Time and Query Complexity Tradeoffs for the Dihedral Coset Problem

Maxime Remaud, André Schrottenloher, Jean-Pierre Tillich

Time and Query Complexity Tradeoffs for the Dihedral Coset Problem

Maxime Remaud[1], André Schrottenloher[2], and Jean-Pierre Tillich[3]

[1] Inria and Eviden Quantum Lab, Paris, France
[2] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
[3] Inria, Paris, France

Abstract. The Dihedral Coset Problem (DCP) in $\mathbb{Z}_N$ has been extensively studied in quantum computing and post-quantum cryptography, as for instance, the Learning with Errors problem reduces to it. While the Ettinger-Høyer algorithm is known to solve the DCP in $O(\log N)$ queries, it runs inefficiently in time $O(N)$. The first time-efficient algorithm was introduced (and later improved) by Kuperberg (SIAM J. Comput. 2005). These algorithms run in a *subexponential* amount of time and queries $\tilde{O}\left(2^{\sqrt{c_{\mathsf{DCP}} \log N}}\right)$, for some constant c_{DCP}.

The sieving algorithms *à la* Kuperberg admit many trade-offs between quantum and classical time, memory and queries. Some of these trade-offs allow the attacker to reduce the number of queries if they are particularly costly, which is notably the case in the post-quantum key-exchange CSIDH. Such optimizations have already been studied, but they typically fall into two categories: the resulting algorithm is either based on Regev's approach of reducing the DCP with quadratic queries to a subset-sum instance, or on a re-optimization of Kuperberg's sieve where the time and queries are both subexponential.

In this paper, we introduce the first algorithm to improve in the linear queries regime over the Ettinger-Høyer algorithm. We then show that we can in fact interpolate between this algorithm and Kuperberg's sieve, by using the latter in a pre-processing step to create several quantum states, and solving a *quantum* subset-sum instance to recover the full secret in one pass from the obtained states. This allows to interpolate smoothly between the linear queries-exponential time complexity case and the subexponential query and time complexity case, thus allowing a fine tuning of the complexity taking into account the query cost. We also give on our way a precise study of quantum subset-sum algorithms in the non-asymptotic regime.

Keywords: Dihedral Hidden Subgroup Problem, Subset-sum, Dihedral Coset Problem, Quantum Algorithms

1 Introduction

Hidden Subgroup Problem. Let G be a known group and H be an unknown subgroup of G. Finding H is a problem known as the Hidden Subgroup Problem

(HSP). To solve it, we can query a function f which satisfies a certain property with respect to H:

Definition 1 (HSP). *The hidden subgroup problem is defined as:*

- *Given: a function $f : G \to S$ that is constant and distinct on the left cosets of an unknown subgroup H of a group G, S being a finite set,*
- *Find: (a generating set of) H.*

Many problems used to construct primitives can be reduced to an HSP instance, for example the Discrete Logarithm and Shortest Vector problems. Shor's algorithm [32], which solves the DLP and breaks the RSA cryptosystem [31] in polynomial time, can actually be extended to solve the HSP for an abelian group G. In the general case, it is well known that the problem requires only a *polynomial* (in $\log_2 |G|$) number of queries to the function f [19]. However, time-efficient quantum algorithms are only known for very specific instances, including abelian groups, and it is widely admitted that the generic problem remains difficult for quantum algorithms.

*Dihedral Hidden Subgroup Problem (*DHSP*).* While the HSP in an abelian group is quantumly easy to solve, many post-quantum primitives are related to the HSP in the *dihedral group*. In this case, even if the group is very close to be abelian (it has namely an abelian subgroup of index 2) no polynomial-time algorithm is known. This is the case of cryptosystems based on the Unique Shortest Vector Problem (uSVP) in lattice-based cryptography (such as [1,29]) or on any problem that can be reduced to the uSVP (because of a chain of reductions between several problems [28,25,33]). More concretely, the security of several primitives reduces to the DHSP. The most prominent example is the isogeny-based post-quantum key-exchange CSIDH [11], which is similar to the Diffie-Hellman protocol [16] except that it does not rely on the period-finding problem in an abelian group (which is solvable in quantum polynomial time), but on the difficulty to invert the group action. Several related constructions [3] such as the signature schemes SeaSign [14,15] and CSI-FiSh [6] also rely on the same problem. It should be noted that these isogeny-based cryptosystems are the only major contenders for which the quantum attacker enjoys more than a quadratic speedup, as opposed to the lattice- and hash-based finalists of the NIST post-quantum standardization process [26,2].

As it has been shown in [9,27,12], a better understanding of the security of CSIDH comes from a careful analysis of quantum DHSP algorithms. This is the motivation of our work.

From the DHSP *to the* DCP. Solving the HSP for the dihedral group of order $2N$ is known to reduce to the specific case where the hidden subgroup is $\{(0,0),(s,1)\}$, where $s \in \mathbb{Z}_N$, which can in turn be reduced to a problem known as the Dihedral Coset Problem (see [18]).

Definition 2 (DCP). *The dihedral coset problem is defined as:*

– *Given: an oracle outputting coset states of the form* $\frac{1}{\sqrt{2}}\left(|0\rangle|x\rangle + |1\rangle|x+s\rangle\right)$ *for random* $x \in [\![0, N]\!]$,
– *Find:* $s \in [\![0, N]\!]$

While the Ettinger-Høyer algorithm [18] solves the DCP with a linear number of queries ($O(\log N)$), it runs in exponential time $O(N)$. This algorithm basically consists in measuring $O(\log N)$ coset states and then classically looking among all possible values for the secret s the one that matches the best a statistical test. It is possible to improve over this running time by reducing the resolution of the DCP to a subset-sum problem, as described by [9,7], at the cost of squaring the query complexity. Though it remains exponential, the time complexity becomes $\tilde{O}(N^{c_{\mathsf{SS}}})$, where c_{SS} is a constant smaller than 1 that depends on the invoked subset-sum subroutine.

In a seminal work [23], Kuperberg initiated a family of *sieving* algorithms which reach subexponential time complexities (at the cost of a subexponential query complexity). The idea here is to iterate a process of combining states to build new ones with a stronger and stronger structure, until building a very specific state that allows us to guess a bit of the secret when measured. The first algorithm [23] requires subexponential quantum time *and* space, but it was quickly followed by an algorithm of Regev [30] which requires only polynomial space. Later, Kuperberg proposed his second algorithm [24], which generalized Regev's while improving its exponents, giving in the end a complexity in time (classical and quantum) and classical space of $\tilde{O}\left(2^{\sqrt{2\log N}}\right)$ with a quantum space of $O(\mathsf{polylog}\, N)$ and $\tilde{O}\left(2^{\sqrt{2\log N}}\right)$ queries, which is the state of the art so far.

Our contributions. Let $n \stackrel{\text{def}}{=} \lceil \log_2 N \rceil$. We first propose a new algorithm using a linear number of queries. It is somewhat analogous to Regev's algorithm where instead of reducing the DCP to a classical subset-sum problem, it reduces the DCP to a *quantum* subset-sum problem. In the first case, the algorithm makes $O(n)$ queries to find one bit of the secret, meaning it has to be iterated $O(n)$ times. With this new algorithm, which is inspired by [28,33], we only need $O(1)$ quantum subset-sum instances, *i.e.*, $O(n)$ queries, to find the whole secret.

Second, we present a simple and natural method of interpolation between Kuperberg's second algorithm (which is the state of the art) and the new algorithm we mentioned above. It consists in using Kuperberg's algorithm to more or less preprocess the states given as input to our algorithm. The difficulty of solving the inherent quantum subset-sum problem instance will depend on the preprocessing step.

Finally, as a building block of our algorithms, we study quantum subset-sum algorithms when the problem to solve is partially in superposition. We show here that we can still improve over Grover's search even under the constraint of a polynomial quantum memory, using an exponential classical memory, with or without quantum access. Specifically, we show that the QRACM-based algorithm of [8] adapts to this case and reaches a complexity $\tilde{O}\left(2^{0.2356n}\right)$. Without

QRACM, we reach a quantum time $\tilde{O}\left(2^{0.4165n}\right)$ using $O\left(2^{0.2334n}\right)$ bits of classical memory, improving over a previous algorithm by Helm and May [21]. In both cases, we also give non-asymptotic estimates of their complexity.

All together, we can summarize the complexity exponents of the different algorithms for solving the DCP in Table 1, including the new one we propose.

Table 1. Costs of algorithms for finding the whole secret s.

Algorithm	Queries	Classical Time	Quantum Time	Classical Space
Kuperberg II	$\sqrt{2n} + \frac{1}{2}\log n + 3$	$\sqrt{2n} + \frac{1}{2}\log n + 3$	$\sqrt{2n} + \frac{1}{2}\log n + 3$	$\sqrt{2n}$
Regev	$2\log n + 3$	$0.283n$	$2\log n + 3$	$0.283n$
Ettinger-Hoyer	$\log n + 6.5$	n	$\log n + 6.5$	$\log n$
Alg. 4 w/ QRACM	$\log n + 3$	$0.238n + 12$	$0.238n + \frac{3}{2}\log n + 12$	$0.238n$
Alg. 4 w/o QRACM	$\log n + 3$	$< 0.2324n$	$0.418n + \frac{3}{2}\log n + 15.5$	$< 0.2324n$

We propose two versions of our algorithm, one with QRACM and one without, both using polynomial quantum space. Note that our algorithm with QRACM outperforms other algorithms using a linear number of queries when we look at the complexity in classical time + quantum time.

Impact on CSIDH. Although Kuperberg's second algorithm is the one with the best time complexity for solving the DCP, it is still interesting to look at algorithms that only use a linear number of queries, since for example, CSIDH cryptanalysis via the resolution of the DCP involves the use of a very expensive oracle.

We give in Table 2 a few examples of complexity exponents for parameters of CSIDH.

Table 2. Complexity exponents for some parameters of CSIDH. The quantum space is polynomial in n.

