


Subset Sum in Time $2^{n/2}/\text{poly}(n)$

Xi Chen ✉ 

Columbia University, New York, NY, USA

Yaonan Jin ✉ 

Columbia University, New York, NY, USA

Tim Randolph ✉ 

Columbia University, New York, NY, USA

Rocco A. Servedio ✉ 

Columbia University, New York, NY, USA

Abstract

A major goal in the area of exact exponential algorithms is to give an algorithm for the (worst-case) n -input Subset Sum problem that runs in time $2^{(1/2-c)n}$ for some constant $c > 0$. In this paper we give a Subset Sum algorithm with worst-case running time $O(2^{n/2} \cdot n^{-\gamma})$ for a constant $\gamma > 0.5023$ in standard word RAM or circuit RAM models. To the best of our knowledge, this is the first improvement on the classical “meet-in-the-middle” algorithm for worst-case Subset Sum, due to Horowitz and Sahni, which can be implemented in time $O(2^{n/2})$ in these memory models [16].

Our algorithm combines a number of different techniques, including the “representation method” introduced by Howgrave-Graham and Joux [17] and subsequent adaptations of the method in Austrin, Kaski, Koivisto, and Nederlof [4], and Nederlof and Węgrzycki [20], and “bit-packing” techniques used in the work of Baran, Demaine, and Pătraşcu [5] on subquadratic algorithms for 3SUM.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Exact algorithms, subset sum, log shaving

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.39

Category RANDOM

Related Version *Full Version:* <https://arxiv.org/abs/2301.07134>

Funding *Xi Chen:* Supported in part by NSF grants IIS-1838154, CCF-2106429 and CCF-2107187. *Rocco A. Servedio:* Supported in part by NSF awards IIS-1838154, CCF-2106429, and CCF-2211238, and by the Simons Collaboration on Algorithms and Geometry.

Acknowledgements We would like to thank Martin Dietzfelbinger for pointing out the work [12], as well as several anonymous referees for helpful suggestions.

1 Introduction

One of the most well-known and simple-to-state NP-complete problems is the *Subset Sum* problem. An instance of Subset Sum consists of a list $X = (x_1, \dots, x_n)$ of n positive integer values and a positive integer target t , and the output is either a subset $S \subseteq X$ such that $\sum_{x_i \in S} x_i = t$ or a report that no such subset exists. Subset Sum was one of the original 21 problems proved NP-complete in Karp’s seminal paper [18] and has been the subject of intensive study from many different perspectives for at least five decades.

This paper is motivated by the following open problem in the theory of exact exponential time algorithms: how quickly can we solve worst-case instances of Subset Sum? Exhaustive search over all possible solutions yields a trivial $2^n \cdot \text{poly}(n)$ -time algorithm. In 1974, Horowitz and Sahni introduced the “meet-in-the-middle” technique, which gives an algorithm that can be implemented in $O(2^{n/2})$ time in standard RAM models [16]. Since then, obtaining



© Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio;
licensed under Creative Commons License CC-BY 4.0

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023).

Editors: Nicole Megow and Adam D. Smith; Article No. 39; pp. 39:1–39:18



Leibniz International Proceedings in Informatics

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

a $2^{(1/2-c)n}$ -time algorithm for some constant $c > 0$ has emerged as a major goal in the exact exponential time algorithms community (explicitly mentioned in [23, 11, 3, 4, 20] and numerous other works) which has attracted the attention of many researchers.

Intriguing progress has been made on a number of variants of the core worst-case Subset Sum problem. More than forty years ago Schroepel and Shamir [22] improved the $2^{n/2} \cdot \text{poly}(n)$ space complexity of the meet-in-the-middle algorithm by giving an algorithm that runs in $2^{n/2} \cdot \text{poly}(n)$ time and $2^{n/4} \cdot \text{poly}(n)$ space. An exciting recent breakthrough by Nederlof and Węgrzycki [20] further improved this space complexity to $2^{0.249999n}$. In [17] Howgrave-Graham and Joux gave an algorithm which can solve *average-case* Subset Sum instances in time $2^{0.337n}$,¹ and this was later improved to $2^{0.291n}$ in the work of Becker et al. [6]. The closely related *Equal Subset Sum* problem, which looks for two subsets with the same sum, can be solved exponentially faster than suggested by meet-in-the-middle [19] and also yields further improvements in the average case [10]. However, to the best of our knowledge, there have been no improvements on the worst-case $O(2^{n/2})$ runtime of the meet-in-the-middle algorithm for Subset Sum since it was first introduced almost fifty years ago.

Our contribution: Worst-case Subset Sum in $2^{n/2}/\text{poly}(n)$ time. Given the longstanding difficulty of achieving a $2^{(1/2-c)n}$ -time worst-case algorithm for Subset Sum, it is natural to consider the relaxed goal of achieving *some* nontrivial speedup of the meet-in-the-middle algorithm. In this paper we achieve this goal; more precisely, we give three different randomized algorithms for worst-case Subset Sum, each of which runs in time $O(2^{n/2} \cdot n^{-\gamma})$ for a specific constant $\gamma > 0$ in a standard word RAM or circuit RAM model (described in detail in Section 1.1 below). Our fastest algorithm, which combines techniques from our other two algorithms, runs in time $O(2^{n/2} \cdot n^{-0.5023})$.

The improvements we achieve over the $O(2^{n/2})$ runtime of the meet-in-the-middle algorithm for Subset Sum are analogous to “log-shaving” improvements on the runtimes of well-known and simple polynomial-time algorithms for various problems which have resisted attempts at polynomial-factor improvements. There is a substantial strand of research along these lines (see [9, 8] for a non-exhaustive overview); indeed, Abboud and Bringmann [1] have recently stated that: “A noticeable fraction of Algorithms papers in the last few decades improve the runtime of well-known algorithms for fundamental problems by logarithmic factors.” In our setting, since the well-known and simple algorithm for Subset Sum (namely, meet-in-the-middle) runs in exponential time, saving a $\text{poly}(n)$ factor, as we do, is analogous to “log-shaving”. Indeed, as we discuss in Section 1.2 below, our first and most straightforward algorithm is based on “bit-packing” techniques that were used by Baran, Demaine, and Pătraşcu [5] to shave log factors from the standard $O(n^2)$ -time algorithm for the 3SUM problem. We find it somewhat surprising that the “log-shaving” perspective has not previously appeared in the literature on Subset Sum, and we hope that our work will lead to further (and more substantial) runtime improvements for Subset Sum and other problems with well-known and simple exponential-time algorithms.

► **Remark 1.** We note that by a straightforward reduction, an algorithm for 4SUM running in time $O(n^2/\log(n)^\alpha)$ for any constant $\alpha > 0$ would immediately imply a Subset Sum algorithm running in time $O(2^{n/2}/n^\alpha)$, which would be a result comparable to ours. However, while log-shaving results for 3SUM are known [5, 14], giving an $o(n^2)$ algorithm for 4SUM is a well-known open problem.

¹ See the last paragraph of [6] for a discussion of the runtime of [17].

1.1 Our Computational Model

A Subset Sum instance is parameterized by the number of inputs n and the size of the target value t (without loss of generality, $x_1, x_2, \dots, x_n \leq t$). Thus it is natural to adopt a memory model with word length $\ell = \Theta(\log t)$ such that each input integer can be stored in a single word. This is the framework used in the work of Pisinger [21], which studies dynamic programming approaches for Subset Sum in the word RAM model (see [21, Equation (1)]). We also note that this memory model is analogous to the standard RAM model that is commonly used for problems such as 3SUM (see e.g. [5]), where it is assumed that each input value is at most $\text{poly}(n)$ and hence fits into a single $O(\log n)$ -bit machine word.

This framework lets us consider arbitrary input instances of Subset Sum with no constraints on the size of the input integers. If $t = 2^{o(n)}$, standard dynamic programming algorithms [7] solve the problem in time $O(nt) = 2^{o(n)}$, which supersedes our $\text{poly}(n)$ -factor improvements over meet-in-the-middle; hence throughout the paper we assume $t = 2^{\Omega(n)}$. It is arguably most natural to think about instances in which $t = 2^{\Theta(n)}$, in which case $\ell = \Theta(n)$, and we encourage the first-time reader to imagine $\ell = \Theta(n)$ for easy digestion. More precisely, we make the assumption throughout the paper that the word size $\ell = \text{poly}(n)$, although some of our results even hold for extremely large word sizes and are footnoted accordingly.

We consider runtime in two standard variants of the RAM model. The first is *circuit RAM*; in this model, any operation that maps a constant number of words to a single word and has a $\text{poly}(\ell)$ -size circuit with unbounded fan-in gates can be performed in time proportional to the depth of the circuit. Consequently, in the circuit RAM model, AC^0 operations on a *constant* number of words can be performed in *constant* time, and multiplying, performing modular division, etc., on two ℓ -bit words can be performed in time $O(\log \ell)$. The second is *word RAM*, in which the usual arithmetic operations, including multiplication, are assumed to take unit time, but arbitrary AC^0 operations are not atomic operations on words. We present each of our algorithms for the stronger circuit RAM model,² and explain adaptations that give corresponding word RAM algorithms.

1.2 Results, Techniques, and Organization

In Section 2 we establish our notation and review some background results and observations that will be used throughout the paper.

