


Improved Space Bounds for Subset Sum

Tatiana Belova ✉

St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia

Nikolai Chukhin ✉

Neapolis University Pafos, Paphos, Cyprus
JetBrains Research, Paphos, Cyprus

Alexander S. Kulikov ✉ 

JetBrains Research, Paphos, Cyprus

Ivan Mihajlin ✉

JetBrains Research, Paphos, Cyprus

Abstract

More than 40 years ago, Schroeppel and Shamir presented an algorithm that solves the Subset Sum problem for n integers in time $O^*(2^{0.5n})$ and space $O^*(2^{0.25n})$. The time upper bound remains unbeaten, but the space upper bound has been improved to $O^*(2^{0.249999n})$ in a recent breakthrough paper by Nederlof and Węgrzycki (STOC 2021). Their algorithm is a clever combination of a number of previously known techniques with a new reduction and a new algorithm for the Orthogonal Vectors problem.

In this paper, we give two new algorithms for Subset Sum. We start by presenting an Arthur–Merlin algorithm: upon receiving the verifier’s randomness, the prover sends an $n/4$ -bit long proof to the verifier who checks it in (deterministic) time and space $O^*(2^{n/4})$. An interesting consequence of this result is the following fine-grained lower bound: assuming that 4-SUM cannot be solved in time $O(n^{2-\varepsilon})$ for all $\varepsilon > 0$, Circuit SAT cannot be solved in time $O(g2^{(1-\varepsilon)n})$, for all $\varepsilon > 0$ (where n and g denote the number of inputs and the number of gates, respectively).

Then, we improve the space bound by Nederlof and Węgrzycki to $O^*(2^{0.246n})$ and also simplify their algorithm and its analysis. We achieve this space bound by further filtering sets of subsets using a random prime number. This allows us to reduce an instance of Subset Sum to a larger number of instances of smaller size.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases algorithms, subset sum, complexity, space, upper bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.21

Related Version *Full Version:* <https://arxiv.org/abs/2402.13170> [3]

Funding Research is partially supported by the grant 075-15-2022-289 for creation and development of Euler International Mathematical Institute, and by the Foundation for the Advancement of Theoretical Physics and Mathematics “BASIS”.

1 Overview

In this paper, we study the well-known Subset Sum problem and its parameterized version, k -SUM. In Subset Sum, given a set of n integers and a target integer t the goal is to check whether there is a subset of them that sum up to t . It is common to assume that the absolute value of all input integers is at most $2^{O(n)}$ (one can achieve this using hashing techniques).

In the k -SUM problem, the goal is to check whether some k of the input integers sum to zero. We consider a slightly different formulation of k -SUM: given k sequences A_1, \dots, A_k , each containing n integers of absolute value at most $n^{O(1)}$, check whether there exist indices



© Tatiana Belova, Nikolai Chukhin, Alexander S. Kulikov, and Ivan Mihajlin; licensed under Creative Commons License CC-BY 4.0

32nd Annual European Symposium on Algorithms (ESA 2024).

Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 21; pp. 21:1–21:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

i_1, \dots, i_k such that¹ $A_1[i_1] + \dots + A_k[i_k] = 0$. It is common to further assume that the bit-length of all integers in the k -SUM problem is at most $k \log n + O(\log \log n)$. This can be achieved by the standard fingerprinting technique.

1.1 Known Results

1.1.1 Time Complexity

2-SUM can be solved in time $\tilde{O}(n)$ using binary search (the $\tilde{O}(\cdot)$ notation hides polylogarithmic factors): sort A_1 ; then, for each $1 \leq i_2 \leq n$, check whether A_1 contains an element $-A_2[i_2]$. Another well known algorithm is based on the two pointers method that proceeds as follows. Sort A_1 and A_2 and let $i_1 = 1$ and $i_2 = n$. Then, keep repeating the following: if $i_1 > n$ or $i_2 < 1$, stop; if $A_1[i_1] + A_2[i_2] = 0$, return (i_1, i_2) ; if $A_1[i_1] + A_2[i_2] > 0$, decrement i_2 (as $A_2[i_2]$ is too large: its sum with the currently smallest element of A_1 is positive); if $A_1[i_1] + A_2[i_2] < 0$, increment i_1 (as $A_1[i_1]$ is too small: its sum with the currently largest element of A_2 is negative).

The algorithm for 2-SUM allows k -SUM to be solved in time $\tilde{O}(n^{\lceil k/2 \rceil})$ via the following reduction. Given k arrays A_1, \dots, A_k , split them into two halves: $A_1, \dots, A_{\lceil k/2 \rceil}$ and $A_{\lceil k/2 \rceil}, \dots, A_k$. Populate arrays B_1 and B_2 with all sums of elements from the first and second halves, respectively. Then, it remains to solve 2-SUM for B_1 and B_2 .

For 3-SUM, Chan [5] proved a slightly better upper bound $O(n^2 \log \log^{O(1)} n / \log^2 n)$. It is a major open problem whether 3-SUM can be solved in time $O(n^{2-\varepsilon})$ and the 3-SUM hypothesis, stating that this is impossible, is a popular hypothesis in the field of fine-grained complexity. It is also currently unknown whether k -SUM can be solved in time $\tilde{O}(n^{\lceil k/2 \rceil - \varepsilon})$ for any $k \geq 3$ and $\varepsilon > 0$.

The 2-SUM algorithm described above is also at the core of the strongest known upper bound $O^*(2^{n/2})$ for Subset Sum ($O^*(\cdot)$ hides polynomial factors) presented by Horowitz and Sahni [7] fifty years ago. To get this running time, partition A into two halves of size $n/2$; then, populate arrays A_1 and A_2 (of size $2^{n/2}$) with sums of all subsets of the two halves, respectively; then, it remains to solve 2-SUM for A_1 and A_2 . It is an important open problem to solve Subset Sum in $O^*(2^{(1/2-\varepsilon)n})$ for a constant $\varepsilon > 0$. The fastest known algorithm shows that Subset Sum can be solved in time $2^{n/2} / \text{poly}(n)$ [6].

1.1.2 Space Complexity

The algorithms that solve k -SUM and Subset Sum via a reduction to 2-SUM have high space complexity: for k -SUM, it is $O(n^{\lceil k/2 \rceil})$, whereas for Subset Sum it is $O(2^{n/2})$. It is natural to ask whether one can lower the space complexity or whether it is possible to trade off time and space. Lincoln et al. [11] gave a positive answer for k -SUM: one can solve 3-SUM in time $O(n^2)$ and space $O(\sqrt{n})$ as well as k -SUM (for $k \geq 4$) in time $O(n^{k-2+2/k})$ and space $O(\sqrt{n})$. For Subset Sum, the well-known algorithm by Schroepel and Shamir [15] solves it in time $O^*(2^{n/2})$ and space $O(2^{n/4})$: they reduce Subset Sum to 4-SUM and note that 4-SUM can be solved in time $\tilde{O}(n^2)$ and space $O(n)$ (since all pairwise sums of two sorted sequences of length n can be enumerated in time $\tilde{O}(n^2)$ and space $O(n)$ using a priority queue data structure). Just recently, Nederlof and Wegrzycki improved the space complexity to $O(2^{0.249999n})$ [13].

¹ These two versions of k -SUM are reducible to each other. One direction is easy: take k copies of the array A . For the other direction, one can use the color coding technique [2]: for every element A , add it to A_i for random $i \in [k]$; then, if a solution for the original problem exists, it survives in the resulting instance with constant probability. This randomized reduction can also be derandomized [2].

1.1.3 Proof Complexity

It is easy to certify a yes-instance of k -SUM or Subset Sum: just write down a solution; it can be checked in deterministic time $O(n)$. Certifying no-instances is trickier. Carmosino et al. [4] showed that, for 3-SUM, there are proofs of size $\tilde{O}(n^{1.5})$ that can be deterministically verified in time $\tilde{O}(n^{1.5})$. By allowing the verifier to be probabilistic, Akmal et al. [1] presented a proof system for 3-SUM where the proof's size and the verification time is $\tilde{O}(n)$. For k -SUM, they give an upper bound of $\tilde{O}(n^{k/3})$. The corresponding proof system is known as a Merlin–Arthur protocol: a computationally unbounded prover (Merlin) prepares a proof and sends it to a probabilistic verifier (Arthur) who needs to be able to check the proof quickly with small constant error probability. We say that a problem can be solved in Merlin–Arthur time $T(n)$, if there exists a protocol where both the proof size and the verification time do not exceed $T(n)$.

For Subset Sum, Nederlof [12] proved an upper bound $O^*(2^{(1/2-\varepsilon)n})$, for some $\varepsilon > 10^{-3}$, on Merlin–Arthur time. Akmal et al. [1] improved the bound to $O^*(2^{n/3})$.

