Electronic Colloquium on Computational Complexity, Report No. 135 (2026)

Improved Quantum Algorithms for Subset Sum and k-SUM

Nikolai Chukhin * Alexander S. Kulikov † Maksim Levitskii ‡ Ivan Mihajlin §

Abstract

The Subset Sum problem asks whether, given n integers and a target, some subset of the integers sums to the target. Its best known worst-case running time is $O^*(2^{n/2})$ (Horowitz and Sahni, 1974), whereas the best quantum upper bound is $O^*(2^{n/3})$ (Bernstein, Jeffery, Lange, and Meurer, 2013). The k-SUM problem is a parameterized version of Subset Sum asking whether there are k integers that sum to the target. The best classical upper bound for it is $\widetilde{O}(n^{\lceil k/2 \rceil})$, whereas the best quantum running time is $\widetilde{O}(n^{k/3})$ (Tani, 2009). For random instances, a quantum algorithm with running time $\widetilde{O}(n^{\Phi_k})$ is known, where

$$\Phi_k = \frac{2k - \lfloor k/7 \rfloor - \lfloor (k+3)/7 \rfloor}{6}$$

(Schrottenloher, 2021).

We present a new quantum algorithm solving worst-case k-SUM in time $\widetilde{O}(n^{\Psi_k})$, where

$$\Psi_k = \Phi_k - \frac{[k \equiv 3 \bmod 7]}{9} - \frac{[k \equiv 6 \bmod 7]}{18}.$$

The algorithm is not only faster for all k congruent to 3 or 6 modulo 7, but also gives a worst-case guarantee rather than a guarantee restricted to single-solution random instances. Combining our algorithm for 7-SUM with the standard block reduction technique yields an $O^*(2^{2n/7})$ quantum algorithm for Subset Sum, improving the previously known $O^*(2^{n/3})$ algorithm.

*JetBrains Research. Email: buyolitsez1951@gmail.com
†JetBrains Research. Email: alexander.s.kulikov@gmail.com
‡Neapolis University Pafos and JetBrains Research. Email: max.levitsky.5@gmail.com
§JetBrains Research. Email: ivmihajlin@gmail.com

ISSN 1433-8092

1 Complexity of Subset Sum and k-SUM

The Subset Sum problem asks whether, given n integers and a target t, some subset of the integers sums to t. Despite its elementary formulation, Subset Sum is one of Karp's original 21 NP-complete problems [Kar72] and has been studied extensively in exact algorithms [HS74, SS81, DDKS12, Bri17, RW26] and cryptography [MH78, LO85, CR88, IN96, LPS10, FMV16, JW26].

Classical complexity of Subset Sum. A classical algorithm due to Horowitz and Sahni [HS74] solves Subset Sum as follows: it partitions the input into two halves, enumerates all subset sums of each half, and searches for a pair of sums, one from each list, whose total sum equals t, thereby achieving $O^*(2^{n/2})$ time and space[1]. The best known worst-case running time improves this bound only by a polynomial factor, to $O(2^{n/2}n^{-\gamma})$ for a constant $\gamma > 0.5023$ [CJRS23]. Designing an $O^*(2^{(1/2-\varepsilon)n})$ algorithm for Subset Sum is a well-known challenge. For random instances of Subset Sum, the classical running time was reduced to $O^*(2^{0.283n})$ [BBSS20], following the line of work [HJ10, BCJ11]. More recently, Li [Li25] proposed a heuristic ternary-representation algorithm running in time $O^*(2^{0.2400n})$. The space usage of the Horowitz–Sahni [HS74] algorithm can, however, be reduced substantially without changing the exponential running time. Schroeppel and Shamir retained $O^*(2^{n/2})$ time while reducing the space to $O^*(2^{n/4})$ [SS81]. Nederlof and Węgrzycki subsequently lowered the space exponent to 0.249999 [NW21], and it was later reduced to 0.246 while preserving the same running time [BCKM24].

Quantum complexity of Subset Sum. For arbitrary inputs, one can partition the input into three blocks, form one list of subset sums for each block, and then use quantum search to find a pair of entries from two lists whose sum occurs in the third list, yielding a running time of $O^*(2^{n/3})$ [BJLM13]. A different construction attains the same asymptotic bound [AHJ+22]. For random instances, the quantum running time was lowered to $O^*(2^{0.218n})$ [BBSS20], following [BJLM13, HM18]. Li [Li25] also proposed a heuristic quantum algorithm running in time $O^*(2^{0.1843n})$.

Classical complexity of k-SUM. An instance of k-SUM consists of k lists $A_1, \ldots, A_k$, each of length n, and a target t; the task is to choose one entry from each list so that their sum equals t.[2] A standard way to solve Subset Sum is to reduce it to k-SUM for some value of k. The standard block reduction partitions the n input integers into k blocks and lists all subset sums within each block, producing a k-SUM instance whose lists have length $O^*(2^{n/k})$. The meet-in-the-middle algorithm of Horowitz and Sahni [HS74] is precisely the case $k = 2$, and the more elaborate algorithms in [SS81, DDKS12, NW21, BCKM24] reduce Subset Sum to k-SUM for larger values of k.

The folklore meet-in-the-middle algorithm for k-SUM partitions the k lists into two groups, enumerates all sums obtained by selecting one entry from each list in a group, and searches for a pair of complementary sums, yielding the running time $\widetilde{O}(n^{\lceil k/2 \rceil})$. The k-SUM Conjecture [Păt10, AL13] asserts that this algorithm is essentially optimal. The conjecture is known to imply lower bounds

[1] $O^*(\cdot)$ suppresses factors that grow polynomially with the input length, while $\widetilde{O}(\cdot)$ suppresses polylogarithmic factors.