Sections 3–5 give our three new algorithms, which augment the standard meet-in-the-middle approach in different ways to achieve their respective runtime improvements. Each of our algorithms is a randomized decision algorithm that runs in the time bound claimed below and on every input instance outputs the correct answer with probability at least $3/4$. Further, each of our algorithms has one-sided error, i.e., it never makes a mistake when it outputs “yes”.

Our first and simplest algorithm, presented in Section 3, achieves a runtime of $\tilde{O}(2^{n/2} \cdot \ell^{-1/2}) \leq \tilde{O}(2^{n/2} \cdot n^{-1/2})$ in the circuit RAM model and $\tilde{O}(2^{n/2} \cdot n^{-1/2})$ in the word RAM model, for all $\ell = \text{poly}(n)$.³ It works by adapting the *bit-packing* trick, a technique developed by Baran, Demaine, and Pătraşcu [5] for the 3SUM problem, for the **Meet-in-the-Middle** algorithm. The idea is to compress the two lists of partial subset sums used in **Meet-in-the-Middle** by packing hashes of multiple values into a machine

² Note that any algorithm in the word RAM model can be simulated in the circuit RAM model with no more than an $O(\log \ell)$ -factor slowdown.

³ The notation $\tilde{O}(\cdot)$ suppresses $\text{polylog}(\ell) = \text{polylog}(n)$ factors.

word, while preserving enough information to make it possible to run (an adaptation of) **Meet-in-the-Middle** on the lists of hashed and packed values. This results in a runtime savings over performing **Meet-in-the-Middle** on the original lists (without hashing and packing), because processing a pair of words, each containing multiple hashed values, takes constant expected time in the circuit RAM model and can be memoized to take constant time in the word RAM model.

Our second algorithm, given in Section 4, achieves a runtime of $O(2^{n/2} \cdot \ell^{-\gamma}) \leq O(2^{n/2} \cdot n^{-\gamma})$ for some constant $\gamma > 0.01$ in the circuit RAM model and $O(2^{n/2} \cdot n^{-\gamma})$ in the word RAM model, for all $\ell = \text{poly}(n)$. Although the time savings is smaller than our first algorithm, we believe that this algorithm is conceptually interesting since it avoids bit-packing and instead combines **Meet-in-the-Middle** with two techniques devised in prior work on Subset Sum. The first of these is the “representation method” introduced by Howgrave-Graham and Joux [17]. Roughly speaking, the idea of this method is to (i) increase the size of the search space in such a way that a single solution has many “representations” in the space of enhanced solutions, and then (ii) search over only a fraction of the enhanced solution space. A consequence of expanding the solution space, though, is that a number of “pseudosolutions”, solutions that contain certain input elements more than once, are introduced. This leads us to the second technique, i.e. the use of a fast subroutine for the Orthogonal Vectors (OV) problem (recall that OV is the problem of deciding whether two lists of $\{0, 1\}$ -vectors contain a pair of vectors, one from each list, that are orthogonal). The fast OV subroutine lets us efficiently rule out pseudosolutions while running (an adaptation of) **Meet-in-the-Middle** on a fraction of the enhanced solution space.

In Section 5 we give our fastest algorithm, which uses a delicate combination of the techniques from Sections 3 and 4 to obtain a runtime of $O(2^{n/2} \cdot n^{-0.5023})$ for all $\ell = \text{poly}(n)$. While the runtime improvement over Section 3 is not large, this algorithm demonstrates that by leveraging insights specific to the Subset Sum problem, we can achieve time savings beyond what is possible with more “generic” log shaving techniques. Finally, in Section 6 we briefly discuss directions for future work.

Note that this version of the work is an extended abstract. Readers interested in a more complete presentation may wish to consult the full version.⁴

2 Preliminaries

To ease readability, we adopt the following notational conventions throughout the paper: lowercase Roman letters (ℓ , n , etc.) denote variables; lowercase Greek letters (ε , α , etc.) denote numerical constants; capital Roman letters (L , W , etc.) denote sets, multisets, or lists; and calligraphic capital letters (\mathcal{W} , \mathcal{Q} , etc.) denote collections of sets of numbers.

Logarithms. When written without a specified base, $\log(\cdot)$ denotes the base-2 logarithm.

Big-O Notation. We augment big- O notation with a tilde (\tilde{O} , $\tilde{\Omega}$, $\tilde{\Theta}$ etc.) to suppress $\text{polylog}(\ell) = \text{polylog}(n)$ factors. For example, we have $\tilde{O}(2^n) = O(2^n \cdot \text{polylog}(n))$ and $\tilde{\Omega}(n) = \Omega(\frac{n}{\text{polylog}(n)})$.

Probability Notation. Random variables are written in boldface. In particular, we write “ $\mathbf{x} \sim S$ ” to indicate that an element \mathbf{x} is sampled uniformly at random from a finite multiset S .

⁴ <https://arxiv.org/abs/2301.07134>

Set Notation. We write $[a : b]$ for the set of integers $\{a, a + 1, \dots, b\}$ and $[a]$ for $\{1, 2, \dots, a\}$. We write $\mathbb{P}[a : b]$ for the set of all primes in the interval $[a : b]$.

Given a multiset or list Y of integers, we adopt several shorthands: $\Sigma(Y) := \sum_{y \in Y} y$ denotes the sum, $W(Y) := \{\Sigma(T) \mid T \subseteq Y\}$ denotes the set of *distinct* sub-multiset sums, and L_Y denotes the list containing the elements of $W(Y)$ sorted in increasing order.

Also, we often write $f(Y)$ for the multiset or list obtained by applying operation f element-wise, such as $Y + \alpha = \{y + \alpha \mid y \in Y\}$ and $\alpha Y = \{\alpha y \mid y \in Y\}$. The only exception is $(Y \bmod p)$, which denotes the set of *distinct* residues $\{(y \bmod p) \mid y \in Y\}$.

Stirling's Approximation for Binomial Coefficients. We use the following well-known consequence of Stirling's approximation (see e.g. [13, Lemma 16.19]): For each integer $j \in [0 : n/2]$,

$$\sum_{i \leq j} \binom{n}{i} \leq 2^{H(j/n) \cdot n}, \tag{1}$$

where $H(y) := -y \log(y) - (1 - y) \log(1 - y)$ is the binary entropy function.

Pseudolinear Hashing. Recall that $\ell = \text{poly}(n)$ is the word length in our memory model. Given an integer $m \leq \ell$, we write \mathbf{h}_m to denote the random hash function defined as

$$\mathbf{h}_m(y) := (\mathbf{u} \cdot y \pmod{2^\ell}) \gg \ell - m. \tag{2}$$

Here the input y is an ℓ -bit integer, \mathbf{u} is selected uniformly at random from all odd ℓ -bit integers, and \gg denotes a bit shift to the right, i.e., dividing $\mathbf{u} \cdot y \pmod{2^\ell}$ by $2^{\ell - m}$ and then truncating the result so that only the higher-order m bits remain.

This hash function $\mathbf{h}_m(y)$ can be evaluated in time $O(\log \ell) = O(\log n)$ for $\ell = \text{poly}(n)$ in the circuit RAM model, i.e., essentially the time to multiply, or constant time $O_n(1)$ in the word RAM model. Further, \mathbf{h}_m has the following useful properties [12, 5].

► **Lemma 2** (Pseudolinear Hashing [12, 5]). *The following hold for the hash function \mathbf{h}_m :*

1. **Pseudolinearity.** *For any two ℓ -bit integers y, z and any outcome of \mathbf{h}_m ,*

$$\mathbf{h}_m(y) + \mathbf{h}_m(z) \in \mathbf{h}_m(y + z) - \{0, 1\} \pmod{2^m}.$$

2. **Pseudouniversality.** *For any two ℓ -bit integers y, z with $y \neq z$,*

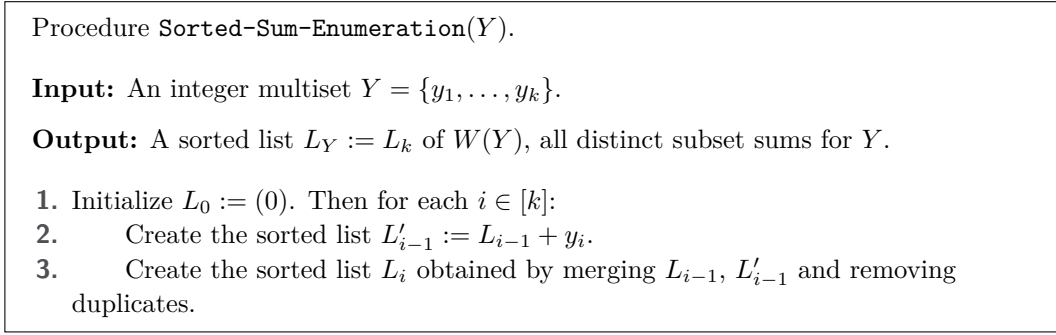
$$\Pr[\mathbf{h}_m(y) = \mathbf{h}_m(z)] = O(2^{-m}).$$

2.1 Preliminary Results

For the purposes of this paper, we may assume that the input target t have size $2^{\Omega(n)}$:

► **Observation 3** (Assumptions about Input Instances). *Given a Subset Sum instance (X, t) with $t \leq 2^{0.499n}$, the standard dynamic programming algorithm [7] takes time $O(nt) = 2^{0.499n + o(n)}$, much faster than the $\text{poly}(n)$ -factor speedups we are targeting, and cannot be improved to time $t^{1-\varepsilon} \cdot 2^{o(n)}$ unless SETH fails [2]. Hence without loss of generality, we assume $t = 2^{\Omega(n)}$ and $\ell = \Omega(\log t) = \Omega(n)$ throughout the paper.*

We further observe that the randomized *decision* algorithms that we give can be converted into randomized *search* algorithms with the same asymptotic runtimes using standard search-to-decision reductions.



