Improved Classical and Quantum Algorithms for Subset-Sum

Xavier Bonnetain[1], Rémi Bricout[2,3], André Schrottenloher[3], and Yixin Shen[4]

[1] Institute for Quantum Computing, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada
[2] Sorbonne Université, Collège Doctoral, F-75005 Paris, France
[3] Inria, France
[4] Université de Paris, IRIF, CNRS, F-75013 Paris, France

Abstract. We present new classical and quantum algorithms for solving random subset-sum instances. First, we improve over the Becker-Coron-Joux algorithm (EUROCRYPT 2011) from $\widetilde{\mathcal{O}}\left(2^{0.291n}\right)$ downto $\widetilde{\mathcal{O}}\left(2^{0.283n}\right)$, using more general representations with values in $\{-1, 0, 1, 2\}$.
Next, we improve the state of the art of quantum algorithms for this problem in several directions. By combining the Howgrave-Graham-Joux algorithm (EUROCRYPT 2010) and quantum search, we devise an algorithm with asymptotic cost $\widetilde{\mathcal{O}}\left(2^{0.236n}\right)$, lower than the cost of the quantum walk based on the same classical algorithm proposed by Bernstein, Jeffery, Lange and Meurer (PQCRYPTO 2013). This algorithm has the advantage of using *classical* memory with quantum random access, while the previously known algorithms used the quantum walk framework, and required *quantum* memory with quantum random access.
We also propose new quantum walks for subset-sum, performing better than the previous best time complexity of $\widetilde{\mathcal{O}}\left(2^{0.226n}\right)$ given by Helm and May (TQC 2018). We combine our new techniques to reach a time $\widetilde{\mathcal{O}}\left(2^{0.216n}\right)$. This time is dependent on a heuristic on quantum walk updates, formalized by Helm and May, that is also required by the previous algorithms. We show how to partially overcome this heuristic, and we obtain an algorithm with quantum time $\widetilde{\mathcal{O}}\left(2^{0.218n}\right)$ requiring only the standard classical subset-sum heuristics.

Keywords: subset-sum, representation technique, quantum search, quantum walk, list merging.

1 Introduction

We study the *subset-sum problem*, also known as *knapsack problem*: given n integers $\mathbf{a} = (a_1, \dots a_n)$, and a target integer S, find an n-bit vector $\mathbf{e} = (e_1, \dots e_n) \in \{0, 1\}^n$ such that $\mathbf{e} \cdot \mathbf{a} = \sum_i e_i a_i = S$. The *density* of the knapsack instance is defined as $d = n/(\log_2 \max_i a_i)$, and for a random instance $\mathbf{a}$, it is related to the number of solutions that one can expect.

The decision version of the knapsack problem is NP-complete [12]. Although certain densities admit efficient algorithms, related to lattice reduction [22,23], the best algorithms known for the knapsack problem when the density is close to 1 are exponential-time, which is why we name these instances "hard" knapsacks. This problem underlies some cryptographic schemes aiming at post-quantum security (see *e.g.* [24]), and is used as a building block in some quantum hidden shift algorithms [7], which have some applications in quantum cryptanalysis of isogeny-based [10] and symmetric cryptographic schemes [8].

In this paper, we focus on the case where $d = 1$, where expectedly a single solution exists. Instead of naively looking for the solution $\mathbf{e}$ via exhaustive search, in time 2^n, Horowitz and Sahni [15] proposed to use a meet-in-the-middle approach in $2^{n/2}$ time and memory. The idea is to find a collision between two lists of $2^{n/2}$ subknapsacks, *i.e.* to merge these two lists for a single solution. Schroeppel and Shamir [33] later improved this to a 4-list merge, in which the memory complexity can be reduced down to $2^{n/4}$.

The Representation Technique. At EUROCRYPT 2010, Howgrave-Graham and Joux [16] (HGJ) proposed a heuristic algorithm solving *random* subset-sum instances in time $\widetilde{\mathcal{O}}\left(2^{0.337n}\right)$, thereby breaking the $2^{n/2}$ bound. Their key idea was to represent the knapsack solution ambiguously as a sum of vectors in $\{0,1\}^n$. This *representation technique* increases the search space size, allowing to merge more lists, with new arbitrary constraints, thereby allowing for a more time-efficient algorithm. The time complexity exponent is obtained by numerical optimization of the list sizes and constraints, assuming that the individual elements obtained in the merging steps are well-distributed. This is the standard heuristic of classical and quantum subset-sum algorithms. Later, Becker, Coron and Joux [3] (BCJ) improved the asymptotic runtime downto $\widetilde{\mathcal{O}}\left(2^{0.291n}\right)$ by allowing even more representations, with vectors in $\{-1,0,1\}^n$.

The BCJ representation technique is not only a tool for subset-sums, as it has been used successfully to speed up generic decoding algorithms, classically [26,4,27] and quantumly [17]. Therefore, the subset-sum problem serves as the simplest application of representations, and improving our understanding of the classical and quantum algorithms may have consequences on these other generic problems.

Quantum Algorithms for the Subset-Sum Problem. Cryptosystems based on hard subset-sums are natural candidates for post-quantum cryptography, but in order to understand precisely their security, we have to study the best generic algorithms for solving subset-sums. The first quantum time speedup for this problem was obtained in [6], with a quantum time $\widetilde{\mathcal{O}}\left(2^{0.241n}\right)$. The algorithm was based on the HGJ algorithm. Later on, [14] devised another algorithm based on BCJ, running in time $\widetilde{\mathcal{O}}\left(2^{0.226n}\right)$. Both algorithms use the corresponding classical merging structure, wrapped in a quantum walk on a Johnson graph, in the MNRS quantum walk framework [25]. However, they suffer from two limitations.

First, both use the model of *quantum memory with quantum random-access* (QRAQM), which is stronger than the standard quantum circuit model, as it allows unit-time lookups in superposition of all the qubits in the circuit. The QRAQM model is used in most quantum walk algorithms to date, but its practical realizations are not clear at the moment. With a more restrictive model, no quantum time speedup over BCJ was previously known. This is not the case for some other hard problems in post-quantum cryptography, *e.g.* heuristic lattice sieving for solving the Shortest Vector Problem, where the best quantum algorithms to date require only *classical* memory with quantum random-access [20].

Second, both use a conjecture (implicit in [6], made explicit in [14]) about quantum walk updates. In short, the quantum walk maintains a data structure, that contains a merging tree similar to HGJ (resp. BCJ), with lists of smaller size. A quantum walk step is made of updates that changes an element in the lowest-level lists, and requires to modify the upper levels accordingly, *i.e.* to track the partial collisions that must be removed or added. In order to be efficient, the update needs to run in polynomial time. Moreover, the resulting data structure shall be a function of the lowest-level list, and not depend on the path taken in the walk. The conjecture states that it should be possible to guarantee sound updates without impacting the time complexity exponent. However, it does not seem an easy task and the current literature on subset-sums lacks further justification or workarounds.

Contributions. In this paper, we improve classical and quantum subset-sum algorithms based on representations. We write these algorithms as sequences of "merge-and-filter" operations, where lists of subknapsacks are first merged with respect to an arbitrary constraint, then *filtered* to remove the subknapsacks that cannot be part of a solution.

First, we propose a more time-efficient classical subset-sum algorithm based on representations. We relax the constraints of BCJ and introduce "2"s in the representations, obtaining a better time complexity exponent of 0.283.

Most of our contributions concern quantum algorithms. As a generic tool, we introduce *quantum filtering*, which speeds up the filtering of representations with a quantum search. We use this improvement in all our new quantum algorithms.

We give an improved quantum walk based on quantum filtering and our extended $\{-1, 0, 1, 2\}$ representations. Our best runtime exponent is 0.216., under the quantum walk update heuristic of [14]. Next, we show how to overcome this heuristic, by designing a new data structure for the vertices in the quantum walk, and a new update procedure with guaranteed time. We remove this heuristic from the previous algorithms [6,14] with no additional cost. However, we find that removing it from our quantum walk increases its cost to 0.218.

In a different direction, we devise a new quantum subset-sum algorithm based on HGJ, with time $\widetilde{\mathcal{O}}\left(2^{0.237n}\right)$. It is the first quantum time speedup on classical algorithms that is *not* based on a quantum walk. The algorithm performs instead a depth-first traversal of the HGJ tree, using quantum search as its only building block. Hence, by construction, it does not require the additional heuris-

tic of [14] *and* it only uses quantum-accessible *classical* memory, giving also the first quantum time speedup for subset-sum in this memory model.

All these complexity exponents are obtained by numerical optimization. Our code is available at `https://github.com/xbonnetain/optimization-subset-sum` .

Outline. In Section 2, we study classical algorithms. We review the representation technique, the HGJ algorithm and introduce our new $\{-1, 0, 1, 2\}$ representations to improve over [3]. In Section 3, we move to the quantum setting, introduce some preliminaries and the previous quantum algorithms for subset-sum. In Section 4, we present and study our new quantum algorithm based on HGJ and quantum search. We give different optimizations and time-memory tradeoffs. In Section 5, we present our new quantum algorithm based on a quantum walk. Finally, in Section 6 we show how to overcome the quantum walk update conjecture, up to a potential increase in the update cost. We conclude, and give a summary of our new results in Section 7.

2 List Merging and Classical Subset-sum Algorithms

In this section, we remain in the classical realm. We introduce the standard subset-sum notations and heuristics and give a new presentation of the HGJ algorithm, putting an emphasis on the *merge-and-filter* operation. We introduce our extended $\{-1, 0, 1, 2\}$ representations and detail our improvements over BCJ.

2.1 Notations and Conventions

Hereafter and in the rest of the paper, all time and memory complexities, classical and quantum, are exponential in n. We use the soft-O notation $\widetilde{\mathcal{O}}$ which removes polynomial factors in n, and focus on the asymptotic exponent, relative to n. We use $\mathrm{negl}(n)$ for any function that vanishes inverse-exponentially in n. We often replace asymptotic exponential time and memory complexities (*e.g.* $\widetilde{\mathcal{O}}\left(2^{\alpha n}\right)$) by their exponents (*e.g.* α). We use capital letters (*e.g.* L) and corresponding letters (*e.g.* ℓ) to denote the same value, in $\log_2$ and relatively to n: $\ell = \log_2(L)/n$.

Definition 1 (Entropies and multinomial functions). *We define the following functions:*

Hamming Entropy: $h(x) = -x \log_2 x - (1-x)\log_2(1-x)$

Binomial: $\mathrm{bin}\,(\omega, \alpha) = h(\alpha/\omega)\omega n$

2-way Entropy: $g(x, y) = -x \log_2 x - y \log_2 y - (1-x-y)\log_2(1-x-y)$

Trinomial: $\mathrm{trin}\,(\omega, \alpha, \beta) = g(\alpha/\omega, \beta/\omega)\omega n$

3-way Entropy: $f(x, y, z) = -x \log_2 x - y \log_2 y - z \log_2 z - (1-x-y-z)\log_2(1-x-y-z)$

Quadrinomial: $\mathrm{quadrin}\,(\omega, \alpha, \beta, \gamma) = f(\alpha/\omega, \beta/\omega, \gamma/\omega)\omega n$

Property 1 (Standard approximations). We have the following approximations, asymptotically in n:

$$\mathrm{bin}\,(\omega,\alpha) \simeq \log_2 \tbinom{\omega n}{\alpha n} \quad ; \quad \mathrm{trin}\,(\omega,\alpha,\beta) \simeq \log_2 \tbinom{\omega n}{\alpha n,\beta n}$$
$$\mathrm{quadrin}\,(\omega,\alpha,\beta,\gamma) \simeq \log_2 \tbinom{\omega n}{\alpha n,\beta n,\gamma n}$$

Definition 2 (Distributions of knapsacks). *A* knapsack *or* subknapsack *is a vector* $\mathbf{e} \in \{-1,0,1,2\}^n$*. The set of* $\mathbf{e}$ *with* αn *"-1",* $(\alpha+\beta-2\gamma)n$ *"1",* γn *"2" and* $(1-2\alpha-\beta+\gamma)n$ *"0" is denoted* $D^n[\alpha,\beta,\gamma]$*. We note* $X^n[\alpha,\beta,\gamma]$ *the random variable sampled uniformly in* $D^n[\alpha,\beta,\gamma]$*. If* $\gamma = 0$*, we may omit the third parameter. This coincides with the notation* $D^n[\alpha,\beta]$ *from [14].*

Property 2 (Size of knapsack sets). We have:

$$\tfrac{1}{n}\log_2 |D^n[0,\beta,0]| \simeq h(\beta) \quad ; \quad \tfrac{1}{n}\log_2 |D^n[\alpha,\beta,0]| \simeq g(\alpha,\alpha+\beta)$$
$$\tfrac{1}{n}\log_2 |D^n[\alpha,\beta,\gamma]| \simeq f(\alpha,\alpha+\beta-2\gamma,\gamma) \ .$$

Subset-sum. The problem we will solve is defined as follows:

Definition 3 (Random subset-sum instance of weight $n/2$**).** *Let* $\mathbf{a}$ *be chosen uniformly at random from* $(\mathbb{Z}_{2^\ell})^n$*, where* $\ell \simeq n$*. Let* $\mathbf{e}$ *be chosen uniformly at random from* $D^n[0,1/2,0]$*. Let* $t = \mathbf{a}\cdot\mathbf{e} \pmod{2^\ell}$*. Then* $\mathbf{a}, t$ *is a random subset-sum instance. A* solution *is a vector* $\mathbf{e}'$ *such that* $\mathbf{a}\cdot\mathbf{e}' \equiv t \pmod{2^\ell}$*.*

Sampling. Throughout this paper, we assume that we can classically sample uniformly at random from $D^n[\alpha,\beta,\gamma]$ in time poly(n). (Since αn, βn and γn will in general not be integer, we suppose to have them rounded to the nearest integer.) This comes from an efficient bijection between representations and integers (see Lemma 10 in Appendix A). In addition, we can efficiently produce the uniform superposition of vectors of $D^n[\alpha,\beta,\gamma]$, using poly(n) quantum gates, and we can perform a quantum search among representations.

