Quantum algorithms for the subset-sum problem

Daniel J. Bernstein[1,2], Stacey Jeffery[3], Tanja Lange[2], and Alexander Meurer[4]

[1] Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org

[3] Institute for Quantum Computing, University of Waterloo, Canada
sjeffery@uwaterloo.ca

[4] Ruhr-University Bochum, Horst Görtz Institute for IT-Security
alexander.meurer@rub.de

Abstract. This paper introduces a subset-sum algorithm with heuristic asymptotic cost exponent below 0.25. The new algorithm combines the 2010 Howgrave-Graham–Joux subset-sum algorithm with a new streamlined data structure for quantum walks on Johnson graphs.

Keywords: subset sum, quantum search, quantum walks, radix trees, decoding, SVP, CVP

1 Introduction

The subset-sum problem is the problem of deciding, given integers $x_1, x_2, \ldots, x_n$ and s, whether there exists a subset I of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in I} x_i = s$; i.e., whether there exists a subsequence of $x_1, x_2, \ldots, x_n$ with sum s. Being able to solve this decision problem implies being able to find such a subset if one exists: for $n > 1$ one recursively tries $x_1, x_2, \ldots, x_{n-1}$ with sum s or, if that fails, sum $s - x_n$.

The reader should imagine, as a typical "hard" case, that $x_1, x_2, \ldots, x_n$ are independent uniform random integers in $\{0, 1, \ldots, 2^n\}$, and that s is chosen as a uniform random integer between $(n/2 - \sqrt{n})2^{n-1}$ and $(n/2 + \sqrt{n})2^{n-1}$. The number of subsets $I \subseteq \{1, 2, \ldots, n\}$ with $\sum_{i \in I} x_i = s$ then has a noticeable chance of being larger than 0 but is quite unlikely to be much larger, say larger than n.

This work was supported by the National Science Foundation under grant 1018836, by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005, by NSERC Strategic Project Grant FREQUENCY, by the US ARO, and by the European Commission under Contract ICT-2007-216676 ECRYPT II. Permanent ID of this document: 797dcbe9c3389410ae4d5ae6dba33ab7. Date: 2013.04.07.

The subset-sum problem is, historically, one of the first problems to be proven NP-complete. A polynomial-time non-quantum algorithm for the subset-sum problem would violate the standard P $\neq$ NP conjecture; a polynomial-time quantum algorithm for the subset-sum problem would violate the standard NP $\not\subseteq$ BQP conjecture. There is, however, a very large gap between polynomial time and the time needed for a naive search through all 2^n subsets of $\{1, 2, \ldots, n\}$. The standard NP $\not\subseteq$ BQP conjecture does not rule out faster exponential-time algorithms, or even subexponential-time algorithms, or even algorithms that take polynomial time for *most* inputs. This paper studies faster exponential-time algorithms.

Variations. Often one is interested only in sums of fixed weight, or of limited weight. We are now given integers $x_1, x_2, \ldots, x_n$, s, and w; the problem is to decide whether there is a subset I of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in I} x_i = s$ and $\#I = w$. In the special case $s = 0$ with $w \neq 0$, such a subset I immediately produces a short nonzero vector in the lattice L of vectors $v \in \mathbf{Z}^n$ satisfying $\sum_i x_i v_i = 0$: specifically, the characteristic function of I is a vector of length $\sqrt{w}$ in L. In many applications this is the shortest nonzero vector in L; in some applications this vector can be found by standard SVP algorithms.

For $s \neq 0$ one can instead compute a vector $r \in \mathbf{R}^n$ satisfying $\sum_i x_i r_i = s$, and then observe that subtracting the characteristic function of I from r produces an element of L. In many applications this is the vector in L closest to r; in some applications this vector can be found by standard CVP algorithms.

A variant of the same problem is the central algorithmic problem in coding theory. The input now consists of vectors $x_1, x_2, \ldots, x_n$, a vector s, and an integer w; these vectors all have the same length and have entries in the field $\mathbf{F}_2$ of integers modulo 2. The problem, as above, is to decide whether there is a subset I of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in I} x_i = s$ and $\#I = w$.

We do not mean to suggest that these problems are identical. However, the algorithmic techniques used to attack subset-sum problems are among the central algorithmic techniques used to attack lattice problems and decoding problems. For example, the best attack known against code-based cryptography, at least asymptotically, is a very recent decoding algorithm by Becker, Joux, May, and Meurer [4], improving upon a decoding algorithm by May, Meurer, and Thomae [24]; the algorithm of [4] is an adaptation of a subset-sum algorithm by Becker, Coron, and Joux [3], improving analogously upon a subset-sum algorithm by Howgrave-Graham and Joux [17].

There is also a line of work on building cryptographic systems whose security is more directly tied to the subset-sum problem. For example, Lyubashevsky, Palacio, and Segev in [22] propose a public-key encryption system and prove that being able to break it implies being able to solve modular subset-sum problems of the following type: find a random subset $I \subseteq \{1, 2, \ldots, n\}$ given random $x_1, x_2, \ldots, x_n$ modulo M and given $\sum_{i \in I} x_i$ modulo M, where M is roughly $(10n \log n)^n$. They claim in [22, Section 1] that "there are currently no known quantum algorithms that perform better than classical ones on the subset sum problem".