■ **Figure 1** Efficiently enumerating subset sums of $k \geq 1$ integers.

A basic primitive for our algorithms is the sorted list L_Y containing the elements of $W(Y)$, all *distinct* subset sums, for a given multiset Y of size $|Y| = k$. Figure 1 shows an $O(2^k)$ -time folklore algorithm that enumerates L_Y :

► **Lemma 4** (Sorted Sum Enumeration; Folklore). *Sorted-Sum-Enumeration runs in time $O(2^k)$.*

Proof. We essentially adapt the classic *merge sort* algorithm. Since $|L_{i-1}| = |L'_{i-1}| \leq |L_i| \leq 2|L_{i-1}|$ for each $i \in [k]$, the runtime of **Sorted-Sum-Enumeration** is

$$\sum_{i \in [k]} O(|L'_{i-1}| + |L_i|) = \sum_{i \in [k]} O(|L_i|) = \sum_{i \in [k]} O(2^i) = O(2^k). \quad \blacktriangleleft$$

We can improve this runtime analysis if $|W(Y)|$ is smaller than 2^k by a $\text{poly}(k)$ factor:

► **Lemma 5** (Sorted Sum Enumeration for Small $W(Y)$). *If Y is a multiset of $|Y| = k$ integers with $|W(Y)| \leq 2^k \cdot k^{-\varepsilon}$ for some constant $\varepsilon > 0$, **Sorted-Sum-Enumeration** runs in time $O(2^k \cdot k^{-\varepsilon} \cdot \log k)$.*

Proof. If $|W(Y)| \leq 2^k \cdot k^{-(1+\varepsilon)}$, then since $|L_i| \leq |L_k| = |W(Y)|$ for each $i \in [k]$, it is easy to check that the algorithm runs in time $O(k \cdot |W(Y)|) = O(2^k \cdot k^{-\varepsilon})$; so suppose that $|W(Y)| \geq 2^k \cdot k^{-(1+\varepsilon)}$. Using the bound $|L_i| \leq 2^i$ for $i \leq \log |W(Y)|$ and the bound $|L_i| \leq |W(Y)|$ for $\log |W(Y)| < i \leq k$, the runtime of **Sorted-Sum-Enumeration** is upper bounded by

$$\begin{aligned} \sum_{i \in [k]} O(|L_i|) &= \sum_{i \leq \log |W(Y)|} O(2^i) + \sum_{\log |W(Y)| < i \leq k} O(|W(Y)|) \\ &= O(|W(Y)|) + (1 + \varepsilon) \cdot \log k \cdot O(|W(Y)|) \\ &= O(|W(Y)| \cdot \log k) = O(2^k \cdot k^{-\varepsilon} \cdot \log k). \quad \blacktriangleleft \end{aligned}$$

► **Remark 6.** Lemma 5 gives a tight analysis of the algorithm when $|W(Y)| = 2^k \cdot k^{-\varepsilon}$, as can be seen by considering the particular size- k multiset $Y = (2^0, 2^1, \dots, 2^{k-\varepsilon \log k - 1}, 1, 1, \dots, 1)$.

Finally, Figure 2 shows the classic meet-in-the-middle algorithm for Subset Sum, by Horowitz and Sahni [16], which serves as our baseline for comparison.

► **Lemma 7.** *The worst-case runtime of **Meet-in-the-Middle** is $O(2^{n/2})$.*

Proof. This follows immediately from Lemma 4 as $|L_A|, |L_B| \leq 2^{n/2}$. ◀

Procedure **Meet-in-the-Middle**(X, t).

Input: An integer multiset $X = \{x_1, x_2, \dots, x_n\}$ and an integer target t .

Output: “yes” if (X, t) is a Subset Sum instance that has a solution, “no” otherwise.

0. Fix any partition of $X = A \cup B$ such that $|A| = |B| = n/2$.
1. Enumerate the sorted lists L_A and L_B using **Sorted-Sum-Enumeration**.
2. Initialize two pointers at the smallest value in L_A and the largest value in L_B .
 If these two values sum to the target t , then return “yes”;
 if they sum to less than t , then increment the pointer into L_A and repeat;
 and if they sum to more than t , then increment the pointer into L_B and repeat.
 If either pointer goes past the end of its list, then return “no”.

■ **Figure 2** The classic **Meet-in-the-Middle** algorithm [16].

3 $\Omega(n^{0.5}/\log n)$ -Factor Speedup via Bit Packing

In this section we analyze our first and simplest algorithm, **Bit-Packing** (see Figure 3).⁵

► **Theorem 8.** ***Bit-Packing** is a zero-error randomized algorithm for the Subset Sum problem with expected runtime $O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell) \leq O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$ in the circuit RAM model.*⁶

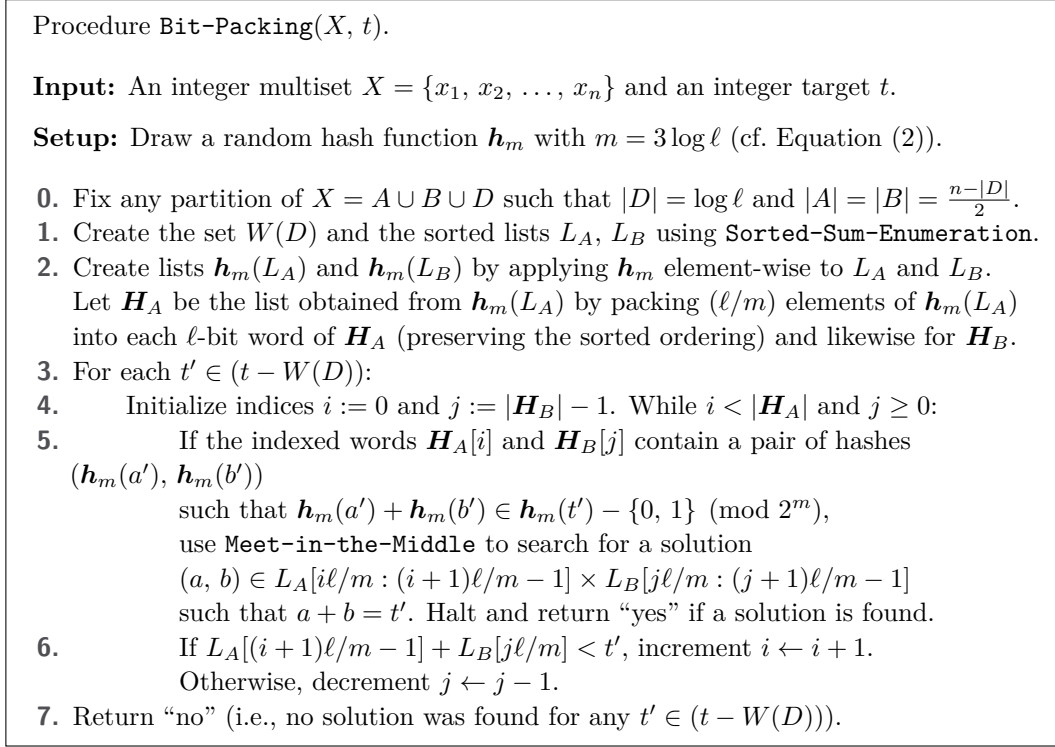
Bit-Packing works by packing ℓ/m hashed values into a single word via our pseudolinear hash function h_m , for $m = 3 \log \ell$, while preserving enough information to run **Meet-in-the-Middle** on the lists of hashed and packed values. This allows us to compare two length- (ℓ/m) sublists of L_A and L_B in constant expected time in the circuit RAM model, since each hashed and packed sublist fits into a constant number of words, instead of time $O(\ell/m)$ like the original **Meet-in-the-Middle** algorithm. So far, this is essentially the approach taken by [5] in their bit-packing algorithm for 3SUM. However, in our context the $O(\ell/m)$ speedup described above is offset by the following issue: if we follow the original **Meet-in-the-Middle** setup and take $|A| = |B| = \frac{n}{2}$, the lists L_A and L_B may have length $\Omega(2^{n/2})$, increasing the runtime.

To deal with this, we set aside a small set $D \subseteq X$ of $|D| = \log \ell$ many input elements and solve the remaining subinstance $X \setminus D$ for each shifted target $t' \in (t - W(D))$. Removing the elements in D shortens the lists L_A and L_B , which are now formed from the elements in $X \setminus D$, and allows us to enumerate and pack them quickly. Balancing the overhead of solving each of these subinstances against the savings described earlier, we get the claimed speedup $\Omega(\ell^{1/2}/\log \ell) \geq \Omega(n^{1/2}/\log n)$.

Proof of Correctness for **Bit-Packing.** The algorithm outputs “yes” only when a triple $(a, b, t') \in L_A \times L_B \times (t - W(D))$ with $a + b = t'$ is found, so it never returns a false positive.