2.2 Merging and Filtering

In all subset-sum algorithms studied in this paper, we repeatedly sample vectors with certain distributions $D^n[*,*,*]$, then combine them. Let $D_1 = D^n[\alpha_1,\beta_1,\gamma_1]$, $D_2 = D^n[\alpha_2,\beta_2,\gamma_2]$ be two input distributions and $D = D^n[\alpha,\beta,\gamma]$ a target. Given two lists $L_1 \in D_1^{|L_1|}$ and $L_2 \in D_2^{|L_2|}$, we define:

- the *merged list* $L = L_1 \bowtie_c L_2$ containing all vectors $\mathbf{e} = \mathbf{e_1} + \mathbf{e_2}$ such that: $\mathbf{e_1} \in L_1, \mathbf{e_2} \in L_2$, $(\mathbf{e}_1+\mathbf{e}_2)\cdot\mathbf{a} = s \mod M$, $s \le M$ is an arbitrary integer and $M \approx 2^{cn}$ (we write $L_1 \bowtie_c L_2$ because s is an arbitrary value, whose choice is without incidence on the algorithm)
- the *filtered list* $L^f = (L \cap D) \subseteq L$, containing the vectors with the target distribution of $1,-1,2$ (the target D will always be clear from context).

In general, L is exponentially bigger than L^f and does not need to be written down, as vectors can be filtered on the fly. The algorithms then repeat the merge-and-filter operation on multiple levels, moving towards the distribution

$D^n[0, 1/2]$ while increasing the bit-length of the modular constraint, until we satisfy $\mathbf{e} \cdot \mathbf{a} = t \mod 2^n$ and obtain a solution.

The standard subset-sum heuristic assumes that vectors in L^f are drawn independently, uniformly at random from D. It simplifies the complexity analysis of both classical and quantum algorithms studied in this paper. Note that this heuristic, which is backed by experiments, actually leads to provable probabilistic algorithms in the classical setting (see [3, Theorem 2]).

Heuristic 1 *If input vectors are uniformly distributed in $D_1 \times D_2$, then the filtered pairs are uniformly distributed in D (more precisely, among the subset of vectors in D satisfying the modular condition).*

Filtering Representations. We note $\ell = (1/n) \log_2 |L|$, and so on for ℓ_1, ℓ_2, ℓ^f. By Heuristic 1, the average sizes of L_1, L_2, L and L^f are related by:

- $\ell = \ell_1 + \ell_2 - c$
- $\ell^f = \ell + \mathsf{pf}$, where pf is negative and $2^{\mathsf{pf}n}$ is the probability that a pair $(\mathbf{e_1}, \mathbf{e_2})$, drawn uniformly at random from $D_1 \times D_2$, has $(\mathbf{e_1} + \mathbf{e_2}) \in D$.

In particular, the occurrence of collisions in L^f is a negligible phenomenon, unless ℓ_f approaches $(\log_2 |D|/n) - c$, which is the maximum number of vectors in D with constraint c. For a given random knapsack problem, with high probability, the size of any list built by sampling, merging and filtering remains very close to its average (by a Chernoff bound and a union bound on all lists).

Here, pf depends only on D_1, D_2 and D. Working with this *filtering probability* is especially useful for writing down our algorithm in Section 4. We give its formula for $\{0, 1\}$ representations below. Two similar results for $\{-1, 0, 1\}$ and $\{-1, 0, 1, 2\}$ can be found in Appendix B.

Lemma 1 (Filtering HGJ-style representations). *Let $\mathbf{e_1} \in D^n[0, \alpha]$ and $\mathbf{e_2} \in D^n[0, \beta]$. The probability that $\mathbf{e_1} + \mathbf{e_2} \in D^n[0, \alpha + \beta]$ is $2^{\mathsf{pf}_1(\alpha,\beta)n}$, with*

$$\mathsf{pf}_1(\alpha, \beta) = \mathrm{bin}(1 - \alpha, \beta) - h(\beta) = \mathrm{bin}(1 - \beta, \alpha) - h(\alpha)$$

if $\alpha + \beta \leq 1$, and the probability is 0 otherwise.

Proof. The probability that a $\mathbf{e}_1 + \mathbf{e}_2$ survives the filtering is:

$$\binom{n - \alpha n}{\beta n} / \binom{n}{\beta n} = \binom{n - \beta n}{\alpha n} / \binom{n}{\alpha n} \ .$$

Indeed, given a choice of αn bit positions among n, the other βn bit positions must be compatible, hence chosen among the $(1 - \alpha)n$ remaining positions. By taking the $\log_2$, we obtain the formula for the filtering probability. □

Time Complexity of Merging. Classically, the time complexity of the merge-and-filter operation is related to the size of the *merged list*.

Lemma 2 (Classical merging with filtering). *Let L_1 and L_2 be two lists stored in classical memory with random access. In $\log_2$, relatively to n, and discarding logarithmic factors, merging and filtering L_1 and L_2 costs a time $\max(\min(\ell_1, \ell_2), \ell_1 + \ell_2 - c)$ and memory $\max(\ell_1, \ell_2, \ell^f)$, assuming that we must store the filtered output list.*

Proof. Assuming sorted lists, there are two symmetric ways to produce a stream of elements of $L_1 \bowtie_c L_2$: we can go through the elements of L_1, and for each one, find the matching elements in L_2 by dichotomy search (time $\ell_1 + \max(0, \ell_2 - c)$) or we can exchange the role of L_1 and L_2. Although we do not need to store $L_1 \bowtie_c L_2$, we need to examine all its elements in order to filter them. □

2.3 The HGJ Algorithm

We start our study of classical subset-sum by recalling the algorithm of Howgrave-Graham and Joux [16], with the corrected time complexity of [3]. The algorithm builds a merging tree of lists of subknapsacks, with four levels, numbered 3 downto 0. Level j contains 2^j lists. In total, 8 lists are merged together into one.

Level 3. We build 8 lists denoted $L_0^3 \dots L_7^3$. They contain *all* subknapsacks of weight $\frac{n}{16}$ on $\frac{n}{2}$ bits, either left or right:

$$\begin{cases} L_{2i}^3 &= D^{n/2}[0, 1/8] \times \{0^{n/2}\} \\ L_{2i+1}^3 &= \{0^{n/2}\} \times D^{n/2}[0, 1/8] \end{cases}$$

Property 2 provides that these level-3 lists have size $\ell_3 = h(1/8)/2$. As the positions set to 1 cannot interfere, these is no filtering when merging L_{2i}^3 and L_{2i+1}^3.

Level 2. We merge the lists pairwise with a (random) constraint on c_2 bits, and obtain 4 filtered lists. The size of the filtered lists plays a role in the memory complexity of the algorithm, but the time complexity depends on the size of the unfiltered lists.

In practice, when we say "with a constraint on c_j bits", we assume that given the subset-sum objective t modulo 2^n, random values r_i^j such that $\sum_i r_i^j = t \mod 2^{c_j}$ are selected at level j, and the r_i^j have $c_j n$ bits only. Hence, at this step, we have selected 4 integers on c_2 bits $r_0^1, r_1^1, r_2^1, r_3^1$ such that $r_0^1 + r_1^1 + r_2^1 + r_3^1 = t \mod 2^{c_2}$. The 4 level-2 lists $L_0^2, L_1^2, L_2^2, L_3^2$ have size $\ell_2 = (h(1/8) - c_2)$, they contain subknapsacks of weight $\frac{n}{8}$ on n bits.

Level 1. We merge the lists pairwise with $c_1 - c_2$ new bits of constraint, ensuring that the constraint is compatible with the previous ones. We obtain two filtered lists L_0^1, L_1^1, containing subknapsacks of weight $n/4$. They have size:

$$\ell_1 = 2\ell_2 - (c_1 - c_2) + \mathsf{pf}_1(1/8, 1/8)$$

where $\mathsf{pf}_1(1/8, 1/8)$ is given by Lemma 1.

Level 0. We find a solution to the subset-sum problem with the complete constraint on n bits. This means that the list L^0 must have expected length $\ell_0 = 0$. Note that there remains $1 - c_1$ bits of constraint to satisfy, and the filtering term is similar as before, so:

$$\ell_0 = 2\ell_1 - (1 - c_1) + \mathsf{pf}_1\left(1/4, 1/4\right) \quad .$$

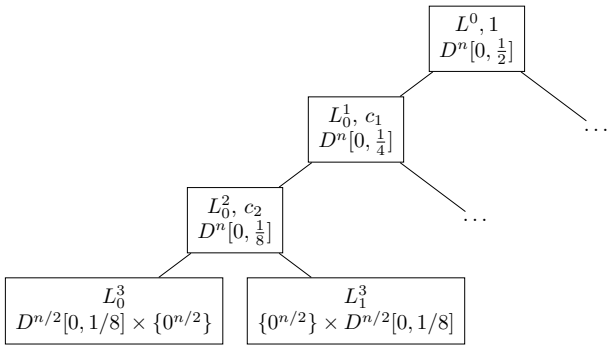

Fig. 1. The HGJ algorithm (duplicate lists are omitted)

By Lemma 2, the time complexity of this algorithm is determined by the sizes of the unfiltered lists:

$$\max\left(\ell_3, 2\ell_3 - c_2, 2\ell_2 - (c_1 - c_2), 2\ell_1 - (1 - c_1)\right)$$

and the memory complexity, by the sizes of the filtered lists $\max\left(\ell_3, \ell_2, \ell_1\right)$. By a numerical optimization, one obtains a time exponent of $0.337n$.

2.4 The BCJ Algorithm and our improvements

The HGJ algorithm uses representations to increase artificially the search space. The algorithm of Becker, Coron and Joux [3] improves the runtime exponent down to 0.291 by allowing even more freedom in the representations, which can now contain "-1". The "-1" have to cancel out progressively, to ensure the validity of the final knapsack solution.

We improved over this algorithm in two different ways. First, we relaxed the constraints $\ell_j + c_j = g(\alpha_j, 1/2^{j+1})$ enforced in [3], as only the inequalities $\ell_j + c_j \leqslant g(\alpha_j, 1/2^{j+1})$ are necessary. When optimizing the parameters under these new constraints, we bring the asymptotic time exponent down to $0.289n$.

Using $\{-1, 0, 1, 2\}$ *representations.* Next, we allow the value "2" in the subknapsacks. The algorithm builds a merging tree with five levels, numbered 4 downto 0. Level j contains 2^j lists. In total, 16 lists are merged together into one.

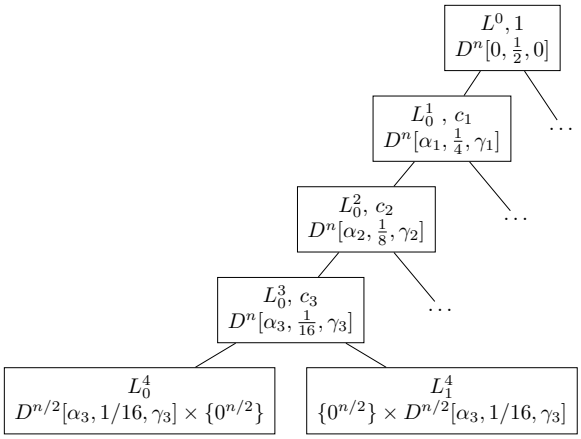

Fig. 2. Our improved algorithm (duplicate lists are omitted).

Level 4. We build 16 lists $L_0^4 \dots L_{15}^4$. They contain complete distributions on $\frac{n}{2}$ bits, either left or right, with $\frac{n}{32} + \frac{\alpha_3 n}{2} - \gamma_3 n$ "1", $\frac{\alpha_3 n}{2}$ "-1" and $\frac{\gamma_3 n}{2}$ "2":

$$\begin{cases} L_{2i}^4 &= D^{n/2}[\alpha_3, 1/16, \gamma_3] \times \{0^{n/2}\} \\ L_{2i+1}^4 &= \{0^{n/2}\} \times D^{n/2}[\alpha_3, 1/16, \gamma_3] \end{cases}$$

As before, this avoids filtering at the first level. These lists have size: $\ell_4 = f(\alpha_3, 1/16 + \alpha_3 - 2\gamma_3, \gamma_3)/2$.

Level 3. We merge into 8 lists $L_0^3 \dots L_7^3$, with a constraint on c_3 bits. As there is no filtering, these lists have size: $\ell_3 = f(\alpha_3, 1/16 + \alpha_3 - 2\gamma_3, \gamma_3) - c_3$.

Level 2. We now merge and filter. We force a target distribution $D^n[\alpha_2, 1/8, \gamma_2]$, with α_2 and γ_2 to be optimized later. There is a first filtering probability p_2 given by Lemma 12. We have $\ell_2 = 2\ell_3 - (c_2 - c_3) + p_2$.

Level 1. Similarly, we have: $\ell_1 = 2\ell_2 - (c_1 - c_2) + p_1$.

Level 0. We have $\ell_0 = 2\ell_1 - (1 - c_1) + p_0 = 0$, since the goal is to obtain one solution in the list L_0.

With these constraints, we find a time $\widetilde{\mathcal{O}}\left(2^{0.2830n}\right)$ (rounded upwards) with the following parameters:

$$\alpha_1 = 0.0340, \alpha_2 = 0.0311, \alpha_3 = 0.0202, \gamma_1 = 0.0041, \gamma_2 = 0.0006, \gamma_3 = 0.0001$$

$$c_1 = 0.8067, c_2 = 0.5509, c_3 = 0.2680, p_0 = -0.2829, p_1 = -0.0447, p_2 = -0.0135$$

$$\ell_1 = 0.2382, \ell_2 = 0.2694, \ell_3 = 0.2829, \ell_4 = 0.2755$$

3 Quantum Preliminaries and Previous Work

In this section, we recall some preliminaries of quantum computation (quantum search and quantum walks) that will be useful throughout the rest of this paper. We also recall previous quantum algorithms for subset-sum.