1.2 New Results

Below, we give an overview of the main results of the paper. Their proofs are given later in the text. The proofs of some of the technical lemmas are omitted due to the page limit, the corresponding lemmas are stated with a reference to the full version [3] of the paper that contains all the proofs.

1.2.1 New Arthur–Merlin Algorithm

Recall that in an Arthur–Merlin protocol the randomness is shared (also known as public coins) and the verifier is deterministic: upon receiving the verifier's randomness, the prover prepares a proof and sends it to the verifier who checks it deterministically with small error probability (taken over public randomness). Formally, we say that a language $L \subseteq \{0, 1\}^*$ belongs to a class $\text{AM}[s(n), t(n)]$, if there exists an Arthur–Merlin protocol such that, for any $x \in \{0, 1\}^n$, the proof length is at most $s(n)$, the verification time is at most $t(n)$, and for each $x \in L$, the verifier accepts with probability at least $2/3$, whereas for each $x \notin L$, the verifier rejects with probability $1/2$.² Such a protocol implies that L can be solved by a randomized algorithm that has running time $t(n)$ and error probability at least $2^{-s(n)}$. It is also easily parallelizable: using $k \leq 2^{s(n)}$ processors, the problem can be solved in time $\frac{2^{s(n)}t(n)}{k}$.

Our first main result is a new Arthur–Merlin algorithm for Subset Sum.

► **Theorem 1.** *Subset Sum* $\in \text{AM}[n/4, O^*(2^{n/4})]$.

As discussed above, it can be parallelized easily: to solve the problem, one can enumerate possible proofs in parallel. Also, by enumerating all possible proofs, one recovers upper bounds on time and space for Subset Sum proved by Schroepel and Shamir in 1979. Interestingly, the resulting algorithm is very simple and does not need to use the priority queue data structure as in the algorithm by Schroepel and Shamir.

As it is the case with the previously known algorithms for Subset Sum, our algorithm follows from an algorithm for 4-SUM.

² In the standard definition of AM protocols, there is either a two-sided error probability or zero error probability of acceptance. We choose to have a zero error probability of rejection as it allows for straightforward transformation to randomized and parallel algorithms.

21:4 Improved Space Bounds for Subset Sum

► **Theorem 2.** $4\text{-SUM} \in \text{AM} \left[\log_2 n, \tilde{O}(n) \right]$.

Since $2k\text{-SUM}$ can be reduced to 4-SUM , this extends to $2k\text{-SUM}$ as follows.

► **Corollary 3.** *For every even integer k , $2k\text{-SUM} \in \text{AM} \left[\frac{k}{2} \log_2 n, \tilde{O}(n^{k/2}) \right]$.*

The main idea of the proof of Theorem 2 is the following. To find integers i_1, i_2, i_3, i_4 such that $A_1[i_1] + A_2[i_2] + A_3[i_3] + A_4[i_4] = 0$, one generates a random prime $p \leq n$ and uses $(A_1[i_1] + A_2[i_2]) \bmod p$ as a proof. A similar idea of using a random prime for filtering subsets was used previously by Howgrave-Graham and Joux [8].

1.2.2 New Conditional Lower Bounds for Circuit SAT

Another interesting consequence of Theorem 2 is a fine-grained lower bound for the Circuit SAT problem. Circuit SAT is a generalization of SAT where instead of a formula in CNF, one is given an arbitrary Boolean circuit with g binary gates and n inputs (and the goal, as usual, is to check whether it is satisfiable). Being a generalization of SAT, Circuit SAT cannot be solved in time $O^*(2^{(1-\varepsilon)n})$, for any constant $\varepsilon > 0$, under the Strong Exponential Time Hypothesis (that states that for every $\varepsilon > 0$, there exists k such that $k\text{-SAT}$ cannot be solved in time $O(2^{(1-\varepsilon)n})$). Even designing a $2^{(1-\varepsilon)n}$ time algorithm solving Circuit SAT for circuits with $g \leq 8n$ gates is challenging: as shown by [9], this would imply new circuit lower bounds.

We prove the following conditional lower bound for Circuit SAT.

► **Theorem 4.** *If, for any $\varepsilon > 0$, 4-SUM cannot be solved in time $O(n^{2-\varepsilon})$, then, for any $\varepsilon > 0$, Circuit SAT cannot be solved in time $O(g2^{(1-\varepsilon)n})$.*

This theorem however does not exclude a $O(g^{O(1)}2^{(1-\varepsilon)n})$ -time algorithm for Circuit-SAT in the same manner. To get such a fine-grained lower bound, one needs to push the algorithm from Corollary 3 further (i.e., to trade most of the n^k time upper bound for nondeterminism).

► **Open Problem 1.** *Prove or disprove: $2k\text{-SUM} \in \text{AM} [k \log_2 n, n^{o(k)}]$.*

► **Corollary 5.** *Assume that $2k\text{-SUM} \in \text{AM} [k \log_2 n, n^{o(k)}]$. Then, if for any $\varepsilon > 0$ and any $k \geq 1$, $2k\text{-SUM}$ cannot be solved in time $O(n^{k-\varepsilon})$, then, for any $\varepsilon > 0$, Circuit SAT cannot be solved in time $O(g^{O(1)}2^{(1-\varepsilon)n})$.*

1.2.3 Improved Space Upper Bound for Subset Sum

Our second main result is an improved space upper bound for Subset Sum. We achieve this by improving and simplifying the algorithm by Nederlof and Węgrzycki.

► **Theorem 6.** *There exists a Monte Carlo algorithm with constant success probability that solves Subset Sum for instances with n integers of absolute value at most $2^{O(n)}$ in time $O^*(2^{0.5n})$ and space $O^*(2^{0.246n})$.*

The algorithm by Nederlof and Węgrzycki proceeds roughly as follows. Let $I \subseteq \mathbb{Z}$, $|I| = n$, be an instance of Subset Sum. Assume that it is a yes-instance and that a solution $S \subseteq I$ has size $n/2$. Assume further that there exists a subset $M \subseteq I$ of size $\Theta(n)$ such that $|S \cap M| = |M|/2$ and M is a perfect mixer: the weights of all subsets of M are (pairwise) distinct. Partition $I \setminus M$ into two parts L and R of equal size, hence, $I = L \sqcup M \sqcup R$. We will be looking for two disjoint sets $S_1 \subseteq L \sqcup M$ and $S_2 \subseteq M \sqcup R$ such

that $|S_1 \cap M| = |S_2 \cap M| = |M|/4$ and $S_1 \sqcup S_2$ is a solution. This problem can be solved by a reduction to the Weighted Orthogonal Vectors (WOV) problem that can be viewed as a hybrid of the Orthogonal Vectors and the 2-SUM problems: given two families \mathcal{L} and \mathcal{R} of weighted sets and a target integer t , find two disjoint sets whose sum of weights is equal to t . A naive such reduction would proceed similarly to the algorithm by Horowitz and Sahni [7]: let

$$\begin{aligned}\mathcal{L} &= \{(S_1 \cap M, w(S_1)) : S_1 \subseteq L \sqcup M, |S_1 \cap M| = |M|/4\}, \\ \mathcal{R} &= \{(S_2 \cap M, w(S_2)) : S_2 \subseteq M \sqcup R, |S_2 \cap M| = |M|/4\}.\end{aligned}$$

Then, it remains to solve WOV for \mathcal{L} , \mathcal{R} and t . The bad news is that $|L \sqcup M| > n/2$, hence enumerating all $S_1 \subseteq L \sqcup M$ leads to a running time worse than $O^*(2^{n/2})$. The good news is that the solution S is represented by exponentially many pairs (S_1, S_2) : there are many ways to distribute $S \cap M$ between S_1 and S_2 . This allows one to use the representation technique: take a random prime p of the order $2^{|M|/2}$ and a random remainder $r \in \mathbb{Z}_p$; filter \mathcal{L} and \mathcal{R} to leave only subsets S_1 and S_2 such that $w(S_1) \equiv_p r$ and $w(S_2) \equiv_p t - r$. This way, one reduces the space complexity since \mathcal{L} and \mathcal{R} become about p times smaller.

In this work, we further introduce the following idea to reduce the space complexity of the algorithm and to simplify it at the same time. We use another prime q to further filter \mathcal{L} and \mathcal{R} : this allows us to reduce the original instance to a larger number of instances that in turn have smaller size. The idea is best illustrated by the following toy example. Consider an instance I of size n of Subset Sum. As in the algorithm by Horowitz and Sahni [7], partition I arbitrarily into two parts L and R of size $n/2$. Assume now that we are given a magic integer $q = \Theta(2^{n/4})$ that hashes all subsets of L and R almost perfectly: for each $r \in \mathbb{Z}_q$,

$$|\{A \subseteq L : w(A) \equiv_q r\}|, |\{B \subseteq R : w(B) \equiv_q t - r\}| = O(2^{n/4}).$$

This allows us to solve I in time $O^*(2^{n/2})$ and space $O^*(2^{n/4})$ without using a reduction to 4-SUM and priority queues machinery of Schroepel and Shamir's algorithm: for each r , construct sequences of all subsets of L and R having weights r and $t - r$ modulo q , respectively, and solve 2-SUM for the two resulting sequences of length $O(2^{n/4})$. Thus, instead of reducing I to a single instance of 2-SUM of size $2^{n/2}$, we reduce it to $2^{n/4}$ instances of size $2^{n/4}$. We show that even though in general a randomly chosen integer q does not give a uniform distribution of remainders, one can still use this idea to reduce the space.