	Algorithm	Queries	Classical Time	Quantum Time	Classical Space
CSIDH-512	Regev	15	80	15	73
($n = 256$)	Alg. 4 w/ QRACM	11	73	85	61
CSIDH-1024	Regev	17	153	17	145
($n = 512$)	Alg. 4 w/ QRACM	12	134	148	122
CSIDH-1792	Regev	18	262	18	254
($n = 896$)	Alg. 4 w/ QRACM	13	226	240	214
CSIDH-3072	Regev	19	443	19	435
($n = 1536$)	Alg. 4 w/ QRACM	14	378	394	366
CSIDH-4096	Regev	20	589	20	580
($n = 2048$)	Alg. 4 w/ QRACM	14	500	516	488

Organization of the Paper. In Section 2, we give some preliminaries on sieving algorithms for the DCP, and subset-sum algorithms that we will use as black boxes afterwards. In Section 3, we recall the reduction from the DCP to the subset-sum problem, and introduce our new idea of using a *quantum* subset-sum solver. Our interpolation between the sieving and subset-sum approaches is detailed in Section 4. Finally, our contributions on quantum subset-sum algorithms, and the details of the black boxes that we used throughout the paper, are provided in Section 5.

2 Preliminaries

In this section, we cover the main principles of sieving algorithms for the DCP, including Kuperberg's and Regev's algorithms [23,24,30,13]. We assume knowledge of the quantum circuit model, *i.e.*, the $|\cdot\rangle$ notation of quantum states, and basic quantum operations such as CNOT, Toffoli, the Quantum Fourier Transform (QFT), *etc.*

We estimate the *time* complexity of a quantum algorithm in the quantum circuit model, as a number of *n-bit arithmetic operations*. That is, instead of counting precisely the quantum gates, we count the n-bit XORs, additions, subtractions, comparisons, QFTs, depending on the complexity parameter n.

We work with different types of memory:

- quantum memory (*i.e.*, qubits): some DCP algorithms (*e.g.*, Kuperberg's first algorithm [23]) need to store many coset states, which creates a subexponential quantum memory requirement;
- classical memory with quantum random-access (QRACM): the QRACM (or qRAM, QROM in some papers) is a specialized hardware which stores classical data and accesses this data in quantum superposition. That is, we assume that given a classical memory of M bits $y_0, \ldots, y_{M-1}$, the following unitary operation:

$$|x\rangle |i\rangle \xmapsto{\mathsf{Access}} |x \oplus y_i\rangle |i\rangle$$

  can be implemented in time $O(1)$. QRACM is a very common assumption in quantum computing, and it appears in several works on the DCP [24,27] but also on collision-finding [10] and subset-sum algorithms [8].
- classical memory without quantum random-access: the Access operation can be implemented in M arithmetic operations using a sequential circuit. This removes the QRACM assumption, and we fall back on the basic quantum circuit model. Some algorithms using QRACM can be re-optimized in a non-trivial way when memory access is costly, and this is the case of subset-sum [21].

2.1 Phase Vectors and Kuperberg's First Algorithm

We will consider in what follows that we have access to an oracle outputting *phase vectors* denoted by $|\psi_k\rangle$ and defined as:

$$|\psi_k\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}\left(|0\rangle + \omega_N^{sk}|1\rangle\right)$$

where $\omega_N = \exp(2\iota\pi/N)$, $\iota = \sqrt{-1}$, and k is a *known* uniformly distributed random element of $\mathbb{Z}_N$. They are obtained from coset states (the input states of the DCP) by applying a QFT on $\mathbb{Z}_N$ on the first register and then measuring this register, since we have

$$(QFT_N \otimes \mathbf{I})\left(\frac{1}{\sqrt{2}}\left(|x\rangle|0\rangle + |x+s\rangle|1\rangle\right)\right) = \frac{1}{\sqrt{N}}\sum_{k\in\mathbb{Z}_N}\omega_N^{kx}|k\rangle|\psi_k\rangle\,.$$

Finding $s \in [\![0, N-1]\!]$ from a collection of phase vectors $|\psi_k\rangle$ for known uniformly distributed random $k \in [\![0, N-1]\!]$ solves both the DCP and the DHSP.

Subexponential Algorithms. We will now give more details on the algorithms solving the DCP in subexponential time. Until the end of this section, it can be assumed that $N = 2^n$ for the sake of simplicity, but the algorithms discussed here work for any value of N.

Kuperberg's initial observation is that one can combine two phase vectors $|\psi_p\rangle$ and $|\psi_q\rangle$ to construct a new phase vector. Indeed, we have:

$$|\psi_p, \psi_q\rangle \xmapsto{\text{CNOT}} \frac{1}{\sqrt{2}}\left(|\psi_{p+q}, 0\rangle + \omega_N^{yq}|\psi_{p-q}, 1\rangle\right) \quad .$$

A measurement of the second qubit will leave the first one either in the state $|\psi_{p-q}\rangle$, or $|\psi_{p+q}\rangle$, depending on the bit measured. With probability 1/2, we get $|\psi_{p-q}\rangle$. By noticing on the other hand that $|\psi_{N/2}\rangle = \mathbf{H}|\text{lsb}(s)\rangle$ ($\text{lsb}(s)$ being the least significant bit of s), Kuperberg designed a quite simple algorithm which groups the phase vectors according to their last non-zero bits. They are then combined two by two using CNOT gates. Half of the time, the difference is obtained, and it contains as many zeroes as there were bits in common. The resulting phase vectors are regrouped and the process is reiterated. As proven in [23], the target state $|\psi_{N/2}\rangle$ is then obtained in subexponential time.

2.2 Regev's Algorithm

Kuperberg's first algorithm requires to store, at each time, a subexponential number of phase vectors; thus, it has subexponential quantum memory complexity. Regev [30] modified the combination routine to reduce the number of qubits to polynomial, while keeping the time complexity subexponential.

The new routine combines m phase vectors for a well-chosen m (to minimize the overall complexity).

Let B be some chosen, arbitrary value. We start with m phase vectors $|\psi_{k_1}\rangle, \ldots, |\psi_{k_m}\rangle$. We tensor the vectors, *i.e.*, we obtain a sum:

$$\bigotimes_i |\psi_{k_i}\rangle = \sum_{\mathbf{b}\in\{0,1\}^m} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle} |\mathbf{b}\rangle$$

We compute $\lfloor \langle \mathbf{b},\mathbf{k}\rangle / B \rfloor$ into a new qubit register, and measure a value V. This projects the state on the vectors $\mathbf{b}$ such that $\lfloor \langle \mathbf{b},\mathbf{k}\rangle / B \rfloor = V$. We choose m and the size of B such that on average two solutions $\mathbf{b}$ and $\mathbf{b}'$ occur. The state becomes proportional to:

$$|\mathbf{b}\rangle + \omega_N^{s(\langle \mathbf{b},\mathbf{k}\rangle - \langle \mathbf{b}',\mathbf{k}\rangle)} |\mathbf{b}'\rangle \quad .$$

Finally, we remap $\mathbf{b}, \mathbf{b}'$ to $0, 1$ respectively. We have obtained a phase vector $|\psi_k\rangle$ with a label $k = \langle \mathbf{b},\mathbf{k}\rangle - \langle \mathbf{b}',\mathbf{k}\rangle \leq B$. Then, step by step, we can make the labels decrease until we obtain the label 1. As remarked in [9], we can also obtain any label whose value is invertible modulo N, by multiplying all initial labels by their inverse, and applying normally the algorithm afterwards. In particular, when N is odd, we can obtain all powers of two.

Regev [30] and later Childs, Jao and Soukharev [13] used this combination routine to get an algorithm with $\tilde{O}\left(2^{\sqrt{2n\log_2 n}}\right)$ queries and $O(n)$ quantum memory.

2.3 Kuperberg's Second Algorithm

Like the two previous ones, Kuperberg's *collimation sieve* [24] is a hybrid quantum/classical procedure starting from the initial phase vectors, where we need to perform both quantum computations which create new vectors, and classical computations which give their description. The difference is that phase vectors are now multi-labeled:

$$|\psi_{k_1,\ldots,k_\ell}\rangle = \frac{1}{\sqrt{\ell}} \sum_i \omega_N^{sk_i} |i\rangle \quad .$$

In order to control these new phase vectors, we need to know the list of all their labels. These lists will become of subexponential size, although the vector itself requires only a polynomial amount of qubits. This is why the algorithm combines a polynomial quantum memory with a subexponential *classical* memory.

The combination subroutine is similar to Regev's, except that it does not necessarily reduce the list of labels down to 2. Instead, the two phase vectors are combined into a new one holding a similar number of labels, as shown in Algorithm 1.

Originally, Kuperberg uses classical memory with quantum random-access (QRACM), an approach later followed by Peikert [27]. However it only improves the trade-offs with respect to the total quantum time, and it is not necessary to reach the optimal complexity. Also, the collimation procedure presented here

Algorithm 1 Combination routine in the collimation sieve.

Input: $|\psi_{k_1,\ldots,k_\ell}\rangle$, $\ldots$, $\left|\psi_{k'_1,\ldots,k'_{\ell'}}\right\rangle$ such that $\forall i \leq \ell, \forall j \leq \ell', k_i < 2^a, k'_j < 2^a$, the lists of the labels

Output: $\left|\psi_{v_1,\ldots,v_{\ell''}}\right\rangle$ such that $\forall i, v_i < 2^{a-r}$

1: *Quantum:* Tensor the vectors: $\sum_{i\leq\ell, j\leq\ell'} \omega_N^{s(k_i+k'_j)} |i\rangle |j\rangle$

2: *Quantum:* Compute the function $i, j \mapsto \lfloor (k_i + k'_j)/2^{a-r} \rfloor$ into an ancilla register

3: *Quantum:* Measure the register, obtain a value V. The state collapses to:

$$\sum_{i,j|\lfloor (k_i+k'_j)/2^{a-r} \rfloor = V} \omega_N^{s(k_i+k'_j)} |i\rangle |j\rangle$$

4: *Classical:* Compute $\{(i,j) | \lfloor (k_i + k'_j)/2^{a-r} \rfloor = V\}$, of size ℓ''

5: *Quantum:* Apply to the state a transformation that maps the pairs (i,j) to $[\![0, \ell'' - 1]\!]$.