3.1 Quantum Preliminaries

As we consider all our algorithms from the point of view of asymptotic complexities, and neglect polynomial factors in n, a high-level overview is often enough, and we will use quantum building blocks as black boxes. The interested reader may find more details in [31].

Quantum Computation Model. All the quantum algorithms considered in this paper run in the quantum circuit model, with quantum random-access memory, often denoted as qRAM. "Baseline" quantum circuits are simply built using a universal gate set. Many quantum algorithms use qRAM access, and require the circuit model to be augmented with the so-called "qRAM gate". This includes subset-sum, lattice sieving and generic decoding algorithms that obtain time speedups with respect to their classical counterparts. Given an input register $1 \leq i \leq r$, which represents the index of a memory cell, and many quantum registers $|x_1, \dots x_r\rangle$, which represent stored data, the qRAM gate fetches the data from register x_i:

$$|i\rangle |x_1, \dots x_r\rangle |y\rangle \mapsto |i\rangle |x_1, \dots x_r\rangle |y \oplus x_i\rangle \quad .$$

This then enables various quantum data structures, such as the ones depicted in [6] and [1], with fast access, even if the queries and the data are superpositions.

We will use the terminology of [19] for the qRAM gate:

- If the input i is classical, then this is the plain quantum circuit model;
- If the x_j are classical, we have *quantum-accessible classical memory* (QRACM)
- In general, we have *quantum-accessible quantum memory* (QRAQM)

All known quantum algorithms for subset-sum with a quantum time speedup over the best classical one require QRAQM. For comparison, speedups on heuristic lattice sieving algorithms exist in the QRACM model [21,18], including the best one to date [20]. While no physical architecture for quantum random access has been proposed, that would indeed produce a constant or negligible overhead in time, some authors [19] consider the separation meaningful. For example, notice that a quantum random access to a QRACM of size N can be emulated in time $\mathcal{O}(N)$, while this is not the case for QRAQM.

Quantum Search. One of the most well-known quantum algorithms is Grover's unstructured search algorithm [13]. We present here its generalization, amplitude amplification [11].

Lemma 3 (Amplitude amplification, from [11]). *Let $\mathcal{A}$ be a reversible quantum circuit, f a computable boolean function over the output of $\mathcal{A}$, O_f its implementation as a quantum circuit, and a be the initial success probability of $\mathcal{A}$, that is, the probability that $O_f\mathcal{A}|0\rangle$ outputs "true". There exists a quantum reversible algorithm that calls $\mathcal{O}\left(\sqrt{1/a}\right)$ times $\mathcal{A}$, $\mathcal{A}^\dagger$ and O_f, uses as many qubits as $\mathcal{A}$ and O_f, and produces an output that passes the test f with probability greater than* $\max(a, 1-a)$.

This is known to be optimal when the functions are black-box oracles [5].

As we will use quantum search as a subprocedure, we make some remarks similar to [28, Appendix A.2] and [9, Section 5.2] to justify that, up to additional polynomial factors in time, we can consider it runs with no errors and allows to return all the solutions efficiently.

Remark 1 (Error in a sequence of quantum searches). Throughout this paper, we will assume that a quantum search in a search space of size S with T solutions runs in exact time $\sqrt{S/T}$. In practice, there is a constant overhead, but since S and T are always exponential in n, the difference is negligible. Furthermore, this is a probabilistic procedure, and it will return a wrong result with a probability of the order $\sqrt{T/S}$. As we can test if an error occurs, we can make it negligible by redoing the quantum search polynomially many times.

Remark 2 (Finding all solutions). Quantum search returns a solution among the T wanted, selected uniformly at random. Finding all solutions is then an instance of the coupon collector problem with T coupons [29]. For any β, the probability that recovering all coupons takes time t satisfies:

$$\Pr\left[t > \beta T \log T\right] \leq T^{-\beta+1} \ .$$

In our algorithms, T is always exponential in n. Therefore, our analysis can assume that all coupons are actually recovered in time $\widetilde{\mathcal{O}}\left(T\right)$.

Quantum Walks. Quantum walks can be seen as a generalization of quantum search. They allow to obtain polynomial speedups on many unstructured problems, with sometimes optimal results (*e.g.* Ambainis' algorithm for element distinctness [1]). In this paper, we will consider walks in the MNRS framework [25].

Let $G = (V, E)$ be an undirected, connected, regular graph, such that some vertices of G are "marked". Let ϵ be the fraction of marked vertices, that is, a random vertex has a probability ϵ of being marked. Let δ be the spectral gap of G, which is defined as the difference between its two largest eigenvalues.

In a *classical* random walk on G, we can start from any vertex and reach the stationary distribution in approximately $\frac{1}{\delta}$ random walk steps. Then, such a random vertex is marked with probability ϵ. Assume that we have a procedure `Setup` that samples a random vertex to start with in time S, `Check` that verifies if a vertex is marked or not in time C and `Update` that performs a walk step in time U, then we will have found a marked vertex in expected time: $\mathsf{S} + \frac{1}{\epsilon}\left(\frac{1}{\delta}\mathsf{U} + \mathsf{C}\right)$.

Quantum walks reproduce the same process, except that their internal state is not a vertex of G, but a superposition of vertices. The walk starts in the uniform superposition $\sum_{v \in V} |v\rangle$, which must be generated by the `Setup` procedure. It repeats $\sqrt{1/\epsilon}$ iterations that, similarly to amplitude amplification, move the amplitude towards the marked vertices. An update produces, from a vertex, the superposition of its neighbors. Each iteration does not need to repeat $\frac{1}{\delta}$ vertex updates and, instead, takes a time equivalent to $\sqrt{1/\delta}$ updates to achieve a good mixing.

Theorem 1 (Quantum walk [25]). *Let $G = (V, E)$ be a regular graph with spectral gap $\delta > 0$. Let $\epsilon > 0$ be a lower bound on the probability that a vertex chosen randomly of G is marked. For a random walk on G, let $\mathsf{S}, \mathsf{U}, \mathsf{C}$ be the setup, update and checking cost. Then there exists a quantum algorithm that with high probability finds a marked vertex in time*

$$\mathsf{S} + \frac{1}{\sqrt{\epsilon}} \left(\frac{1}{\sqrt{\delta}} \mathsf{U} + \mathsf{C} \right).$$

3.2 Solving Subset-sum with Quantum Walks

In 2013, Bernstein, Jeffery, Lange and Meurer [6] constructed quantum subset sum algorithms inspired by Schroeppel-Shamir [33] and HGJ [16]. We briefly explain the idea of their quantum walk for HGJ.

The graph G that they consider is a product Johnson graph. We recall formal definitions from [17].

Definition 4 (Johnson graph). *A Johnson graph $J(N, R)$ is an undirected graph whose vertices are the subsets of R elements among a set of size N, and there is an edge between two vertices S and S' iff $|S \cap S'| = R - 1$, in other words, if S' can be obtained from S by replacing an element. Its spectral gap is given by $\delta = \frac{N}{R(N-R)}$.*

Theorem 2 (Cartesian product of Johnson graphs [17]). *Let $J^m(N, R)$ be defined as the cartesian product of m Johnson graphs $J(N, R)$,* i.e.*, a vertex in $J^m(N, R)$ is a tuple of m subsets $S_1, \dots S_m$ and there is an edge between $S_1, \dots S_m$ and $S'_1, \dots S'_m$ iff all subsets are equal except two of them, which have $|S_i \cap S'_i| = R - 1$. Then it has $\binom{N}{R}^m$ vertices and its spectral gap is greater than $\frac{1}{m} \frac{N}{R(N-R)}$.*

In our case, a vertex contains a product of 8 sublists $L'^3_0 \subset L^3_0, \dots, L'^3_7 \subset L^3_7$ of a smaller size than the classical lists: $\ell < \ell_3$. There is an edge between two vertices if we can transform one into the other by replacing only one element in one of the sublists. The spectral gap of such a graph is (in $\log_2$, relative to n) $-\ell$.

In addition, each vertex has an internal data structure which reproduces the HGJ merging tree, from level 3 to level 0. Since the initial lists are smaller, the list L^0 is now of expected size $8(\ell - \ell_3)$ (in $\log_2$, relative to n), *i.e.* the walk needs to run for $4(\ell_3 - \ell)$ steps. Each step requires $\ell/2$ updates.

In the setup procedure, we simply start from all choices for the sublists and build the tree by merging and filtering. Assuming that the merged lists have decreasing sizes, the setup time is ℓ. The vertex is marked if it contains a solution at level 0. Hence, checking if a vertex is marked takes time $\mathsf{C} = 1$, but the update procedure needs to ensure the consistency of the data structure. Indeed, when updating, we remove an element $\mathbf{e}$ from one of the lists L'^3_i and replace it by a $\mathbf{e}'$ from L^0_i. We then have to track all subknapsacks in the upper levels where $\mathbf{e}$ intervened, to remove them, and to add the new collisions where $\mathbf{e}'$ intervenes.

Assuming that the update can run in poly(n), an optimization with the new parameter ℓ yields an exponent 0.241. In [6], the parameters are such that on average, a subknapsack intervenes only in a single sum at the next level. The authors propose to simply limit the number of elements to be updated at each level, in order to guarantee a constant update time.

Quantum Walk Based on BCJ. In [14], Helm and May quantize, in the same way, the BCJ algorithm. They add "-1" symbols and a new level in the merging tree data structure, reaching a time exponent of 0.226. But they remark that this result depends on a conjecture, or a heuristic, that was implicit in [6].

Heuristic 2 (Helm-May) *In these quantum walk subset-sum algorithms, an update with expected constant time* U *can be replaced by an update with exact time* U *without affecting the runtime of the algorithm, up to a polynomial factor.*

Indeed, it is easy to construct "bad" vertices and edges for which an exact update, *i.e.* the complete reconstruction of the merging tree, will take exponential time: by adding a single new subknapsack $\mathbf{e}$, we find an exponential number of pairs $\mathbf{e} + \mathbf{e}'$ to include at the next level. So we would like to update only a few elements among them. But in the MNRS framework, the data structure of a vertex must depend solely on the vertex itself (*i.e.* on the lowest-level lists in the merging tree). And if we do as proposed in [6], we add a dependency on the path that lead to the vertex, and lose the consistency of the walk.

In a related context, the problem of "quantum search with variable times" was studied by Ambainis [2]. In a quantum search for some x such that $f(x) = 1$, in a set of size N, if the time to evaluate f on x is always 1, then the search requires time $\mathcal{O}\left(\sqrt{N}\right)$. Ambainis showed that if the elements have different evaluation times $t_1, \dots t_N$, then the search now requires $\widetilde{\mathcal{O}}(\sqrt{t_1^2 + \ldots + t_N^2})$, the geometric mean of $t_1, \dots t_N$. As quantum search can be seen as a particular type of quantum walk, this shows that Heuristic 2 is wrong in general, as we can artificially create a gap between the geometric mean and expectation of the update time U; but also, that it may be difficult to actually overcome. In this paper, we will obtain different heuristic and non-heuristic times.

4 Quantum Asymmetric HGJ

In this section, we give the first quantum algorithm for the subset-sum problem, *in the QRACM model*, with an asymptotic complexity smaller than BCJ.

4.1 Quantum Match-and-Filter

We open this section with some technical lemmas that replace the classical merge-and-filter Lemma 2. In this section, we will consider a merging tree as in the HGJ algorithm, but this tree will be built quantumly, with a quantum search. The following lemmas bound the expected time of merge-and-filter *and*

match-and-filter operations performed quantumly, in the QRACM model. This will have consequences both in this section and in the next one.

First, we remark that by using an adapted data structure as in [6], we can use the QRACM model for finding *partial matching elements*, which is more general than a simple lookup. When there are multiple solutions, we obtain their uniform superposition.

Lemma 4 (QRACM query with multiple answers). *Let M be a memory of size 2^m holding n-bit strings $x_0, \dots x_{2^m-1}$. Then in the QRACM model, there exists a data structure for M that can answer, in* $\mathrm{poly}(m,n)$ *time, partial matching queries with multiple answers:*

$$|y\rangle |0\rangle , s \mapsto |y\rangle \sum_{i|y\oplus x_i \ starts \ with \ s} |i\rangle \quad .$$

Next, we write a lemma for quantum *matching* with filtering, in which one of the lists is not written down. We start from a unitary that produces the uniform superposition of the elements of a list L_1, and we wrap it into an amplitude amplification, in order to obtain a unitary that produces the uniform superposition of the elements of the merged-and-filtered list.

Lemma 5 (Quantum matching with filtering). *Let L_2 be a list stored in QRACM. Assume given a unitary U that produces in time t_{L_1} the uniform superposition of $L_1 = x_0, \dots x_{2^m-1}$:*

$$|0\rangle |0\rangle \mapsto \sum_i |i\rangle |x_i\rangle \quad .$$

We merge L_1 and L_2 with a modular condition of c bits and a filtering probability p. Let L be the merged list and L^f the filtered list. Assume $|L^f| \geq 1$. Then there exists a unitary U' producing the uniform superposition of L^f in time: $\widetilde{\mathcal{O}}\left(\frac{t_{L_1}}{\sqrt{p}} \max(\sqrt{2^c/|L_2|}, 1)\right)$.

Notice that this is also the time complexity to produce a single random element of L^f. If we want to produce and store the whole list L^f, it suffices to multiply this complexity by the number of elements in L^f (*i.e.* $p|L_1||L_2|/2^c$). We would obtain: $\widetilde{\mathcal{O}}\left(t_{L_1}\sqrt{p}\max\left(|L_1|\sqrt{\frac{|L_2|}{2^c}}, \frac{|L_1||L_2|}{2^c}\right)\right)$.

Proof. We separate three cases.