⁵ As explained in Section 1.1, we assume $\ell = \text{poly}(n)$ throughout; however, our results for **Bit-Packing** apply for superpolynomial word length as long as $\ell = O(2^{n/3})$. We leave the extensional modifications to the proofs for this regime as an exercise for the interested reader.

⁶ By halting **Bit-Packing** and returning “no” if its runtime exceeds $C \cdot 2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell$ for a large enough constant $C > 0$, we get an one-sided error algorithm with success probability $\geq 3/4$, as claimed in Section 1.2.



■ **Figure 3** The Bit-Packing algorithm.

It remains to show that for any $t' \in (t - W(D))$, we are guaranteed to find a shifted solution $(a, b) \in L_A \times L_B$ such that $a + b = t'$, if one exists. Without loss of generality, we consider two sublists $L_A[i\ell/m : (i+1)\ell/m - 1]$ and $L_B[j\ell/m : (j+1)\ell/m - 1]$ that contain such a shifted solution (a, b) and correspond to two packed words $\mathbf{H}_A[i]$ and $\mathbf{H}_B[j]$ for some indices i and j . The existence of such a shifted solution $a + b = t'$ combined with the condition in Line 6 ensures that the algorithm will not step past either the packed word $\mathbf{H}_A[i]$ or $\mathbf{H}_B[j]$ before reaching the other one, so the algorithm will compare these two packed words at some point. Following Lemma 2, we have $\mathbf{h}_m(a) + \mathbf{h}_m(b) \in \mathbf{h}_m(t') - \{0, 1\} \pmod{2^m}$, satisfying the condition in Line 5. Thus we are guaranteed to find the shifted solution (a, b) by running **Meet-in-the-Middle** to check all pairs in $L_A[i\ell/m : (i+1)\ell/m - 1] \times L_B[j\ell/m : (j+1)\ell/m - 1]$. ◀

Proof of Runtime for Bit-Packing. Recalling the assumption $\ell = \text{poly}(n)$:

- Line 1 takes time $O(2^{|A|} + 2^{|B|} + 2^{|D|}) = O(2^{n/2} \cdot \ell^{-1/2} + \ell) = O(2^{n/2} \cdot \ell^{-1/2})$ by Lemma 4, for the choices of $|A|$, $|B|$, and $|D|$.
- Line 2 takes time $(|L_A| + |L_B|) \cdot O(\log \ell) = O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell)$, where $O(\log \ell)$ bounds the time for each evaluation of the hash function \mathbf{h}_m .
- Line 3 (the outer loop) is performed for at most $|W(D)| \leq 2^{|D|} = \ell$ iterations.
- Line 4 (the inner loop) is performed for at most $(|L_A| + |L_B|) \cdot \frac{1}{\ell/m} = O(2^{n/2} \cdot \ell^{-3/2} \cdot \log \ell)$ iterations, since each iteration either increments $i \leftarrow i + 1$ or decrements $j \leftarrow j - 1$.
- Line 5: (i) Checking whether the “If” condition holds for any two words $\mathbf{H}_A[i]$ and $\mathbf{H}_B[j]$ requires a single \mathbf{AC}^0 operation on three words, taking constant time in the circuit RAM model. (ii) Finding a solution (a, b) using **Meet-in-the-Middle** on the two length- (ℓ/m) sublists takes time $O(\ell/m) = O(\ell/\log \ell)$.

The “If” test is passed (i) at most once for a correct solution and (ii) each time we encounter a hash collision. Note that the sequence of pairs of words $(\mathbf{H}_A[i], \mathbf{H}_B[j])$ we compare is completely determined by L_A and L_B and is unaffected by the outcome of the random hash function \mathbf{h}_m . Thus by Lemma 2, each of the $(\ell/m)^2$ hash pairs in $(\mathbf{H}_A[i], \mathbf{H}_B[j])$ incurs a collision with probability $O(2^{-m})$. By a union bound, the expected time taken for Line 5 because of hash collisions is at most $(\ell/m)^2 \cdot O(2^{-m}) \cdot O(\ell/m) = O(1/\log^3(\ell)) = o_n(1)$, since $m = 3 \log \ell$ and $\ell = \Omega(n)$.

■ Line 6 clearly takes time $O_n(1)$.

Consequently, **Bit-Packing** has expected runtime

$$\begin{aligned} \text{TIME}(n, \ell) &= O(2^{n/2} \cdot \ell^{-1/2} + \ell) + O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell) && \text{Lines 1 and 2} \\ &+ 1 \cdot (O_n(1) + O(\ell/\log \ell) + O_n(1)) && \text{Lines 3 to 6} \\ &+ \ell \cdot O(2^{n/2} \cdot \ell^{-3/2} \cdot \log \ell) \cdot (O_n(1) + o_n(1) + O_n(1)) && \text{Lines 3 to 6} \\ &= O(2^{n/2} \cdot \ell^{-1/2} \cdot \log \ell). && \blacktriangleleft \end{aligned}$$

► **Observation 9** (Adapting **Bit-Packing** to Word RAM). *In the word RAM model, multiplication and evaluation of our pseudolinear hash function \mathbf{h}_m each take constant time. Hence, for any word length $\ell = \Omega(n)$ we can get a variant of **Bit-Packing** with expected runtime $O(2^{n/2} \cdot n^{-1/2} \cdot \log n)$, essentially by performing **Bit-Packing** as if the word length were $\ell' := 0.1n$. By a similar conversion from [5]:*

Run **Bit-Packing** as if the word length were $\ell' = 0.1n$, which results in the modified parameters $m' = 3 \log \ell'$, $|D'| = \log \ell'$, and $|A'| = |B'| = (n - |D'|)/2$ etc., except for two modifications:

1. Line 2 packs $q' := \min\{\ell', \ell\}/m' = \Theta(\frac{n}{\log n})$ many m' -bit hashes into each word of $\mathbf{H}_{A'}$, $\mathbf{H}_{B'}$, so every (ℓ'/m') hashes are stored in $\ell'/(m'q') = \Theta_n(1)$ words rather than a single word.
2. Before Line 5, create a table that memoizes the result of every comparison of two ℓ' -bit strings in time $(2^{\ell'})^2 \cdot \text{poly}(\ell') = O(2^{0.21n})$. This table can then be accessed via a $2^{\ell'}/(m'q') = \Theta_n(1)$ -word index in constant time. Line 5 replaces the constant-time AC^0 circuit RAM operation on two ℓ' -bit strings with a constant-time lookup into this table.

Compared with running **Bit-Packing** itself for $\ell' = 0.1n$, the only difference of this variant is that $\mathbf{H}_{A'}$ and $\mathbf{H}_{B'}$ are stored in $\Theta_n(1)$ times as many words, so the correctness is easy to check. The expected time taken for the collisions in each execution of Line 5 is $(q')^2 \cdot O(2^{-m'}) \cdot O(q') = o_n(1)$. Thus, the overall runtime is as claimed:

$$\begin{aligned} &\underbrace{O(2^{|A'|} + 2^{|B'|} + 2^{|D'|} + 2^{0.21n})}_{\text{Lines 1 and 2}} + \underbrace{O(q') + 2^{|D'|} \cdot O((2^{|A'|} + 2^{|B'|})/q')}_{\text{Lines 3 to 6; solution versus collisions}} \\ &= O(2^{n/2} \cdot n^{-1/2} \cdot \log n). \end{aligned}$$

4 $\Omega(n^{0.01})$ -Factor Speedup via Orthogonal Vectors and the Representation Method

Our second algorithm, **Representation-OV** (see Figure 4), achieves a speedup of $\Omega(\ell^\gamma) \geq \Omega(n^\gamma)$ over **Meet-in-the-Middle** for a constant $\gamma > 0.01$.⁷ While this is a smaller speedup than that achieved by the **Bit-Packing** algorithm, the **Representation-OV** algorithm does

⁷ Similar to Section 3, while we investigate **Representation-OV** in the regime $\ell = \text{poly}(n)$, analogous results apply for word length ℓ as large as $O(2^{cn})$ for an absolute constant $c > 0$.

not use “bit tricks”. Instead, **Representation-OV** combines **Meet-in-the-Middle** with the *representation method* of Howgrave-Graham and Joux [17], so as to reduce the Subset Sum problem to many small instances of the Orthogonal Vectors (OV) problem: namely, instances with $O(\ell/\log \ell)$ many binary vectors of dimension $\Theta(\log \ell) = \Theta(\log n)$. Such instances of OV can be solved quickly through a single AC^0 word operation in the circuit RAM model (or through constantly many operations in the word RAM model after an initial memoization step), which leads to our speedup.

The high-level idea behind our algorithm is to partition the input $X = A \cup B \cup C$ into two large subsets⁸ A and B and one small subset C of size $|C| = \Theta(\log \ell)$, and to run **Meet-in-the-Middle** on two lists formed from subsets of $(A \cup C)$ and $(B \cup C)$. We describe in more detail below just how these lists are formed, but roughly speaking they are created by modifying the representation method (in a way somewhat similar to the algorithm of Nederlof and Węgrzycki [20]) to ensure that the lists are not too long. Further, to eliminate the false positives due to overlapping subsets of C , we exploit a fast implementation of a function that computes a batch of small instances of Orthogonal Vectors. (To see the relevance of the Orthogonal Vectors problem in this context, note that a solution to an instance of OV on k -dimensional Boolean vectors corresponds to two disjoint subsets of the set $[k]$.) Before giving more details we provide some helpful notation:

Notation and setup. We write $\mathcal{Q}(C) := \{T \mid T \subseteq C \text{ and } |T| \leq \frac{|C|}{4}\}$ to denote the collection of all quartersets for C . While $\mathcal{Q}(C)$ is useful for intuition, in fact, as explained below, our algorithm will use a slight variant of it: we define $\mathcal{Q}^{+\varepsilon_2}(C)$ to be the collection of all subsets of size at most $(1 + \varepsilon_2)\frac{|C|}{4}$, where $\varepsilon_2 > 0$ is a small constant that is fixed in the detailed description of the algorithm.