[2] Another common formulation gives a single list A of n integers and asks whether there exist k distinct indices $i_1, \ldots, i_k$ such that $A[i_1] + \cdots + A[i_k] = t$. For fixed k, the two versions have roughly the same complexity [LVWW16]: one of them can be solved in time $\widetilde{O}(n^\alpha)$ if and only if the other can be solved in the same time.

for various computational problems [GO95, Her96, Păt10, AL13, VW13, AV14, JV16, KPP16]. Moreover, its connection to other fine-grained conjectures has recently been established [BKM+24, Lam26].

Quantum complexity of k-SUM. For $k = 2$, Ambainis [Amb07], based on the result of [BDH+05], gave an $\widetilde{O}(n^{2/3})$ quantum algorithm. For all values of k, the upper bound $\widetilde{O}(n^{k/3})$ can be obtained via the Claw Finding algorithm of Tani [Tan09]. For $k = 4$, an algorithm for random instances running in time $\widetilde{O}(n^{6/5})$ was obtained in [BJLM13]. A classical foundation for this line of work on random instances is Wagner's merging-tree algorithm for the Generalized Birthday Problem [Wag02]. *Random instances* of k-SUM were studied in [GNS18, NS20, Sch21]. For single-solution random instances, the best known k-SUM algorithm is due to Schrottenloher [Sch21][3].

Theorem 1.1 ([Sch21])**.** *For every integer $k > 2$, single-solution random instances of k-SUM can be solved in quantum time $\widetilde{O}(n^{\Phi_k})$, where*

$$\Phi_k = \frac{2k - \lfloor k/7 \rfloor - \lfloor (k+3)/7 \rfloor}{6}.$$

Here, *single-solution random instances* are those for which the expected number of witnesses is one; see [Sch21] for the exact definition.

Our Results

Our main result is the quantum algorithm for the *worst-case* k-SUM problem.

Theorem 1.2. *For every integer $k > 3$, worst-case k-SUM can be solved in quantum time $\widetilde{O}(n^{\Psi_k})$, where*[4]

$$\Psi_k = \underbrace{\frac{2k - \lfloor k/7 \rfloor - \lfloor (k+3)/7 \rfloor}{6}}_{\Phi_k} - \frac{[k \equiv 3 \bmod 7]}{9} - \frac{[k \equiv 6 \bmod 7]}{18}.$$

This improves Tani's $\widetilde{O}(n^{k/3})$ algorithm for every value of $k > 3$, and, asymptotically, $\Psi_k = \frac{2k}{7} + O(1)$.

A simple corollary of our k-SUM algorithm is a new upper bound for the Subset Sum problem, which improves the previous worst-case quantum bound $O^*(2^{n/3})$ [BJLM13, AHJ+22].

Corollary 1.3. *The* Subset Sum *problem can be solved in quantum time $O^*(2^{2n/7})$.*

Beyond Subset Sum, our k-SUM algorithm also improves the quantum upper bound for a related variant. In Pigeonhole Equal Subset Sum, one seeks two subsets with equal sums; here, the n input integers are promised to sum to less than $2^n - 1$, so that two such subsets exist by the pigeonhole principle. An $O^*(2^{n/2})$ classical algorithm for this problem [AHJ+22] was improved by Jin and Wu [JW24] to $O^*(2^{0.4n})$, and subsequently by Jin, Williams, and Zhang to $O^*(2^{n/3})$ [JWZ25]. No algorithm faster than $O^*(2^{n/3})$ is known for the Pigeonhole Equal Subset Sum problem, even in the quantum setting. These works also raised the question of whether the *modular* version of this problem can be solved faster than $O^*(2^{0.5n})$. We give a positive answer in the quantum setting, showing that the Pigeonhole Modular Equal Subset Sum problem can be solved in quantum time $O^*(2^{0.453n})$ by a simple reduction to the k-SUM problem and an application of Theorem 1.2. Formal definitions and proofs are provided in Theorem 4.2 of Section 4.

[3] Some of the bounds above were proven for the k-XOR problem, but they carry over to k-SUM.

[4] $[\cdot]$ is the Iverson bracket: for a condition $\mathcal{E}$, $[\mathcal{E}] = 1$ if $\mathcal{E}$ holds, and $[\mathcal{E}] = 0$ otherwise.

Techniques and comparison with previous works. Schrottenloher's algorithm uses a merging tree [Wag02, NS20, Sch21]. For random instances, it recursively combines partial sums and uses modular constraints to keep the intermediate lists small. At the end, it uses Claw Finding to find a solution. Consequently, the analysis of both the success probability and the efficiency depends on the partial sums being sufficiently well distributed. Our algorithm replaces this merging-tree paradigm with a single decomposition into four blocks and one modular filtering step based on a random prime. We then use a quantum walk to reduce the search space, quantum search to find a suitable residue, and, at the very end, Claw Finding to find a solution.

Below, we describe the main ideas underlying our algorithm in greater detail. We group k input lists into four consecutive blocks of sizes approximately $\frac{3}{14}k$, $\frac{4}{14}k$, $\frac{3}{14}k$, and $\frac{4}{14}k$. A tuple in block i records one choice from each list in that block; we write $\mathcal{X}_i$ for the set of these tuples and Σ_i for their sums. This transforms the problem into an unbalanced 4-SUM instance: the first and third grouped lists are smaller, whereas the second and fourth are too large to enumerate.

Instead of constructing the two larger lists explicitly, the algorithm stores m-element subsets $S_2 \subseteq \mathcal{X}_2$ and $S_4 \subseteq \mathcal{X}_4$, where m is a carefully chosen number that is a bit smaller than $\min\{|\mathcal{X}_2|, |\mathcal{X}_4|\}$. However, m is chosen sufficiently small so that there is time to examine S_2 and S_4. We then construct a checker for a fixed pair of subsets S_2, S_4. Hence, using quantum search, one can find suitable subsets S_2, S_4 using roughly $\sqrt{\frac{|\mathcal{X}_2|\cdot|\mathcal{X}_4|}{m^2}}$ calls to the checker.