- If $|L_2| < 2^c$, then we have no choice but to make a quantum search on elements of L_1 that match the modular constraint and pass the filtering step, in time: $\widetilde{\mathcal{O}}\left(t_{L_1}\sqrt{\frac{2^c}{L_2 p}}\right)$.
- If $|L_2| > 2^c$ but $|L_2| < 2^c/p$, an element of L_1 will always pass the modular constraint, with more than one candidate, but in general all these candidates will be filtered out. Given an element of L_1, producing the superposition of

these candidates is done in time 1, so finding the one that passes the filter, if there is one, takes time $\sqrt{|L_2|/2^c}$. Next, we wrap this in a quantum search to find the "good" elements of L_1 (passing the two conditions), with $\sqrt{(2^c)/pL_2}$ iterations. The total time is:

$$\widetilde{\mathcal{O}}\left(\sqrt{\frac{2^c}{L_2 p}} \times \left(\sqrt{|L_2|/2^c} \times t_{L_1}\right) = \frac{t_{L_1}}{\sqrt{p}}\right) \ .$$

- If $|L_2| > 2^c/p$, then an element of L_1 yields on average more than one filtered candidate. Producing the superposition of the modular candidates is done in time 1 with a QRACM query, then finding the superposition of filtered candidates requires $1/\sqrt{p}$ iterations. The total time is: $\widetilde{\mathcal{O}}\left(t_{L_1}/\sqrt{p}\right)$.

The total time in all cases is: $\widetilde{\mathcal{O}}\left(\frac{t_{L_1}}{\sqrt{p}} \max(\sqrt{2^c/|L_2|}, 1)\right)$. □

In the QRACM model, we have the following corollary for merging and filtering two lists of equal size. This result will be helpful in Section 4.3 and 5.

Corollary 1. *Consider two lists L_1, L_2 of size $|L_1| = |L_2| = |L|$ exponential in n. We merge L_1 and L_2 with a modular constraint c, and filter, obtaining a list L^f. Assume that $|L^f| \geq 1$. Then this operation can be done in quantum time $\widetilde{\mathcal{O}}\left(|L^f|\sqrt{\frac{|L|^2}{2^{cn}|L^f|}}\right)$.*

Proof. We do a quantum search to find each element of L^f. Finding an element of L^f costs time $\widetilde{\mathcal{O}}\left(\sqrt{\frac{|L|^2}{2^{cn}|L^f|}}\right)$. We then find all elements of L^f by repeating this search (this is a coupon collector problem). □

4.2 Revisiting HGJ

We now introduce our new algorithm for subset-sum in the QRACM model.

Our starting point is the HGJ algorithm. Similarly to [28], we use a merging tree in which the lists at a given level may have different sizes. Classically, this does not improve the time complexity. However, quantumly, we will use quantum filtering. Since our algorithm does not require to write data in superposition, only to read from classical registers with quantum random access, we require only QRACM instead of QRAQM.

In the following, we consider that all lists, except L_0^3, L_0^2, L_0^1, L^0, are built with classical merges. The final list L^0, containing (expectedly) a single element, and a branch leading to it, are part of a nested quantum search. Each list L_0^3, L_0^2, L_0^1, L^0 corresponds either to a search space, the solutions of a search, or both. We represent this situation on Fig. 3. Our procedure runs as follows:

1. (Classical step): build the *intermediate lists* L_1^3, L_1^2, L_1^1 and store them in QRACM
2. (Quantum step): perform a quantum search on L_0^3. In order to test a vector $\mathbf{e} \in L_0^3$:

- Find $\mathbf{e}_3 \in L_1^3$ such that $\mathbf{e} + \mathbf{e}_3$ passes the c_0^2-bit modular constraint (assume that there is at most one such solution). There is no filtering here.
- Find $\mathbf{e}_2 \in L_1^2$ such that $(\mathbf{e} + \mathbf{e}_3) + \mathbf{e}_2$ passes the additional $(c^1 - c_0^2)$-bit constraint.
- If it also passes the filtering step, find $\mathbf{e}_1 \in L_1^1$ such that $(\mathbf{e}+\mathbf{e}_3+\mathbf{e}_2)+\mathbf{e}_1$ is a solution to the knapsack problem (and passes the filter).

Structural constraints are imposed on the tree, in order to guarantee that there exists a knapsack solution. The only difference between the quantum and classical settings is in the optimization goal: the final time complexity.

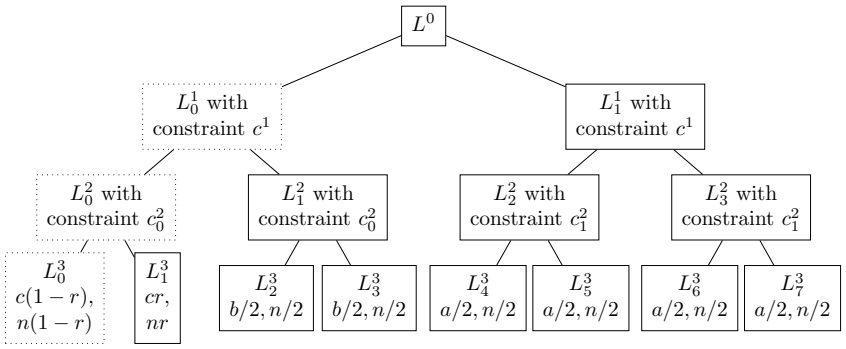

Fig. 3. Quantum HGJ algorithm

Structural Constraints. We now introduce the variables and the structural constraints that determine the shape of the tree in Fig. 3. The asymmetry happens both in the weights at level 0 and at the constraints at level 1 and 2. Let us denote $\ell_i^j = (\log_2 |L_i^j|)/n$. With the lists built classically,we expect a symmetry to be respected, so we have: $\ell_2^3 = \ell_3^3$, $\ell_4^3 = \ell_5^3 = \ell_6^3 = \ell_7^3$, $\ell_2^2 = \ell_3^2$. We also tweak the left-right split at level 0: lists from L_2^3 to L_7^3 have a standard balanced left-right split; however, we introduce a parameter r that determines the proportion of positions set to zero in list L_0^3: in L_0^3, the vectors weigh $c(1-r)$ on a support of size $n(1-r)$, instead of $c/2$ on a support of size $n/2$. In total we have $c + b + 2a = \frac{1}{2}$, as the weight of the solution is supposed to be exactly $n/2$.

Then we note that:

- The lists at level 3 have a maximal size depending on the corresponding weight of their vectors:

$$\ell_0^3 \leq h(c)(1-r), \ell_1^3 \leq h(c)r, \ell_2^3 = \ell_3^3 \leq h(b)/2, \ell_4^3 \leq h(a)/2$$

- The lists at level 2 cannot contain more representations than the filtered list of all subknapsacks of corresponding weight:

$$\ell_0^2 \leq h(c) - c_0^2, \ell_1^2 \leq h(b) - c_0^2, \ell_2^2 = \ell_3^2 \leq h(a) - c_1^2$$

- Same at levels 1 and 0:

$$\ell_0^1 \leq h(c+b) - c^1, \ell_1^1 \leq h(2a) - c^1$$

- The merging at level 2 is exact (there is no filtering):

$$\ell_0^2 = \ell_0^3 + \ell_1^3 - c_0^2, \quad \ell_1^2 = \ell_2^3 + \ell_3^3 - c_0^2, \quad \ell_2^2 = \ell_4^3 + \ell_5^3 - c_1^2, \quad \ell_3^2 = \ell_6^3 + \ell_7^3 - c_1^2,$$

- At level 1, with a constraint $c^1 \geq c_0^2, c_1^2$ that subsumes the previous ones:

$$\ell_0^1 = \ell_0^2 + \ell_1^2 - c^1 + c_0^2 + \mathsf{pf}_1(b,c) \ , \qquad \ell_1^1 = \ell_2^2 + \ell_3^2 - c^1 + c_1^2 + \mathsf{pf}_1(a,a)$$

- And finally at level 0:

$$\ell^0 = 0 = \ell_0^1 + \ell_1^1 - (1 - c^1) + \mathsf{pf}_1(b+c, 2a)$$

Classical Optimization. All the previous constraints depend on the problem, not on the computation model. Now we can get to the time complexity in the classical setting, that we want to minimize:

$$\max\big(\ell_4^3, \ell_2^3, \ell_1^3, \ell_2^3 + \ell_3^3 - c_0^2, \ell_4^3 + \ell_5^3 - c_1^2, \ell_2^2 + \ell_3^2 - c^1 + c_1^3, \\ \ell_0^3 + \max(\ell_1^3 - c_0^2, 0) + \max(\ell_1^2 - c^1 + c_0^2, 0)\big) \ .$$

The last term corresponds to the exhaustive search on ℓ_0^3. In order to keep the same freedom as before, it is possible that an element of L_0^3 matches against *several* elements of L_1^3, all of which yield a potential solution that has to be matched against L_1^2, *etc.* Hence for each element of L_0^3, we find the expected $\max(\ell_1^3 - c_0^2, 0)$ candidates matching the constraint c_0^1. For each of these candidates, we find the expected $\max(\ell_1^2 - c^1 + c_0^2, 0)$ candidates matching the constraint c^1. For each of these candidates, if it passes the filter, we search for a collision in L_1^1. We verified that optimizing the classical time under our constraints gives the time complexity of HGJ.

Quantum Optimization. The time complexity for producing the intermediate lists is unchanged. The only difference is the way we find the element in L_0^3 that will lead to a solution, which is a nested sequence of quantum searches.

- We can produce the superposition of all elements in L_0^2 in time

$$t_2 = \frac{1}{2}\max(c_0^2 - \ell_1^3, 0)$$

- So by Lemma 5, we can produce the superposition of all elements in L_0^1 in time

$$t_2 - \frac{1}{2}\mathsf{pf}_1(b,c) + \frac{1}{2}\max\left(c^1 - c_0^2 - \ell_1^2, 0\right)$$

- Finally, we expect that there are

$$\ell_0^2 + \ell_1^2 - c^1 + c_0^2 + \mathsf{pf}_1(b,c)$$

elements in L_0^1, which gives the number of iterations of the quantum search.

The time of this search is:

$$\frac{1}{2}\left(\ell_0^1 + \max(c_0^2 - \ell_1^3, 0) - \mathsf{pf}_1(b,c) + \max\left(c^1 - c_0^2 - \ell_1^2, 0\right)\right)$$

and the total time complexity is:

$$\max\Big(\ell_4^3, \ell_2^3, \ell_1^3, \ell_2^3 + \ell_3^3 - c_0^2, \ell_4^3 + \ell_5^3 - c_1^2, \ell_2^2 + \ell_3^2 - c^1 + c_1^2, \\ \frac{1}{2}\left(\ell_0^1 + \max(c_0^2 - \ell_1^3, 0) - \mathsf{pf}_1(b,c) + \max\left(c^1 - c_0^2 - \ell_1^2, 0\right)\right)\Big)$$

We obtain a quantum time complexity exponent of 0.2374 with this method (the detailed parameters are given in Table 1).

4.3 Improvement via Quantum Filtering

Let us keep the tree structure of Figure 3 and its structural constraints. The final quantum search step is already made efficient with respect to the filtering of representations, as we only pay half of the filtering term $\mathsf{pf}_1(b,c)$. However, we can look towards the *intermediate lists* in the tree, *i.e.* $L_1^3, L_2^3, L_1^2, L_2^2$. The merging at the first level is exact: due to the left-right split, there is no filtering of representations, hence the complexity is determined by the size of the output list. However, the construction of L_2^2 contains a filtering step. In the QRACM model, with quantum search, we can produce the elements of L_2^2 faster and reduce the time complexity from: $\ell_2^2 + \ell_3^2 - c^1 + c_1^2$ to: $\ell_2^2 + \ell_3^2 - c^1 + c_1^2 + \frac{1}{2}\mathsf{pf}_1(a,a)$. By optimizing with this time complexity, we obtain a time exponent 0.2356 (the detailed parameters are given in Table 1).

Table 1. Optimization results for the quantum asymmetric HGJ algorithm (in $\log_2$ and relative to n), rounded to four digits. The time complexity is an upper bound.

Variant	Time	a	b	c	ℓ_0^3	ℓ_1^3	ℓ_2^3	ℓ_4^3	ℓ_0^2	ℓ_0^1
Classical	0.3370	0.1249	0.11	0.1401	0.3026	0.2267	0.25	0.2598	0.3369	0.3114
Section 4.2	0.2374	0.0951	0.0951	0.2146	0.4621	0.2367	0.2267	0.2267	0.4746	0.4395
Section 4.3	0.2356	0.0969	0.0952	0.2110	0.4691	0.2356	0.2267	0.2296	0.4695	0.4368

Remark 3 (More improvements). We have tried increasing the tree depth or changing the tree structure, but it does not seem to bring any improvement. In theory, we could allow for more general representations involving "-1" and "2". However, computing the filtering probability, when merging two lists of subknapsacks in $D^n[\alpha, \beta, \gamma]$ *with different distributions* becomes much more technical. We managed to compute it for $D^n[\alpha, \beta]$, but the number of parameters was too high for our numerical optimizer, which failed to converge.

4.4 Quantum Time-Memory Tradeoff

In the original HGJ algorithm, the lists at level 3 contain full distributions $D^{n/2}[0, 1/8]$. By reducing their sizes to a smaller exponential, one can still run the merging steps, but the final list L^0 is of expected size exponentially small in n. Hence, one must redo the tree many times. This general time-memory tradeoff is outlined in [16] and is also reminiscent of Schroeppel and Shamir's algorithm [33], which can actually be seen as repeating $2^{n/4}$ times a merge of lists of size $2^{n/4}$, that yields $2^{-n/4}$ solutions on average.

Tradeoff à la *Schroeppel-Shamir.* The first time-memory tradeoff that we can propose is to dimension the lists sizes in the tree so that, instead of a solution to the knapsack problem, we obtain only on average a solution with a modular constraint of $c_0 n$ bits. Then, we would like to do a quantum search and repeat the tree for only $2^{(1-c_0)n/2}$ iterations. However, this solution is unsatisfactory. If we do a quantum search at this step, we now require quantum memory with quantum random access (QRAQM). Indeed, the tree and all its merges are now in superposition.