On a technical side, we choose a mixer M more carefully. This allows us to cover various corner cases, when the size of the solution S is not $n/2$ and when the mixer M is far from being perfect, in the same manner, thus simplifying the algorithm by Nederlof and Węgrzycki.

2 General Setting

2.1 Modular Arithmetic

We write $a \equiv_m b$ to denote that integers a and b have the same remainder modulo m . By \mathbb{Z}_m we denote the ring of remainders modulo m . We make use of the following well known fact.

► **Theorem 7** (Chinese Remainder Theorem). *For any coprime positive integers p and q and any $a \in \mathbb{Z}_p$ and $b \in \mathbb{Z}_q$, there exists a unique $x \in \mathbb{Z}_{pq}$ such that $x \equiv_p a$ and $x \equiv_q b$. Furthermore, it can be computed (using the extended Euclidean algorithm) in time $O(\log^2 p + \log^2 q)$.*

2.2 Prime Numbers

Given an integer t , one can generate a uniform random prime from $[t, 2t]$ in time $O(\log^{O(1)} t)$: to do this, one selects a random integer from $[t, 2t]$ and checks whether it is prime [10]; the expected number of probes is $O(\log t)$ due to the law of distribution of prime numbers. We also use the following estimate: any positive integer k has at most $\log_2 k$ distinct prime divisors; hence the probability that a random prime from $[t, 2t]$ divides k is at most $O(\frac{\log k \log t}{t})$.

2.3 Probability Amplification

We use the following standard success probability amplification trick: if a “good” event A occurs with probability at least p , then for A to occur at least once with constant probability, $1/p$ independent repetitions are sufficient: $(1 - p)^{1/p} \leq (e^{-p})^{1/p} = e^{-1}$.

2.4 Growth Rate and Entropy

Much like $O(\cdot)$ hides constant factors, $O^*(\cdot)$ and $\tilde{O}(\cdot)$ hide factors that grow polynomially in the input size and polylogarithmically, respectively: for example, $n^2 \cdot 1.5^n = O^*(1.5^n)$ and $\log^3 n \cdot n^2 = \tilde{O}(n^2)$. $\Omega^*(\cdot)$, $\Theta^*(\cdot)$, $\tilde{\Omega}(\cdot)$, and $\tilde{\Theta}(\cdot)$ are defined similarly.

We use the following estimates for binomial coefficients: for any constant $0 \leq \alpha \leq 1$,

$$\Omega(n^{-1/2})2^{h(\alpha)n} \leq \binom{n}{\alpha n} \leq 2^{h(\alpha)n} \text{ and hence } \binom{n}{\alpha n} = \Theta^*(2^{h(\alpha)n}), \quad (1)$$

where $h(x) = -x \log_2 x - (1 - x) \log_2(1 - x)$ is the binary entropy function. The function h is concave: for any $0 \leq \beta \leq 1$ and any $0 \leq x, y \leq 1$, $\beta h(x) + (1 - \beta)h(y) \leq h(\beta x + (1 - \beta)y)$.

2.5 Sets and Sums

For a positive integer n , $[n] = \{1, 2, \dots, n\}$. For disjoint sets A and B , by $A \sqcup B$ we denote their union.

Let $S \subseteq \mathbb{Z}$ be a finite set of integers. By $w(S)$ we denote the weight of S : $w(S) = \sum_{a \in S} a$. By 2^S we denote the power set of S , i.e., the set of its subsets: $2^S = \{A : A \subseteq S\}$. By $\binom{S}{k}$ we denote the set of all subsets of S of size k : $\binom{S}{k} = \{A \subseteq S : |A| = k\}$. For a family of sets $\mathcal{F} \subseteq 2^S$, by $w(\mathcal{F})$ we denote the set of their weights: $w(\mathcal{F}) = \{w(A) : A \in \mathcal{F}\}$.

Given S , one can compute $|w(2^S)|$ in time and space $O^*(2^{|S|})$ by iterating over all of its subsets. If $|w(2^S)| = 2^{(1-\varepsilon)|S|}$, we call S an ε -mixer.

► **Lemma 8** ([3]). *Let $S \subseteq A$ and $|A| = n$. For $m = O(1)$ uniformly randomly chosen disjoint non-empty sets $A_1, \dots, A_m \subseteq A$,*

$$\Pr \left[\frac{|A_i \cap S|}{|A_i|} \approx \frac{|S|}{|A|}, \text{ for all } i \in [m] \right] = \Omega^*(1),$$

where $\frac{|A_i \cap S|}{|A_i|} \approx \frac{|S|}{|A|}$ means that $\left| |A_i \cap S| - |A_i| \frac{|S|}{|A|} \right| = O(1)$.

2.6 Weighted Orthogonal Vectors

► **Definition 9** (Weighted Orthogonal Vectors, WOV). *Given families of N weighted sets $\mathcal{A}, \mathcal{B} \subseteq 2^{[d]} \times \mathbb{N}$, and a target integer t , the problem is to find $(A, w_A) \in \mathcal{A}$ and $(B, w_B) \in \mathcal{B}$ such that $A \cap B = \emptyset$ and $w_A + w_B = t$.*

► **Lemma 10** ([3]). *For any $\sigma \in [0, \frac{1}{2}]$, there is a Monte-Carlo algorithm that, given $\mathcal{A} \subseteq \binom{[d]}{\sigma d} \times \mathbb{N}$, $\mathcal{B} \subseteq \binom{[d]}{(1/2-\sigma)d} \times \mathbb{N}$ and a target integer t , solves $\text{WOV}(\mathcal{A}, \mathcal{B})$ in time $\tilde{O}((|\mathcal{A}| + |\mathcal{B}|)2^{d(1-h(1/4))})$ and space $\tilde{O}(|\mathcal{A}| + |\mathcal{B}| + 2^d)$.*

To prove the lemma, we utilize the algorithm from [13], but with a different parameter range.

3 New Arthur–Merlin Algorithm

► **Theorem 2.** $4\text{-SUM} \in \text{AM} \left[\log_2 n, \tilde{O}(n) \right]$.

Proof. Given arrays A_1, A_2, A_3, A_4 each consisting of n $4 \log n$ -bit integers, our goal is to find four indices $i_1, i_2, i_3, i_4 \in [n]$ such that $A_1[i_1] + A_2[i_2] + A_3[i_3] + A_4[i_4] = 0$. Without loss of generality, assume that A_i 's are sorted and do not contain duplicates: for every $i \in [4]$, $A_i[1] < A_i[2] < \dots < A_i[n]$. Assume that $i_1^*, i_2^*, i_3^*, i_4^*$ is a solution and let $s = A_1[i_1^*] + A_2[i_2^*]$.

Let \mathcal{P} be a randomized algorithm that, given an integer n , returns a uniform random prime from $[2..n]$. Since one can check whether a given integer k is prime or not in time $O(\log^{O(1)} k)$, the expected running time of \mathcal{P} is $O(\log^{O(1)} n)$: one picks a random $k \in [2..n]$ and checks whether it is prime; if it is not, one repeats; the expected number of probes is $O(\log n)$.

We are ready to describe the protocol.

Stage 0: Shared randomness. Using the shared randomness, Arthur and Merlin generate a random prime $p = \mathcal{P}(n)$.

Stage 1: Preparing a proof. Merlin sends an integer $0 \leq r < p$ to Arthur.

Stage 2: Verifying the proof. Arthur takes all elements of A_1 and A_2 modulo p and sorts both arrays. Using the two pointers method (or binary search), he enumerates all pairs $(i_1, i_2) \in [n]^2$ such that $(A_1[i_1] + A_2[i_2]) \equiv r \pmod{p}$. For each such pair (i_1, i_2) , he adds $A_1[i_1] + A_2[i_2]$ to a new array A_{12} . If the size of A_{12} becomes larger than $O(n \log^3 n)$, Arthur rejects immediately. Then, he does the same for A_3 and A_4 , but for the remainder $-r$ (rather than r) and creates an array A_{34} . Finally, he solves 2-SUM for A_{12} and A_{34} (the two arrays may have different lengths, but this can be fixed easily by padding one of them with dummy elements).