Before initiating the quantum search, the algorithm chooses a random prime $p = \widetilde{O}(m)$. Within the checker, the algorithm groups the sums from S_2 and S_4 according to their residues modulo p and stores one canonical representative of every nonempty residue bucket in the dictionary. Intuitively, a suitable random prime isolates the relevant exact sums: with constant probability, the block-2 and block-4 sums of a fixed solution are canonical and can therefore be detected by our algorithm.

If the checker terminates without finding a solution involving tuples from S_2 and S_4, the entire dictionary must be recomputed for a new pair of subsets S_2' and S_4'. This recomputation is costly, so we use a quantum walk on pairs (S_2, S_4) instead of the quantum search. Informally, the advantage of the quantum walk over quantum search in this setting is that the quantum walk changes the state (S_2, S_4) to a *neighbouring* state (S_2', S_4'), allowing the dictionary to be recomputed efficiently.

To check a quantum state (S_2, S_4), we use a quantum search over the possible residue $q \in \mathbb{Z}_p$ of the left-hand sum $\Sigma_1 + \Sigma_2$; the right-hand residue is then forced to be $-q$ (assuming that the target is equal to zero). For a fixed q, we iterate over all elements $\alpha_1 \in \mathcal{X}_1$ and store the value of $\Sigma_1(\alpha_1) + \Sigma_2(\alpha_2)$, where α_2 is the canonical tuple from S_2 with the compatible residue, that is, such that $\Sigma_1(\alpha_1) + \Sigma_2(\alpha_2) \equiv q \bmod p$. The right collection is indexed by $\alpha_3 \in \mathcal{X}_3$ and contains the analogous values $-\Sigma_3(\alpha_3) - \Sigma_4(\alpha_4)$ obtained from S_4. We then use the Claw Finding procedure, which searches for an equal exact value in the two collections. If no claw is found, the walk moves to a neighbouring state and updates its data.

2 Preliminaries

For positive integers n and a, $[n] = \{1, \ldots, n\}$ and $[n]^a$ denotes the set of all a-tuples of elements of $[n]$. For a list A, we write $A[i]$ for its i-th entry. For a tuple or vector u, we write $u[i]$ for its i-th coordinate. For a finite set $\mathcal{X}$, $x \in_R \mathcal{X}$ denotes that x is chosen uniformly at random from $\mathcal{X}$. For a condition $\mathcal{E}$, the Iverson bracket $[\mathcal{E}]$ equals 1 if $\mathcal{E}$ holds and 0 otherwise. All logarithms are base two.

2.1 The Subset Sum and k-SUM Problems

Definition 2.1. *An instance of* Subset Sum *consists of a list A of n integers and a target $t \in \mathbb{Z}$. The task is to determine whether there exists a subset $S \subseteq [n]$ such that*

$$\sum_{i \in S} A[i] = t.$$

Definition 2.2. *An instance of k-SUM consists of k lists $A_1, \ldots, A_k$ of integers, each of size n, and a target $t \in \mathbb{Z}$. The task is to find $(i_1, \ldots, i_k) \in [n]^k$ satisfying*

$$A_1[i_1] + \cdots + A_k[i_k] = t,$$

or to report that no such tuple exists.

It is common to assume in k-SUM that the target t is zero, which is achieved by replacing every $A_1[i]$ with $A_1[i] - t$. It is also common to assume that the bit-length of all integers in the k-SUM problem is at most $k \log n + O(\log \log n)$, that is, that every integer has absolute value $\widetilde{O}(n^k)$; this can be achieved by the standard fingerprinting technique [ALW14].

2.2 Prime Numbers

Given an integer $M \geq 2$, one can generate a uniformly random prime from $\{M, \ldots, 2M\}$ in expected time $\log^{O(1)} M$ as follows. One selects a uniformly random integer from $\{M, \ldots, 2M\}$ and tests it for primality in time $\log^{O(1)} M$ [Len02, AKS04]. By the prime number theorem [Had96, dlVP96], the expected number of trials is $O(\log M)$. We also use the following estimate.

Proposition 2.3. *Let $M \geq 2$ and $z \neq 0$ be integers. The probability that a uniformly random prime from $\{M, \ldots, 2M\}$ divides z is*

$$O\left(\frac{\log |z|}{M}\right).$$

Proof. Suppose that d distinct primes from $\{M, \ldots, 2M\}$ divide z. Their product divides $|z|$, and hence $M^d \leq |z|$, which implies $d \leq \log |z| / \log M$. By the prime number theorem [Had96, dlVP96], the interval contains $\Omega(M / \log M)$ primes, and therefore the probability in question is $O(\log |z| / M)$. □

2.3 Quantum Primitives

Our algorithm employs the standard circuit model augmented with coherent read–write quantum random-access memory, denoted QRAQM. We count each single- or two-qubit gate as one operation and assume that coherent random access to any entry of a list represented using m qubits can be performed in time polylogarithmic in m. See [NS20, Sch21] for a formal definition.

We use the following standard primitives in our algorithm.

Coherent dictionaries.

Theorem 2.4 ([Amb07, Section 6.2])**.** *Let $M = n^{O(1)}$, and let $\mathcal{U}$ be a totally ordered universe whose elements have* polylog(n)*-bit encodings. In the* QRAQM *model, a dynamic set $D \subseteq \mathcal{U}$ of at most M elements can be stored in $\widetilde{O}(M)$ space and constructed in $\widetilde{O}(M)$ time. The structure supports coherent successor lookup, insertion, and deletion in $\widetilde{O}(1)$ time, up to inverse-polynomial error.*

Quantum search.

Theorem 2.5 ([Gro96, BHMT00, HMdW03]). *Let $f : [N] \to \{0,1\}$ be a predicate with a bounded-error coherent evaluation algorithm of cost T. There is a bounded-error quantum algorithm that finds an x satisfying $f(x) = 1$, or reports that none exists, in time $\widetilde{O}(\sqrt{N}\,T)$.*

Quantum walks. Let P be a reversible ergodic Markov chain on a finite state space, with stationary distribution π and spectral gap δ. Fix a marked set $\mathcal{M}$ and let $\mathcal{D}(x)$ denote the data stored with a state x. Let T_{set}, T_{upd}, and T_{chk} denote the costs of preparing the stationary state and its data, coherently performing one transition and updating the data, and checking the vertex, respectively. The last operation maps, up to inverse-polynomial error,

$$|x\rangle|\mathcal{D}(x)\rangle \longmapsto (-1)^{[x\in\mathcal{M}]}|x\rangle|\mathcal{D}(x)\rangle,$$

and returns all ancillary registers to $|0\rangle$.

