]} (-1)^{j-i-1} E_i \cdot P_{j-i} \right) \mod q \,.$$

Here, by E_j (respectively P_j) respresents $E_j(\mu^{w_1}, \ldots, \mu^{w_\ell})$ (respectively P_j). We can assume that P_j's have been pre-computed, contributing to the complexity only once. This calculation is very similar to [6, Lemma 2], with a similar relation. But we give the details, for the completeness.

Once we have computed P_j's, we can utilize FFT (Algorithm 1) to compute the values E_j's, which gives $T(m) \leq \widetilde{O}(k(n+t))$.

To elaborate, in the for-loop 7-8 in Algorithm 1, we aim to compute $\sum_{i=\ell}^{s} (-1)^{j-i} E_i \cdot P_{j-i}$ for all $j \in \{s+1, \ldots, u\}$. To achieve this, we define the polynomials:

$$F(x) := \sum_{k=0}^{u-\ell} (-1)^{k-1} P_k x^k, \text{ and } G(x) := \sum_{j=0}^{s-\ell} E_{j+\ell} x^j \,.$$

It is important to note that our $F(x)$ differs from the one used in [6], because of slightly different recurrence relation. We can compute $H(x) = F(x) \cdot G(x)$, in time $\widetilde{O}((u-\ell))$. Observe that $\sum_{i=\ell}^{u} (-1)^{j-i-1} P_{j-i} \cdot E_i = \text{coef}_{x^{j-\ell}}(H(x))$ because $(-1)^{j-i-1} P_{j-i} = \text{coef}_{x^{j-i}}(F(x))$ and $E_i = \text{coef}_{x^{i-\ell}}(G(x))$. Therefore, the inner for loop can be computed in $\widetilde{O}((u-\ell))$ time.

a) Final time complexity.: Let $T'(m)$ is the complexity of computing $E_1, \ldots, E_m$ assuming precomputations of P_j and j^{-1}. Then,

$$T'(m) \leq 2T'(m/2) + \widetilde{O}(m) \implies T'(m) \leq \widetilde{O}(m) \,.$$

Therefore, the total complexity of computing $E_1, \ldots, E_m$, is $T(m) = T'(m) + \widetilde{O}(k(n+t))$, where $\widetilde{O}(k(n+t))$ is for the time for computing P_j's (and j^{-1}). Since, $q = O(n+k+t)$ and $m \leq k$, we get $T(m) = \widetilde{O}(k(n+t))$, as we wanted.

Algorithm 1: Algorithm for computing E_i

Input: P_i, for $i \in [m]$, q and $E_0 = 1$
Output: E_i for $i \in [m]$
1 Initialize $E_j \leftarrow 0$, for $j \in [m]$;
2 **return** Compute$(0, m)$;
3 **Procedure** Compute(ℓ, u) ▷ the values returned by Compute(ℓ, u) are the final values $E_\ell, \ldots, E_u$ are computed;
4 **for** $\ell < u$ **do**
5 $\quad s \leftarrow \lfloor \frac{\ell+u}{2} \rfloor$
6 $\quad$ Compute(ℓ, s);
7 $\quad$ **for** $j \leftarrow s+1, \ldots, u$ **do**
8 $\quad\quad E_j \leftarrow E_j + j^{-1} \cdot \left(\sum_{i=\ell}^{s} (-1)^{j-i} E_i \cdot P_{j-i}\right) \mod q$;
9 $\quad$ Compute$(s+1, u)$
10 **return** $E_\ell, \ldots, E_u$;