6: Return the state and the vector of corresponding labels $k_i + k'_j$.

is from later works such as [27], as it allows to easily deal with arbitrary group orders.

Without QRACM, Steps 2 and 5 in Algorithm 1 require a time complexity $O(\max(\ell, \ell', \ell''))$. This is also the classical time complexity required by Step 4, assuming that the lists of labels are sorted.

The Algorithm as a Merging Tree. Starting from a certain set of multi-labeled phase vectors, we can identify them with the classical lists of their labels. The combination step operates on these lists like a purely classical list-merging algorithm, in which new lists of labels are formed from the pairs of labels satisfying a certain condition. This algorithm can be represented as a *merging tree* in which all nodes are lists of labels (resp. phase vectors).

On the classical side, Kuperberg's algorithm is thus similar to Wagner's generalized birthday algorithm [34], which is a binary merging tree of depth $\sqrt{n}$. In Wagner's algorithm, the goal is to impose stronger conditions at each level which culminate in a full-zero sum. Here, the same conditions are imposed on the labels in the phase vectors. A success in the list-merging routine is equivalent to a success in the collimation routine (we obtain a phase vector with the wanted label).

The query, time and memory complexities depend on the shape of the tree. Even though the conditions are actually chosen at random at the measurement step in Algorithm 1, we can consider them chosen at random *before* the combination to analyze the algorithm.

Example: Optimal Time. The optimal time complexity is obtained with a tree with $\sqrt{2n}$ levels. It starts with lists of size 2, *i.e.*, two-labeled phase vectors. At level i starting from the leaves, the lists have (expected) size 2^i, and they are merged pairwise into a list of size 2^{i+1}. This means that we can eliminate

$2i - (i+1) = i - 1$ bits. So we should use h levels, where:

$$1 + \ldots + h - 1 = n \implies h \simeq \sqrt{2n} \ .$$

Since each level of merging doubles the number of lists, there are in total $2^{\sqrt{2n}}$ leaves (hence queries). The (classical) cost of merging, over the whole tree, is equal to the sum of all list sizes. It is also the (quantum) cost of the relabeling operations: $\sum_i 2^{\sqrt{2n}-i} \times 2^i = O\left(\sqrt{2n}2^{\sqrt{2n}}\right)$.

To compute the memory complexity, one must note that it is not required to store whole levels of the merging tree. Instead, we compute the lists (resp. the phase vectors) depth-first, and store only one node of each level at most, *i.e.* $\sqrt{2n}$ phase vectors. For the same reason, the classical memory complexity is $O\left(2^{\sqrt{2n}}\right)$.

Precise Analysis. The analysis above is only performed on average, and in practice, there is a significant variance in the list sizes. More precise analyses were performed in [27,12]. It follows from them that the list size after merging should be considered smaller than the expected one by an "adjusting factor" $\sqrt{3/(2\pi)}$. Furthermore, the combination may create lists that are too large, which must be discarded. The empirical analysis of Peikert [27] gives a factor $(1-\delta)$ of loss at each level, with $\delta = 0.02$.

The smaller factor in list sizes simply means that at level i, we will not exactly eliminate $i-1$ bits, but $i-c$ where $c = \log_2\left(1+\sqrt{\frac{3}{2\pi}}\right) \simeq 0.76$. (We can control the interval size in Algorithm 1 very precisely.) Thus h is solution to:

$$\sum_{i=1}^{h}(i-c) = n \implies \frac{h^2}{2} - ch = n \implies h \simeq c + \sqrt{2n + 4c^2} \ .$$

Finally, the loss at each level induces a global multiplicative factor $(1-\delta)^{-h} = 2^{-\log_2(1-\delta)h} \simeq 2^{0.029h}$ on the complexity. Therefore, accounting for the adjusting factor and discards, the query complexity of the sieve is:

$$2^{1.029\left(0.76+\sqrt{2n+2.30}\right)} \tag{1}$$

and the quantum time complexity multiplies this by a factor $0.76 + \sqrt{2n + 2.30}$. The difference with the exact $2^{\sqrt{2n}}$ is not negligible, but not large either. At $n = 4608$, the two exponents are respectively 99.6 and 96.

Obtaining All the Bits of the Solution. The analysis above applies if we want to obtain a specific label, *e.g.*, the label 1. Afterwards, the algorithm can be repeated n times. For a generic N (not a power of 2), one typically produces all labels which are powers of 2 and uses a QFT to directly recover the secret. This is done for example in [9]. Peikert [27] proposed a more advanced method to recover multiple bits of the secret with each phase vector.

Lemma 1. *Let $\alpha > 0$ and n be a positive integer. We have*

$$\sum_{i=1}^{n} 2^{\alpha\sqrt{i}} = O\left(\sqrt{n}2^{\alpha\sqrt{n}}\right).$$

Proof. When i is a perfect square, let say $i = j^2$, we have that $2^{\alpha\sqrt{i}} = 2^{\alpha j}$. Now for any i between the two perfect squares $(j-1)^2$ and j^2, we have the upper bound $2^{\alpha\sqrt{i}} < 2^{\alpha j}$. In order to use this, we rewrite the sum:

$$\begin{aligned}
\sum_{i=1}^{n} 2^{\alpha\sqrt{i}} &\leq \sum_{j=0}^{\lceil\sqrt{n}\rceil-1} \sum_{k=j^2+1}^{(j+1)^2} 2^{\alpha\sqrt{k}} \\
&\leq \sum_{j=0}^{\lceil\sqrt{n}\rceil-1} \sum_{k=j^2+1}^{(j+1)^2} 2^{\alpha(j+1)} \\
&= \sum_{j=0}^{\lceil\sqrt{n}\rceil-1} (2j+1)2^{\alpha(j+1)}
\end{aligned}$$

Using the formula for geometric series, we obtain:

$$\begin{aligned}
\sum_{i=1}^{n} 2^{\alpha\sqrt{i}} &\leq 2^{\alpha+1}\frac{(2^\alpha-1)\lceil\sqrt{n}\rceil 2^{\alpha\lceil\sqrt{n}\rceil} - 2^\alpha(2^{\alpha\lceil\sqrt{n}\rceil}-1)}{(2^\alpha-1)^2} + 2^\alpha\frac{2^{\alpha\lceil\sqrt{n}\rceil}-1}{2^\alpha-1} \\
&= \frac{2^\alpha}{2^\alpha-1}\left((2\lceil\sqrt{n}\rceil+1)2^{\alpha\lceil\sqrt{n}\rceil} - \frac{2^{\alpha+1}}{2^\alpha-1}(2^{\alpha\lceil\sqrt{n}\rceil}-1) - 1\right) \\
&\leq \frac{2^\alpha}{2^\alpha-1}(2\lceil\sqrt{n}\rceil+1)2^{\alpha\lceil\sqrt{n}\rceil} \quad .
\end{aligned}$$

which allows us to conclude the proof, α being fixed. □

Obtaining Partially Collimated Labels. In this paper, we will consider the task of obtaining labels which, instead of reaching a prescribed k, match k on a certain number of bits only (we can say that the phase vectors are *partially collimated*), let say i: this complexity is of order $2^{\sqrt{2i}}$. By Lemma 1, we can obtain a sequence of i phase vectors collimated on $1, \ldots, i$ bits with a query complexity: $\sum_{j=1}^{i} 2^{\sqrt{2j}} = O\left(\sqrt{i}2^{\sqrt{2i}}\right)$.

2.4 The Subset-Sum Problem

As we will see in Section 3, the DCP can be reduced to the Subset-sum problem; this leads to the most query-efficient algorithms, and depending on the cost of queries, to the best optimization for some instances.

Definition 3 (Subset-sum). *A subset-sum instance is given by $(v, \mathbf{k}), v \in \mathbb{Z}_N, \mathbf{k} \in \mathbb{Z}_N^m$ for some modulus N and integer m. The problem is to find a vector (or all vectors) $\mathbf{b} \in \{0,1\}^m$ such that $\langle \mathbf{b}, \mathbf{k} \rangle = v \mod N$.*

When $m \simeq n = \lceil \log_2 N \rceil$, there is one solution on average. The instance is said to be of *density one*. Heuristic classical and quantum algorithms based on the *representation technique* [22,4] allow to solve it in exponential time in n. In the following, we will use these algorithms as black boxes. We first need a classical subset-sum solver.

Fact 1. *We have a classical algorithm $\mathscr{S}^{\mathrm{C}}$ which, on input a subset-sum instance $(v, \mathbf{k})$ of density one, finds all solutions. It has a time complexity in $\tilde{O}\left(2^{c_{c\mathsf{SS}} n}\right)$ where $c_{c\mathsf{SS}} < 1$.*

Here, the parameter $c_{c\mathsf{SS}}$ is the best asymptotic exponent that we can obtain for classical subset-sum algorithms. If there are no constraints on the memory, we can take $c_{c\mathsf{SS}} = 0.283$ which is the best value known at the moment [8].

In this paper, we will also need (quantum) algorithms solving a more difficult problem, in which $\mathbf{k}$ is fixed, but the target v is *in superposition*. We will call this type of algorithm a *quantum* subset-sum solver.

Fact 2. *We have a quantum algorithm $\mathscr{S}^{\mathrm{Q}}$ which has a complexity cost in $\tilde{O}\left(2^{c_{q\mathsf{SS}} n}\right)$ (where $c_{q\mathsf{SS}} < 1$), which, given an error bound ε, given a known (classical) $\mathbf{k} \in \mathbb{Z}_N^m$ and on input a quantum v, maps:*

$$|v\rangle |\mathbf{b}\rangle \mapsto |v\rangle \left|\mathbf{b} \oplus \mathscr{S}^{\mathrm{Q}}(v)\right\rangle$$

where, for a proportion at least $1 - \varepsilon$ of all v admitting a solution, $\mathscr{S}^{\mathrm{Q}}(v)$ is selected u.a.r. from the solutions to the subset-sum problem, i.e., from the set $\{\mathbf{b} |\ \langle \mathbf{b}, \mathbf{k} \rangle = v\}$.

Notice that in the way we implement the solver, we can only guarantee that it succeeds on a large proportion of inputs (there remains some probability of error). However, it depends on some precomputations that we can redo, to obtain a heuristically independent solver which allows to reduce ε and / or to ensure that we get more solutions.