Theorem 2.6 ([Sze04, MNRS11]). *If $\pi(\mathcal{M}) \geq \mu$, then a state in $\mathcal{M}$ can be found with bounded error in time*

$$\widetilde{O}\left(T_{\text{set}} + \frac{1}{\sqrt{\mu}}\left(\frac{1}{\sqrt{\delta}}T_{\text{upd}} + T_{\text{chk}}\right)\right).$$

Quantum walks on Johnson graphs. For a finite set $\mathcal{X}$ and $1 \leq m \leq |\mathcal{X}|/2$, the Johnson graph $J(\mathcal{X}, m)$ has the m-subsets of $\mathcal{X}$ as its vertices, with two subsets adjacent when their intersection has size $m-1$. Our algorithm uses the following lazy walk on $J(\mathcal{X}, m) \times J(\mathcal{X}, m)$. At each step, the walk successively samples one of the two component subsets, a member of that subset, and an element of $\mathcal{X}$ from their respective uniform distributions. It replaces the sampled member by the sampled element unless the latter already belongs to the subset, in which case it stays in the same vertex. This walk is reversible and has the uniform stationary distribution. Kachigar and Tillich [KT17] show that its spectral gap is $\delta = \Omega(1/m)$. Consequently, if at least a μ fraction of its vertices are marked, Theorem 2.6 specializes to the running time

$$\widetilde{O}\left(T_{\text{set}} + \frac{1}{\sqrt{\mu}}\left(\sqrt{m}\,T_{\text{upd}} + T_{\text{chk}}\right)\right).$$

Claw Finding.

Theorem 2.7 ([Tan09]). *Let $\mathcal{L}$, $\mathcal{R}$, and $\mathcal{Y}$ be finite sets with $|\mathcal{Y}| = \text{poly}(|\mathcal{L}|, |\mathcal{R}|)$, and let $f : \mathcal{L} \to \mathcal{Y}$ and $g : \mathcal{R} \to \mathcal{Y}$. A* claw *is a pair $(x, y) \in \mathcal{L} \times \mathcal{R}$ satisfying $f(x) = g(y)$. Suppose that $|\mathcal{L}| \leq |\mathcal{R}|^2$, $|\mathcal{R}| \leq |\mathcal{L}|^2$, and each function can be evaluated coherently in $\log^{O(1)}(|\mathcal{L}| + |\mathcal{R}|)$ time. Then there is a bounded-error quantum algorithm that finds a claw, or reports that none exists, in time*

$$\widetilde{O}\left((|\mathcal{L}| \cdot |\mathcal{R}|)^{1/3}\right).$$

3 The k-SUM Algorithm

We are now ready to prove Theorem 1.2.

Theorem 1.2. *For every integer $k > 3$, worst-case k-SUM can be solved in quantum time $\widetilde{O}(n^{\Psi_k})$, where*[5]

$$\Psi_k = \underbrace{\frac{2k - \lfloor k/7 \rfloor - \lfloor (k+3)/7 \rfloor}{6}}_{\Phi_k} - \frac{[k \equiv 3 \bmod 7]}{9} - \frac{[k \equiv 6 \bmod 7]}{18}.$$

3.1 Partition

Let k_1, k_2, k_3, k_4 be positive integer parameters satisfying $k_1 + k_2 + k_3 + k_4 = k$; their values will be chosen at the end of the proof. Group the k input lists into four consecutive blocks of sizes (k_1, k_2, k_3, k_4), numbered 1, 2, 3, and 4. Thus, block 1 consists of the first k_1 input lists, and blocks 2, 3, and 4 consist of the next k_2, k_3, and k_4 input lists, respectively.

The index tuples of the four blocks range over the product domains

$$\mathcal{X}_i = [n]^{k_i} \qquad \text{for every } i \in [4],$$

and we write $\alpha_i \in \mathcal{X}_i$ for an index tuple of block i. For each $i \in [4]$, let $\Sigma_i : \mathcal{X}_i \to \mathbb{Z}$ map a tuple of block i to its sum. For example, if $\alpha_1 \in \mathcal{X}_1$, then $\Sigma_1(\alpha_1) = \sum_{j \in [k_1]} A_j[\alpha_1[j]]$.

3.2 Construction

Searching in a random subspace. The straightforward approach would construct complete lists of sums for the blocks used in the search. Instead, we retain only a small sample of tuples from some of the blocks and use a quantum walk to find the appropriate subset of tuples.

Let $r \in (0, \min\{k_2, k_4\}]$ be a parameter, whose value is fixed at the end of the proof, and set

$$m = \lfloor n^r/2 \rfloor.$$

We perform a quantum walk on the graph

$$\mathcal{G} = J(\mathcal{X}_2, m) \times J(\mathcal{X}_4, m),$$

whose vertices are of the form

$$v = (S_2, S_4) \in \binom{\mathcal{X}_2}{m} \times \binom{\mathcal{X}_4}{m}.$$

We will choose $k_2 = k_4$, so $|\mathcal{X}_2| = |\mathcal{X}_4|$ and $\mathcal{G}$ is isomorphic to $J(\mathcal{X}, m) \times J(\mathcal{X}, m)$ for a set $\mathcal{X}$ of that cardinality. Each $\mathcal{X}_i$ is the set of all index tuples for block i; for $i \in \{2, 4\}$, $S_i \subseteq \mathcal{X}_i$ is the m-element sample stored at the vertex, and $x \in S_i$ denotes a single sampled tuple. One step of the walk chooses $i \in_R \{2, 4\}$, $x \in_R S_i$, and $x' \in_R \mathcal{X}_i$. If $x' \notin S_i$, the step replaces x by x' in that coordinate; otherwise it remains at v.