Now, we analyze the protocol. It is clear that the proof size is at most $\log_2 n$ and that the running time of the verification stage is $\tilde{O}(n)$. Also, if there is no solution for the original instance, Arthur rejects with probability 1, as he only accepts if the solution is found. Below, we show that if Merlin sends $r = s \pmod{p}$ (recall that $s = A_1[i_1^*] + A_2[i_2^*]$), Arthur accepts with good enough probability.

Consider the set $S = \{(i_1, i_2) \in [n]^2 : A_1[i_1] + A_2[i_2] \equiv s \pmod{p}\}$ (hence, $|A_{12}| = |S|$). It contains at most n true positives, that is, pairs (i_1, i_2) such that $A_1[i_1] + A_2[i_2] = s$: as the arrays do not contain duplicates, for every i_1 there is at most one matching i_2 . All the other pairs (i_1, i_2) in S are false positives: $A_1[i_1] + A_2[i_2] - s \neq 0$, but $A_1[i_1] + A_2[i_2] - s \equiv 0 \pmod{p}$. Recall that $|A_1[i_1] + A_2[i_2] - s| = O(n^4)$, hence the number of prime divisors of $A_1[i_1] + A_2[i_2] - s$ is $O(\log n)$, Section 2.2. The probability that a random prime $2 \leq p \leq n$ divides $(A_1[i_1] + A_2[i_2] - s)$ is $O(\frac{\log^2 n}{n})$. Thus, the expected number of false positives is at most $n^2 \cdot O(\frac{\log^2 n}{n}) = O(n \log^2 n)$. By Markov's inequality, the probability that the number of false positives is larger than $O(n \log^3 n)$ (and hence that $|S| = O(n \log^3 n)$) is at most $1/\log n$.

Thus, with probability at least $1 - 2/\log n$, both A_{12} and A_{34} have length at most $O(n \log^3 n)$ and by solving 2-SUM for them, Arthur finds a solution for the original yes-instance. ◀

► **Theorem 4.** *If, for any $\varepsilon > 0$, 4-SUM cannot be solved in time $O(n^{2-\varepsilon})$, then, for any $\varepsilon > 0$, Circuit SAT cannot be solved in time $O(g2^{(1-\varepsilon)n})$.*

Proof. Assume that, for some $\varepsilon > 0$, there exists an algorithm \mathcal{A} that checks whether a given Boolean circuit with g binary gates and n inputs is satisfiable in time $O(g2^{(1-\varepsilon)n})$. Using \mathcal{A} , we will solve 4-SUM in time $O(n^{2-\varepsilon})$.

Consider the verification algorithm from Theorem 2 as a Turing machine M : M has three read-only input tapes (I_R, I_P, I) : the tape I_R contains random bits, the tape I_P contains a proof, and the tape I contains four input arrays. For any four arrays (A_1, A_2, A_3, A_4) of size n , if it is a yes-instance, there exists a proof of size at most $\log_2 n$ such that M accepts with probability at least $2/3$, whereas for a no-instance, M rejects every proof with probability 1. The machine M compares $2 \log n$ -bit integers and moves two pointers through two arrays. Hence, the running time of M is $\tilde{O}(n)$.

Given (A_1, A_2, A_3, A_4) , fix the contents of the tape I and fix the random bits of the tape I_R . This turns M into a machine M' with $\log_2 n$ input bits and running time $\tilde{O}(n)$.

As proved by [14], a Turing machine recognizing a language $L \subseteq \{0, 1\}^*$ with running time $t(n)$ can be converted to an infinite series $\{C_n\}_{n=1}^\infty$ of circuits of size $O(t(n) \log t(n))$: for every n , C_n has n inputs and $O(t(n) \log t(n))$ gates and computes a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ such that $f^{-1}(1) = L \cap \{0, 1\}^n$. The circuit C_n can be produced in time $O(t(n) \log t(n))$.

Applying this to the machine M' , we get a circuit C with $\tilde{O}(n)$ gates and $\log_2 n$ input bits. If A_1, A_2, A_3, A_4 is a yes-instance of 4-SUM, the circuit C is satisfiable with probability at least $2/3$; otherwise it is unsatisfiable. Using the algorithm \mathcal{A} , we can check the satisfiability of C in time $\tilde{O}(n2^{(1-\varepsilon)\log_2 n}) = \tilde{O}(n^{2-\varepsilon})$. ◀

► **Theorem 1.** *Subset Sum \in AM $[n/4, O^*(2^{n/4})]$.*

Proof. We use the standard reduction from Subset SUM to 4-SUM. Given a sequence A of n integers, partition them into four parts of size $n/4$ and create sequences A_1, A_2, A_3, A_4 of size $2^{n/4}$ containing sums of all subsets of the corresponding parts. Use the protocol from Theorem 2 for the resulting instance of 4-SUM. ◀

The proofs of the following two corollaries can be found in the full version of the paper [3].

► **Corollary 3.** *For every even integer k , $2k$ -SUM \in AM $[\frac{k}{2} \log_2 n, \tilde{O}(n^{k/2})]$.*

► **Corollary 5.** *Assume that $2k$ -SUM \in AM $[k \log_2 n, n^{o(k)}]$. Then, if for any $\varepsilon > 0$ and any $k \geq 1$, $2k$ -SUM cannot be solved in time $O(n^{k-\varepsilon})$, then, for any $\varepsilon > 0$, Circuit SAT cannot be solved in time $O(g^{O(1)}2^{(1-\varepsilon)n})$.*

4 Improved Space Upper Bound for Subset Sum

4.1 Algorithm

Let (I, t) be an instance of Subset Sum (as usual, $n = |I|$) and assume that it is a yes-instance: we will present a randomized algorithm that is correct on no-instances with probability 1, so, as usual, the main challenge is to obtain $\Omega^*(1)$ success probability for yes-instances. Let $S \subseteq I$ be a solution: $w(S) = t$. It is unknown to us, but we may assume that we know its size: there are just n possibilities for $|S|$, so with a polynomial overhead we can enumerate all of them. Thus, assume that $|S| = \alpha n$ where $0 < \alpha \leq 1$ is known to us.

For a parameter $\beta = \beta(\alpha) < 0.15$ to be specified later, select randomly pairwise disjoint sets $M_L, M, M_R \in \binom{I}{3\beta n}$. In time and space $O^*(2^{0.15n})$, find $\varepsilon_L, \varepsilon, \varepsilon_R$ such that M_L is an ε_L -mixer, M is an ε -mixer, and M_R is ε_R -mixer. Without loss of generality, we assume that $\varepsilon \leq \varepsilon_L, \varepsilon_R$. Consider the probability that S touches exactly half of $M_L \sqcup M \sqcup M_R$:

$$\begin{aligned} \Pr[|(M_L \cup M \cup M_R) \cap S| = 3\beta n/2] &= \frac{\binom{\alpha n}{3\beta n/2} \binom{(1-\alpha)n}{3\beta n/2}}{\binom{n}{3\beta n}} = \Theta^*(2^{-\lambda n}) \\ \frac{\binom{\alpha n}{3\beta n/2} \binom{(1-\alpha)n}{3\beta n/2}}{\binom{n}{3\beta n}} = 2^{-\lambda n} &\iff \frac{1}{n} \log \frac{\binom{n}{3\beta n}}{\binom{\alpha n}{3\beta n/2} \binom{(1-\alpha)n}{3\beta n/2}} = \lambda \iff \\ &\iff \lambda = h(3\beta) - \alpha h\left(\frac{3\beta}{2\alpha}\right) - (1-\alpha)h\left(\frac{3\beta}{2(1-\alpha)}\right), \end{aligned} \quad (\text{by (1)})$$

hence,

$$\lambda = h(3\beta) - \alpha h\left(\frac{3\beta}{2\alpha}\right) - (1-\alpha)h\left(\frac{3\beta}{2(1-\alpha)}\right). \quad (2)$$

By repeating this process $2^{\lambda n}$ times, we ensure that the event $|(M_L \cup M \cup M_R) \cap S| = 3\beta n/2$ happens with probability $\Omega^*(1)$ (recall Section 2.3). Further, we may assume that

$$|M_L \cap S| = |M \cap S| = |M_R \cap S| = \beta n/2,$$

as by Lemma 8, $|M_L \cap S| = |M \cap S| = |M_R \cap S| = \beta n/2 + O(1)$ with probability $\Omega^*(1)$ conditioned on the event that S touches $3\beta n/2$ elements of $M_L \sqcup M \sqcup M_R$. An exact equality can be assumed since we can search for an exact value in the neighbourhood of $\beta n/2$ and the constant difference will not affect the memory and time complexity. In the following applications of Lemma 8, we omit $O(1)$ factors for similar considerations.