Though we could implement the function $\mathscr{S}^{\mathrm{Q}}$ by running an available classical (or quantum) subset-sum algorithm, it would then require exponential amounts of qubits. Using only $\mathsf{poly}(n)$ qubits, we know for sure that $c_{q\mathsf{SS}} \leq 0.5$, because we can use Grover's algorithm to exhaustively search for a solution $\mathbf{b}$. This search uses $\mathsf{poly}(n)$ qubits only. In Section 5, we will show that we can reach smaller values for $c_{q\mathsf{SS}}$, which differ depending on whether we allow QRACM or not.

3 Reducing DCP to a Subset-sum Problem

Recall that we note $n = \lceil \log_2 N \rceil$, where N is not necessarily a power of 2. We will focus in this section on two algorithms to solve the DCP: the first one (from Regev [30]) uses a classical subset-sum solver and the other (ours) uses a quantum one.

Algorithm 2 Finding $\mathrm{lsb}(s)$ using a classical subset-sum solver $\mathscr{S}^{\mathrm{C}}$

Require: $|\psi_{k_1}\rangle, \ldots, |\psi_{k_n}\rangle$ with $\mathbf{k} \stackrel{\text{def}}{=} (k_1 \ldots k_n) \in \mathbb{Z}_N^n$.
Ensure: $\mathrm{lsb}(s)$.

1: Tensor the phase vectors and append a register on $\mathbb{F}_2^{n-1}$

$$\bigotimes_{i=1}^{n} |\psi_{k_i}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{b}\in\mathbb{F}_2^n} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle} |\mathbf{b}\rangle$$

2: Compute the inner product of $\mathbf{b}$ and $\mathbf{k}$ in the ancillary register

$$\frac{1}{\sqrt{2^n}} \sum_{\mathbf{b}\in\mathbb{F}_2^n} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle} |\mathbf{b}\rangle \left|\langle \mathbf{b},\mathbf{k}\rangle \mod 2^{n-1}\right\rangle$$

3: Measure the ancillary register ▷ Z is a normalizing constant

$$\frac{1}{\sqrt{Z}} \sum_{\substack{\mathbf{b}\in\mathbb{F}_2^n:\\ \langle \mathbf{b},\mathbf{k}\rangle = z \mod 2^{n-1}}} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle} |\mathbf{b}\rangle |z\rangle$$

4: Search for vectors $\mathbf{b_i}$ such that $\langle \mathbf{b_i},\mathbf{k}\rangle = z \mod 2^{n-1}$ using $\mathscr{S}^{\mathrm{C}}$
5: Project the superposition onto a pair of solutions, *e.g.*, $(\mathbf{b}_1, \mathbf{b}_2)$

$$\frac{1}{\sqrt{2}} \left(\omega_N^{s\langle \mathbf{b_1},\mathbf{k}\rangle} |\mathbf{b}_1\rangle + \omega_N^{s\langle \mathbf{b_2},\mathbf{k}\rangle} |\mathbf{b}_2\rangle\right)$$

6: Relabel the basis states to $(|0\rangle, |1\rangle)$, resulting in

$$\frac{\omega_N^{s\langle \mathbf{b_1},\mathbf{k}\rangle}}{\sqrt{2}} \left(|0\rangle + \omega_N^{s\langle \mathbf{b_2}-\mathbf{b_1},\mathbf{k}\rangle} |1\rangle\right)$$

7: Apply a Hadamard gate on the qubit, measure it and output the result.

3.1 Using a Classical Subset-sum Solver

By reducing Regev's algorithm to a single level, as described in [9], we can directly produce $\mathrm{lsb}(s)$ from n phase vectors. This is detailed in Algorithm 2.

It can be proven that in Step 4, the number of solutions is quite small but generally enough for our purpose. In Step 5, the solution vectors we want to project our superposition on are marked in an ancillary register which is then measured. Either we will get what we want, or we will end up with a superposition of the solution vectors that were not marked, in which case we start the process again with two other solution vectors. For more details, we refer to the extensive study of Regev's algorithm by Childs, Jao and Soukharev [13].

The following lemma gives us the complexity of Algorithm 2, derived from Regev's algorithm.

Lemma 2 (Subsection 3.3 [9]). *There exists an algorithm which finds* $\mathrm{lsb}(s)$ *with* $O(n)$ *queries and quantum time and space. It has the same usage in classical time and space as the subset-sum solver* $\mathscr{S}^{\mathrm{C}}$.

Algorithm 2 finds one bit of the secret. In order to retrieve the whole secret, we will have to repeat this procedure n times. Thus, we get an algorithm using a quadratic number of calls to the oracle, exponential classical time and space because of the subset-sum solver, linear quantum space and quadratic quantum time.

It turns out that we could solve the classical subset-sum problem on the side with a quantum computer, leading to some tradeoffs described in [7]. But we show hereafter that we can also build an algorithm which directly uses a quantum subset-sum solver instead of having to measure the ancillary register to get a classical instance of a subset-sum problem.

3.2 Using a Quantum Subset-sum Solver

The main observation that led to the design of the algorithm we introduce hereafter is that on one hand, we would like to build the superposition

$$\frac{1}{\sqrt{N}} \sum_{j \in \mathbb{Z}_N} \omega_N^{sj} \left| j \right\rangle \tag{2}$$

since applying the inverse QFT on $\mathbb{Z}_N$ on it would directly give the secret s, and on the other hand, we know that it would be possible, thanks to a quantum subset-sum solver, to prepare the state

$$\frac{1}{\sqrt{Z(\mathbf{k})}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \left| \langle \mathbf{b}, \mathbf{k} \rangle \mod N \right\rangle \tag{3}$$

where $Z(\mathbf{k})$ is a normalizing constant depending on $\mathbf{k}$. Indeed, preparing this state is done by using Regev's trick (see [28,33]), *i.e.*,

(i) by tensoring m phase vectors

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \left| \mathbf{b} \right\rangle \left| \mathbf{0}_n \right\rangle,$$

(ii) then computing the subset-sum in the second register to get the entangled state

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \left| \mathbf{b} \right\rangle \left| \langle \mathbf{b}, \mathbf{k} \rangle \mod N \right\rangle,$$

(iii) and finally disentangle it thanks to a quantum subset-sum algorithm which from $\langle \mathbf{b}, \mathbf{k} \rangle \mod N$ and $\mathbf{k}$ (which is classical) recovers $\mathbf{b}$ and subtracts it from the first register to get the state we want.

As one can see, if we could take $m = n$ and have an isomorphism between the vectors $\mathbf{b}$ and the knapsack sums $\langle \mathbf{b}, \mathbf{k} \rangle \mod N$, the prepared state (3) would be exactly the superposition (2).

Algorithm 3 Ideal algorithm

Require: A parameter $m < n$ and phase vectors $|\psi_{k_i}\rangle$ for $i \in [\![1, m]\!]$.
Ensure: An element $j \in \mathbb{Z}_N$.

1: Tensor the m phase vectors and append a register on $\mathbb{Z}_N$

$$\bigotimes_{i=1}^{m} |\psi_{k_i}\rangle \, |\mathbf{0}_n\rangle = \frac{1}{\sqrt{M}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \, |\mathbf{b}\rangle \, |\mathbf{0}_n\rangle$$

2: Compute the inner product of $\mathbf{b}$ and $\mathbf{k}$ in the ancillary register

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \, |\mathbf{b}\rangle \, |\langle \mathbf{b}, \mathbf{k} \rangle \mod N\rangle$$

3: Uncompute $\mathbf{b}$ thanks to $\mathbf{k}$ and $|\langle \mathbf{b}, \mathbf{k} \rangle \mod N\rangle$

$$\frac{1}{\sqrt{Z(\mathbf{k})}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b}, \mathbf{k} \rangle} \, |\mathbf{0}_m\rangle \, |\langle \mathbf{b}, \mathbf{k} \rangle \mod N\rangle$$

4: Apply the inverse QFT on $\mathbb{Z}_N$ on the second register

$$\frac{1}{\sqrt{N}} \sum_{j \in \mathbb{Z}_N} \left(\frac{1}{\sqrt{Z(\mathbf{k})}} \sum_{\mathbf{b} \in \mathbb{F}_2^m} \omega_N^{(s-j)\langle \mathbf{b}, \mathbf{k} \rangle} \right) |\mathbf{0}_m\rangle \, |j\rangle$$

5: Measure the state and output the resulting j.

However, there would be many cases in which multiple solutions to the subset-sum problem exist. Thus we take $m < n$ and define $M \stackrel{\text{def}}{=} 2^m < N$. This is different from Algorithm 2, where such collisions are needed. We obtain Algorithm 3, which uses Regev's trick with a quantum subset-sum solver in Step 3.

Despite M being smaller than N, some cases still yield multiple solutions, and furthermore the subset-sum solver (as given by Fact 2) fails on some instances. This is why we distinguish between Algorithm 3 in which we consider the quantum subset-sum solver to be ideal (*i.e.*, it finds back $\mathbf{b}$ from $\langle \mathbf{b}, \mathbf{k} \rangle$ and $\mathbf{k}$ with certainty), and the algorithm that we actually build in practice: Algorithm 4.

The analysis of Algorithm 4 is related to the set of $\mathbf{b}$ on which the quantum subset-sum solver succeeds: $\mathscr{S}^{\mathrm{Q}}(\langle \mathbf{b}, \mathbf{k} \rangle) = \mathbf{b}$ for a fixed $\mathbf{k}$.

Notation 3. *Let us denote by $\mathscr{G}(\mathbf{k})$ the set of $\mathbf{b}$'s that are correctly found back by $\mathscr{S}^{\mathrm{Q}}$ for a given $\mathbf{k}$:*

$$\mathscr{G}(\mathbf{k}) \stackrel{def}{=} \{\mathbf{b} \in \mathbb{F}_2^m : \mathscr{S}^{\mathrm{Q}}(\langle \mathbf{b}, \mathbf{k} \rangle) = \mathbf{b}\}$$

and let $G(\mathbf{k})$ be the size of the set $\mathscr{G}(\mathbf{k})$.