We denote by **OV** the Boolean function on $2\ell = \text{poly}(n)$ many input bits that takes as input two lists $(x^1, \dots, x^{\ell/|C|})$, $(y^1, \dots, y^{\ell/|C|})$ of (at most) $\ell/|C|$ many binary vectors each, where each binary vector $x^i, y^j \in \{0, 1\}^{|C|}$, and returns 1 if and only if the two lists contain an orthogonal pair, i.e., a pair (i, j) such that $x_k^i \cdot y_k^j = 0$ for every $1 \leq k \leq |C|$. It is easy to see that **OV** is an AC^0 operation on two ℓ -bit words, and thus it takes constant time in the circuit RAM model.

We return to the intuitive overview of our approach. At a high level, the representation method works by first expanding the search space of possible solutions; for our algorithm this is done by writing down the list $\mathcal{Q}^{+\varepsilon_2}(C)$ of all “near-quartersets”. If a certain Subset Sum solution $S \subseteq X$ satisfies $|S \cap C| \leq \frac{|C|}{2}$, the restricted solution $S \cap C = Q_1 \cup Q_2$ is the union of *many* different pairs of disjoint quartersets, namely $Q_1, Q_2 \in \mathcal{Q}(C)$ with $Q_1 \cap Q_2 = \emptyset$. In fact, we work on the list of near-quartersets, $\mathcal{Q}^{+\varepsilon_2}(C)$, rather than $\mathcal{Q}(C)$, so as to cover all disjoint pairs (Q_1, Q_2) with $|Q_1| + |Q_2| \approx |C|/2$.

We then *filter* the list $\mathcal{Q}^{+\varepsilon_2}(C)$: given a random prime modulus \mathbf{p} we extract those near-quartersets that fall into two particular residue classes that sum to $\Sigma(S \cap C) \pmod{\mathbf{p}}$. With appropriate preprocessing checks and parameter settings, this significantly reduces the search space while ensuring that we retain some disjoint pair of near-quartersets $Q_1, Q_2 \in \mathcal{Q}^{+\varepsilon_2}(C)$ that recover the restricted solution, $Q_1 \cup Q_2 = S \cap C$ with $Q_1 \cap Q_2 = \emptyset$.

Finally, we use a modified **Meet-in-the-Middle** procedure to search for a solution, i.e., two sum-subset couples $(a, Q_1) \in (L_A \times \mathcal{Q}^{+\varepsilon_2}(C))$, $(b, Q_2) \in (L_B \times \mathcal{Q}^{+\varepsilon_2}(C))$ with $a + \Sigma(Q_1) + b + \Sigma(Q_2) = \Sigma(S \cap A) + \Sigma(S \cap B) + \Sigma(S \cap C) = \Sigma(S) = t$, verifying $Q_1 \cap Q_2 = \emptyset$ via the Boolean function **OV**.

⁸ Technically, sub-multisets. We make the same simplification hereafter.

Procedure **Representation-0V**(X, t).

Input: An integer multiset $X = \{x_1, x_2, \dots, x_n\}$ and an integer target t .

Setup: Constants $\varepsilon_1 \approx 0.1579$ and $\varepsilon_2 \approx 0.2427$, the solutions to Equations (3) and (4). Parameters $\beta \approx 1.1186$, $\lambda \approx 0.0202$, $s(\ell)$, and $k(n, \ell)$, all to be specified in the proof.

A uniform random prime modulus $p \sim \mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$.

0. Fix any partition of $X = A \cup B \cup C$ such that $|C| = \beta \log(\frac{\ell}{\beta \log \ell})$ and $|A| = |B| = \frac{n-|C|}{2}$.
1. Use Lemma 13 to solve (X, t) if there exists a solution $S \subseteq X$ with $|S \cap C| \notin (1 \pm \varepsilon_1) \frac{|C|}{2}$.
2. Use Lemma 12 to solve (X, t) if $|W(C)| \leq 2^{|C|} \cdot \ell^{-\lambda}$.
3. Create the sorted lists L_A and L_B using **Sorted-Sum-Enumeration**. Let $\{\mathbf{L}_{A,i}\}_{i \in [p]}$ be the sorted sublists given by $\mathbf{L}_{A,i} := \{a \in L_A \mid a \equiv_p i\}$ and likewise for $\{\mathbf{L}_{B,i}\}_{i \in [p]}$.
Create the collection $\mathcal{Q}^{+\varepsilon_2}(C) = \{Q \mid Q \subseteq C \text{ and } |Q| \leq (1 + \varepsilon_2) \frac{|C|}{4}\}$.
4. Repeat Lines 5 to 6 either $s(\ell)$ times or until a total of $k(n, \ell)$ many sum-subset couples have been created in Line 5 (whichever comes first):
5. Sample a uniform random residue $r \sim [p]$. Then use Lemma 15 and the subroutine **Residue-Couple-List** to create the sorted lists $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$:
 $\mathbf{R}_{A,r} = \{(a' := a + \Sigma(Q_1), Q_1) \mid (a, Q_1) \in L_A \times \mathcal{Q}^{+\varepsilon_2}(C) \text{ with } a' \equiv_p r\}$ and
 $\mathbf{R}_{B,r} = \{(b' := b + \Sigma(Q_2), Q_2) \mid (b, Q_2) \in L_B \times \mathcal{Q}^{+\varepsilon_2}(C) \text{ with } b' \equiv_p (t - r)\}$.
▷ In fact, $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ are stored in a succinct format by **Residue-Couple-List**.
6. Use Lemma 15 (based on **Meet-in-the-Middle** and the Boolean function **0V**) to search the sorted lists $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ for a solution $(a', Q_1), (b', Q_2)$ such that
 $a' + b' = t$ and $Q_1 \cap Q_2 = \emptyset$. Halt and return “yes” if a solution is found.
7. Return “no” (i.e., no solution was found). ▷ Possibly a false negative.

■ **Figure 4** The **Representation-0V** algorithm.

► **Theorem 10.** *Representation-0V is a one-sided error randomized algorithm for the Subset Sum problem (with no false positives) with worst-case runtime $O(2^{n/2} \cdot \ell^{-\gamma}) \leq O(2^{n/2} \cdot n^{-\gamma})$, for some constant $\gamma > 0.01$, and a success probability of $1/3$ in the circuit RAM model.*

The proofs of correctness and runtime below rely on several auxiliary lemmas, namely Lemmas 12–15. These lemmas, along with an adaptation of the result to the Word RAM model, can be found in Appendix A.

Proof of Correctness for Representation-0V. The constants $\varepsilon_1 \approx 0.1579$ and $\varepsilon_2 \approx 0.2427$ given in the description of the algorithm are the solutions to the following equations:

$$\frac{1-H((1-\varepsilon_1)/2)}{(1-\varepsilon_1)/2} = 1 - H\left(\frac{1-\varepsilon_2}{2}\right), \quad (3)$$

$$1 + \varepsilon_1 - 3H\left(\frac{1-\varepsilon_1}{2}\right) = -2H\left(\frac{1+\varepsilon_2}{4}\right). \quad (4)$$

Also, we set $\beta := \frac{1}{H((1+\varepsilon_2)/4)} \approx 1.1186$ and $\lambda := (1 - 10^{-5}) \cdot \frac{1-\varepsilon_1}{2} \cdot \beta \cdot (1 - H(\frac{1-\varepsilon_2}{2})) \approx 0.0202$.

39:12 Subset Sum in Time $2^{n/2}/\text{poly}(n)$

Lines 1 and 2 preprocess the instance (X, t) , solving it deterministically via Lemmas 12 and 13 unless both Conditions (1) and (2) hold:

Condition (1). (X, t) is either a “yes” instance with $|S \cap C| \in [(1 - \varepsilon_1)\frac{|C|}{2} : \frac{|C|}{2}]$ for each solution $S \subseteq X$,⁹ or a “no” instance.

Condition (2). $|W(C)| > 2^{|C|} \cdot \ell^{-\lambda}$. Note that this implies $|W(T)| > 2^{|T|} \cdot \ell^{-\lambda}$ for each $T \subseteq C$.

Lines 3 to 7 accept only if a solution $t = a' + b' = (a + \Sigma(Q_1)) + (b + \Sigma(Q_2))$ is found in Line 6, for which $a \in L_A$, $b \in L_B$, and $Q_1, Q_2 \in \mathcal{Q}^{+\varepsilon_2}(C)$ are disjoint. As a consequence, the algorithm never reports false positives.