Asymmetric Tradeoff. The new tradeoff that we propose is compliant with the QRACM model. It consists in increasing the asymmetry of the tree: we reduce the sizes of the intermediate lists L_1^3, L_1^2, L_1^1 in order to use less memory; this in turn increases the size of L_0^3, L_0^2 and L_0^1 in order to ensure that a solution exists. We find that this tradeoff is close to the curve $TM = 2^{n/2}$, from which it deviates only slightly (the optimal point when $m = 0.2356$ has $TM = 2^{0.4712n}$). This is shown on Figure 4. At $m = 0$, we start at $2^{n/2}$, where L_0^3 contains all vectors of Hamming weight $n/2$.

Fact 1 *For any memory constraint $m \leq 0.2356$ (in $\log_2$ and proportion of n), the optimal time complexity in the quantum asymmetric HGJ algorithm of Section 4.3 is lower than $2^{n/2-m}$.*

Improving the QRACM usage. In trying to reduce the quantum or quantum-accessible hardware used by our algorithm, it makes sense to draw a line between QRACM and classical RAM, *i.e.* between the part of the memory that is actually accessed quantumly, and the memory that is used only classically. We now try to enforce the constraint only on the QRACM, using possibly more RAM. In this context, we cannot produce the list L_1^1 via quantum filtering. The memory constraint on lists L_1^3, L_1^2, L_1^1 still holds; however, we can increase the size of the other intermediate lists.

Fact 2 *For any QRACM constraint $m \leq 0.2356$, the optimal time complexity obtained by using more RAM is always smaller than the best optimization of Section 4.3.*

The difference remains only marginal, as can be seen in Table 2, but it shows a tradeoff between quantum and classical resources.

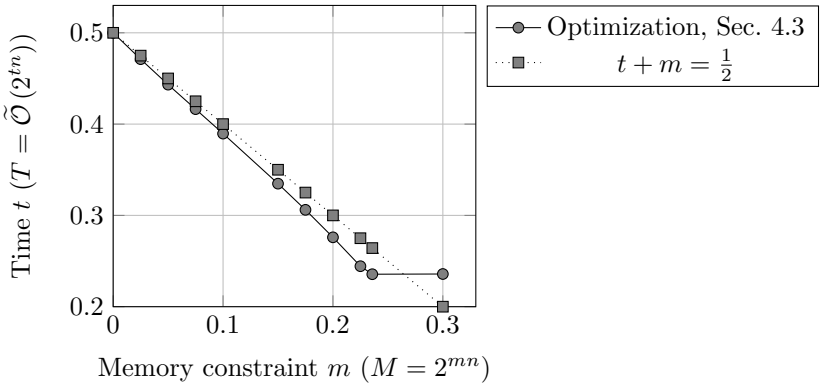

Fig. 4. Quantum time-memory tradeoff of the asymmetric HGJ algorithm, with quantum filtering

Table 2. Time-memory tradeoffs (QRACM) for three variants of our asymmetric HGJ algorithm, obtained by numerical optimization, and rounded upwards. The last variant uses more classical RAM than the QRACM constraint.

QRACM	Section 4.2		Section 4.3		With more RAM	
bound	Time	Memory	Time	Memory	Time	Memory
0.0500	0.4433	0.0501	0.4433	0.0501	0.4412	0.0650
0.1000	0.3896	0.1000	0.3896	0.1000	0.3860	0.1259
0.1500	0.3348	0.1501	0.3348	0.1501	0.3301	0.1894
0.3000	0.2374	0.2373	0.2356	0.2356	0.2373	0.2373

5 New Algorithms Based on Quantum Walks

In this section, we improve the algorithm by Helm and May [14] based on BCJ and the MNRS quantum walk framework. There are two new ideas involved.

5.1 Asymmetric 5th level

In our new algorithm, we can afford one more level than BCJ. We then have a 6-level merging tree, with levels numbered 5 downto 0. Lists at level i all have the same size ℓ_i, *except at level 5*. Recall that the merging tree, and all its lists, is the additional data structure attached to a node in the Johnson graph. In the original algorithm of [14], there are 5 levels, and a node is a collection of 16 lists, each list being a subset of size ℓ_4 among the $g(1/16 + \alpha_3, \alpha_3)/2$ vectors having the right distribution.

In our new algorithm, at level 5, we separate the lists into "left" lists of size ℓ_5^l and "right" lists of size ℓ_5^r. The quantum walk will only be performed on the left lists, while the right ones are full enumerations. Each list at level

4 is obtained by merging a "left" and a "right" list. The left-right-split at level 5 is then asymmetric: vectors in one of the left lists L_5^l are sampled from $D^{\delta n}[\alpha_4, 1/32, \gamma_4] \times \{0^{(1-\delta)n}\}$ and the right lists L_5^r contain *all* the vectors from $\{0^{\delta n}\} \times D^{(1-\delta)n}[\alpha_4, 1/32, \gamma_4]$. This yields a new constraint: $\ell_5^r = f(1/32 + \alpha_4 - 2\gamma_4, \alpha_4, \gamma_4)(1-\delta)$.

While this asymmetry does not bring any advantage classically, it helps in reducing the update time. We enforce the constraint $\ell_5^r = c_4$, so that for each element of L_5^l, there is on average one matching element in L_5^r. So updating the list L_4 at level 4 is done on average time 1. Then we also have $\ell_4 = \ell_5^l$.

With this construction, ℓ_5^r and ℓ_5^l are actually unneeded parameters. We only need the constraints $c_4 (= \ell_5^r) = f(1/32 + \alpha_4 - 2\gamma_4, \alpha_4, \gamma_4)(1-\delta)$ and $\ell_4 (= \ell_5^l) \leq f(1/32 + \alpha_4 - 2\gamma, \alpha_4, \gamma_4)\delta$. The total setup time is now:

$$\mathsf{S} = \max\Bigg(\underbrace{c_4, \ell_4}_{\text{Lv. 5 and 4}}, \underbrace{2\ell_4 - (c_3 - c_4)}_{\text{Level 3}}, \underbrace{2\ell_3 - (c_2 - c_3)}_{\text{Level 2}}, \underbrace{2\ell_2 - (c_1 - c_2)}_{\text{Level 1}}, \underbrace{\ell_1 + \max(\ell_1 - (1 - c_1), 0)}_{\text{Level 0}} \Bigg)$$

and the expected update time for level 5 (inserting a new element in a list L_5^l at the bottom of the tree) and at level 4 (inserting a new element in L_4) is 1. The spectral gap of the graph is $\delta = \Omega\left(\frac{1}{|L_5^l|}\right)$.

Saturation Constraints. In the quantum walk, we have $\ell_0 < 0$, since we expect only some proportion of the nodes to be marked (to contain a solution). This proportion is hence ℓ_0. The saturation constraints are modified as follows:

$$\begin{aligned}
\ell_5^l &\leq \frac{\ell_0}{16} + f(1/32 + \alpha_4 - 2\gamma_4, \alpha_4, \gamma_4)\delta \\
\ell_4 &\leq \frac{\ell_0}{16} + f(1/32 + \alpha_4 - 2\gamma_4, \alpha_4, \gamma_4) - c_4 \\
\ell_3 &\leq \frac{\ell_0}{8} + f(1/16 + \alpha_3 - 2\gamma_3, \alpha_3, \gamma_3) - c_3 \\
\ell_2 &\leq \frac{\ell_0}{4} + f(1/8 + \alpha_2 - 2\gamma_2, \alpha_2, \gamma_2) - c_2 \\
\ell_1 &\leq \frac{\ell_0}{2} + f(1/4 + \alpha_1 - 2\gamma_1, \alpha_1, \gamma_1) - c_1
\end{aligned}$$

Indeed, the *classical* walk will go through a total of $-\ell_0$ trees before finding a solution. Hence, it needs to go through $-\ell_0/16$ different lists at level 5 (and 4), which is why we need to introduce ℓ_0 in the saturation constraint: there must be enough elements, not only in L_5^l, but in the whole search space that will be spanned by the walk.

5.2 Better Setup and Updates using quantum search

Along the lines of Lemmas 5 and 1, we now show how to use a quantum search to speed up the Setup and Update steps in the quantum walk.

Setup. Let $p_i, (1 \leq i \leq 3)$ be the filtering probabilities at level i, *i.e.* the (logarithms of the) probabilities that an element that satisfies the modulo condition resp. at level i also has the desired distribution of 0s, 1s, -1s and 2s, and appears in list L_i. Notice that $p_i \leq 0$. Due to the left-right split, there is no filtering at level 4.

We remark that we can find the elements that pass the filter with a quantum search; thus, we can speed up the computation of lists at levels 3, 2 and 1 in the setup. We use Lemma 1 with these lists. However, it does not apply for level 0, since L_0 has a negative expected size. At this level, we will simply perform a quantum search over L_1. If there is too much constraint, *i.e.* $(c_1 - c_2) > \ell_1$, then for a given element in L_1, there is on average less than one modular candidate. If $(c_1 - c_2) < \ell_1$, there is on average more than one (although less than one with the filter) and we have to do another quantum search on them all. This is why the setup time at level 0, in full generality, becomes $(\ell_1 + \max(\ell_1 - (c_1 - c_2), 0))/2$. The setup time can thus be improved to :

$$\mathsf{S} = \max\Bigg(\underbrace{c_4, \ell_4}_{\text{Lv. 5 and 4}}, \underbrace{2\ell_4 - (c_3 - c_4) + p_3/2}_{\text{Level 3}}, \underbrace{2\ell_3 - (c_2 - c_3) + p_2/2}_{\text{Level 2}}, \underbrace{2\ell_2 - (c_1 - c_2) + p_1/2}_{\text{Level 1}}, \underbrace{(\ell_1 + \max(\ell_1 - (1 - c_1), 0))/2}_{\text{Level 0}} \Bigg) .$$

Update. Our update will also use a quantum search. First of all, recall that the updates of levels 5 and 4 are performed in (expected) time 1. Having added an element in list L_4, we need to update the upper level. There are on average $\ell_4 - (c_3 - c_4)$ candidates satisfying the modular condition. In order to avoid a blowup in the time complexity, we forbid to have more than one element inserted in L_3 on average, which means: $\ell_4 - (c_3 - c_4) + p_3 \leq 0 \iff \ell_3 \leq \ell_4$. We can then find this element, if it exists, with a quantum search among the $\ell_4 - (c_3 - c_4)$ candidates.

Similarly, as at most one element is updated in L_3, we can move on to the upper levels 2, 1 and 0 and use the same argument. We forbid to have more than one element inserted in L_2 on average: $\ell_3 - (c_2 - c_3) + p_2 \leq 0 \iff \ell_2 \leq \ell_3$, and in L_1: $\ell_1 \leq \ell_2$. At level 0, a quantum search may not be needed, hence a time

$\max(\ell_1 - (1 - c_1), 0)/2$. The expected update time becomes:

$$\mathsf{U} = \max\Bigg(0, \underbrace{(\ell_4 - (c_3 - c_4))/2}_{\text{Level 3}}, \underbrace{(\ell_3 - (c_2 - c_3))/2}_{\text{Level 2}}, \underbrace{(\ell_2 - (c_1 - c_2))/2}_{\text{Level 1}}, \underbrace{(\ell_1 - (1 - c_1))/2}_{\text{Level 0}}\Bigg) .$$

5.3 Parameters

Using the following parameters, we found that this algorithm would run in time $\widetilde{\mathcal{O}}\left(2^{0.2156n}\right)$:

$$\ell_0 = -0.1899, \ell_1 = 0.1994, \ell_2 = 0.2033, \ell_3 = 0.2127, \ell_4 (= \ell_5^l) = 0.2127$$
$$c_1 = 0.6228, c_2 = 0.4480, c_3 = 0.2505, c_4 (= \ell_5^r) = 0.0435$$
$$\alpha_1 = 0.0180, \alpha_2 = 0.0157, \alpha_3 = 0.0133, \alpha_4 = 0.0066$$
$$\gamma_1 = 0.0020, \gamma_2 = \gamma_3 = \gamma_4 = 0, \delta = 0.8496$$

6 Mitigating Quantum Walk Heuristics for Subset-Sum

In this section, we provide a modified quantum walk NEW-QW for any quantum walk subset-sum algorithm QW, including [6,14] and ours, that will no longer rely on Heuristic 2. In NEW-QW, the Johnson graph is the same, but the vertex data structure and the update procedure are different (Section 6.1). It allows us to guarantee the update time, at the expense of having some vertices no longer marked. In Section 6.2, we will show that most marked vertices in QW remain marked.

6.1 New Data Structure for Vertices

The algorithms that we consider use multiple levels of merging. However, we will focus only on a single level. Our arguments can be generalized to any constant number of merges (with an increase in the polynomial factors involved).

Storing Lists of Elements. The main requirement of the vertex data structure is to store lists of subknapsacks with modular constraints in QRAQM. We use the augmented radix tree data structure proposed in [6]. In addition to fast superposition access, it allows to insert or to delete an element in a sorted list in time $\text{poly}(n)$.

In the following, we will consider the merging of two lists L_l and L_r of subknapsacks of respective sizes ℓ_l and ℓ_r, with a modular constraint c and a filtering probability pf. The merged list is denoted $L' = L_l \bowtie_c L_r$ and the filtered list is denoted L^f. We assume that pairs $(\mathbf{e_1}, \mathbf{e_2})$ in L must satisfy $(\mathbf{e_1}+\mathbf{e_2})\cdot\mathbf{a} = 0 \mod 2^{cn}$ (the generalization to any value modulo any moduli is straightfoward).

There are two situations that we need to consider:

1. when $\ell_l > \ell_r$, but only L_l needs to be updated, and $\mathsf{pf} = 0$: this is the lowest level of our vertex data structure in our new quantum walk.

2. When $\ell_l = \ell_r$ and either L_l or L_r are updated: this happens at all upper levels in our algorithm. In previous quantum walks, we had $\ell' = 2\ell - c \leq \ell$, *i.e.* $\ell \leq c$; now we will have $2\ell - c \geq \ell$ and $2\ell - c + \mathsf{pf} \leq \ell$. We will focus on the situation 2., because the other one can easily be studied with the same arguments.