We argue that either $S \cap M$ or $M \setminus S$ is an $(\leq \varepsilon)$ -mixer. Indeed, M is an ε -mixer and hence $|w(2^M)| = 2^{(1-\varepsilon)|M|}$. Now, $|S \cap M| = |M \setminus S| = |M|/2$ and $|w(2^{S \cap M})| \cdot |w(2^{M \setminus S})| \geq |w(2^M)|$. Hence,

$$\max\{|w(2^{S \cap M})|, |w(2^{M \setminus S})|\} \geq 2^{(1-\varepsilon)|M|/2},$$

implying that either $S \cap M$ or $M \setminus S$ is indeed a $(\leq \varepsilon)$ -mixer. In the following, we assume that it is $S \cap M$ that is $(\leq \varepsilon)$ -mixer. To assume this without loss of generality, we run the final algorithm twice – for (I, t) and $(I, w(I) - t)$.

For any (finite and non-empty) set A , there exists $1 \leq k \leq |A|$, such that $|w(\binom{A}{k})| \geq |w(2^A)|/|A|$ (by the pigeonhole principle). Since $|w(\binom{A}{k})| = |w(\binom{A}{|A|-k})|$, we can even assume that $1 \leq k \leq |A|/2$. Consider the corresponding k for the set $A = S \cap M$ and let $\mu = k/(2|S \cap M|)$ (hence, $0 \leq \mu \leq 0.25$). Then,

$$\left|w\left(\binom{S \cap M}{2\mu|S \cap M|}\right)\right| \geq \frac{|w(2^{S \cap M})|}{|S \cap M|}. \quad (3)$$

Partition $I \setminus (M_L \sqcup M \sqcup M_R)$ randomly into $L_1 \sqcup L_2 \sqcup L_3 \sqcup L_4 \sqcup R_1 \sqcup R_2 \sqcup R_3 \sqcup R_4$ where the exact size of all eight parts will be specified later. Let $L = L_1 \sqcup L_2 \sqcup L_3 \sqcup L_4$ and $R = R_1 \sqcup R_2 \sqcup R_3 \sqcup R_4$. By Lemma 8, with probability $\Omega^*(1)$ the set S covers the same fraction of each of L_i 's and R_i 's. We denote this fraction by γ :

$$\gamma = \frac{|(L \sqcup R) \cap S|}{|L \sqcup R|} = \frac{\alpha - 3\beta/2}{1 - 3\beta}. \quad (4)$$

21:10 Improved Space Bounds for Subset Sum

Now, we apply the representation technique. Note that there exist many ways to partition S into $S_L \sqcup S_R$ such that

$$S_L \subseteq M_L \sqcup L \sqcup M \text{ and } S_R \subseteq M \sqcup R \sqcup M_R. \quad (5)$$

Indeed, the elements from $S \cap M$ can be distributed arbitrarily between S_L and S_R . Below, we show that for a large random prime number p and a random remainder r modulo p , the probability that there exists a partition $S = S_L \sqcup S_R$ such that $w(S_L) \equiv_p r$ is $\Omega^*(1)$ (informally, at least one representative $S_L \sqcup S_R$ of the original solution S survives, even if we “hit” all such partitions with a large prime).

► **Lemma 11** ([3]). *With constant probability over choices of a prime number p such that $p \in [X/2, X]$, where $X \leq |w(2^{S \cap M})|$, the number of remainders $a \in \mathbb{Z}_p$ such that there exists a subset $M' \subseteq S \cap M$ with $|M'| = 2\mu|S \cap M|$ and $w(M') \equiv_p a$ is close to p :*

$$\Pr \left[\left| \left\{ a \in \mathbb{Z}_p : \text{there exists } M' \subseteq S \cap M, |M'| = 2\mu|S \cap M|, w(M') \equiv_p a \right\} \right| \geq \Omega(p/n^2) \right] \geq 9/10.$$

Take a random prime $2^{\frac{1-\varepsilon}{2}\beta n-1} \leq p \leq 2^{\frac{1+\varepsilon}{2}\beta n}$ and a random remainder $r \in \mathbb{Z}_p$. Lemma 11 ensures that with probability $\Omega^*(1)$, there exists $M' \subseteq S \cap M$ such that $|M'| = 2\mu|S \cap M| = \mu\beta n$ and $w(M') \equiv_p r$.

To find the partition $S_L \sqcup S_R$, we are going to enumerate all sets $A \subseteq M_L \sqcup L \sqcup M$ with $w(A) \equiv_p r$ and $|A \cap (M \cap S)| = \mu\beta n$ as well as all sets $B \subseteq M \sqcup R \sqcup M_R$ with $w(B) \equiv_p t - r$ and $|B \cap (M \cap S)| = (1/2 - \mu)\beta n$. Then, it suffices to solve WOV for the sets of A 's and B 's.

To actually reduce to WOV, we first construct set families $\mathcal{M}_L \subseteq 2^{M_L}$ and $\mathcal{M}_R \subseteq 2^{M_R}$ such that $|\mathcal{M}_L| = |w(2^{M_L})|$ and $|\mathcal{M}_R| = |w(2^{M_R})|$. That is, \mathcal{M}_L contains a single subset of M_L for each possible weight. One can construct \mathcal{M}_L (as well as \mathcal{M}_R) simply by going through all subsets of M_L and checking, for each subset, whether its sum is already in \mathcal{M}_L .

Then, construct the following families (see Figure 1).

$$\mathcal{Q}_1 = \{M' \cup L' : M' \in \mathcal{M}_L, L' \subseteq L_1, |L'| = \gamma|L_1|\}, \quad (6)$$

$$\mathcal{Q}_2 = \{L' : L' \subseteq L_2, |L'| = \gamma|L_2|\}, \quad (7)$$

$$\mathcal{Q}_3 = \{L' : L' \subseteq L_3, |L'| = \gamma|L_3|\}, \quad (8)$$

$$\mathcal{Q}_4 = \{L' \cup M' : L' \subseteq L_4, M' \subseteq M, |L'| = \gamma|L_4|, |M'| = \mu\beta n\}, \quad (9)$$

$$\mathcal{Q}'_1 = \{M' \cup R' : M' \in \mathcal{M}_R, R' \subseteq R_1, |R'| = \gamma|R_1|\}, \quad (10)$$

$$\mathcal{Q}'_2 = \{R' : R' \subseteq R_2, |R'| = \gamma|R_2|\}, \quad (11)$$

$$\mathcal{Q}'_3 = \{R' : R' \subseteq R_3, |R'| = \gamma|R_3|\}, \quad (12)$$

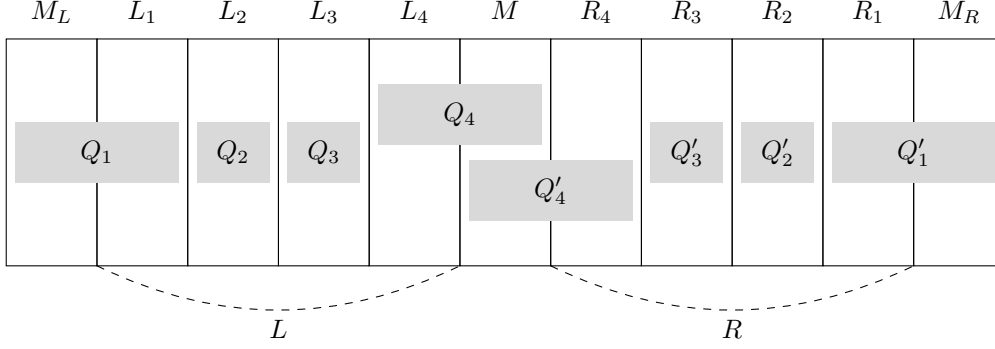
$$\mathcal{Q}'_4 = \{R' \cup M' : R' \subseteq R_4, M' \subseteq M, |R'| = \gamma|R_4|, |M'| = (1/2 - \mu)\beta n\}. \quad (13)$$

Let \mathcal{Q}_{max} stand for \mathcal{Q}_i or \mathcal{Q}'_i with the maximum size. For $a, p \in \mathbb{Z}_{>0}$, let

$$\mathcal{X}_a = \left\{ (Q_1, Q_2, Q_3, Q_4) : Q_i \in \mathcal{Q}_i, \sum w(Q_i) = a \right\} \text{ and } \mathcal{X}_{a,p} = \bigcup_{a' \equiv_p a} \mathcal{X}_{a'}.$$

Then, it remains to solve WOV for $\mathcal{X}_{r,p}$ and $\mathcal{X}'_{(t-r),p}$. To do this, we use the idea used in the Arthur–Merlin protocol: we reduce to 4-SUM and use a cutoff for the number of found candidates. The following lemmas estimate the size of these set families.

► **Lemma 12** ([3]). *With probability $\Omega^*(1)$ over the choice of r , the size of $\mathcal{X}_{r,p}$ is at most $O^*(|\mathcal{Q}_{max}|^4/p)$ and there exists $M_1 \subseteq M \cap S$ such that $w((S \cap (L \cup M_L)) \cup M_1) \equiv_p r$.*



■ **Figure 1** Partition of the instance I (white) and its solution S (gray) into parts.