It remains to show that Lines 4 to 6 accept a “yes” instance (X, t) with probability at least $1/3$ when (X, t) satisfies Conditions (1) and (2). Consider a solution $S \subseteq X$ and the following set W' containing distinct sums of all “ ε_2 -balanced” subsets of $S \cap C$:

$$W' := \{ \Sigma(Q) \mid Q \subseteq (S \cap C) \text{ and } |Q| \in (1 \pm \varepsilon_2)\frac{|S \cap C|}{2} \}.$$

Line 3 creates the residue sublists $L_A = \{\mathbf{L}_{A,i}\}_{i \in [p]}$ and $L_B = \{\mathbf{L}_{B,i}\}_{i \in [p]}$. We say that a residue $i \in [p]$ is *good* if it satisfies $i - \Sigma(S \cap A) \in (W' \bmod \mathbf{p})$, namely there exists a subset $Q_1 \subseteq (S \cap C)$ of size $|Q_1| \in (1 \pm \varepsilon_2)\frac{|S \cap C|}{2}$ such that $\Sigma(S \cap A) + \Sigma(Q_1) \equiv_{\mathbf{p}} i$. On sampling a good residue $\mathbf{r} = i$ in Line 5, both Q_1 and $Q_2 := (S \cap C) \setminus Q_1$ are of size at most $(1 + \varepsilon_2)\frac{|S \cap C|}{2} \leq (1 + \varepsilon_2)\frac{|C|}{4}$, so they are included in the collection $\mathcal{Q}^{+\varepsilon_2}(C)$ and, respectively, in the lists $\mathbf{R}_{A,\mathbf{r}}$ and $\mathbf{R}_{B,\mathbf{r}}$ created in Line 5. Then using Lemma 15, we are ensured to find the solution $S = (S \cap A) \cup (S \cap B) \cup (Q_1 \cup Q_2)$.

Hence it suffices to **(i)** lower bound the probability that at least one of the $s(\ell)$ samples $\mathbf{r} \sim [p]$ is good, and **(ii)** upper bound the probability that these samples generate a total of $k(n, \ell)$ or more sum-subset couples.

(i). We claim that the size of set W' is at least $\Omega(2^{|S \cap C|} \cdot \ell^{-\lambda}) \geq \tilde{\Omega}(\ell^{(1-\varepsilon_1)\cdot\beta/2-\lambda})$ and is at most $2^{|S \cap C|} \leq \ell^{\beta/2}$. The lower bound on the size comes from a combination of two observations. First, the set $(S \cap C)$ has at least $2^{|S \cap C|} \cdot \ell^{-\lambda}$ many distinct subset sums, by Condition (2). Second, the number of subsets $Q \subseteq (S \cap C)$ of size $|Q| \notin (1 \pm \varepsilon_2)\frac{|S \cap C|}{2}$ is at most $2 \cdot 2^{H(\frac{1-\varepsilon_2}{2})\cdot|S \cap C|} = o(2^{|S \cap C|} \cdot \ell^{-\lambda})$, given Stirling’s approximation (Equation (1)) and the technical condition

$$\frac{|S \cap C|}{\log \ell} \cdot (1 - H(\frac{1-\varepsilon_2}{2})) \geq (1 - o_n(1)) \cdot \frac{1-\varepsilon_1}{2} \cdot \beta \cdot (1 - H(\frac{1-\varepsilon_2}{2})) > \lambda,$$

which is true for our choice of λ .

The upper and lower bounds on $|W'|$ allow us to apply Lemma 14: with probability at least $3/4$ over the modulus $\mathbf{p} \sim \mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$, there are $|W' \bmod \mathbf{p}| = \Omega(|W'|) = \tilde{\Omega}(\ell^{(1-\varepsilon_1)\cdot\beta/2-\lambda})$ many good residues. Conditioned on this event, taking

$$s(\ell) := \tilde{\Theta}(\ell^{1+\lambda+\varepsilon_1\cdot\beta/2}) \geq \tilde{\Omega}\left(\frac{\mathbf{p}}{|W' \bmod \mathbf{p}|}\right)$$

many samples $\mathbf{r} \sim [p]$ yields at least one good residue with probability $\geq 2/3$.

⁹ Given Line 1, a solution $S \subseteq X$ or its complementary $(X \setminus S)$, as a solution to the complementary instance $(X, \Sigma(X) - t)$, must have this property.

(ii). All candidate lists $\{\mathbf{R}_{A,i}\}_{i \in [p]}$, $\{\mathbf{R}_{B,i}\}_{i \in [p]}$ together have a total of

$$(|L_A| + |L_B|) \cdot |\mathcal{Q}^{+\varepsilon_2}(C)| \leq 2 \cdot 2^{(n-|C|)/2} \cdot 2^{|C|/\beta} = O(2^{n/2} \cdot \ell^{1-\beta/2})$$

sum-subset couples, by construction (Line 3), the choices of $|A|$, $|B|$, $|C|$, and Stirling's approximation (Equation (1)). For any outcome of the random modulus $\mathbf{p} = p = \Theta(\ell^{1+\beta/2})$, a number of $s(\ell)$ samples $\mathbf{r} \sim [p]$ generate a total of $O(2^{n/2} \cdot \ell^{1-\beta/2}) \cdot s(\ell)/p = \tilde{O}(2^{n/2} \cdot \ell^{-(1-\varepsilon_1/2) \cdot \beta - (1+\lambda)})$ sum-subset couples in expectation. By setting a large enough cutoff

$$k(n, \ell) := \tilde{\Theta}(2^{n/2} \cdot \ell^{-(1-\varepsilon_1/2) \cdot \beta - (1+\lambda)}),$$

the probability that $k(n, \ell)$ or more sum-subset couples are generated is at most $1/6$.

Overall, our algorithm succeeds with probability $\geq (3/4) \cdot (2/3) - (1/6) = 1/3$. ◀

Proof of Runtime for Representation-0V. Recalling the assumption $\ell = \text{poly}(n)$:

- Line 1 takes time $\tilde{O}(2^{n/2} \cdot \ell^{-(1-H((1-\varepsilon_1)/2)) \cdot \beta/2})$, by Lemma 13.
- Line 2 takes time $\tilde{O}(2^{n/2} \cdot \ell^{-\lambda/2})$, by Lemma 12.
- Line 3 takes time $O(2^{|A|} + 2^{|B|} + 2^{|C|}) \cdot O(\log \ell) = \tilde{O}(2^{n/2} \cdot \ell^{-\beta/2})$, by Lemma 4, the choices of $|A|$, $|B|$, $|C|$, and that the modulo operation in the circuit RAM model takes time $O(\log \ell)$.
- Lines 4 to 6 take time $\tilde{O}(k(n, \ell)) = \tilde{O}(2^{n/2} \cdot \ell^{-(1-\varepsilon_1/2) \cdot \beta - (1+\lambda)})$, because a single iteration (Lemma 15) takes time $\tilde{O}(|\mathbf{R}_{A,\mathbf{r}}| + |\mathbf{R}_{B,\mathbf{r}}|)$ and by construction we create a total of at most $\sum_{\mathbf{r}} (|\mathbf{R}_{A,\mathbf{r}}| + |\mathbf{R}_{B,\mathbf{r}}|) \leq k(n, \ell)$ many sum-subset couples.

The runtime of Line 3 is dominated by that of Line 1, so the bottleneck occurs in Line 1, Line 2, or Lines 4 to 6. For the choices of constants given in the algorithm, we achieve a speedup of $\Omega(\ell^\gamma)$ for any constant $\gamma \in (0, \gamma_*)$, where the

$$\gamma_* := \min \left\{ \lambda/2, \quad (1 - H(\frac{1-\varepsilon_1}{2})) \cdot \beta/2, \quad (1 - \varepsilon_1/2) \cdot \beta - (1 + \lambda) \right\} = \lambda/2 \approx 0.0101. \blacktriangleleft$$

5 Subset Sum in Time $O(2^{n/2} \cdot n^{-0.5023})$

Our last algorithm is a delicate combination of **Bit-Packing** and **Representation-0V**. Our main result, Theorem 11, demonstrates that problem-specific features of Subset Sum can be exploited to obtain an additional nontrivial time savings beyond what can be achieved by augmenting **Meet-in-the-Middle** with generic bit-packing tricks. Although the bookkeeping to analyze the algorithm of this section is somewhat intricate, many of the ideas in this section are previewed in Section 4.

► **Theorem 11.** *Packed-Representation-0V is a one-sided error randomized algorithm for the Subset Sum problem (with no false positives) with worst-case runtime $O(2^{n/2} \cdot n^{-(1/2+\gamma)})$, for some constant $\gamma > 0.0023$, and success probability at least $1/12$ in the circuit RAM model.*

This extended abstract presentation omits the proof of Theorem 11 and the adaptation of Theorem 11 to the word RAM model. For the proof, the reader is referred to the full version of the paper.¹⁰ Below, we describe a difficulty that arises in the attempt to directly combine the two building block algorithms, as well as the new AC^0 operation used in the approach.