Fix a particular k-SUM solution and denote its four tuples of selected indices, one for each block, by $\alpha^* = (\alpha_1^*, \alpha_2^*, \alpha_3^*, \alpha_4^*)$. For $i \in \{2, 4\}$, a uniformly random m-subset of $\mathcal{X}_i$ contains a fixed tuple with probability m/n^{k_i}. Consequently, the fraction of vertices $v = (S_2, S_4)$ with $\alpha_2^* \in S_2$ and $\alpha_4^* \in S_4$ is

$$\Pr_{v=(S_2,S_4) \in_R V(\mathcal{G})}[\alpha_2^* \in S_2 \text{ and } \alpha_4^* \in S_4] = \frac{m^2}{n^{k_2+k_4}}.$$

[5] $[\cdot]$ is the Iverson bracket: for a condition $\mathcal{E}$, $[\mathcal{E}] = 1$ if $\mathcal{E}$ holds, and $[\mathcal{E}] = 0$ otherwise.

Filtering by a random prime. Let $\mathcal{P}$ be the set of primes in $\{m \log^2 n, \ldots, 2m \log^2 n\}$. We sample $p \in_R \mathcal{P}$ and keep it fixed throughout the walk. For every fixed choice of p, the functions defined below determine a fixed marked subset of $V(\mathcal{G})$. When checking a vertex, we use quantum search over residues $q \in \mathbb{Z}_p$. Once q is fixed, we retain only the left- and right-hand tuple combinations whose partial sums satisfy $\Sigma_1(\alpha_1) + \Sigma_2(\alpha_2) \equiv q \bmod p$ and $\Sigma_3(\alpha_3) + \Sigma_4(\alpha_4) \equiv -q \bmod p$, respectively.

Data stored at a vertex. At a vertex $v = (S_2, S_4)$, the walk stores, for each $i \in \{2, 4\}$, a dictionary $D_{v,i}$ whose bucket indices are residues $s \in \mathbb{Z}_p$. The bucket indexed by s is

$$D_{v,i}[s] = \{(\Sigma_i(x), x) \colon x \in S_i \text{ and } \Sigma_i(x) \equiv s \bmod p\}.$$

By Theorem 2.4, those dictionaries can be built in $\widetilde{O}(m)$ time, and each of coherent lookup, insertion, and deletion costs $\widetilde{O}(1)$ time. The setup procedure prepares both dictionaries as part of the walk state. If a transition replaces x by x' in coordinate i, its update coherently transforms the vertex data from $(D_{v,2}, D_{v,4})$ to $(D_{v',2}, D_{v',4})$ by deleting $(\Sigma_i(x) \bmod p, (\Sigma_i(x), x))$ and inserting $(\Sigma_i(x') \bmod p, (\Sigma_i(x'), x'))$.

Define the lookup function $R_{v,i} : \mathbb{Z}_p \to (\mathbb{Z} \times \mathcal{X}_i) \cup \{\perp\}$, where

$$R_{v,i}(s) = \begin{cases} \min D_{v,i}[s], & D_{v,i}[s] \neq \varnothing, \\ \perp, & D_{v,i}[s] = \varnothing. \end{cases}$$

Here, the minimum is taken lexicographically, first by sum and then by tuple. If $R_{v,i}(s) = (z, x)$, we call (z, x) the *canonical representative* of the bucket indexed by s. Later, we show that, with high probability, every entry in the bucket containing a solution has the same sum, so the choice of representative is immaterial.

Checking a vertex. Let $\alpha_1 \in \mathcal{X}_1$ and $\alpha_3 \in \mathcal{X}_3$. Fix $q \in \mathbb{Z}_p$ and define the two target bucket indices by

$$s_2^q(\alpha_1) = (q - \Sigma_1(\alpha_1)) \bmod p, \qquad s_4^q(\alpha_3) = (-q - \Sigma_3(\alpha_3)) \bmod p.$$

Thus, for $i \in \{2, 4\}$, s_i^q is the bucket index required for the block-i sum. Define $F_{v,q}$ and $G_{v,q}$ by

$$F_{v,q}(\alpha_1) = \begin{cases} \Sigma_1(\alpha_1) + z, & R_{v,2}(s_2^q(\alpha_1)) = (z, \alpha_2), \\ \perp, & R_{v,2}(s_2^q(\alpha_1)) = \perp, \end{cases}$$

and

$$G_{v,q}(\alpha_3) = \begin{cases} -\Sigma_3(\alpha_3) - z, & R_{v,4}(s_4^q(\alpha_3)) = (z, \alpha_4), \\ \top, & R_{v,4}(s_4^q(\alpha_3)) = \perp. \end{cases}$$

Here, $\perp$ and $\top$ are two distinct symbols outside $\mathbb{Z}$. If $F_{v,q}(\alpha_1) = G_{v,q}(\alpha_3)$, then the corresponding representatives satisfy $\Sigma_1(\alpha_1) + \Sigma_2(\alpha_2) = -\Sigma_3(\alpha_3) - \Sigma_4(\alpha_4)$, so the four tuples form a solution. Define the marked set

$$\mathcal{M}_p = \{v \in V(\mathcal{G}) : \exists q \in \mathbb{Z}_p,\ \alpha_1 \in \mathcal{X}_1,\ \alpha_3 \in \mathcal{X}_3 \text{ such that } F_{v,q}(\alpha_1) = G_{v,q}(\alpha_3)\}.$$

The checking procedure coherently evaluates the predicate $v \in \mathcal{M}_p$ by quantum search over q and Claw Finding on $F_{v,q}$ and $G_{v,q}$, and then applies the transformation

$$|v\rangle |D_{v,2}, D_{v,4}\rangle \longmapsto (-1)^{[v \in \mathcal{M}_p]} |v\rangle |D_{v,2}, D_{v,4}\rangle.$$

The pseudocode is presented in Algorithm 1.