To further reduce the size of these families, we choose another random prime number

$$q = \Theta^* \left(\frac{2^{n/2}}{2^{\lambda n} |\mathcal{Q}_{max}|^2} \right),$$

and iterate over all possible remainders $s \in \mathbb{Z}_q$. We aim to find two families of sets as the above ones, but with weights congruent to $w(S_L)$ and $(t - w(S_L))$, respectively, modulo p and q . While iterating on s , at some point we try $s \equiv_q w(S_L)$, so we can assume that we guessed s correctly and construct the families for the fixed r and s . We now focus on the first family only, since we can then find the second one independently using the same algorithm.

For $a \in \mathbb{Z}_{>0}$, let $\mathcal{L}_a = \{(a, Q_4 \cap M) : (Q_1, Q_2, Q_3, Q_4) \in \mathcal{X}_a\}$. For $a_1, a_2, p, q \in \mathbb{Z}_{>0}$, let

$$\mathcal{X}_{a_1, p, a_2, q} = \bigcup_{\substack{a \equiv_p a_1 \\ a \equiv_q a_2}} \mathcal{X}_a \text{ and } \mathcal{L}_{a_1, p, a_2, q} = \bigcup_{\substack{a \equiv_p a_1 \\ a \equiv_q a_2}} \mathcal{L}_a.$$

Our goal is to find a family \mathcal{L} containing $\mathcal{L}_{w(S_L)}$ in order to guarantee that we later find the solution using the WOV algorithm. But we need \mathcal{L} to be small enough. Let $\ell = \frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\max\{\mu, 1/2 - \mu\}}$ represent an upper bound on the sum of the average size of $\mathcal{X}_{a_1, p, a_2, q}$ and the size of $\mathcal{L}_{w(S_L)}$. We want \mathcal{L} to be of size $\tilde{O}(\ell)$. The following lemma shows that $\mathcal{X}_{w(S_L), p, w(S_L), q}$ contains not too many redundant sets.

► **Lemma 13** ([3]). *With probability $\Omega^*(1)$ over the choice of q , $|\mathcal{X}_{r, p, w(S_L), q} \setminus \mathcal{X}_{w(S_L)}| \leq \ell$.*

Let us find the first $\leq 2\ell + 1$ elements of $\mathcal{X}_{r, s}$ using the algorithm from Lemma 14 which runs in time $\tilde{O}(|\mathcal{Q}_{max}|^2 + \ell)$ and uses $\tilde{O}(|\mathcal{Q}_{max}| + \ell)$ space.

► **Lemma 14** ([3]). *There is an algorithm that given Q_1, Q_2, Q_3, Q_4 , (r, p, s, q) and integer m , finds $\min(|\mathcal{X}_{r, p, s, q}|, m)$ elements of $\mathcal{X}_{r, p, s, q}$ using $\tilde{O}(|\mathcal{Q}_{max}|^2 + m)$ time and $\tilde{O}(|\mathcal{Q}_{max}| + m)$ space.*

Assume that $s \equiv_q w(S_L)$. If the algorithm yields at most 2ℓ elements then we know the whole $\mathcal{X}_{r, p, w(S_L), q}$, so we can now just construct \mathcal{L} from $\mathcal{X}_{r, p, s, q}$ in $\tilde{O}(\ell)$ time and space. Otherwise, by Lemma 13, we have that $|\mathcal{X}_{w(S_L)}| > \ell$ with probability $\Omega^*(1)$. Since strictly more than half of the outputted tuples have weight exactly $w(S_L)$, we can determine $w(S_L)$ in $\tilde{O}(\ell)$ time. When we know $w(S_L)$, we use Lemma 15 to obtain $\mathcal{L}_{w(S_L)}$ using $\tilde{O}(|\mathcal{Q}_{max}|^2)$ time and $\tilde{O}(|\mathcal{Q}_{max}|)$ space.

► **Lemma 15.** *There is an algorithm that given Q_1, Q_2, Q_3, Q_4 and a , finds \mathcal{L}_a using $\tilde{O}(|\mathcal{Q}_{max}|^2)$ time and $\tilde{O}(|\mathcal{Q}_{max}|)$ space.*

21:12 Improved Space Bounds for Subset Sum

Proof. Schroepel and Shamir [15] use priority queues to implement a data structure D_1 for the following task: output all elements from $\{w(Q_1) + w(Q_2) : Q_1 \in \mathcal{Q}_1, Q_2 \in \mathcal{Q}_2\}$ in non-decreasing order using $\tilde{O}(|\mathcal{Q}_1| \cdot |\mathcal{Q}_2|)$ time and $\tilde{O}(|\mathcal{Q}_1| + |\mathcal{Q}_2|)$ space. Similarly, a data structure D_2 outputs all elements from $\{(w(Q_3) + w(Q_4), Q_4 \cap M) : Q_3 \in \mathcal{Q}_3, Q_4 \in \mathcal{Q}_4\}$ in non-increasing order (we compare elements by their first coordinate) using $\tilde{O}(|\mathcal{Q}_3| \cdot |\mathcal{Q}_4|)$ time and $\tilde{O}(|\mathcal{Q}_3| + |\mathcal{Q}_4|)$ space. By $\text{inc}(w(\mathcal{Q}_1, \mathcal{Q}_2))$ and $\text{dec}(w(\mathcal{Q}_3, \mathcal{Q}_4))$ we denote initialization methods for these data structures, by $\text{pop}()$ we denote their method that gives the next element.

Algorithm 1 solves the problem using data structures D_1 and D_2 . It works in time $\tilde{O}(|\mathcal{Q}_1| \cdot |\mathcal{Q}_2| + |\mathcal{Q}_3| \cdot |\mathcal{Q}_4|)$, uses $\tilde{O}(|\mathcal{Q}_1| + |\mathcal{Q}_2| + |\mathcal{Q}_3| + |\mathcal{Q}_4|)$ space, and outputs $|\mathcal{L}_a|$ elements. Since the number of outputted subsets is at most $|\mathcal{Q}_4|$, even if we store the output, we still need only $\tilde{O}(|\mathcal{Q}_{max}|^2)$ time and $\tilde{O}(|\mathcal{Q}_{max}|)$ space.

■ **Algorithm 1** Pseudocode for Lemma 15.

Input : $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4, a$
Output : $\mathcal{L}_a = \{(a, Q_4 \cap M) : (Q_1, Q_2, Q_3, Q_4) \in \mathcal{X}_a\}$

- 1 Initialize $D_1 = \text{inc}(w(\mathcal{Q}_1, \mathcal{Q}_2))$
- 2 Initialize $D_2 = \text{dec}(w(\mathcal{Q}_3, \mathcal{Q}_4))$
- 3 **while** $(d_2, A) = D_2.\text{pop}()$ **do**
- 4 **while** $d_1 = D_1.\text{pop}()$ and $d_1 + d_2 < a$ **do**
- 5 **skip** d_1
- 6 **if** $d_1 + d_2 = a$ and *not* $\text{used}[A]$ **then**
- 7 **output** $((a, A))$
- 8 $\text{used}[A] = \text{True}$

Note that the size of \mathcal{L}_a is at most $\binom{|M|}{\mu|M|}$. Summarizing, we get the following.

► **Lemma 16.** *There is an algorithm that given $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4$ and (r, p, s, q) , uses $\tilde{O}\left(|\mathcal{Q}_{max}|^2 + \frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\mu|M|}\right)$ time and $\tilde{O}\left(|\mathcal{Q}_{max}| + \frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\mu|M|}\right)$ space. It returns a set $\mathcal{L} \subseteq \mathbb{Z} \times \mathcal{Q}_4$ of size $\tilde{O}\left(\frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\mu|M|}\right)$ that satisfies the following.*

If there exist a solution $S \subseteq I$ with $w(S) = t$, and a partition $S = S_L \sqcup S_R$, such that:

- $|S| = \alpha n$;
- $|M_L \cap S| = |M \cap S| = |M_R \cap S| = \beta n/2$;
- $S \cap M$ is a $(\leq \varepsilon)$ -mixer;
- $|S \cap L_i| = \gamma |L_i|$ for each $i \in [4]$, the same for R_i ;
- $S_L \subseteq M_L \sqcup L \sqcup M$, $S_R \subseteq M \sqcup R \sqcup M_R$;
- $|S_L \cap M| = \mu \beta n$, $|S_R \cap M| = (1/2 - \mu) \beta n$;
- $w(S_L) \equiv_p r$, $w(S_L) \equiv_q s$, $w(S_R) \equiv_p t - r$, $w(S_R) \equiv_q t - s$;
- $|\mathcal{X}_{r,p,s,q} \setminus \mathcal{X}_{w(S_L)}| \leq \ell$,

then $\mathcal{L}_{w(S_L)} \subseteq \mathcal{L}$. Otherwise, \mathcal{L} may be any set.