We apply in Step 4 a measurement in order to disentangle the superposition we have, so we can apply an inverse QFT in the same natural way as in the ideal algorithm. We show that the probability of success of the measurement (*i.e.*, of

Algorithm 4 Finding s using a quantum subset-sum solver $\mathscr{S}^{\mathrm{Q}}$

Require: A parameter $m < n$ and phase vectors $|\psi_{k_i}\rangle$ for $i \in [\![1, m]\!]$.
Ensure: An element $j \in \mathbb{Z}_N$.

1: Tensor the phase vectors and append a register on $\mathbb{Z}_N$

$$\bigotimes_{i=1}^{m} |\psi_{k_i}\rangle\,|\mathbf{0}_n\rangle = \frac{1}{\sqrt{M}} \sum_{\mathbf{b}\in\mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle}\,|\mathbf{b}\rangle\,|\mathbf{0}_n\rangle$$

2: Compute the inner product of $\mathbf{b}$ and $\mathbf{k}$ in the ancillary register

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b}\in\mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle}\,|\mathbf{b}\rangle\,|\langle \mathbf{b},\mathbf{k}\rangle \mod N\rangle$$

3: Apply $\mathscr{S}^{\mathrm{Q}}$ to uncompute $\mathbf{b}$

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b}\in\mathbb{F}_2^m} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle}\,\Big|\mathbf{b} \oplus \mathscr{S}^{\mathrm{Q}}(\langle \mathbf{b},\mathbf{k}\rangle)\Big\rangle\,|\langle \mathbf{b},\mathbf{k}\rangle \mod N\rangle$$

4: Measure the first register. If the result is not $\mathbf{0}_m$, abort and restart with new coset states. Otherwise, we obtain

$$\frac{1}{\sqrt{G(\mathbf{k})}} \sum_{\mathbf{b}\in\mathscr{G}} \omega_N^{s\langle \mathbf{b},\mathbf{k}\rangle}\,|\mathbf{0}_m\rangle\,|\langle \mathbf{b},\mathbf{k}\rangle \mod N\rangle$$

5: Apply the inverse QFT on $\mathbb{Z}_N$ on the second register

$$\frac{1}{\sqrt{N}} \sum_{j\in\mathbb{Z}_N} \left(\frac{1}{\sqrt{G(\mathbf{k})}} \sum_{\mathbf{b}\in\mathscr{G}} \omega_N^{(s-j)\langle \mathbf{b},\mathbf{k}\rangle} \right) |\mathbf{0}_m\rangle\,|j\rangle$$

6: Measure the state and output the resulting j.

measuring 0) is good enough for our purpose when taking m close to n. We also prove under the same assumption that the algorithm outputs the secret with good probability. All in all, these two properties lead to our main result.

Theorem 4. *There exists an algorithm which finds s using $O(n)$ queries and the same usage in time and space as the subset-sum solver $\mathscr{S}^{\mathrm{Q}}$.*

In order to analyze Algorithm 4 and prove Theorem 4, we will proceed in two steps.

Step 1. The first step is to give a lower bound on $\mathbb{E}_{\mathbf{k}}[G(\mathbf{k})]$. This lower bound is given by estimating the number of vectors which admit more than one possible solution.

To arrive here, we first take a look at the normalization constant $Z(\mathbf{k})$ and we compute $\mathbb{E}_{\mathbf{k}}[Z(\mathbf{k})]$ (the average over all choices of $\mathbf{k}$). This can be done by simply looking at the measurement step in Algorithm 3.

Lemma 3. *We have*

$$\mathbb{E}_{\mathbf{k}}\left[Z(\mathbf{k})\right] = M\left(1 + \frac{M-1}{N}\right).$$

Proof. Fix $\mathbf{k} = (k_1, \cdots, k_m)$. For all $j \in \mathbb{Z}_N$, the measurement in Algorithm 3 returns j with probability:

$$\begin{aligned}
\mathbb{P}_{ideal}\left[j|\mathbf{k}\right] &= \frac{1}{NZ(\mathbf{k})}\left|\sum_{\mathbf{b}\in\mathbb{F}_2^m} \omega_N^{(s-j)\langle \mathbf{b},\mathbf{k}\rangle}\right|^2 \\
&= \frac{1}{NZ(\mathbf{k})}\left|\prod_{i=1}^{m}\left(1+\omega_N^{(s-j)k_i}\right)\right|^2 \quad = \frac{1}{NZ(\mathbf{k})}\prod_{i=1}^{m}\left|1+\omega_N^{(s-j)k_i}\right|^2 \\
&= \frac{1}{NZ(\mathbf{k})}\prod_{i=1}^{m} 4\cos^2\left(\pi k_i \frac{s-j}{N}\right) \quad = \frac{M^2}{NZ(\mathbf{k})}\prod_{i=1}^{m}\cos^2\left(\pi k_i\frac{s-j}{N}\right).
\end{aligned}$$

Furthermore, we have $\sum_{j\in\mathbb{Z}_N}\mathbb{P}_{ideal}\left[j|\mathbf{k}\right] = 1$, so we can write:

$$Z(\mathbf{k}) = \frac{M^2}{N}\sum_{j\in\mathbb{Z}_N}\prod_{i=1}^{m}\cos^2\left(\pi k_i\frac{s-j}{N}\right) \tag{4}$$

It follows that

$$\mathbb{E}_{\mathbf{k}}\left[Z(\mathbf{k})\right] = \frac{M^2}{N}\sum_{j\in\mathbb{Z}_N}\mathbb{E}\left[\prod_{i=1}^{m}\cos^2\left(\pi k_i\frac{s-j}{N}\right)\right]$$

and since the k_i are i.i.d., we have

$$\begin{aligned}
\mathbb{E}_{\mathbf{k}}\left[Z(\mathbf{k})\right] &= \frac{M^2}{N}\left(1 + \sum_{j\in\mathbb{Z}_N\setminus\{s\}}\prod_{i=1}^{m}\mathbb{E}\left[\cos^2\left(\pi k_i\frac{s-j}{N}\right)\right]\right) \\
&= \frac{M^2}{N}\left(1 + (N-1)\prod_{i=1}^{m}\frac{1}{2}\right) = \frac{M}{N}(N+M-1)\ . \qquad \square
\end{aligned}$$

Next, we give a relation between $G(\mathbf{k})$ and $Z(\mathbf{k})$.

Lemma 4. *For any* $\mathbf{k}$*:*

$$G(\mathbf{k}) \geq (1-\varepsilon)\left(2M - Z(\mathbf{k})\right).$$

Proof. Fix $\mathbf{k}$. Let $\mathscr{B}(j)$ be the set of vectors whose knapsack sum is j:

$$\mathscr{B}(j) \stackrel{\text{def}}{=} \{\mathbf{b}\in\mathbb{F}_2^m |\ \langle \mathbf{b},\mathbf{k}\rangle = j \mod N\}$$

and let $\mathscr{C}_i$ be the set of vectors $\mathbf{b}$ that have i collisions:

$$\mathscr{C}_i \stackrel{\text{def}}{=} \{\mathbf{b}\in\mathbb{F}_2^m |\ \#\mathscr{B}(\langle \mathbf{b},\mathbf{k}\rangle) = i\}.$$

We denote by C_i the size of the set $\mathscr{C}_i$.

If we take a closer look at $Z(\mathbf{k})$, we have that

$$\begin{aligned} Z(\mathbf{k}) &= \sum_{j\in\mathbb{Z}_N}\left|\sum_{\mathbf{b}\in B(j)}\omega_N^{s\langle\mathbf{b},\mathbf{k}\rangle}\right|^2 & &= \sum_{j\in\mathbb{Z}_N}\left|\omega_N^{sj}\right|^2\left|\sum_{\mathbf{b}\in B(j)}1\right|^2 \\ &= \sum_{j\in\mathbb{Z}_N}\sum_{\mathbf{b}\in B(j)}\sum_{\mathbf{b}'\in B(j)}1 & &= \sum_{j\in\mathbb{Z}_N}\sum_{\mathbf{b}\in B(j)}\sum_{\mathbf{b}'\in B(\langle\mathbf{b},\mathbf{k}\rangle)}1 \\ &= \sum_{j\in\mathbb{Z}_N}\sum_{\mathbf{b}\in B(j)}\#\mathscr{B}(\langle\mathbf{b},\mathbf{k}\rangle) & &= \sum_{\mathbf{b}\in\mathbb{F}_2^m}\#\mathscr{B}(\langle\mathbf{b},\mathbf{k}\rangle) \\ &= \sum_{i\geq 1}\sum_{\mathbf{b}\in\mathbb{F}_2^m:\ \#\mathscr{B}(\langle\mathbf{b},\mathbf{k}\rangle)=i} i & &= \sum_{i\geq 1} iC_i \end{aligned}$$

Letting $C_{>1}$ be the number of vectors $\mathbf{b}$ with at least one collision (*i.e.*, for which there exists $\mathbf{b}' \neq \mathbf{b}$ such that they have the same knapsack sum), we have $C_{>1} = \sum_{i>1} C_i$. From

$$Z(\mathbf{k}) = \sum_{i\geq 1} iC_i = C_1 + 2\sum_{i\geq 2} C_i + \sum_{i\geq 3}(i-2)C_i \ ,$$

it follows that we have the lower bound:

$$Z(\mathbf{k}) \geq C_1 + 2C_{>1} \ .$$

Injecting twice the equation $C_1 = M - C_{>1}$ in this inequality and using the trivial bound $G(\mathbf{k}) \geq (1-\varepsilon)C_1$, we conclude the proof. □

From Lemma 3 and 4, we immediately deduce:

Lemma 5.

$$\mathbb{E}_{\mathbf{k}}\left[G(\mathbf{k})\right] \geq (1-\varepsilon)M\left(1 - \frac{M-1}{N}\right).$$

Step 2. The second step in our proof computes the probability of success of the "real" algorithm by relating it to $\mathbb{E}_{\mathbf{k}}\left[G(\mathbf{k})\right]$.