Our heuristic time complexity analysis assumes an update time $(\ell - c)/2$. Indeed, the update of an element in L_l or L_r modifies on average $(\ell - c)$ elements in $L_l \bowtie_c L_r$, among which we expect at most one filtered pair $(\mathbf{e_1}, \mathbf{e_2})$ (by the inequality $2\ell - c + \mathsf{pf} \leq \ell$). We find this solution with a quantum search. In the following, we modify the data structure of vertices in order to guarantee the best update time possible, up to additional polynomial factors. We will see however that it does not reach $(\ell - c)/2$.

The product Johnson graph on which we run the quantum walk is unchanged. We will later analyze the number of marked vertices.

Preliminary: Ordering. We need the following property to select subsets of representations in our data structure:

Fact 3 *There exists an efficient mapping ϕ of representations to integers by Lemma 10, hence a canonical ordering of vectors and pairs of vectors.*

First Level: Moduli Bounds. For an integer $C \leq 2^{cn}$, we expect on average $2^{n(\ell-c)}$ subknapsacks in L_r and L_l such that $\mathbf{e} \cdot \mathbf{a} = C \mod 2^{cn}$. A potential blowup in the update cost would happen if L_l (resp. L_r) contains too many elements with an identical modulo. Hence, our data structure will prevent this situation. We will only propagate an update when the number of subknapsacks with a given modulo C does not differ too much from its expectation.

More precisely, for each $T \leq 2^{\ell n}$, in L_r and L_l, we expect on average only one element $\mathbf{e}$ such that $\mathbf{e} \cdot \mathbf{a} = T \mod 2^{\ell n}$. Let us set an arbitrary bound $B = \mathrm{poly}(n)$, to be chosen later. We sort L_l and L_r according to the (precomputed) values of $\mathbf{e} \cdot \mathbf{a} \mod 2^{\ell n}$. We also keep track of how many elements in L_l have the same value modulo $2^{\ell n}$ and we make sure that we can, for a given $T \mod 2^{\ell n}$, access all elements in L_l with $\mathbf{e} \cdot \mathbf{a} = T \mod 2^{\ell n}$.

Now we construct a sublist $L'_l \subseteq L_l$ that contains, for each $T \mod 2^{\ell n}$, at most B subknapsacks with $\mathbf{e} \cdot \mathbf{a} = T \mod 2^{\ell n}$: if there are more vectors in L_l satisfying this condition, we only consider the first B in the canonical order.

$$L'_l = \{\mathbf{e} \in L_l, |\{\mathbf{e}' \in L_l, \mathbf{e} \cdot \mathbf{a} = \mathbf{e}' \cdot \mathbf{a} \mod 2^{\ell n} \wedge \mathbf{e}' \leq \mathbf{e}\}| \leq B\} \ .$$

We do the same for L_r. Obviously, this removes some fraction of the subknapsacks, but only a polynomial fraction at most (see 6.2 for a formal proof). Now, for each integer $C \leq 2^{cn}$, it holds that the number of elements in L'_l and L'_r with modulo $C \mod 2^{cn}$ is upper bounded by $B2^{n(\ell-c)}$.

Updating the First Level. When removing an element $\mathbf{e}$ in L_l (resp. L_r), let $\mathbf{e} \cdot \mathbf{a} = T \mod 2^{\ell n}$, there are two cases for L'_l:

- If $|\{\mathbf{e}' \in L_l, \mathbf{e}' \cdot \mathbf{a} = T \wedge \mathbf{e}' \leq \mathbf{e}\}| > B$, L'_l is unchanged.
- If $|\{\mathbf{e}' \in L_l, \mathbf{e}' \cdot \mathbf{a} = T \wedge \mathbf{e}' \leq \mathbf{e}\}| \leq B$, we remove $\mathbf{e}$ from L'_l. If some elements of L_ℓ are not in L'_l, we add the next element in canonical order.

When adding an element $\mathbf{e}$ in L_l (resp. L_r), there are also two cases for L'_l:

- If $|\{\mathbf{e}' \in L_l, \mathbf{e}' \cdot \mathbf{a} = T \wedge \mathbf{e}' \leq \mathbf{e}\}| > B$, L'_l is unchanged.
- If $|\{\mathbf{e}' \in L_l, \mathbf{e}' \cdot \mathbf{a} = T \wedge \mathbf{e}' \leq \mathbf{e}\}| < B$, we add $\mathbf{e}'$ in L_l. If L'_ℓ would become larger than B, we remove its largest element.

In all cases, there is at most one insertion and one deletion in L'_l or L'_r.

Second Level: Modular Sublists. We cut L'_l and L'_r in chunks of subknapsacks having the same modulo $C \mod 2^{cn}$ on cn bits. Hence, each sublist $L'_{l,C}$ and $L'_{r,C}$ has size at most $B2^{(\ell-c)n}$. The joint list $L'_l \bowtie_c L'_r$ is the union of the joints $L'_{l,C} \times L'_{r,C}$ for each $C \mod 2^{cn}$. We expect an element in $L'_l \bowtie_c L'_r$ to be filtered with probability pf, and we expect these filtered elements to be uniformly distributed among the sublist joints $L'_{l,C} \times L'_{r,C}$. Hence, for each of these (of size $2\ell - 2c$) we expect $2\ell - 2c + \mathsf{pf}$ filtered pairs to occur. We focus on one of these sublists.

Second Level: Case Disjunction. Let us first consider the case where $2\ell - 2c + \mathsf{pf} \leq 0$. A pair of modular sublists $L'_{l,C} \times L'_{r,C}$ is now responsible for (on average) less than one element in the filtered list L^f. When we update an element $\mathbf{e}$ in $L'_{l,C}$, there are potentially more than one pair $\mathbf{e}, \mathbf{e}'$ in $L'_{l,C} \times L'_{r,C}$ to insert in L^f, although there is on average less than one. To make our updates consistent, we keep the B smallest pairs, as before.

When we add an element $\mathbf{e}$ in $L'_{l,C}$, we look in $L'_{r,C}$ for the vectors $\mathbf{e}'$ such that $\mathbf{e} + \mathbf{e}'$ is a valid subknapsack. We retrieve only the $\mathbf{e}'$ that minimizes $\phi(\mathbf{e} + \mathbf{e}')$. This is done in time $(\ell - c)/2$, using quantum search, thanks to the following:

Lemma 6. *Let X be a set of size 2^n and $\phi : X \to [0; 2^n - 1[$ a bijection computable in time $\mathsf{poly}(n)$. Let $g : X \to \{0, 1\}$ be a test function. While $|g^{-1}(1)|$ is unknown, there exists a quantum algorithm that computes the "smallest solution" $x = argmin_{|g^{-1}(1)|}\phi$, in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.*

Proof. The algorithm is similar to a quantum search with an unknown number of solutions [11], in which the search is repeated in subspaces of increasing sizes (we double the size at each step). In our case, we combine quantum search with dichotomy search: we separate the interval $I_n = [0; 2^n - 1[$ in $\left[0; 2^{n-1} - 1\right[\cup \left[2^{n-1}; 2^{n-1} - 1\right[$. If we find a solution in the first interval, then we set $I_{n-1} = \left[0; 2^{n-1} - 1\right[$ and continue; if we don't find a solution, then we set $I_{n-1} = \left[2^{n-1}; 2^n - 1\right[$, and proceed recursively. In the end, we will not only find if there is a solution, but we will also find the smallest one. □

Once we have found the smallest valid pair that involves $\mathbf{e}$, we compare it with the previously stored pairs with the moduli C. If it is to be included, we replace it, and do another dichotomy search for the second smallest pair, else we discard it, and so on. It is easy to see that after an update, the stored pairs remains the smallest among all valid pairs in $L'_{l,C} \times L'_{r,C}$. However, when we remove an element from $L'_{l,C}$, we now must look among all pairs in $L'_{l,C} \times L'_{r,C}$ for the new smallest valid ones. We do this using a quantum search on the product space, hence the total update time at this step is $\ell - c$.

Second Level in General. When $2\ell - 2c + \mathsf{pf} > 0$, we cut $L'_{l,C}$ into sublists $L_{l,C,i}$, and $L'_{r,C}$ into sublists $L_{r,C,j}$, of size $-\mathsf{pf}/2$ each, so that any product $L_{l,C,i} \times L_{r,C,j}$ is supposed to contain on average a single valid pair. There are $\ell - c + \mathsf{pf}/2$ sublists. This cut is done with an arbitrary modular constraint, so the sizes of $L_{l,C,i}$ and $L_{r,C,j}$ are not bigger than $-\mathsf{pf}/2$ by more than a constant factor. As above, we take only the smallest pairs. When we add an element to $L'_{l,C}$, or when we remove an element from it, there are potentially $\ell - c + \mathsf{pf}/2$ valid pairs to update, and there is a total cost of $\ell - c$ for finding the new smallest pairs.

Fact 4 *In log scale, our data structure has update time $\ell - c$ and an update modifies $\max(\ell - c + \mathsf{pf}/2, 0)$ elements in the filtered list at the next level.*

6.2 Fraction of Marked Vertices

Now that we have computed the update time of NEW-QW, it remains to compute its fraction ϵ_{new} of marked vertices. We will show that $\epsilon_{new} = \epsilon\left(1 - \frac{1}{\mathrm{poly}(n)}\right)$ with overwhelming probability on the random subset-sum instance, where ϵ is the previous fraction in QW.

Let $\mathcal{N}$ be a marked vertex in QW. There is a path in the data structure leading to the solution, hence a constant number of subknapsacks $\mathbf{e_1}, \ldots, \mathbf{e_t}$ such that the vertex will remain marked if none of them is "accidentally" discarded by our new data structure. A subknapsack $\mathbf{e_i}$ in a list L at any level of the QW merging tree can be discarded for two reasons:

- we have $|\{\mathbf{e} \in L, \mathbf{e} \cdot \mathbf{a} = \mathbf{e_i} \cdot \mathbf{a} \mod 2^{\ell n}\}| > B$, or:
- at the next level, there is more than B pairs $\mathbf{e_i}, \mathbf{e}$ such that $(\mathbf{e_i} + \mathbf{e}) \cdot \mathbf{a} = 0 \mod 2^{cn}$ and $\mathbf{e_i} + \mathbf{e}$ is filtered, where the probability of being filtered is $\mathsf{pf} \leq 0$ and $(\ell - c) + \mathsf{pf} \leq 0$.

Estimation. Let G be the Johnson graph of the walk. We are interested in upper bounding:

$$\Pr_{v \in G}\begin{pmatrix} v \text{ is marked in QW and} \\ \text{not marked in NEW-QW} \end{pmatrix} \leq \sum_{\mathbf{e_i}, 1 \leq i \leq t} \Pr_{v \in G}\begin{pmatrix} \mathbf{e_i} \in v \text{ in QW} \\ \mathbf{e_i} \notin v \text{ in NEW-QW} \end{pmatrix} .$$

So we focus on some level in the tree, on a list L of average size $2^{\ell n}$, and on a single vector $\mathbf{e_0}$ that must appear in L. Subknapsacks in L are taken from

$\mathcal{B} \subseteq D^n[\alpha, \beta, \gamma]$. We study the event that $\mathbf{e_0}$ is accidentally discarded from L. We remark the following to make our computations easier.

Fact 5 *We can replace the L from our new data structure* NEW-QW *by a list of exact size $2^{\ell n}$, which is a sublist from the list L in* QW*.*

At successive levels, our new data structure discards more and more vectors. Hence, the actual lists are smaller than in QW. However, removing a vector $\mathbf{e}$ from a list, if it does not unmark the vertex, does not increase the probability of unmarking it at the next level, since $\mathbf{e}$ does not belong to the unique solution.

Fact 6 *When a vertex in* NEW-QW *is sampled uniformly at random, given a list L at some merging level, we can assume that L are sampled uniformly at random with replacement from its distribution $\mathcal{B}$ (with a modular constraint).*

This fact translates Heuristic 1 as a global property of the Johnson graph. At the first level, nodes contain lists of exponential size which are sampled without replacement. However, when sampling with replacement, the probability of collisions is exponentially low. Thus, we can replace $\Pr_{v \in G}$ by $\Pr_{v \in G'}$ where G' is a "completed" graph containing all lists sampled uniformly at random with replacement. This adds only a negligible number of vertices and does not impact the probability of being discarded.

Number of Vectors Having the Same Modulus. Given a particular $\mathbf{e}_0 \in \mathcal{B}$ and a vector $\mathbf{a} \in \mathbb{Z}_{2^n}^n$, define

$$Y(\mathcal{B}, \mathbf{e}_0; \mathbf{a}) = \left|\{\mathbf{x} \in \mathcal{B} : \mathbf{a} \cdot \mathbf{e} = \mathbf{a} \cdot \mathbf{e}_0 \pmod{M}\}\right|.$$

For simplicity, we write $Y(\mathbf{a})$ for $Y(\mathcal{B}, \mathbf{e}_0; \mathbf{a})$ in the following. We are interested in $Y(\mathbf{a})$ as a random variable when $\mathbf{a}$ is drawn uniformly from $\mathbb{Z}_{2^n}^n$. We prove the following lemmas in Appendix C:

Lemma 7. *With probability* $1 - \mathrm{negl}(n)$*,*

$$Y(\mathbf{a}) \leq 2\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] \sim 2 \cdot \frac{|\mathcal{B}|}{M} \tag{1}$$

For $\mathbf{e} \in \mathcal{B}$, define

$$X_{\mathbf{e}}(a) = \begin{cases} 1 & \text{if } \mathbf{e} \cdot \mathbf{a} = \mathbf{e}_0 \cdot \mathbf{a} \pmod{M} \\ 0 & \text{otherwise} \end{cases}$$

We also prove the following Lemma.