Algorithm 1 Quantum k-SUM Algorithm

Input: Integer lists $A_1, \ldots, A_k$, block sizes k_1, k_2, k_3, k_4, and a parameter $r \in (0, \min\{k_2, k_4\}]$.
Output: A tuple of indices $(i_1, \ldots, i_k) \in [n]^k$ satisfying $\sum_{j=1}^{k} A_j[i_j] = 0$, if one exists, and NO otherwise.

```
Partition the lists into four blocks of sizes (k_1, k_2, k_3, k_4)
Set m ← ⌊n^r/2⌋ and choose a prime p ∈_R P
Run a quantum walk over G:
    Quantum search over q ∈ Z_p
    Construct F_{v,q} and G_{v,q} and run Claw Finding on them
    if a claw is found then
        return the indices corresponding to the found claw
    end if
return NO
```

3.3 Correctness

The algorithm cannot output an incorrect solution; so it remains to show that, whenever a solution exists, the algorithm finds one with constant probability. Fix a solution $\alpha^* = (\alpha_1^*, \alpha_2^*, \alpha_3^*, \alpha_4^*)$, and let $z_2^* = \Sigma_2(\alpha_2^*)$ and $z_4^* = \Sigma_4(\alpha_4^*)$. We say that z_2^* is *exposed* at a vertex if the canonical representative of its residue has exact sum z_2^*, and define exposure of z_4^* analogously.

Let $\mathcal{V}^*$ be the set of vertices whose two sampled subsets contain α_2^* and α_4^*. For a fixed prime p, call a vertex $v \in \mathcal{V}^*$ *successful* if both target sums z_2^* and z_4^* are exposed at v. Call p *good* if at least half of the vertices in $\mathcal{V}^*$ are successful.

Lemma 3.1. *A prime $p \in_R \mathcal{P}$ is good with constant probability.*

Proof. Fix a vertex $v \in \mathcal{V}^*$. For $i \in \{2, 4\}$, an entry $x \in S_i$ with $\Sigma_i(x) \neq z_i^*$ can lie in the same residue bucket as z_i^* only if p divides $\Sigma_i(x) - z_i^*$. The difference $\Sigma_i(x) - z_i^*$ is nonzero and has absolute value $\widetilde{O}(n^k)$, therefore applying Proposition 2.3 bounds the collision probability by

$$O\left(\frac{\log \widetilde{O}(n^k)}{m \log^2 n}\right) = O\left(\frac{k}{m \log n}\right) = O\left(\frac{1}{m \log n}\right),$$

where the last equality uses that k is fixed. A union bound over at most $2m$ tuples in the two sampled subsets shows that, except with probability $O(1/\log n)$, both relevant buckets contain only entries of exact sums z_2^* and z_4^*, respectively; in particular, v is successful. So, the expected fraction of unsuccessful vertices of $\mathcal{V}^*$ is $O(1/\log n)$. Markov's inequality now implies that more than half of the vertices of $\mathcal{V}^*$ are unsuccessful with probability $O(1/\log n)$. □

Every successful vertex belongs to $\mathcal{M}_p$, hence if p is good, at least half of $\mathcal{V}^*$ is contained in $\mathcal{M}_p$, and then

$$\frac{|\mathcal{M}_p|}{|V(\mathcal{G})|} \geq \frac{1}{2}\frac{|\mathcal{V}^*|}{|V(\mathcal{G})|} = \frac{1}{2}\frac{m^2}{n^{k_2+k_4}} = \Omega(n^{2r-k_2-k_4}). \tag{1}$$

By Lemma 3.1, the bound in Equation (1) holds with constant probability over the choice of p. The residue search and the claw-finding procedure are amplified to error $n^{-\omega(1)}$, as required by Theorem 2.6. Thus the algorithm finds a solution with constant probability whenever one exists.

3.4 Running Time

By Lemma 3.1 and Equation (1), with constant probability over $p \in_R \mathcal{P}$, the quantum walk has a marked fraction $\mu = \Omega(n^{2r-k_2-k_4})$. The quantum search over $q \in \mathbb{Z}_p$ contributes a factor $\widetilde{O}(\sqrt{p})$ to the checking time by Theorem 2.5. For each fixed residue, Claw Finding searches the domains $\mathcal{X}_1$ and $\mathcal{X}_3$. For the parameter choices made below, we force $k_1/2 \le k_3 \le 2k_1$, so Theorem 2.7 gives a claw finding of a fixed residue in time $\widetilde{O}(n^{(k_1+k_3)/3})$. By Theorem 2.4, the walk setup builds the two dictionaries in $\widetilde{O}(m) = \widetilde{O}(n^r)$ time. Since adjacent vertices differ by one tuple, a walk update maintains them using one deletion and one insertion in $\widetilde{O}(1)$ time. Using Theorem 2.6 on $\mathcal{G}$ with parameters

$$T_{\text{set}} = \widetilde{O}(n^r), \qquad T_{\text{upd}} = \widetilde{O}(1),$$
$$T_{\text{chk}} = \widetilde{O}(\sqrt{p}\, n^{(k_1+k_3)/3}) = \widetilde{O}(n^{(k_1+k_3)/3+r/2}), \qquad \mu = \Omega(n^{2r-k_2-k_4})$$

gives a total running time of the algorithm

$$\widetilde{O}\left(n^r + n^{(k_2+k_4)/2-r}\left(n^{r/2} + n^{(k_1+k_3)/3+r/2}\right)\right) = \widetilde{O}\left(n^{\max\left\{r,\ \frac{k_1+k_3}{3}+\frac{k_2+k_4-r}{2}\right\}}\right).$$

It remains to choose the block sizes and r. The bound depends on k_2 and k_4 only through their sum, except for the constraint $r \le \min\{k_2, k_4\}$; hence we choose $k_2 = k_4$. Similarly, it depends on k_1 and k_3 only through their sum, and choosing them as evenly as possible ensures the conditions of Theorem 2.7. For convenience in Table 1, put $\ell = k_1 + k_3$; then $k_1 = \lfloor \ell/2 \rfloor$, $k_3 = \lceil \ell/2 \rceil$, and $k = \ell + 2k_2$. With these choices, put

$$r = \min\left\{k_2, \frac{2}{3}\left(k_2 + \frac{k_1+k_3}{3}\right)\right\}.$$

Let $k = 7d + j$, where $j \in \{0, 1, \ldots, 6\}$.