Similarly, using Lemma 16, we find subset \mathcal{R} for the right side. We call $\text{WOV}(\mathcal{L}, \mathcal{R})$ and return True if it finds a solution. The WOV algorithm works in time $\tilde{O}((|\mathcal{L}| + |\mathcal{R}|)2^{\beta n(1-h(1/4))})$ and space $\tilde{O}(|\mathcal{L}| + |\mathcal{R}| + 2^{\beta n})$ by Lemma 10.

It is worth noting that for most choices of s our assumption that $s \equiv_q w(S_L)$ does not hold. Also, with probability $1 - \Omega^*(1)$, $|\mathcal{X}_{r,p,w(S_L),q} \setminus \mathcal{X}_{w(S_L)}|$ may happen to be greater than ℓ . If any of these cases occurs, we may find a set \mathcal{L} that was not intended (i.e. it

does not contain $\mathcal{L}_{w(S_L)}$, or we may even not find anything at all (if after applying the algorithm from Lemma 14 we obtain $2\ell + 1$ elements, each of which occurs less than $\ell + 1$ times). In the second scenario, we simply continue with the next choice of s . Otherwise, we cannot determine whether we have found the correct \mathcal{L} or not. However, if an incorrect \mathcal{L} is found, the outcome of our algorithm remains unaffected. If $\text{OV}(\mathcal{L}, \mathcal{R})$ finds a solution, it is a solution to the original problem; if not, we simply proceed to the next iteration of the loop in line 8.

Now we are ready to present the final Algorithm 2. We will choose the exact values for its parameters later.

■ **Algorithm 2** The main algorithm.

Input : (I, t) .
Output : True, if $S \subseteq I$ with $w(S) = t$ exists, otherwise False.

- 1 **repeat** $2^{\lambda n}$ **times**
- 2 Select random disjoint subsets $M_L, M, M_R \subseteq I$ of size βn
- 3 **foreach** μ such that $\mu\beta n \in [0, \beta n/2]$ **do**
- 4 Randomly partition $I \setminus (M_L \sqcup M \sqcup M_R)$ into L, R with size as in (15)–(16)
- 5 Pick a prime $p = \Theta(2^{(1-\varepsilon)\beta n/2})$ and $r \in \mathbb{Z}_p$ at random
- 6 Construct $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4$ and $\mathcal{Q}'_1, \mathcal{Q}'_2, \mathcal{Q}'_3, \mathcal{Q}'_4$ as defined in (6)–(13)
- 7 Pick a random prime $q = \Theta^*(2^{n/2}/(2^{\lambda n}|\mathcal{Q}_{max}|^2))$
- 8 **foreach** $s \in \mathbb{Z}_q$ **do**
- 9 Construct \mathcal{L} and \mathcal{R} (as in Lemma 16)
- 10 **if** $\text{WOV}(\mathcal{L}, \mathcal{R})$ (as in Lemma 10) **then**
- 11 | **return** True
- 12 **return** False

4.2 Correctness

► **Lemma 17.** *The success probability of the algorithm is $\Omega^*(1)$.*

Proof. To detect that I is indeed a yes-instance, an algorithm iteration relies on a number of events. The first one is $|(M_L \cup M \cup M_R) \cap S| = 3\beta n/2$, that occurs with probability $\Theta^*(2^{-\lambda n})$. All the others occur with probability $\Omega^*(1)$, conditioned on the event that all the previous events occur. This implies that all of them occur with probability $\Omega^*(2^{-\lambda n})$, and after repeating it $2^{\lambda n}$ times, the success probability of the resulting algorithm is $\Omega^*(1)$. Below, we recall all the considered events and provide links to lemmas proving their conditional probabilities.

1. $|(M_L \cup M \cup M_R) \cap S| = 3\beta n/2$, see Eq. (2).
2. $|M_L \cap S| = |M \cap S| = |M_R \cap S| = \frac{\beta}{2}n$, see Lemma 8.
3. $|S \cap L_i| = \gamma|L_i|$ and $|S \cap R_i| = \gamma|R_i|$, see Lemma 8.
4. p is a prime number, see Section 2.2.
5. $|\{a \in \mathbb{Z}_p : \exists M' \subseteq S \cap M, |M'| = 2\mu|S \cap M|, w(M') \equiv_p a\}| \geq \Omega(p/n^2)$, see Lemma 11.
6. $|\mathcal{X}_{r,p}| = O^*(|\mathcal{Q}_{max}|^4/p)$ and there exists a partition $M_1 \sqcup M_2 = S \cap M$ such that $w(S \cap (M_L \cup L \cup M_1)) \equiv_p r$ and $w(S \cap (M_2 \cup R \cup M_R)) \equiv_p t - r$, see Lemma 12.
7. q is a prime number, see Section 2.2.
8. $|\mathcal{X}_{r,p,w(S_L),q} \setminus \mathcal{X}_{w(S_L)}| \leq \ell$, see Lemma 13.

In case any of those events does not happen, the algorithm completes the iteration without affecting the final result. ◀

4.3 Setting the Parameters

Recall that $\alpha = \frac{|S|}{n}$ and we iterate over all possible α . We define $\beta = \beta(\alpha)$ as follows:

$$\beta(\alpha) = \begin{cases} 0.13, & \alpha \in [0.45, 0.55] \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

Let us consider an iteration of the loop on line 3. To this point, we know the values of ε , ε_L , ε_R and μ , so we consider them as fixed numbers. Since $|M_L| = |M| = |M_R| = \beta n$, we have that $|I \setminus (M_L \sqcup M \sqcup M_R)| = (1 - 3\beta)n$. Let us consider a partition $I \setminus (M_L \sqcup M \sqcup M_R) = L_1 \sqcup L_2 \sqcup L_3 \sqcup L_4 \sqcup R_1 \sqcup R_2 \sqcup R_3 \sqcup R_4$ and estimate the size of \mathcal{Q}_i and \mathcal{Q}'_i . Recall that $\gamma = \frac{|(L \cup R) \cap S|}{|L \cup R|} = \frac{\alpha - \frac{3}{2}\beta}{1 - 3\beta}$. Observe that we use ε instead of ε_L and ε_R since $\varepsilon \leq \varepsilon_L, \varepsilon_R$.

$$\begin{aligned} |\mathcal{Q}_1| &\leq \binom{|L_1|}{\gamma|L_1|} \cdot 2^{(1-\varepsilon)\beta n} & |\mathcal{Q}'_1| &\leq \binom{|R_1|}{\gamma|R_1|} \cdot 2^{(1-\varepsilon)\beta n} \\ |\mathcal{Q}_2| &= \binom{|L_2|}{\gamma|L_2|} & |\mathcal{Q}'_2| &= \binom{|R_2|}{\gamma|R_2|} \\ |\mathcal{Q}_3| &= \binom{|L_3|}{\gamma|L_3|} & |\mathcal{Q}'_3| &= \binom{|R_3|}{\gamma|R_3|} \\ |\mathcal{Q}_4| &= \binom{|L_4|}{\gamma|L_4|} \cdot \binom{|M|}{\mu\beta n} & |\mathcal{Q}'_4| &= \binom{|R_4|}{\gamma|R_4|} \cdot \binom{|M|}{(\frac{1}{2} - \mu)\beta n}. \end{aligned}$$

We construct \mathcal{L} and \mathcal{R} in time depending on $|\mathcal{Q}_{max}|^2 + |\mathcal{Q}'_{max}|^2$. Note that this function is minimized when all $|\mathcal{Q}_i|$ and $|\mathcal{Q}'_i|$ are roughly the same. We can achieve that by a careful choice of $|L_i|$ and $|R_i|$ (in the full version of the paper [3], we provide evidence of the non-contradictory nature of the chosen sizes). To make all $|\mathcal{Q}_i|$ and $|\mathcal{Q}'_i|$ equal $n \frac{(h(\gamma) - 3\beta h(\gamma) + 2\beta - 2\varepsilon\beta + \beta h(\mu) + \beta h(\frac{1}{2} - \mu))}{8}$ we set sizes of L_i and R_i as follows:

$$\begin{aligned} |L_1| &= n \frac{h(\gamma) - 3\beta h(\gamma) - 6\beta + 6\varepsilon\beta + \beta h(\mu) + \beta h(\frac{1}{2} - \mu)}{8h(\gamma)} = |R_1| \\ |L_2| &= |L_3| = n \frac{h(\gamma) - 3\beta h(\gamma) + 2\beta - 2\varepsilon\beta + \beta h(\mu) + \beta h(\frac{1}{2} - \mu)}{8h(\gamma)} = |R_2| = |R_3| \\ |L_4| &= n \frac{h(\gamma) - 3\beta h(\gamma) + 2\beta - 2\varepsilon\beta - 7\beta h(\mu) + \beta h(\frac{1}{2} - \mu)}{8h(\gamma)} \\ |R_4| &= n \frac{h(\gamma) - 3\beta h(\gamma) + 2\beta - 2\varepsilon\beta + \beta h(\mu) - 7\beta h(\frac{1}{2} - \mu)}{8h(\gamma)}. \end{aligned}$$

We define the sizes of $|L|$ and $|R|$ as shown below and require that $|L_1| + |L_2| + |L_3| + |L_4| = |L|$ and $|R_1| + |R_2| + |R_3| + |R_4| = |R|$.

$$|L| = \frac{1 - 3\beta - \chi\beta}{2} n, \quad (15)$$

$$|R| = \frac{1 - 3\beta + \chi\beta}{2} n, \quad (16)$$

where $\chi = \frac{h(\mu) - h(\frac{1}{2} - \mu)}{h(\gamma)}$ is a balancing parameter needed to ensure that $|\mathcal{Q}_{max}| \approx |\mathcal{Q}'_{max}|$.