Lemma 8. *For a* $1 - \mathrm{negl}(n)$ *proportion of* $\mathbf{a} \in \mathbb{Z}_{2^n}^n$*:*

$$\Pr_{\mathbf{e}_1, \cdots, \mathbf{e}_{|L|} \sim Unif(\mathcal{B})} \left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B - 1 \right] > 1 - \frac{1}{\mathrm{poly}(n)} \tag{2}$$

Number of Filtered Pairs. For $\mathbf{e} \in \mathcal{B}$, define

$$Z_{\mathbf{e}}(a) = \begin{cases} 1 & \text{if } \mathbf{e} \cdot \mathbf{a} = \mathbf{e_0} \cdot \mathbf{a} \pmod{C} \text{ and } \mathbf{e} + \mathbf{e_0} \text{ is filtered at the next level} \\ 0 & \text{otherwise} \end{cases}$$

where the condition of being filtered or not depends on $\mathbf{e_0}$, and $\mathbb{E}(Z_{\mathbf{e}}(a)) \leq \frac{1}{|L|}$ for some r (which is implied by our constraints). Then a similar proof gives, via a Chernoff bound:

Lemma 9. *For a* $1 - \text{negl}(n)$ *proportion of* $\mathbf{a} \in \mathbb{Z}_{2^n}^n$*:*

$$\Pr_{\mathbf{e}_1, \cdots, \mathbf{e}_{|L|} \sim Unif(\mathcal{B})} \left[\sum_{i=1}^{|L|} Z_{\mathbf{e}_i}(\mathbf{a}) < B - 1 \right] > 1 - \frac{1}{\text{poly}(n)} \tag{3}$$

By a union bound argument, since there is a constant number of intermediate subknapsacks to keep, the fraction of unmarked vertices is less than $\frac{1}{\text{poly}(n)}$.

6.3 Time Complexities without Heuristic 2

Previous quantum subset-sum algorithms [6,14] have the same time complexities without Heuristic 2, as they fall in parameter ranges where only "moduli bounds" are necessary. However, this is not the case of our new quantum walk. We keep the same set of constraints and optimize with a new update time. Although using the extended $\{-1, 0, 1, 2\}$ representations brings an improvement, neither do the fifth level, nor the left-right split. This simplifies our constraints. Let $\widehat{\max}(\cdot) = \max(\cdot, 0)$. The guaranteed update time becomes:

$$\mathsf{U} = \widehat{\max}\Bigg(\underbrace{\ell_3 - (c_2 - c_3)}_{\text{Level 2}}, \underbrace{\widehat{\max}(\ell_3 - (c_2 - c_3) + \frac{p_2}{2})}_{\substack{\text{Number of elements} \\ \text{to update at level 1}}} + \widehat{\max}(\ell_2 - (c_1 - c_2)),$$

$$\underbrace{\frac{1}{2}\left(\widehat{\max}\left(\ell_3 - (c_2 - c_3) + \frac{p_2}{2}\right) + \widehat{\max}\left(\ell_2 - (c_1 - c_2) + \frac{p_1}{2}\right) + \widehat{\max}(\ell_1 - (1 - c_1))\right)}_{\text{Final quantum search among all updated elements}}\Bigg)$$

We obtain the time exponent 0.2182 (rounded upwards) with the following parameters (rounded):

$$\ell_0 = -0.2021, \ell_1 = 0.1883, \ell_2 = 0.2102, \ell_3 = 0.2182, \ell_4 = 0.2182$$
$$c_3 = 0.2182, c_2 = 0.4283, c_1 = 0.6305, p_0 = -0.2093, p_1 = -0.0298, p_2 = -0.0160$$
$$\alpha_1 = 0.0172, \alpha_2 = 0.0145, \alpha_3 = 0.0107, \gamma_1 = 0.0020$$

7 Conclusion

In this paper, we proposed improved classical and quantum heuristic algorithms for subset-sum, building upon several new ideas. First, we used extended representations ($\{-1, 0, 1, 2\}$) to improve the current best classical and quantum algorithms. In the quantum setting, we showed how to use a quantum search to speed up the process of *filtering* representations, leading to an overall improvement on existing work. We built an "asymmetric HGJ" algorithm that uses a nested quantum search, leading to the first quantum speedup on subset-sum in the model of *classical memory with quantum random access*. By combining all our ideas, we obtained the best quantum walk algorithm for subset-sum in the MNRS framework. Although its complexity still relies on Heuristic 2, we showed how to partially overcome it and obtained the first quantum walk results that require only the classical subset-sum heuristic, and the best to date for this problem. A summary of our contributions is given in Table 3.

Table 3. Time complexity exponents of previous and new algorithms for subset-sums, classical and quantum, rounded upwards. We note that the removal of Heuristic 2 in [6,14] is implied by our new analysis in Section 6.3. QW: Quantum Walk. QS: Quantum Search. QF: Quantum filtering.

	Time exponent	Representations	Memory model	Techniques	Requires Heuristic 2	Reference
Classical	0.3370	$\{0, 1\}$	RAM			[16]
	0.2909	$\{-1, 0, 1\}$	RAM			[3]
	0.2830	$\{-1, 0, 1, 2\}$	RAM			Sec. 2.4
Quantum	0.241	$\{0, 1\}$	QRAQM	QW	**No**	[6], Sec. 6.3
	0.226	$\{-1, 0, 1\}$	QRAQM	QW	**No**	[14], Sec. 6.3
	0.2356	$\{0, 1\}$	**QRACM**	QS + QF	No	Sec 4.3
	0.2156	$\{-1, 0, 1, 2\}$	QRAQM	QW + QF	Yes	Sec. 5.3
	0.2182	$\{-1, 0, 1, 2\}$	QRAQM	QW + QF	No	Sec. 6.3

Open Questions. We leave as open the possibility to use representations with "-1"s (or even "2"s) in a quantum asymmetric merging tree, as in Section 4.3. Another question is how to bridge the gap between heuristic and non-heuristic quantum walk complexities. In our work, the use of an improved vertex data structure seems to encounter a limitation, and we may need a more generic result about the quantum walk framework, similar to [2]. Finally, we managed to improve the classical algorithm by adding "2"s, it would be of interest to study if adding "-2"s (or, more generally, using a larger set of integers) can lead to better algorithms.

Acknowledgments.

The authors want to thank André Chailloux, Stacey Jeffery, Antoine Joux, Frédéric Magniez, Amaury Pouly, Nicolas Sendrier for helpful discussions and comments. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 - acronym QUASYModo).

References

1. Ambainis, A.: Quantum Walk Algorithm for Element Distinctness. SIAM J. Comput. 37(1), 210–239 (2007)
2. Ambainis, A.: Quantum search with variable times. Theory Comput. Syst. 47(3), 786–807 (2010), https://doi.org/10.1007/s00224-009-9219-1
3. Becker, A., Coron, J., Joux, A.: Improved generic algorithms for hard knapsacks. In: EUROCRYPT. LNCS, vol. 6632, pp. 364–385. Springer (2011)
4. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: EUROCRYPT. LNCS, vol. 7237, pp. 520–536. Springer (2012)
5. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.V.: Strengths and weaknesses of quantum computing. SIAM J. Comput. 26(5), 1510–1523 (1997), https://doi.org/10.1137/S0097539796300933
6. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: PQCrypto. LNCS, vol. 7932, pp. 16–33. Springer (2013)
7. Bonnetain, X.: Improved low-qubit hidden shift algorithms. CoRR (2019)
8. Bonnetain, X., Naya-Plasencia, M.: Hidden shift quantum cryptanalysis and implications. In: ASIACRYPT (1). LNCS, vol. 11272, pp. 560–592. Springer (2018)
9. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. 2019(2), 55–93 (2019)
10. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: EUROCRYPT 2020. LNCS, Springer (May 2020)
11. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics 305, 53–74 (2002)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
13. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996. pp. 212–219. ACM (1996), http://doi.acm.org/10.1145/237814.237866
14. Helm, A., May, A.: Subset sum quantumly in 1.17^{n}. In: TQC. LIPIcs, vol. 111, pp. 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)
15. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21(2), 277–292 (1974)
16. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: EUROCRYPT. LNCS, vol. 6110, pp. 235–256. Springer (2010)
17. Kachigar, G., Tillich, J.: Quantum information set decoding algorithms. In: PQCrypto. LNCS, vol. 10346, pp. 69–89. Springer (2017)

18. Kirshanova, E., Mårtensson, E., Postlethwaite, E.W., Moulik, S.R.: Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In: ASIACRYPT. LNCS, vol. 11921, pp. 521–551. Springer (2019), https://doi.org/10.1007/978-3-030-34578-5_19
19. 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 fuer Informatik (2013)
20. Laarhoven, T.: Search problems in cryptography. Ph.D. thesis, PhD thesis, Eindhoven University of Technology (2015)
21. Laarhoven, T., Mosca, M., van de Pol, J.: Finding shortest lattice vectors faster using quantum search. Des. Codes Cryptogr. 77(2-3), 375–400 (2015), https://doi.org/10.1007/s10623-015-0067-5
22. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. In: FOCS. pp. 1–10. IEEE Computer Society (1983)
23. Lyubashevsky, V.: The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In: APPROX-RANDOM. LNCS, vol. 3624, pp. 378–389. Springer (2005)
24. Lyubashevsky, V., Palacio, A., Segev, G.: Public-key cryptographic primitives provably as secure as subset sum. In: TCC. LNCS, vol. 5978, pp. 382–400. Springer (2010)
25. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. SIAM Journal on Computing 40(1), 142–164 (2011)
26. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\widetilde{\mathcal{O}}\left(2^{0.054n}\right)$. In: ASIACRYPT. LNCS, vol. 7073, pp. 107–124. Springer (2011)
27. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: EUROCRYPT (1). LNCS, vol. 9056, pp. 203–228. Springer (2015)
28. Naya-Plasencia, M., Schrottenloher, A.: Optimal merging in quantum k-xor and k-sum algorithms. In: EUROCRYPT 2020. LNCS, Springer (May 2020)
29. Newman, D.J., Shepp, L.: The double dixie cup problem. The American Mathematical Monthly 67(1), 58–61 (1960)
30. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Lam, K.Y., Shparlinski, I., Wang, H., Xing, C. (eds.) Cryptography and Computational Number Theory. pp. 331–342. Birkhäuser Basel, Basel (2001)
31. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
32. Schalkwijk, J.: An algorithm for source coding. IEEE Transactions on Information Theory 18(3), 395–399 (May 1972)
33. Schroeppel, R., Shamir, A.: 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), https://doi.org/10.1137/0210033

Appendices

A Bijection between Representations and Integers

It is well known that there exists a bijection between $\left[0, \binom{n}{m}\right[$ and n-bit vectors of Hamming weight m, and this bijection can be computed in polynomial time in n [32]. In our case, $m = \beta n$ and such vectors are subknapsacks from $D^n[0, \beta]$. If $i_1, \ldots i_m$ are the bit-positions of the m "1" in this vector, we map it to the m-tuple of integers: $(i_1, \ldots i_m)$, and define the bijection as:

$$\phi \ : (i_1, \ldots i_m) \mapsto \binom{i_m - 1}{m} + \ldots + \binom{i_1 - 1}{1}$$

where $\binom{i}{j}$ has supposedly been precomputed for all $i \leq n, j \leq m$. In order to compute the inverse ϕ^{-1}, we find for each $j \leq t$ the unique integer i_j such that $\binom{i_j - 1}{m} \leq x \leq \binom{i_j}{m}$. We can generalize this to an arbitrary number of nonzero symbols (3 in our paper: "1", "-1" and "2"), that we denote $1, 2, \ldots t$. Let $k_1, \ldots k_t$ be the counts of each symbol in the vector $\mathbf{v}$. We map it to a tuple of tuples: $(i_1^1, \ldots i_{k_1}^1), \ldots, (i_1^t, \ldots i_{k_t}^t)$ where the first vector represents the positions of the "1" among the n bit positions, the second vector represents the positions of the "2" *after having removed the "1"*, and so on. Consequently, we have $0 \leq i_j^1 \leq n - 1$, $0 \leq i_j^2 \leq n - 1 - k_1$, *etc.*

Next, we map each of these t tuples individually to an integer, as was done above: $\phi^j(i_1^j, \ldots i_{k_j}^j) = x^j$ where $0 \leq x^j \leq \binom{n - k_1 \ldots - k_{j-1}}{k_j}$. Finally, we compute:

$$\begin{aligned}
\phi(\mathbf{v}) &= x^1 + \binom{n}{k_1} x^2 + \binom{n}{k_1, k_2} x^3 + \ldots + \binom{n}{k_1, \ldots k_t} x^t \\
&= x^1 + \binom{n}{k_1} \left(x^2 + \binom{n - k_1}{k_2} \left(x^3 + \ldots + \binom{n - k_1 \ldots - k_{t-2}}{k_{t-1}} x^t \right) \ldots \right) \ .
\end{aligned}$$

By the bounds on the x^j, we remark that $0 \leq \phi(\mathbf{v}) \leq \binom{n}{k_1, \ldots k_t} - 1$. Furthermore, we observe that one can easily retrieve the x^j by successive euclidean divisions, and use the bijections ϕ^j to finish the computation.

Lemma 10. *Let $n, k_1, \ldots k_t$ be $t + 1$ integers such that $k_1 + \ldots + k_t \leq n$. There exists a quantum unitary, realized with poly(n) gates, that on input a number j in $\left[0, \binom{n}{k_1, \ldots k_t}\right[$, writes on its output register the j-th vector $\phi^{-1}(j) \in \{0, \ldots, t\}^n$ having, for each $1 \leq i \leq t$, exactly k_i occurrences of the symbol "i". There exists another unitary which writes, on input $\mathbf{v}$, the integer value $\phi(\mathbf{v})$.*

Using this unitary in combination with a Quantum Fourier Transform, we can, for example, easily produce superpositions of subsets of $D^n[\alpha, \beta, \gamma]$, by taking arbitrary integer intervals.