Lemma 6. *Algorithm 4 outputs the secret s with probability $\geq (1-\varepsilon)\frac{M(N-M+1)}{N^2}$.*

Proof. We compute the probability of measuring $j \in \mathbb{Z}_N$ at the end of Algorithm 4. In particular, we have for s

$$\mathbb{P}_{real}\left[s|\mathbf{k}\right] = \frac{1}{NG(\mathbf{k})}\left|\sum_{\mathbf{b}\in\mathscr{G}}\omega_N^0\right|^2 = \frac{G(\mathbf{k})}{N}$$

We have by Lemma 5 that $\mathbb{E}\left[G(\mathbf{k})\right] \geq (1-\varepsilon)M\left(1-\frac{M-1}{N}\right)$. We finish the proof by observing that $\mathbb{P}_{real}\left[s\right] = \mathbb{E}\left[\mathbb{P}_{real}\left[s|\mathbf{k}\right]\right] \geq (1-\varepsilon)\frac{M(N-M+1)}{N^2}$. □

Finally, we can prove Theorem 4.

Proof. Step 4 of Algorithm 4 succeeds with average probability $\frac{\mathbb{E}[G(\mathbf{k})]}{M}$ which is greater than $(1-\varepsilon)\frac{N-M+1}{N}$ (by Lemma 5). The final measurement of the algorithm outputs the secret with probability $\geq (1-\varepsilon)\frac{M(N-M+1)}{N^2}$ (by Lemma 6). We will thus have to repeat the algorithm an expected number smaller than $\frac{N^3}{(1-\varepsilon)^2 M(N-M+1)^2}$ times. By letting m be equal to $n-1$, we obtain that the algorithm will have to be repeated less than $8/(1-\varepsilon)^2$ times. Thus, we can conclude that our algorithm needs $O(n)$ queries and has complexity costs identical to the ones of the subset-sum solver, since the subset-sum resolution is the only exponential step of the algorithm. □

4 Interpolation algorithm

If we take a look at the ideal algorithm and consider that there is no collision, we can see that we would like 2^m to be as close to N as possible in order for the sum

$$\frac{1}{\sqrt{M}} \sum_{\mathbf{b}\in\mathbb{F}_2^m} \omega_N^{(s-j)\langle \mathbf{b},\mathbf{k}\rangle}$$

to contain as many as possible elements of the sum

$$\frac{1}{\sqrt{N}} \sum_{\mathbf{b}\in\mathbb{F}_2^n} \omega_N^{(s-j)\langle \mathbf{b},\mathbf{k}\rangle}.$$

In the mean time, it is clear that the closest M gets to N, the more likely collisions $\langle \mathbf{b_1},\mathbf{k}\rangle = \langle \mathbf{b_2},\mathbf{k}\rangle$ for $\mathbf{b}_1 \neq \mathbf{b}_2$ become. We thus have to find a compromise on the value m or more interestingly play with the values k_i used in the algorithm, to avoid collisions and to simplify the resolution of the subset-sum problem.

In fact, we can reduce the size of the subset-sum problem we have to solve by pre-processing the states to get values of k_i that will allow us to solve the subset-sum problem on some bits by Gaussian elimination. Constructing these k_i's can be achieved by Kuperberg's second algorithm (or any improvement). Given a threshold parameter $t \in [\![1, m]\!]$, we can consider the following configuration for the k_i to use as inputs in Algorithm 4 (dots represent unknown bits and the i-th bit of the j-th row is the j-th bit of the binary expansion of k_i):

$$\begin{array}{c} \\ k_1 \\ k_2 \\ \vdots \\ k_{m-t} \\ k_{m-t+1} \\ \vdots \\ k_m \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & \cdots & m-t & m-t+1 & \cdots & n \end{array} \\ \begin{pmatrix} 1 & \bullet & \cdots & \bullet & \bullet & \cdots & \bullet \\ 0 & 1 & \ddots & \bullet & \bullet & \cdots & \bullet \\ \vdots & \vdots & \ddots & \ddots & \vdots & \cdots & \bullet \\ 0 & 0 & \cdots & 1 & \bullet & \cdots & \bullet \\ 0 & 0 & \cdots & 0 & \bullet & \cdots & \bullet \\ \vdots & \vdots & & \vdots & \vdots & \cdots & \bullet \\ 0 & 0 & \cdots & 0 & \bullet & \cdots & \bullet \end{pmatrix} \end{array} \tag{5}$$

In Algorithm 4, it turns out that we can keep a good probability of finding the secret s by letting m be equal to $n-1$ so that is what we will assume afterwards.

To build phase vectors that satisfy the configuration in (5), we will approximately have to query the oracle

$$\sum_{i=1}^{n-t} 2^{\sqrt{c_{\mathsf{DCP}} i}} + t2^{\sqrt{c_{\mathsf{DCP}}(n-t)}}$$

leading to a query and time complexities of $O\left((\sqrt{n-t}+t)2^{\sqrt{c_{\mathsf{DCP}}(n-t)}}\right)$ (by Lemma 1), where c_{DCP} is the constant of the algorithm used to construct the states ($c_{\mathsf{DCP}} = 2$ for Kuperberg's second algorithm). For the subset-sum problem, solving it on the first $n-t$ bits is easy (thanks to a Gaussian elimination), the difficulty comes from the last t bits, leading to a complexity in $O\left(2^{c_{q\mathsf{SS}}t}\right)$ time, where $c_{q\mathsf{SS}}$ is the complexity exponent of the quantum subset-sum solver. This parameter t can be used in a natural way to obtain an interpolation algorithm, since it allows to obtain a tradeoff between the preparation of the states and the resolution of the problem (which amounts to solving a quantum subset-sum problem).

We can now give an interpolation algorithm derived from Algorithm 4. We note that letting q be the query complexity exponent, it is possible to determine t from n and the value q we can afford. Using Kuperberg's second algorithm (or any improvement) to compute suitable phase vectors as described before and then giving them as inputs to Algorithm 4, we can retrieve the secret s as described by Algorithm 5 with the complexities given by Section 5.

Algorithm 5 Interpolation algorithm (using a quantum SS solver)

Require: q such that 2^q is the number of queries we are allowed to do.
Ensure: The secret s.
1: Use Kuperberg's second algorithm (or any improvement) to create states $|\psi_{k_i}\rangle$ for $i \in [\![1, m]\!]$ satisfying the configuration represented by Matrix (5), where $t \approx \frac{q}{c_{\mathsf{SS}}}$.
2: Apply Algorithm 4 on these m states to obtain a value $j \in \mathbb{Z}_N$.
3: Check if j is the secret. If not, return to Step 1. Otherwise, output j.

Theorem 5. *Let $t \in [\![1, m]\!]$. Algorithm 5 finds s with $O\left((\sqrt{n-t}+t)2^{\sqrt{c_{\mathsf{DCP}}(n-t)}}\right)$ queries in $O\left((\sqrt{n-t}+t)2^{\sqrt{c_{\mathsf{DCP}}(n-t)}} + 2^{c_{q\mathsf{SS}}t}\right)$ quantum time, classical space $O\left(2^{\sqrt{c_{\mathsf{DCP}}(n-t)}} + 2^{c_{q\mathsf{SS}}t}\right)$ and $O\left(\mathsf{poly}(n)\right)$ quantum space.*

We notice that when $t = m$, the k_i are kept random and we have to solve the "full rank" subset-sum, matching with Algorithm 4. On the other side, when $t = 1$, we fall back on Kuperberg's second algorithm since we have in this case

to construct a collection of states divisible by all the successive powers of 2, see Section 2.3. Finally, when $1 < t < m$, we have new algorithms working for any number of queries between $O(n)$ and $\tilde{O}\left(2^{\sqrt{c_{\mathrm{DCP}}\, n}}\right)$.

5 Quantum Subset-sum Algorithms

In this section, we consider quantum algorithms solving the *quantum* subset-sum problem introduced in Section 2.4. We give both asymptotic complexities and numerical estimates.

Recall that we consider a subset-sum instance $(v, \mathbf{k}), \mathbf{k} \in \mathbb{Z}_N^m$, where v is *in superposition*, and $\mathbf{k}$ will remain fixed. The problem is to find $\mathbf{b}$ such that $\langle \mathbf{b}, \mathbf{k} \rangle = v \mod N$ for a given (fixed) modulus N. For a given v, if there are many solutions, we want to find one selected uniformly at random (under heuristics). If we want all solutions, then we can run multiple instances of the solver (we will have to redo the pre-computations that we define below). A given solver, defined for a specific $\mathbf{k}$, is expected to work only for some (large) proportion $1 - \varepsilon$ of v. We can check whether the output is a solution or not and measure the obtained bit to collapse on the cases of success.

5.1 Algorithms Based on Representations

The best algorithms to solve the subset-sum problem with density one are list-merging algorithms using the *representation technique* [22,4]. The best asymptotic complexities (both classical and quantum) are given in [8]. We detail the representation framework following the depiction given in [8]. To ease the description, we start with the case $v = 0$, *i.e.*, the *homogeneous* case, and we will show below how to extend it easily to $v \neq 0$.

Guessed Weight. We assume that the solution $\mathbf{b}$ is of weight $\lceil m/2 \rceil$. This is true only with probability: $p_m := 2^{-m}\binom{m}{\lceil m/2 \rceil} = 1/\mathsf{poly}(m)$. If not, we re-randomize the subset-sum instance by multiplying $\mathbf{b}$ by a random invertible matrix. Thus if we manage to solve an instance of weight $\lceil m/2 \rceil$, the total complexity to solve any instance will introduce a multiplicative factor $\frac{1}{p_m}$ that we will have to estimate.

Distributions. We consider *distributions* of vectors having certain relative weights: $D^m[\alpha] \subseteq \{0,1\}^m$ is the set of vectors having weight αm. The basic idea of representations is to write the solution $\mathbf{b}$ as a sum of vectors of smaller relative weights, *e.g.*, $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_2$ where $\mathbf{b}_1 \in D^m[\alpha_1], \mathbf{b}_2 \in D^m[\alpha_2]$ and $\alpha_1 + \alpha_2 = \frac{1}{2}$. In this paper, we consider only representations with coefficients 0 or 1. Extended representations can be considered, using more coefficients (which have to cancel out each other). However, the advantage of using extended representations becomes quickly insignificant in practice. It is also harder to compute the number of representations, or the filtering probabilities that we define below.