We have set all the parameters and now we are ready to estimate the time and space complexity of Algorithm 2.

4.4 Time Complexity

The running time is dominated by the following parts.

- Repeating the whole algorithm $2^{\lambda n}$ times (Line 1).
- Constructing $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4$ and $\mathcal{Q}'_1, \mathcal{Q}'_2, \mathcal{Q}'_3, \mathcal{Q}'_4$ takes $O^*(|\mathcal{Q}_{max}| + 2^{\beta n})$ time (Line 6).
- Repeating q times (Line 8).
- Finding \mathcal{L}, \mathcal{R} using Lemma 16 in time $O^*\left(|\mathcal{Q}_{max}|^2 + \frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\max\{\mu, 1/2-\mu\}}|M|\right)$ (Line 9).
- Solving $\text{WOV}(\mathcal{L}, \mathcal{R})$ using Lemma 10 in time $O^*((|\mathcal{L}| + |\mathcal{R}|)2^{\beta n(1-h(1/4))})$ (Line 10). The O^* here hides logarithmic factor in $|\mathcal{L}| + |\mathcal{R}|$ that grows polynomially in n .

Let us consider an iteration of the loop on line 3. Note that the values of $p, q, |\mathcal{Q}_{max}|$ and μ are now determined. The running time of Algorithm 2 on this specific iteration is

$$O^*\left(q \cdot \left(|\mathcal{Q}_{max}|^2 + \frac{|\mathcal{Q}_{max}|^4}{pq} + \left(\frac{|\mathcal{Q}_{max}|^4}{pq} + 2^{\beta n h(\max\{\mu, 1/2-\mu\})}\right) 2^{\beta n(1-h(1/4))}\right)\right).$$

To bound the total running time of Algorithm 2, we can multiply the number of the iterations $\Theta^*(2^{\lambda n})$ by the upper bound on the slowest iteration (see calculations in the full version [3]). As a result, we bound the running time of Algorithm 2 by $O^*(2^{n/2} + 2^{T(\alpha, \beta) \cdot n})$, where

$$T(\alpha, \beta) = \frac{1}{2} (h(\gamma) - 3\beta h(\gamma) + 3\beta + 2\lambda). \quad (17)$$

► **Lemma 18.** *If for every $\alpha \in [0, 1]$, $T(\alpha, \beta(\alpha)) \leq 0.5$, then Algorithm 2 has running time $O^*(2^{0.5n})$.*

We complete the analysis in Section 4.6.

4.5 Space Complexity

The space usage is dominated by the following parts.

- Constructing $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4$ and $\mathcal{Q}'_1, \mathcal{Q}'_2, \mathcal{Q}'_3, \mathcal{Q}'_4$ with $O^*(|\mathcal{Q}_{max}|)$ space (Line 6).
- Finding \mathcal{L}, \mathcal{R} using Lemma 16 with $O^*\left(|\mathcal{Q}_{max}| + \frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\max\{\mu, 1/2-\mu\}}|M|\right)$ space (Line 9).
- Solving $\text{WOV}(\mathcal{L}, \mathcal{R})$ using Lemma 10 with $O^*(|\mathcal{L}| + |\mathcal{R}| + 2^{\beta n})$ space (Line 10).

For a specific iteration of the loop on line 3, the space usage is at most the sum of the above expressions, where $|\mathcal{L}|, |\mathcal{R}| = O^*\left(\frac{|\mathcal{Q}_{max}|^4}{pq} + \binom{|M|}{\max\{\mu, 1/2-\mu\}}|M|\right)$. The total space complexity of Algorithm 2 can be bounded by the space usage of an iteration, requiring the most space.

Therefore, it suffices to prove that $2^{\beta n}$, $\binom{|M|}{\max\{\mu, 1/2-\mu\}}|M|$, $|\mathcal{Q}_{max}|$ and $\frac{|\mathcal{Q}_{max}|^4}{pq}$ are at most $O^*(2^{0.246n})$ on each iteration. By some simplifications (see calculations in the full version [3]) we can show that the total space usage of our algorithm is $O^*(2^{0.246n} + 2^{S(\alpha, \beta) \cdot n})$, where

$$S(\alpha, \beta) = \frac{1}{4} \left(3h(\gamma) - 9\beta h(\gamma) + 6\beta h\left(\frac{1}{4}\right) + 4\beta - 2 + 4\lambda \right). \quad (18)$$

We can see that the following lemma holds.

► **Lemma 19.** *If for every $\alpha \in [0, 1]$, $S(\alpha, \beta(\alpha)) \leq 0.246$, then the total space usage of Algorithm 2 does not exceed $O^*(2^{0.246n})$.*

We complete the analysis in the following section.

4.6 Parameter Substitution

The proofs of the following two lemmas can be found in the full version [3].

► **Lemma 20.** [Maximum point for function of time complexity] The function $T(\alpha, \beta)$ (see (17)) satisfies $T(\alpha, \beta) \leq T(0.5, 0.13)$ when β is defined as in (14).

► **Lemma 21.** [Maximum point for function of space complexity] The function $S(\alpha, \beta)$ (see (18)) satisfies $S(\alpha, \beta) \leq S(0.5, 0.13)$ when β is defined as in (14).

Hence, the time and space complexity do not exceed $T(0.5, 0.13)$ and $S(0.5, 0.13)$, respectively (for any α and β as in (14)). Recall that $\lambda = h(3\beta) - \alpha h(\frac{3\beta}{2\alpha}) - (1 - \alpha)h(\frac{3\beta}{2(1-\alpha)})$, and $\gamma = \frac{\alpha - \frac{3}{2}\beta}{1 - 3\beta}$, see Equation (4). By plugging $\alpha = 0.5, \beta = 0.13$, we get the following values for the parameters:

$$\begin{aligned} \gamma &= 0.5, \\ h(\gamma) &= 1, \\ \lambda &= 0, \\ T(0.5, 0.13) &= \frac{1}{2} \cdot (1 - 3 \cdot 0.13 + 3 \cdot 0.13) = \frac{1}{2}, \\ S(0.5, 0.13) &\leq \frac{1}{4} \cdot (3 - 9 \cdot 0.13 + 6 \cdot 0.13 \cdot 0.812 + 4 \cdot 0.13 - 2) \leq 0.246. \end{aligned}$$

Combining Lemma 18 and Lemma 19, we get that the algorithm runs in time $O^*(2^{0.5n})$ and space $O^*(2^{0.246n})$, which completes the proof of Theorem 6.

References

- 1 Shyan Akmal, Lijie Chen, Ce Jin, Malvika Raj, and Ryan Williams. Improved Merlin–Arthur protocols for central problems in fine-grained complexity. In *ITCS*, volume 215 of *LIPICs*, pages 3:1–3:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
- 3 Tatiana Belova, Nikolai Chukhin, Alexander S. Kulikov, and Ivan Mihajlin. Improved space bounds for subset sum. *CoRR*, abs/2402.13170, 2024.
- 4 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS 2016*, pages 261–270. ACM, 2016.
- 5 Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020.
- 6 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset sum in time $2^{n/2}/\text{poly}(n)$. In *APPROX/RANDOM*, volume 275 of *LIPICs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
- 7 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.
- 8 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.
- 9 Hamidreza Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *ICALP 2015*, pages 749–760. Springer, 2015.
- 10 Hendrik W. Lenstra, Jr. and Carl Pomerance. Primality testing with gaussian periods. *Journal of the European Mathematical Society*, 21(4):1229–1269, 2019.

- 11 Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic time-space trade-offs for k -SUM. In *ICALP*, volume 55 of *LIPICs*, pages 58:1–58:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
- 12 Jesper Nederlof. A short note on Merlin–Arthur protocols for subset sum. *Inf. Process. Lett.*, 118:15–16, 2017.
- 13 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppele and Shamir’s algorithm for subset sum via orthogonal vectors. In *STOC*, pages 1670–1683. ACM, 2021.
- 14 Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979. doi:10.1145/322123.322138.
- 15 Richard Schroeppele and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.