B Filtering Probabilities

We give below the filtering probabilities for representations that use "-1" and "2". The principle is similar to Lemma 1, but the details are more technical. Notice that both of these lemmas assume symmetric input distributions, and thus are less general than Lemma 1.

Lemma 11 (Filtering BCJ-style representations). *Let* $\mathbf{e_1}, \mathbf{e_2} \in D^n[\alpha, \beta]$ *and* $\gamma \leq 2\alpha$*. Then the logarithm of the probability that* $\mathbf{e_1} + \mathbf{e_2} \in D^n[\gamma, 2\beta]$ *is:*

$$\begin{aligned}\mathsf{pf}_2(\alpha, \beta, \gamma) = \mathrm{bin}(\beta + \alpha, \alpha - \gamma/2) + \mathrm{bin}(\alpha, \alpha - \gamma/2) \\ + \mathrm{trin}(1 - \beta - 2\alpha, \gamma/2, \beta + \gamma/2) - \mathrm{trin}(1, \beta + \alpha, \alpha)\end{aligned}$$

Proof. In order to estimate the success probability, we need to estimate the number of well-formed representations, and how they can be decomposed. Given a fixed vector $\mathbf{e_1} \in D$, we count the number of compatible $\mathbf{e_2}$ such that xn positions with a -1 from $\mathbf{e_2}$ are cancelled by a 1 from $\mathbf{e_1}$. As $\mathbf{e_1} + \mathbf{e_2} \in D^n[\gamma, 2\beta]$, there are $\binom{(\beta+\alpha)n}{xn}\binom{\alpha n}{(2\alpha-\gamma-x)n)}\binom{(1-\beta-2\alpha)n}{(\alpha-x)n,(\beta+\gamma-\alpha+x)n}$ such vectors.

Taking the logarithm and the standard approximations, its derivative is $\log\left(\frac{\beta+\alpha-x}{x}\frac{2\alpha-\gamma-x}{\gamma-\alpha+x}\frac{\alpha-x}{\beta+\gamma-\alpha+x}\right)$. This term is strictly decreasing for $O < x < \gamma-\alpha$, and equals 0 for $x = \alpha - \gamma/2$. Hence, this is the maximum, which correspond to the balanced case. It is equal, up to a polynomial loss, to the total number of compatible vectors. Hence, the log of the number of compatible vectors is

$\mathrm{bin}(\beta + \alpha, \alpha - \gamma/2) + \mathrm{bin}(\alpha, \alpha - \gamma/2) + \mathrm{trin}(1 - \beta - 2\alpha, \gamma/2, \beta + \gamma/2)$. As there are $\mathrm{trin}(1, \beta + \alpha, \alpha)$ vectors in $D^n[\alpha, \beta]$, the lemma holds. □

Lemma 12 (Filtering representations using "2"s). *Let* $\mathbf{e_1}, \mathbf{e_2} \in D^n[\alpha_1, \beta, \gamma_1]$ *and* $\alpha_0, \gamma_0 \geq 0$*. Let us define :* $\begin{cases} x_{min} = \max(0, \alpha_1 + \beta_1 - \frac{1-\alpha_0+\gamma_0}{2}, \gamma_1 - \gamma_0/2) \\ x_{max} = \min(\alpha_1 - \alpha_0/2, \alpha_0/2 + \beta_1 - \gamma_0, \gamma_1) \end{cases}$. *If* $x_{min} \leqslant x_{max}$*, then the logarithm of the probability that* $\mathbf{e_1}+\mathbf{e_2} \in D^n[\alpha_0, 2\beta, \gamma_0]$ *is at least:*

$$\begin{aligned}\mathsf{pf}_3(\alpha_0, \beta, \gamma_0, \alpha_1, \gamma_1) = \max_{x \in [x_{min}, x_{max}]} \mathrm{bin}(\alpha_0, \alpha_0/2) + \mathrm{trin}(\gamma_0, \gamma_1 - x, \gamma_1 - x) \\ + \mathrm{trin}(1 - 2\alpha_0 - 2\beta + \gamma_0, \alpha_1 - \alpha_0/2 - x, \alpha_1 - \alpha_0/2 - x) \\ + \mathrm{quadrin}(\alpha_0 + 2\beta - 2\gamma_0, x, x, \alpha_0/2 + \beta - \gamma_0 - x) \\ - \mathrm{quadrin}(1, \alpha_1, \alpha_1 + \beta - 2\gamma_1, \gamma_1)\end{aligned}$$

Proof. To avoid the explosion of the number of variables, we restrain ourselves to the *symmetric* cases (for which there are as much 1s given by $0+1$ as 1s given by $1+0$, etc.) Given some $\mathbf{e_1} \in D^n[\alpha_1, \beta, \gamma_1]$, we compute the number of compatible $\mathbf{e_2} \in D^n[\alpha_1, \beta, \gamma_1]$. These vectors can be sorted according to the number xn of positions where a -1 from $\mathbf{e_1}$ cancels out a 2 from $\mathbf{e_2}$. The number of such vectors $\mathbf{e_2}$ is $\binom{\alpha_0 n}{\alpha_0 n/2}\binom{\gamma_0 n}{(\gamma_1-x)n,(\gamma_1-x)n}\binom{(1-2\alpha_0-2\beta+\gamma_0)n}{(\alpha_1-\alpha_0/2-x)n,(\alpha_1-\alpha_0/2-x)n}\binom{(\alpha_0+2\beta-2\gamma_0)n}{xn,xn,(\alpha_0/2+\beta-\gamma_0-x)n}$. This quantity is only defined for $x_{min} \leqslant x \leqslant x_{max}$, if x is outside of these

bounds, there is no compatible $\mathbf{e_2}$. As xn must be an integer, there are only a linear number of possible choices for x. Therefore the number of all possible $\mathbf{e_2}$ is given, up to a polynomial factor, by the number of $\mathbf{e_2}$s for the best x. In order to obtain a probability, we divide this number by $\binom{n}{\alpha_1 n,(\alpha_1+\beta-2\gamma_1)n,\gamma_1 n)}$, which is the size of $D^n[\alpha_1,\beta,\gamma_1]$. We observe here that the logarithm of the probability that $\mathbf{e_2}$ is compatible with $\mathbf{e_1}$ is exactly $\mathsf{pf}_3(\alpha_0,\beta,\gamma_0,\alpha_1,\gamma_1)$. As we discarded all asymmetric cases, the real value of the logarithm of the probability that $\mathbf{e_1}$ and $\mathbf{e_2}$ are compatible is at least $\mathsf{pf}_3(\alpha_0,\beta,\gamma_0,\alpha_1,\gamma_1)$. □

C Computing the Fraction of Marked Vertices

This section contains proofs for Lemmas 7 and 8 that were omitted from the main body of the paper. Recall that we defined:

$$Y(\mathbf{a}) = |\{\mathbf{x} \in \mathcal{B} : \mathbf{a}\cdot\mathbf{e} = \mathbf{a}\cdot\mathbf{e}_0 \pmod M\}|$$

and

$$X_{\mathbf{e}}(\mathbf{a}) = \begin{cases} 1 & \text{if } \mathbf{e}\cdot\mathbf{a} = \mathbf{e}_0\cdot\mathbf{a} \pmod M \\ 0 & \text{otherwise} \end{cases}$$

Lemma 7. *With probability* $1-\mathrm{negl}(n)$*,*

$$Y(\mathbf{a}) \le 2\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] \sim 2\cdot\frac{|\mathcal{B}|}{M} \tag{4}$$

Proof. Following [30], for any $M\in\mathbb{N}$ and $z\in\mathbb{C}$, define $\mathbf{e}(z)=\exp(2\pi iz/M)$. It satisfies the identity

$$\forall k\in\mathbb{N},\ \sum_{\lambda=0}^{kM-1}\mathbf{e}(\lambda u) = \begin{cases} 0 & \text{if } u\neq 0 \pmod M \\ kM & \text{if } u = 0 \pmod M \end{cases} \tag{5}$$

for any $u\in\mathbb{Z}$. We have:

$$Y(\mathbf{a}) = \sum_{\mathbf{x}\in\mathcal{B}}\frac{1}{M}\sum_{\lambda=0}^{M-1}\mathbf{e}(\lambda\mathbf{a}\cdot(\mathbf{x}-\mathbf{x}_0))$$

and:

$$\begin{aligned}
\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] &= \frac{1}{2^{n^2}} \sum_{\mathbf{a}\in\mathbb{Z}_{2^n}^n} \sum_{\mathbf{x}\in\mathcal{B}} \frac{1}{M} \sum_{\lambda=0}^{M-1} \mathbf{e}(\lambda\mathbf{a}\cdot(\mathbf{x}-\mathbf{x}_0)) \\
&= \frac{1}{2^{n^2}} \sum_{\mathbf{a}\in\mathbb{Z}_{2^n}^n} \sum_{\mathbf{x}\in\mathcal{B}} \frac{1}{M} + \frac{1}{2^{n^2}} \sum_{\mathbf{a}\in\mathbb{Z}_{2^n}^n} \sum_{\mathbf{x}\in\mathcal{B}} \frac{1}{M} \sum_{\lambda=1}^{M-1} \mathbf{e}(\lambda\mathbf{a}\cdot(\mathbf{x}-\mathbf{x}_0)) \\
&= \frac{|\mathcal{B}|}{M} + \frac{M-1}{M} + \frac{1}{2^{n^2}} \sum_{\mathbf{a}\in\mathbb{Z}_{2^n}^n} \sum_{\mathbf{x}\in\mathcal{B}\setminus\{\mathbf{x}_0\}} \frac{1}{M} \sum_{\lambda=1}^{M-1} \mathbf{e}(\lambda\mathbf{a}\cdot(\mathbf{x}-\mathbf{x}_0)) \\
&= \frac{|\mathcal{B}|+M-1}{M} + \frac{1}{2^{n^2}} \sum_{\mathbf{x}\in\mathcal{B}\setminus\{\mathbf{x}_0\}} \sum_{\lambda=1}^{M-1} \frac{1}{M} \sum_{\mathbf{a}\in\mathbb{Z}_{2^n}^n} \mathbf{e}(\lambda\mathbf{a}\cdot(\mathbf{x}-\mathbf{x}_0)) \\
&= \frac{|\mathcal{B}|+M-1}{M}
\end{aligned}$$

The last step is because for any $\mathbf{x} \neq \mathbf{x}_0$, there exists $i \in [1, n]$ such that $\mathbf{x}^i \neq \mathbf{x}_0^i$, where $\mathbf{x}^i$ is the ith component of $\mathbf{x}$. We then apply equation (5) on this component.

Similarly,

$$\begin{aligned}
\mathbb{V}_{\mathbf{a}}(Y(\mathbf{a})) &= \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})^2] - \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]^2 \\
&= \frac{M^2 + (|\mathcal{B}|-1)(|\mathcal{B}|-2+3M) - (|\mathcal{B}|+M-1)^2}{M^2} \\
&= \frac{(M-1)(|\mathcal{B}|-1)}{M^2}.
\end{aligned}$$

Thus, $\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})] \approx \mathbb{V}_{\mathbf{a}}(Y(\mathbf{a}))$ when we look at their order of magnitude.

According to Tchebychev's inequality,

$$\Pr_{\mathbf{a}}\left[|Y(\mathbf{a}) - \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]| > \mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]\right] \leq \frac{\mathbb{V}_{\mathbf{a}}(Y(\mathbf{a}))}{\mathbb{E}_{\mathbf{a}}[Y(\mathbf{a})]^2} = \mathrm{negl}(n)$$

which completes the proof. □

Lemma 8. *For a* $1 - \mathrm{negl}(n)$ *proportion of* $\mathbf{a} \in \mathbb{Z}_{2^n}^n$*:*

$$\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B - 1\right] > 1 - \frac{1}{\mathrm{poly}(n)} \tag{6}$$

Proof.

$$\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right]$$
$$= \sum_{y=1}^{|\mathcal{B}|} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a})=y\right] \Pr\left[Y(\mathbf{a})=y\right]$$

Under the condition of $Y(\mathbf{a})=y$, for all $i\in[1,|L|]$, $X_{\mathbf{e}_i}(\mathbf{a})$ can be seen as a random variable following $\mathrm{Ber}(\frac{y}{|\mathcal{B}|})$, since $\mathbf{e}_i$s' are randomly chosen from $\mathcal{B}$. Here $\mathrm{Ber}(p)$ is a Bernoulli distribution of parameter p.

Using equation (1), for a $1-\mathrm{negl}(n)$ portion of $\mathbf{a}\in\mathbb{Z}_{2^n}^n$, we have

$$\Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right]$$
$$> \sum_{y=1}^{2\cdot\frac{|\mathcal{B}|}{M}} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a})=y\right] \Pr\left[Y(\mathbf{a})=y\right]$$
$$= (1-\mathrm{negl}(n)) \sum_{y=1}^{2\cdot\frac{|\mathcal{B}|}{M}} \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a})=y\right]$$
$$> (1-\mathrm{negl}(n)) \Pr_{\mathbf{e}_1,\cdots,\mathbf{e}_{|L|}\sim Unif(\mathcal{B})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1 | Y(\mathbf{a})=2\cdot\frac{|\mathcal{B}|}{M}\right]$$
$$= (1-\mathrm{negl}(n)) \Pr_{X_{\mathbf{e}_i}\sim \mathrm{Ber}(2\cdot\frac{|\mathcal{B}|}{M}\cdot\frac{1}{|\mathcal{B}|})=\mathrm{Ber}(\frac{2}{M})}\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) < B-1\right]$$

Chernoff's inequality gives that for any $\delta\geq 1$:

$$\Pr\left[\sum_{i=1}^{|L|} X_{\mathbf{e}_i}(\mathbf{a}) \geq (1+\delta)\frac{2|L|}{M}\right] \leq e^{-\frac{\delta}{3}\frac{2|L|}{M}} \quad .$$

Hence, when $M=|L|$, by taking B linear in n, we obtain that the probability of being unmarked due to this $\mathbf{e_0}$ is less than $\frac{1}{\mathrm{poly}(n)}$. □