$j = k \bmod 7$	$\ell = k_1 + k_3$	$k_2 = k_4$	r	$\Psi_k = \max\{r, \ell/3 + k_2 - r/2\}$
0	$3d$	$2d$	$2d$	$2d$
1	$3d+1$	$2d$	$2d$	$2d + 1/3$
2	$3d$	$2d+1$	$2d + 2/3$	$2d + 2/3$
3	$3d+1$	$2d+1$	$2d + 8/9$	$2d + 8/9$
4	$3d+2$	$2d+1$	$2d+1$	$2d + 7/6$
5	$3d+3$	$2d+1$	$2d+1$	$2d + 3/2$
6	$3d+2$	$2d+2$	$2d + 16/9$	$2d + 16/9$

Table 1: Optimal parameters for Algorithm 1, where $d = \lfloor k/7 \rfloor$ and $j = k \bmod 7$.

This completes the proof of Theorem 1.2.

4 Application to Pigeonhole Modular Equal Subset Sum

Definition 4.1. *An instance of* Pigeonhole Modular Equal Subset Sum *consists of a list A of n positive integers and a modulus $q \le 2^n - 1$. The task is to find two distinct subsets $S_1, S_2 \subseteq [n]$ such that*

$$\sum_{i \in S_1} A[i] \equiv \sum_{i \in S_2} A[i] \bmod q.$$

Theorem 4.2. *The* Pigeonhole Modular Equal Subset Sum *problem can be solved in quantum time* $O^*(3^{2n/7})$.

Proof. Replacing every $A[i]$ by $A[i] \bmod q$ does not change the set of solutions, so we assume that $0 \le A[i] < q$ for all $i \in [n]$.

A solution is equivalent to a nonzero vector $\varepsilon \in \{-1, 0, 1\}^n$ satisfying

$$\sum_{i \in [n]} \varepsilon[i]\, A[i] \equiv 0 \bmod q.$$

As $0 \le A[i] < q$ for all i, the left-hand side lies in the interval $(-nq, nq)$, so the congruence holds if and only if

$$\sum_{i \in [n]} \varepsilon[i]\, A[i] = cq \quad \text{for some integer } c \text{ with } |c| < n.$$

Partition $[n]$ into seven blocks $B_1, \ldots, B_7$, each of size at most $\lceil n/7 \rceil$. For each $j \in [7]$, construct a list L_j indexed by vectors $\varepsilon \in \{-1, 0, 1\}^{B_j}$, with $L_j[\varepsilon] = \sum_{i \in B_j} \varepsilon[i]\, A[i]$. Each list has size at most $M = 3^{\lceil n/7 \rceil}$. For a fixed c, finding a vector $\varepsilon \in \{-1, 0, 1\}^n$ with $\sum_{i \in [n]} \varepsilon[i]\, A[i] = cq$ is exactly a 7-SUM instance on the lists $L_1, \ldots, L_7$ with target cq.

For $c = 0$, one may get the zero vector as an answer, so we need to force at least one coordinate of ε to be nonzero. For $i \in [n]$, $\sigma \in \{-1, 1\}$, and an integer c with $|c| < n$, let $j(i)$ denote the index of the block containing i, and let $I_{i,\sigma,c}$ be the 7-SUM instance with target cq obtained from $L_1, \ldots, L_7$ by restricting $L_{j(i)}$ to the indices $\varepsilon \in \{-1, 0, 1\}^{B_{j(i)}}$ with $\varepsilon[i] = \sigma$. Every solution of $I_{i,\sigma,c}$ satisfies $\varepsilon[i] = \sigma$ and is therefore a nonzero solution of the displayed congruence. By Theorem 1.2 with $k = 7$, each of them is solved in time $\widetilde{O}(M^2)$, so the total running time is

$$O(n^2) \cdot \widetilde{O}(M^2) = O^*(3^{2n/7}). \qquad \square$$

Acknowledgments

The authors used ChatGPT to assist with verifying all proofs, identifying typos, and conducting the literature review. The authors assume responsibility for all content.

References

[AHJ+22] Jonathan Allcock, Yassine Hamoudi, Antoine Joux, Felix Klingelhöfer, and Miklos Santha. Classical and Quantum Algorithms for Variants of Subset-Sum via Dynamic Programming. In *European Symposium on Algorithms*, volume 244 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2022.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[AL13] Amir Abboud and Kevin Lewi. Exact Weight Subgraphs and the k-Sum Conjecture. In *Automata, Languages, and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.

[ALW14] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing Weight by Gaining Edges. In *Algorithms—ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2014.

[Amb07] Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[AV14] Amir Abboud and Virginia Vassilevska Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *IEEE Symposium on Foundations of Computer Science*, pages 434–443. IEEE Computer Society, 2014.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved Classical and Quantum Algorithms for Subset-Sum. In *Advances in Cryptology—ASIACRYPT*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved Generic Algorithms for Hard Knapsacks. In *Advances in Cryptology—EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.

[BCKM24] Tatiana Belova, Nikolai Chukhin, Alexander S. Kulikov, and Ivan Mihajlin. Improved Space Bounds for Subset Sum. In *European Symposium on Algorithms*, volume 308 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2024.

[BDH+05] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum Algorithms for Element Distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005.

[BHMT00] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum Amplitude Amplification and Estimation. *arXiv preprint quant-ph/0005055*, 2000.

[BJLM13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum Algorithms for the Subset-Sum Problem. In *Post-Quantum Cryptography*, volume 7932 of *Lecture Notes in Computer Science*, pages 16–33. Springer, 2013.