First, we describe a difficulty that arises when attempting to combine our two previous algorithms. **Bit-Packing** saves time by removing a subset $D \subseteq X$ from the input, then running a **Meet-in-the-Middle** variant on the resulting subinstance multiple times, while

¹⁰<https://arxiv.org/abs/2301.07134>

Representation-OV runs a **Meet-in-the-Middle** variant on multiple subinstances indexed by residue classes modulo a random prime p . This presents a problem: to get the time savings from bit packing, we would like to reuse subinstances multiple times, but to get the time savings from the representation method we need to build separate subinstances with respect to each residue class $(\text{mod } p)$ that contains elements of $W(D)$. To solve this problem, we construct D in a way that ensures the elements of $W(D)$ fall into few residue classes $(\text{mod } p)$. Specifically, we fix p and consider two cases. In one case, the elements of X fall into few residue classes $(\text{mod } p)$ and it is possible to choose a small set D such that the elements of D (and $W(D)$) fall into very few residue classes $(\text{mod } p)$. In this case we need to construct just $|W(D) \text{ mod } p| = \text{polylog}(n)$ distinct subinstances. In the other case, the elements of X fall into many residue classes $(\text{mod } p)$. Here a carefully selected D satisfies the weaker bound $|W(D) \text{ mod } p| = \tilde{O}(n^\delta)$ for a small constant $\delta > 0$, increasing the number of subinstances we need to construct. However, the fact that the elements of X fall into many residue classes $(\text{mod } p)$ lets us select a larger set C such that the subset sums in $W(S \cap C)$ distribute well $(\text{mod } p)$ for any solution S , offsetting the increase in runtime.

Like the AC^0 operation **OV** in the **Representation-OV** algorithm, the core of our new algorithm is another AC^0 operation, **Hash-OV**, that solves Orthogonal Vectors on small instances. Like **OV**, **Hash-OV** takes as input two ℓ -bit words containing multiple bit vectors of length $O(\log n)$, but now the bit vectors in either word may come from two or more lists, and each list is indexed by the m -bit hash $h_m(s)$ of a corresponding subset sum s , for $m = 3 \log \ell$. Thus **Hash-OV** may solve multiple small Orthogonal Vectors instances at once.

6 Extensions and Future Work

Our results open up several natural directions for future investigation:

- **Derandomization:** All of the $2^{n/2}/\text{poly}(n)$ -time algorithms we have given for Subset Sum use randomness. Can our results be extended to achieve deterministic algorithms with worst-case running time $2^{n/2}/\text{poly}(n)$?
- **Counting:** It is straightforward to modify the **Meet-in-the-Middle** algorithm to output a count of the number of Subset Sum solutions in time $O(2^{n/2})$ (essentially, by keeping track of the multiplicity with which each value occurs in each list L_A, L_B). Can our techniques be extended to give counting algorithms for Subset Sum that run in time $2^{n/2}/\text{poly}(n)$? In time $2^{n/2}/n^{0.501}$?
- **Faster runtimes:** Finally, an obvious goal is to quantitatively strengthen our results by developing faster algorithms for worst-case Subset Sum. It would be particularly interesting to achieve running times of the form $2^{n/2}/f(n)$ for some $f(n) = n^{\omega(1)}$.

References

- 1 Amir Abboud and Karl Bringmann. Tighter Connections Between Formula-SAT and Shaving Logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 2 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. *ACM Transactions on Algorithms (TALG)*, 18(1):1–22, 2022.
- 3 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.

- 4 Per Austrin, Mikko Koivisto, Petteri Kaski, and Jesper Nederlof. Dense subset sum may be the hardest. *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, pages 13:1–13:14, 2016.
- 5 Ilya Baran, Erik D Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. In *Workshop on Algorithms and Data Structures*, pages 409–421. Springer, 2005.
- 6 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 364–385. Springer, 2011.
- 7 Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
- 8 Timothy Chan. The art of shaving logs. Presentation at WADS 2013, slides available at https://tmc.web.engr.illinois.edu/talks/wads13_talk.pdf, 2013.
- 9 Timothy M. Chan. The art of shaving logs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures – 13th International Symposium, WADS 2013, London, ON, Canada, August 12–14, 2013. Proceedings*, volume 8037 of *Lecture Notes in Computer Science*, page 231. Springer, 2013.
- 10 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A Servedio. Average-case subset balancing problems. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–778. SIAM, 2022.
- 11 Marek Cygan, Fedor Fomin, Bart M.P. Jansen, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Open problems for fpt school. Available at <https://fptschool.mimuw.edu.pl/opl.pdf>, 2014.
- 12 Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.
- 13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.
- 14 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. doi:10.1145/3185378.
- 15 Godfrey Harold Hardy, Edward Maitland Wright, et al. *An introduction to the theory of numbers*. Oxford university press, 1979.
- 16 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.
- 17 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.
- 18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103. Plenum Press, New York, 1972.
- 19 Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Wegrzycki. Equal-subset-sum faster than the meet-in-the-middle. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9–11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 73:1–73:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.73.
- 20 Jesper Nederlof and Karol Wegrzycki. Improving Schroepel and Shamir’s algorithm for subset sum via orthogonal vectors. In *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021*, pages 1670–1683. ACM, 2021.
- 21 David Pisinger. Dynamic programming on the word RAM. *Algorithmica*, 35(2):128–145, 2003.
- 22 Richard Schroepel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM journal on Computing*, 10(3):456–464, 1981.
- 23 Gerhard J Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.

A

 Auxiliary Lemmas for Representation-0V

A first “preprocessing lemma” stems from the observation that a not-too-large subset $Y \subseteq X$ with few distinct subset sums (i.e., $|W(Y)| \ll 2^{|Y|}$) can help speed up **Meet-in-the-Middle**. This idea goes back to [3, 4], although the condition of “few distinct subset sums” we use refers to sets with polynomially rather than exponentially fewer subset sums than the maximum possible number.

► **Lemma 12** (Speedup via Additive Structure). *Let (X, t) be an n -integer Subset Sum instance. Given as input (X, t) and a subset $Y \subseteq X$ of size $|Y| \leq n/2$ such that $|W(Y)| \leq 2^{|Y|} \cdot \ell^{-\varepsilon}$ for some constant $\varepsilon > 0$, the instance (X, t) can be solved deterministically in time $\tilde{O}(2^{n/2} \cdot \ell^{-\varepsilon/2})$.*

Proof. Fix any size- $\binom{n+\varepsilon \log \ell}{2}$ subset A such that $Y \subseteq A \subseteq X$; we have $|W(A)| \leq 2^{|A \setminus Y|} \cdot |W(Y)| \leq 2^{n/2} \cdot \ell^{-\varepsilon/2}$. We also have $|W(X \setminus A)| \leq 2^{|X \setminus A|} = 2^{n/2} \cdot \ell^{-\varepsilon/2}$. By Lemmas 4 and 5, it takes overall time $O(2^{n/2} \cdot \ell^{-\varepsilon/2} \cdot \log n)$ in the regime $\ell = \text{poly}(n)$ to create the sorted lists $L_A, L_{X \setminus A}$ and to run **Meet-in-the-Middle**. ◀

Another useful preprocessing lemma shows that the existence of a solution that is “unbalanced” vis-a-vis a given small subset yields a speedup:

► **Lemma 13** (Speedup via Unbalanced Solutions). *Let (X, t) be an n -integer Subset Sum instance that has a solution. Given as input (X, t) and a subset $Y \subseteq X$ of size $|Y| = \beta \log(\ell / (\beta \log \ell))$ for β as specified in **Representation-0V** such that some solution $S \subseteq X$ satisfies $|S \cap Y| \notin (1 \pm \varepsilon) \frac{|Y|}{2}$ for some constant $\varepsilon > 0$, the solution S can be found deterministically in time $\tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$, where the constant $\delta := (1 - H(\frac{1-\varepsilon}{2})) \cdot \beta$.*

Proof. We can assume without loss of generality that the solution S satisfies $|S \cap Y| \leq \frac{|Y|}{2}$ (since either the original instance (X, t) or the complementary instance $(X, \Sigma(X) - t)$ must satisfy this property, and we can attempt both instances and only double the runtime). Hence we can suppose $|S \cap Y| \leq (1 - \varepsilon) \frac{|Y|}{2}$. Then the sorted list L'_Y of the set $\{\Sigma(T) \mid T \subseteq Y \text{ and } |T| \leq (1 - \varepsilon) \frac{|Y|}{2}\}$ can be created in time $O(2^{|Y|}) = O(\ell^\beta) = \text{poly}(n)$ using **Sorted-Sum-Enumeration** (restricted to subsets of sizes $< (1 - \varepsilon) \frac{|Y|}{2}$).

Fix any size- $\binom{n+(\delta/\beta)|Y|}{2}$ subset A with $Y \subseteq A \subseteq X$. In the regime $\ell = \text{poly}(n)$, we can use L'_Y and Lemma 5 to create the sorted list L'_A of $\{\Sigma(T) \mid T \subseteq A \text{ and } |T \cap Y| < (1 - \varepsilon) \frac{|Y|}{2}\}$ in time

$$\tilde{O}(|L'_A|) \leq \tilde{O}(2^{|A \setminus Y|} \cdot |L'_Y|) \leq \tilde{O}(2^{|A \setminus Y|} \cdot 2^{H(\frac{1-\varepsilon}{2}) \cdot |Y|}) \leq \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2}),$$

following Stirling’s approximation (Equation (1)) and the choices of $|Y|$ and $|A|$. Moreover, we can use Lemma 4 to create the sorted list $L_{X \setminus A}$ in time $O(2^{|X \setminus A|}) = \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$.

Provided $|S \cap Y| < (1 - \varepsilon) \frac{|Y|}{2}$, running **Meet-in-the-Middle** on L'_A and $L_{X \setminus A}$ solves (X, t) and takes time $O(|L'_A| + |L_{X \setminus A}|) = \tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$. The overall runtime is $\tilde{O}(2^{n/2} \cdot \ell^{-\delta/2})$. ◀

The next lemma specifies a parameter space within which any set of distinct integers is likely to fall into many residue classes modulo a random prime. This allows us to reduce the search space by considering only solutions that fall into certain residue classes.