Merging Tree. A subset-sum algorithm is defined by a *merging tree*. A node in this tree is a *list* $L[\ell, \alpha, c]$, which represents a set of vectors drawn from $\{0,1\}^m$ under several conditions: 1. the size of the list is $2^{m\ell}$; 2. the vectors are sampled u.a.r. from a prescribed distribution $D^m[\alpha]$; 3. the vectors satisfy a *modular condition* of cm bits. With $v = 0$, the following condition can be used: $\langle \mathbf{e}, \mathbf{k} \rangle \bmod N \in [-N/2^{cm}; N/2^{cm}]$ for some number c. More generally, the modular conditions can be chosen arbitrarily, as long as they remain compatible with the target v.

Once the tree structure is chosen, its parameters are optimized under several constraints. First, the lists have a certain maximal size. A distribution $D^m[\alpha]$ has size $\binom{m}{\alpha m}$, which is asymptotically estimated as $\simeq 2^{h(\alpha)m}$ where $h(x) := -x\log_2 x - (1-x)\log_2(1-x)$ is the Hamming entropy. This creates the constraint $\ell \leq h(\alpha) - c$. Second, we expect the root list to contain the solution of the problem, *i.e.*, $\ell = 0$ (one element), $\alpha = \frac{1}{2}$ and $c = 1$. Finally, each non-leaf list L has its parameters determined by its two children L_1, L_2. Indeed, it is obtained via the *merging-filtering* operation which selects, among all pairs of vectors $(\mathbf{e}_1, \mathbf{e}_2) \in L_1 \times L_2$, the pairs such that: $\mathbf{e}_1 + \mathbf{e}_2$ satisfies the modular condition (merging) and satisfies the weight condition (filtering). The parameters are:

$$\begin{cases} \alpha = \alpha_1 + \alpha_2 \text{ (increasing weights)} \\ \ell = \ell_1 + \ell_2 - (c - \min(c_1, c_2)) - \mathsf{pf}(\alpha_1, \alpha_2) \end{cases} \tag{6}$$

Here, pf is the probability that two vectors chosen u.a.r. in their respective distributions will not have colliding 1s.

Lemma 7 (Lemma 1 in [8]). *Let $\mathbf{e_1}, \mathbf{e_2}$ be drawn u.a.r. from $D^m[\alpha_1], D^m[\alpha_2]$ with $\alpha_1 + \alpha_2 \leq 1$. The probability that $\mathbf{e}_1 + \mathbf{e}_2 \in D^m[\alpha_1 + \alpha_2]$ is equal to:*

$$\mathsf{PF}(\alpha_1, \alpha_2, m) := \binom{m - \alpha_1 m}{\alpha_2 m} \Big/ \binom{m}{\alpha_2 m} \simeq 2^{m\mathsf{pf}(\alpha_1, \alpha_2)}$$

where $\mathsf{pf}(\alpha_1, \alpha_2) := h\left(\frac{1-\alpha_2}{\alpha_1}\right)\alpha_1 - h(\alpha_1)$.

Classical Computation of the Tree. To any correctly parameterized merging tree corresponds a classical subset-sum algorithm that runs as follows: it creates the leaf lists by sampling their distributions at random. It then builds the parent lists by *merging-filtering* steps. The *merging* operation is efficient, since elements can be ordered according to the modular condition to be satisfied.

Lemma 8 (Lemma 2 in [8]). *Let L_1, L_2 be two sorted lists stored in classical memory with random access. In $\log_2$, relatively to m, the parent list L can be built in time:* $\max(\min(\ell_1, \ell_2), \ell_1 + \ell_2 - (c - \min(c_1, c_2)))$ *and in memory* $\max(\ell_1, \ell_2, \ell)$.

Quantum Computation of the Tree. While the more advanced quantum subset-sum algorithms use quantum walks [5,20,8], we want to focus here on algorithms using few qubits, which at the moment, rely only on quantum merging with Grover search. They replace the classical merging operation by the following.

Lemma 9 (Lemma 4 in [8]). *Let L_2 be a sorted list stored in QRACM. Assume given a unitary U that produces, in time t_{L_1}, a uniform superposition of elements of L_1. Then there exists a unitary U' that produces a uniform superposition of elements of L, in time $O\left(\frac{t_{L_1}}{\sqrt{\mathsf{pf}(\alpha_1,\alpha_2)}}\max(\sqrt{2^{cm}/|L_2|},1)\right)$.*

Since the goal is only to sample u.a.r. from the root list, only half of the lists in the tree need actually to be stored in QRACM. The others are sampled using the unitary operators given by Lemma 9. In short, the obtained subset-sum algorithm is a sequence of Grover searches which use existing lists stored in memory to sample elements in new lists with more constraints.

Heuristics. The standard subset-sum heuristic assumes that the elements of all lists in the tree (not only the leaf lists) behave as if they were uniformly sampled from the set of vectors of right weight, satisfying the modular condition. This heuristic ensures that the list sizes are very close to their average: for each L obtained by merging and filtering $L_1[\ell_1,\alpha_1,c_1]$ and $L_2[\ell_2,\alpha_2,c_2]$, we have:

$$|L| \simeq \frac{|L_1||L_2|}{2^{m(c-\min(c_1,c_2))}\mathsf{PF}(\alpha_1,\alpha_2,m)} \ ,$$

where the approximation is exact down to a factor 2. This is true with overwhelming probability for all lists of large expected size via Chernoff-Hoeffding bounds, and even if the root list is of expected size 1, the probability that it actually ends up empty is smaller than $e^{-0.5} \simeq 0.61$.

5.2 From Asymptotic to Exact Optimizations

As the time and memory complexities of a subset-sum algorithm are determined by its merging tree, we seek to select a tree which minimizes these parameters. Given a certain subset-sum problem, we first select a tree shape. As an example, the best subset-sum algorithm with low qubits (using QRACM) is the "quantum HGJ" algorithm of [8], whose structure is reproduced in Figure 1. At level 3, it splits the vectors into two halves, and merges without filtering. While all lists are obtained via quantum merging/filtering, the main computation is performed after obtaining L_1^3, L_1^2, L_1^1, where the main branch is explored using Grover's algorithm: we search through the lists $L_0^3, L_0^2 L_0^1$ without representing them in memory. The quadratic speedup of Grover search makes the tree unbalanced, which is reflected on the naming of its parameters in Figure 1.

The *asymptotic* time complexity of the algorithms has the form $\tilde{O}\left(2^{\beta m}\right)$, and is the result of summing together the costs of all merging steps. Through the approximation of binomial coefficients, the list sizes are approximated in $\log_2$ and relatively to m. The parameters (relative weights, modular conditions and sizes) are numerically optimized. The optimization of Figure 1 in [8] yields the complexity $\tilde{O}\left(2^{0.2356m}\right)$.

In this paper, we also perform non-asymptotic optimizations for a given m. Since we use only $\{0,1\}$-representations, the filtering probability is well known

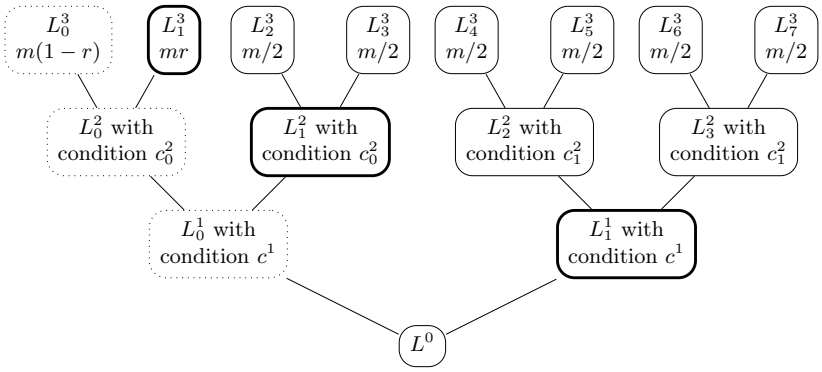

Fig. 1. Quantum HGJ algorithm. Dotted lists are search spaces (they are not stored). Bold lists are stored in QRACM. The first level uses a left-right split of vectors, without filtering.

and has a simple expression (Lemma 7). Since the binomial coefficients can be extended as functions of $\mathbb{R}^2$, we can perform an exact numerical optimization of list sizes for a given m. Afterwards, the numbers obtained are rounded, in particular the weights of representations, and we take the point which gives us the best results: smallest complexity and biggest average size for L^0.

Example. Let us take $n = \log_2 N = 256$, $m = n - 1 = 255$, and the structure of Figure 1. We adapt the optimization code of [8] by taking the exact exponents (not relative to n) and optimize numerically under the constraint $|L^0| = 2^2$ (to ensure that there are solutions). The asymptotic formula would give $2^{0.2356n} \simeq 2^{60.31}$. Numerical optimization gives us a time $2^{63.81}$, but this admits non-integer parameters and it is only the *maximum* between all steps. By rounding the parameters well, we obtain Figure 2.

To compute the quantum time complexity, we consider the list sizes to be exact and use the formula of Lemma 9 without the O. The subtrees on the right can be computed in $2^{65.71}$ operations; the slight increase is due to the fact that we take a sum of their respective terms and not a maximum. In the left branch, we sample from L^0 in $2^{63.48}$ operations.

The actual time complexity is slightly bigger, due to the variation in list sizes, and the constant complexity overhead ($\pi/2$) of Grover search. More importantly, these operations require: • to recompute a sum, using m (controlled) additions modulo N; • to test membership in some distribution; • to sample from input distributions D^n. The latter can be done using a circuit given in [17], which for a weight k and n bits, has a gate count $\tilde{O}(nk)$ and uses $n+2\lceil\log(k+1)\rceil$ qubits. All of this boils down to m arithmetic operations or $O\left(m^2\right)$ quantum gates.