[BKM+24] Tatiana Belova, Alexander S. Kulikov, Ivan Mihajlin, Olga Ratseeva, Grigory Reznikov, and Denil Sharipov. Computations with Polynomial Evaluation Oracle: Ruling Out Superlinear SETH-Based Lower Bounds. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1853. SIAM, 2024.

[Bri17] Karl Bringmann. A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017.

[CJRS23] Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset Sum in time $2^{n/2}/\operatorname{poly}(n)$. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 275 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2023.

[CR88] Benny Chor and Ronald L. Rivest. A Knapsack-Type Public Key Cryptosystem Based on Arithmetic in Finite Fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In *Advances in Cryptology—CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.

[dlVP96] Charles-Jean de la Vallée Poussin. Recherches analytiques sur la théorie des nombres premiers, I–III. *Annales de la Société scientifique de Bruxelles*, 20:183–256, 281–362, 363–397, 1896.

[FMV16] Sebastian Faust, Daniel Masny, and Daniele Venturi. Chosen-Ciphertext Security from Subset Sum. In *Public-Key Cryptography—PKC*, volume 9614 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 2016.

[GNS18] Lorenzo Grassi, María Naya-Plasencia, and André Schrottenloher. Quantum Algorithms for the k-XOR Problem. In *Advances in Cryptology—ASIACRYPT*, volume 11272 of *Lecture Notes in Computer Science*, pages 527–559. Springer, 2018.

[GO95] Anka Gajentaan and Mark H. Overmars. On a Class of $O(n^2)$ Problems in Computational Geometry. *Computational Geometry*, 5:165–185, 1995.

[Gro96] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.

[Had96] Jacques Hadamard. Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques. *Bulletin de la Société Mathématique de France*, 24:199–220, 1896.

[Her96] Antonio Hernández-Barrera. Finding an $o(n^2 \log n)$ Algorithm Is Sometimes Hard. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 289–294. Carleton University Press, 1996.

[HJ10] Nick Howgrave-Graham and Antoine Joux. New Generic Algorithms for Hard Knapsacks. In *Advances in Cryptology—EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.

[HM18] Alexander Helm and Alexander May. Subset Sum Quantumly in 1.17^n. In *Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 111 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2018.

[HMdW03] Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum Search on Bounded-Error Inputs. In *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 291–299. Springer, 2003.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM*, 21(2):277–292, 1974.

[IN96] Russell Impagliazzo and Moni Naor. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. *Journal of Cryptology*, 9(4):199–216, 1996.

[JV16] Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, Triangles. *Algorithmica*, 74(1):326–343, 2016.

[JW24] Ce Jin and Hongxun Wu. A Faster Algorithm for Pigeonhole Equal Sums. In *International Colloquium on Automata, Languages, and Programming*, volume 297 of *LIPIcs*, pages 94:1–94:11. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2024.

[JW26] Antoine Joux and Karol Węgrzycki. Improving Lagarias-Odlyzko Algorithm for Average-Case Subset Sum: Modular Arithmetic Approach. In *International Symposium on Theoretical Aspects of Computer Science*, volume 364 of *LIPIcs*, pages 57:1–57:19. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2026.

[JWZ25] Ce Jin, Ryan Williams, and Stan Zhang. New Algorithms for Pigeonhole Equal Subset Sum. In *European Symposium on Algorithms*, volume 351 of *LIPIcs*, pages 86:1–86:12. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2025.

[Kar72] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1272–1287. SIAM, 2016.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum Information Set Decoding Algorithms. In *Post-Quantum Cryptography*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017.

[Lam26] Michael Lampis. k-SUM Hardness Implies Treewidth-SETH. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1916–1944. SIAM, 2026.

[Len02] Hendrik W. Lenstra, Jr. Primality Testing with Gaussian Periods. In *FSTTCS*, volume 2556 of *Lecture Notes in Computer Science*, page 1. Springer, 2002.

[Li25] Yang Li. From Subset-Sum to Decoding: Improved Classical and Quantum Algorithms via Ternary Representation Technique. *Information*, 16(10):887, 2025.

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving Low-Density Subset Sum Problems. *Journal of the ACM*, 32(1):229–246, 1985.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-Key Cryptographic Primitives Provably as Secure as Subset Sum. In *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 382–400. Springer, 2010.

[LVWW16] Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic Time-Space Trade-Offs for k-SUM. In *International Colloquium on Automata, Languages, and Programming*, volume 55 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2016.

[MH78] Ralph C. Merkle and Martin E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via Quantum Walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

[NS20] María Naya-Plasencia and André Schrottenloher. Optimal Merging in Quantum k-XOR and k-XOR-SUM Algorithms. In *Advances in Cryptology—EUROCRYPT*, volume 12106 of *Lecture Notes in Computer Science*, pages 311–340. Springer, 2020.

[NW21] Jesper Nederlof and Karol Węgrzycki. Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1670–1683. ACM, 2021.

[Păt10] Mihai Pătraşcu. Towards Polynomial Lower Bounds for Dynamic Problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 603–610. ACM, 2010.

[RW26] Tim Randolph and Karol Węgrzycki. Beating Meet-in-the-Middle for Subset Balancing Problems. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing*, pages 1314–1325. ACM, 2026.

[Sch21] André Schrottenloher. Improved Quantum Algorithms for the k-XOR Problem. In *Selected Areas in Cryptography*, volume 13203 of *Lecture Notes in Computer Science*, pages 311–331. Springer, 2021.

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM Journal on Computing*, 10(3):456–464, 1981.

[Sze04] Mario Szegedy. Quantum Speed-Up of Markov Chain Based Algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 32–41. IEEE Computer Society, 2004.

[Tan09] Seiichiro Tani. Claw Finding Algorithms Using Quantum Walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

[VW13] Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

[Wag02] David A. Wagner. A Generalized Birthday Problem. In *Advances in Cryptology—CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il