► **Lemma 14** (Distribution of Integer Sets modulo Random Primes). *Fix a set Y of at most $|Y| \leq \ell^{\beta/2}$ distinct ℓ -bit integers for β as specified in **Representation-0V**. For a uniform random modulus $\mathbf{p} \sim \mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$, the residue set has size $|(Y \bmod \mathbf{p})| = \Theta(|Y|)$ with probability at least $3/4$.*

Subroutine **Residue-Couple-List**($\{L_{A,i}\}_{i \in [p]}$, $\mathcal{Q}^{+\varepsilon}(C)$, r).

Input: A collection of $p = \text{poly}(\ell)$ sorted sublists $L_A = \bigcup_{i \in [p]} L_{A,i}$, for some subset $A \subseteq X$, indexed by residue class modulo p .

Output: A sorted sum-subset list $R_{A,r}$ with elements in the sum-collection format $(a', \mathcal{Q}_{a'})$.

1. For each $Q \in \mathcal{Q}^{+\varepsilon}(C)$, create the sum-subset sublist $R_Q := f_Q(L_{A,j(Q)})$ by applying f_Q , the element-to-couple operation $a \mapsto (a' := a + \Sigma(Q), Q)$, to the particular input sublist of index $j(Q) := ((r - \Sigma(Q)) \bmod p)$. ▷ Each R_Q is sorted by the sums a' .
2. Let $R_{A,r}$ be the list obtained by merging $\{R_Q\}_{Q \in \mathcal{Q}^{+\varepsilon}(C)}$, sorted by sums $a' = a + \Sigma(Q)$.
For each distinct sum a' , compress all couples $(a', Q_1), (a', Q_2), \dots$ with the same first element $= a'$ into a single data object $(a', \mathcal{Q}_{a'} := \{Q_1, Q_2, \dots\})$.
▷ Note that for each sum a' we have $a' \equiv_p r$ and $|\mathcal{Q}_{a'}| \leq |\mathcal{Q}^{+\varepsilon}(C)|$.

■ **Figure 5** The **Residue-Couple-List** subroutine.

Proof. By the prime number theorem [15, Equation (22.19.3)], there are at least $\frac{\ell^{1+\beta/2}}{(1+\beta/2) \cdot \log \ell}$ primes in $\mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$ (for any sufficiently large ℓ). Given any two distinct integers $y \neq z \in Y$, the difference $|y - z| \leq 2^\ell$ has at most $\frac{\ell}{(1+\beta/2) \cdot \log \ell}$ distinct prime factors in $\mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$. Thus under a uniform random choice of modulus $\mathbf{p} \sim \mathbb{P}[\ell^{1+\beta/2} : 2\ell^{1+\beta/2}]$, the second frequency moment $\mathbf{f}_{(2)} := |\{(y, z) \mid y, z \in Y \text{ with } y \equiv_{\mathbf{p}} z\}|$ has the expectation

$$\mathbf{E}_{\mathbf{p}}[\mathbf{f}_{(2)}] = |\{y = z \in Y\}| + |\{y \neq z \in Y\}| \cdot \ell^{-\beta/2} = |Y| + (|Y|^2 - |Y|) \cdot \ell^{-\beta/2} \leq 2|Y|.$$

Therefore, with an arbitrarily high constant probability, we have $\mathbf{f}_{(2)} = O(|Y|)$ and, by the Cauchy-Schwarz inequality $|Y \bmod \mathbf{p}| \cdot \mathbf{f}_{(2)} \geq |Y|^2$,
a residue set of size $|Y \bmod \mathbf{p}| = \Omega(|Y|)$. ◀

► **Lemma 15** (Sorted Lists $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$; Lines 5 and 6). *In Representation-0V:*

(i) *Line 5 uses the subroutine **Residue-Couple-List** to create the sorted lists $\mathbf{R}_{A,r}$, $\mathbf{R}_{B,r}$ in time $\tilde{O}(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$. Moreover, (ii) *Line 6 finds a solution pair $(a', Q_1) \in \mathbf{R}_{A,r}$, $(b', Q_2) \in \mathbf{R}_{B,r}$, if one exists, in time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$.**

Proof. Line 5 creates $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ using the subroutine **Residue-Couple-List** (see Figure 5). First, we claim that **Residue-Couple-List** takes time $\tilde{O}(|\mathbf{R}_A|)$ in the regime $\ell = \text{poly}(n)$:

- Line 1 takes time $\sum_{Q \in \mathcal{Q}^{+\varepsilon_2}(C)} O(|\mathbf{R}_Q|) = O(|\mathbf{R}_{A,r}|)$, since all shifts $\Sigma(Q)$ for $Q \in \mathcal{Q}^{+\varepsilon_2}(C)$ can be precomputed and memoized when the collection $\mathcal{Q}^{+\varepsilon_2}(C)$ is created in Line 3.
- Line 2 builds one sorted list $\mathbf{R}_{A,r}$ from $|\mathcal{Q}^{+\varepsilon_2}(C)| \leq 2^{|C|/\beta} = \ell/(\beta \log \ell) = \text{poly}(n)$ sorted sublists $\{\mathbf{R}_Q\}_{Q \in \mathcal{Q}^{+\varepsilon_2}(C)}$, taking time $O(|\mathbf{R}_{A,r}| \cdot \log n)$ via the classic *merge sort* algorithm.

After Line 5 creates the sorted lists $\mathbf{R}_{A,r} = \{(a', \mathcal{Q}_{a'})\}$ and $\mathbf{R}_{B,r} = \{(b', \mathcal{Q}_{b'})\}$, Line 6 can run **Meet-in-the-Middle** based on the (ordered) indices a' and b' in time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$. This ensures that we discover every pair $(a', \mathcal{Q}_{a'})$, $(b', \mathcal{Q}_{b'})$ such that $a' + b' = t$.

39:18 Subset Sum in Time $2^{n/2}/\text{poly}(n)$

Such a pair yields a solution if and only if it contains two disjoint near-quartersets $Q_1 \in \mathcal{Q}_{a'}$, $Q_2 \in \mathcal{Q}_{b'}$ with $Q_1 \cap Q_2 = \emptyset$, which we can check in constant time by one call of the Boolean function OV . Namely, each near-quarterset $Q \in \mathcal{Q}^{+\varepsilon_2}(C)$ is stored in $|C| < \beta \log \ell$ bits, so each collection $\mathcal{Q}_{a'}$, $\mathcal{Q}_{b'} \subseteq \mathcal{Q}^{+\varepsilon_2}(C)$ can be stored a single word $|C| \cdot |\mathcal{Q}^{+\varepsilon_2}(C)| \leq \ell$. Thus one call of OV suffices to check a given pair $(a', \mathcal{Q}_{a'})$, $(b', \mathcal{Q}_{b'})$. Overall, Line 6 takes time $O(|\mathbf{R}_{A,r}| + |\mathbf{R}_{B,r}|)$. \blacktriangleleft

► **Observation 16** (Adapting Representation-OV to Word RAM). *In the word RAM model, it may take superconstant time to evaluate the Boolean function OV . Similar to the strategy used in Section 3, our word RAM variant avoids this issue by performing Representation-OV as if the word length were $\ell' := 0.1n$ (using sets A' , B' , C' , etc., with appropriate size modifications), except for three modifications:*

1. *In lines 3 and 2, on creating a (sub)collection $\mathcal{Q} \subseteq \mathcal{Q}^{+\varepsilon_2}(C')$ as a bit string, store it in at most $\lceil |\mathcal{Q}^{+\varepsilon_2}(C')| \cdot |C'|/\ell \rceil \leq \lceil \ell'/\ell \rceil = \Theta(1)$ words (since a single word with ℓ bits may be insufficient).*
2. *In Line 3, after creating the collection $\mathcal{Q}^{+\varepsilon_2}(C')$, create a lookup table OV' that memoizes the input-output result of the Boolean function OV on each subcollection pair $\mathcal{Q}_{a'}$, $\mathcal{Q}_{b'} \subseteq \mathcal{Q}^{+\varepsilon_2}(C')$ in time $(2^{|\mathcal{Q}^{+\varepsilon_2}(C')|})^2 \cdot \text{poly}(|\mathcal{Q}^{+\varepsilon_2}(C')|) \leq (2^{\ell'})^2 \cdot \text{poly}(\ell') = O(2^{0.21n})$. This table can then be accessed using a $2^{\lceil \ell'/\ell \rceil} = \Theta(1)$ -word index in constant time.*
3. *Line 6 replaces the Boolean function OV (namely a constant-time AC^0 circuit RAM operation) with constant-time lookup into OV' .*

Compared with running Representation-OV itself for $\ell' = 0.1n$, the only difference of this variant is that $\mathbf{R}_{A,r}$ and $\mathbf{R}_{B,r}$ are stored in $\lceil \ell'/\ell \rceil = \Theta(1)$ times as many words, given $\ell = \Omega(n)$. Hence, it is easy to check the correctness and the runtime $O(2^{n/2} \cdot \ell'^{-\gamma} \cdot \lceil \ell'/\ell \rceil) = O(2^{n/2} \cdot n^{-\gamma})$, for the same constant $\gamma > 0.01$.