Finally, this sampler works only for a proportion $\frac{1}{p_m} = 2^{-5.33}$ of subset-sum instances, so we need to re-randomize accordingly. After running the optimization for $128 \leq m \leq 1024$ and $n = m + 1$, we found that the subset-sum solver would use approximately $2^{0.238m+9.203}$ arithmetic operations, for a final list L^0

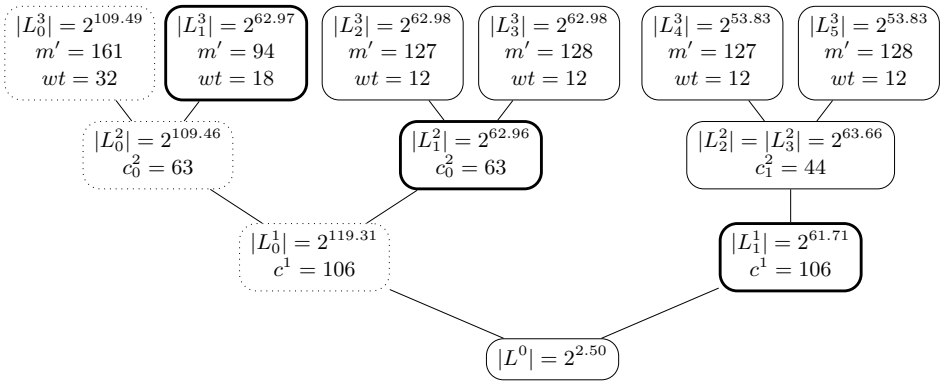

Fig. 2. Optimization of Figure 1 for $m = 255$. The size of the support is indicated by m' and the weight by wt.

of size 2 on average. Under the subset-sum heuristic, we assume an independence between all tuples of elements in the initial lists. Using Chernoff-Hoeffding bounds the probability that the final list is empty is smaller than $e^{-1} \simeq 0.37$. To reduce it to a smaller constant ε, we may simply run multiple independent instances of the solver. This increases the asymptotic complexity by a factor $O(-\log \varepsilon)$.

5.3 Solving Subset-Sum in Superposition

We now show that we can reuse the structure of the QRACM-based subset-sum algorithm of Figure 1 to solve the problem in *superposition* over the target v, while still keeping the number of qubits polynomial.

The basic idea is to reduce the problem with a given $v \neq 0$ to $v = 0$: $\langle \mathbf{k}', \mathbf{b} \rangle = 0 \mod N$, where $\mathbf{k}'$ is a length $m + 1$ vector where we append $-v$ to $\mathbf{k}$. We can then modify *any* existing tree-based subset-sum algorithm solving this instance to force all vectors in the leftmost leaf list to have a 1 in the last coordinate, and all vectors in the other leaves to have 0 in this coordinate. Then only the lists in the left branch of the tree depend on v. The complexity is unchanged.

Following the tree structure of Figure 1, we create the lists L_1^3, L_1^2, L_1^1 in a precomputation step. Then we define a quantum algorithm that outputs an element in L^0 (or a superposition of such elements), and we run this algorithm in superposition over v.

Subset-Sum without QRACM. Helm and May [21] showed that quantum subset-sum algorithms using a small classical memory (without quantum access) can have better time-memory tradeoffs than classical ones. They obtained a time $\tilde{O}\left(2^{0.428m}\right)$ for a memory $O\left(2^{0.285m}\right)$, however their algorithm does not have an unbalanced structure like ours.

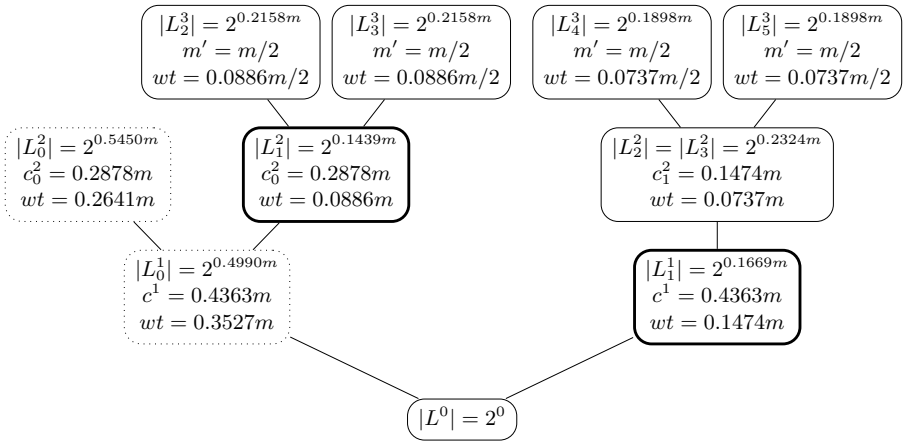

Fig. 3. Asymptotic optimization of our quantum subset-sum algorithm without QRACM. The lists on the right of the tree are constructed with classical computations, using classical RAM. The lists L_1^2 and L_1^1 are stored in classical memory without random access.

We improve on this time-memory tradeoff by adapting the tree of Figure 1 as follows: we remove L_0^3 and L_1^3 and their parameters, and directly sample in L_0^2. Assuming that the lists L_1^2 and L_1^1 are precomputed classically, we sample from L^0 with the same algorithm, except that it replaces each QRACM access (in time 1) by a sequential memory access (in time $|L_1^2|$ and $|L_1^1|$ for L_1^2 and L_1^1 respectively), *i.e.*, a quantum circuit which encodes the elements of the lists as a sequence of standard gates. The asymptotic optimization gives a time $\tilde{O}\left(2^{0.4165m}\right)$ with a memory $O\left(2^{0.2324m}\right)$. The parameters are displayed in Figure 3.

The difference between asymptotic and non-asymptotic optimization is bigger here. For $m = n - 1 = 127$, with the constraint $|L^0| = 2^2$, we obtain a time $2^{60.01} > 2^{128\times 0.4165} = 2^{53.31}$ and a memory $2^{26.82} < 2^{128\times 0.2324} = 2^{29.75}$. On top of this, we must also take p_m into account.

After running optimizations for $n = 128$ to 1024, we obtained a count of about $2^{0.418m+12.851}$ blocks of m arithmetic operations (m^2 quantum gates). The point at which the algorithm starts improving over Grover search lies around $n = 157$.

Acknowledgments. This work has been partially supported by the French Agence Nationale de la Recherche through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the

Theory of Computing, El Paso, Texas, USA, May 4-6, 1997. pp. 284–293 (1997), http://doi.acm.org/10.1145/258533.258604
2. Alagic, G., Cooper, D., Dang, Q., Dang, T., Kelsey, J.M., Lichtinger, J., Liu, Y.K., Miller, C.A., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D., Apon, D.: Status report on the third round of the nist post-quantum cryptography standardization process (2022-07-05 04:07:00 2022), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934458
3. Alamati, N., Feo, L.D., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 12492, pp. 411–439. Springer (2020)
4. Becker, A., Coron, J., Joux, A.: Improved generic algorithms for hard knapsacks. In: EUROCRYPT 2011. pp. 364–385. Lecture Notes in Computer Science, Springer (2011)
5. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Post-Quantum Cryptography 2011. Lecture Notes in Computer Science, vol. 7932, pp. 16–33. Limoges, France (Jun 2013)
6. Beullens, W., Kleinjung, T., Vercauteren, F.: Csi-fish: Efficient isogeny based signatures through class group computations. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 11921, pp. 227–247. Springer (2019)
7. Bonnetain, X.: Improved low-qubit hidden shift algorithms (2019), arXiv:1901.11428
8. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. pp. 633–666. Springer International Publishing, Cham (2020)
9. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 493–522. Springer (2020)
10. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN. Lecture Notes in Computer Science, vol. 1380, pp. 163–169. Springer (1998)
11. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: Csidh: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. pp. 395–427. Springer International Publishing, Cham (2018)
12. Chávez-Saab, J., Chi-Domínguez, J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. J. Cryptogr. Eng. 12(3), 349–368 (2022)
13. Childs, A., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. Journal of Mathematical Cryptology 8(1), 1–29 (2014)
14. De Feo, L., Galbraith, S.D.: Seasign: Compact isogeny signatures from class group actions. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 11478, pp. 759–789. Springer (2019)
15. Decru, T., Panny, L., Vercauteren, F.: Faster seasign signatures through improved rejection sampling. In: PQCrypto. Lecture Notes in Computer Science, vol. 11505, pp. 271–285. Springer (2019)
16. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory 22(6), 644–654 (1976)
17. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: An optimized quantum implementation of ISD on scalable quantum resources. IACR Cryptol. ePrint Arch. p. 1608 (2021)
18. Ettinger, M., Høyer, P.: On quantum algorithms for noncommutative hidden subgroups. Lecture Notes in Computer Science p. 478–487 (1999)

19. Ettinger, M., Høyer, P., Knill, E.: The quantum query complexity of the hidden subgroup problem is polynomial (2004), arXiv:quant-ph/0401083
20. Helm, A., May, A.: Subset sum quantumly in 1.17^n. In: Jeffery, S. (ed.) TQC 2018. Leibniz International Proceedings in Informatics (LIPIcs), vol. 111, pp. 5:1–5:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018)
21. Helm, A., May, A.: The power of few qubits and collisions - subset sum below grover's bound. In: PQCrypto. Lecture Notes in Computer Science, vol. 12100, pp. 445–460. Springer (2020)
22. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) Eurocrypt 2010. Lecture Notes in Computer Science, vol. 6110. Sringer (2010)
23. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. SIAM J. Comput. 35(1), 170–188 (2005), https://doi.org/10.1137/S0097539703436345
24. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: TQC. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
25. Lyubashevsky, V., Micciancio, D.: On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In: CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 577–594. IACR, Springer (2009)
26. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
27. Peikert, C.: He gives c-sieves on the csidh. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. pp. 463–492. Springer International Publishing, Cham (2020)
28. Regev, O.: Quantum computation and lattice problems. In: FOCS. pp. 520–529. IEEE Computer Society (2002)
29. Regev, O.: New lattice-based cryptographic constructions. J. ACM 51(6), 899–942 (2004)
30. Regev, O.: A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space (2004), arXiv:quant-ph/0406151
31. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
32. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994)
33. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 617–635. Springer (2009), https://doi.org/10.1007/978-3-642-10366-7_36
34. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer (2002)