Exponent	Quantum	Split	Algorithm
1	No	1	Brute force
0.5	Yes	1	Quantum search; §2
0.5	No	1/2	Left-right split; §2
0.5	No	1/4	Left-right split with a modulus; §4
0.375	Yes	1/4	Quantum search with a modulus; §4
0.337...	No	1/16	Moduli + representations; §5
0.333...	Yes	1/3	Quantum left-right split; §2
0.333...	Yes	1/2	Quantum walk; §3
0.3	Yes	1/4	Quantum walk with a modulus; §4
0.291...	No	1/16	Moduli + representations + overlap; [3]
0.241...	Yes	1/16	**New**; quantum walk + moduli + representations; §5

Table 1.1. Heuristic asymptotic performance of various subset-sum algorithms. An algorithm using $2^{(e+o(1))n}$ operations is listed as "exponent" e.

Contents of this paper. We introduce the first subset-sum algorithm that beats $2^{n/4}$. Specifically, we introduce a quantum algorithm that, under reasonable assumptions, uses at most $2^{(0.241\ldots+o(1))n}$ qubit operations to solve a subset-sum problem. This algorithm combines quantum walks with the central "representations" idea of [17]. Table 1.1 compares this exponent $0.241\ldots$ to the exponents of other algorithms.

One can reasonably speculate that analogous quantum speedups can also be applied to the algorithms of [24] and [4]. However, establishing this will require considerable extra work, similar to the extra work of [24] and [4] compared to [17] and [3] respectively.

Cost metric and other conventions. This paper follows the tradition of measuring algorithm cost as the number of bit operations or, more generally, qubit operations. In particular, random access to an array of size $2^{O(n)}$ is assumed to cost only $n^{O(1)}$, even if the array index is a quantum superposition.

We systematically suppress cost factors polynomial in n; our concern is with asymptotic exponents such as the $0.241\ldots$ in $2^{(0.241\ldots+o(1))n}$. We also assume that the inputs $x_1, x_2, \ldots, x_n, s$ have $n^{O(1)}$ bits. These conventions mean, for example, that reading the entire input $x_1, x_2, \ldots, x_n, s$ costs only 1.

Almost all of the algorithms discussed here split size-n sets into parts, either 2 or 3 or 4 or 16 parts, as indicated by the "Split" column in Table 1. Any reasonably balanced split is adequate, but to simplify the algorithm statements we assume that n is a multiple of 2 or 3 or 4 or 16 respectively.

The algorithms in this paper are designed to work well for random inputs, particularly in the "hard" case that $x_1, x_2, \ldots, x_n, s$ each have about n bits. Our analyses — like the analyses of state-of-the-art algorithms for integer factorization, discrete logarithms, and many other problems of cryptographic interest — are heuristic. We do not claim that the algorithms work for *all* inputs, and we do not claim that what we call the "hard" case is the worst case. Even for random inputs we do not claim that our analyses are proven, but we do speculate that they are provable by an adaptation of the proof ideas stated in [17, eprint version].

Acknowledgments. This work was initiated during the Post-Quantum Cryptography and Quantum Algorithms workshop at the Lorentz Center in November 2012. We acknowledge helpful discussions with Andris Ambainis, Frédéric Magniez, Nicolas Sendrier, and Jean-Pierre Tillich.

2 Searches

Define Σ as the function that maps $I \subseteq \{1, 2, \ldots, n\}$ to $\sum_{i \in I} x_i$. Recall that we assume that $x_1, x_2, \ldots, x_n, s$ have $n^{O(1)}$ bits, and that we suppress polynomial cost factors; evaluating Σ therefore has cost 1.

The subset-sum problem is the problem of deciding whether there exists I with $\Sigma(I) = s$, i.e., whether the function $\Sigma - s$ has a root. A classical search for a root of $\Sigma - s$ uses 2^n evaluations of $\Sigma - s$, for a total cost of 2^n. Of course, the search can finish much sooner if it finds a root (one expects only 2^{n-1} evaluations on average if there is 1 root, and fewer if there are more roots); but as discussed in Section 1 we focus on "hard" cases where there are not many roots, and then the cost is 2^n (again, suppressing polynomial factors) with overwhelming probability.

This section reviews two standard ways to speed up this brute-force search. The first way is Grover's quantum search algorithm. The second way is decomposing $\Sigma(I)$ as $\Sigma(I_1) + \Sigma(I_2)$, where $I_1 = I \cap \{1, 2, \ldots, n/2\}$ and $I_2 = I \cap \{n/2 + 1, \ldots, n\}$; this split was introduced by Horowitz and Sahni in [16].

Review: The performance of quantum search. Consider any computable function f with a b-bit input and a unique root. Grover's algorithm [15] finds the root (with negligible failure chance) using approximately $2^{b/2}$ quantum evaluations of f and a small amount of overhead.

More generally, consider any computable function f with a b-bit input and $r > 0$ roots. Boyer, Brassard, Høyer, and Tapp in [6] introduced a generalization of Grover's algorithm (almost exactly the same as Grover's original algorithm but stopping after a particular r-dependent number of iterations) that finds a root (again with negligible failure chance) using approximately $(2^b/r)^{1/2}$ quantum evaluations of f and a small amount of overhead. One can easily achieve the same result by using Grover's original algorithm sensibly (as mentioned in [15]): choose a fast but sufficiently random map from $b - \lceil \lg r \rceil$ bits to b bits; the composition of this map with f has a good chance of having a unique root; apply Grover's algorithm to this composition; repeat several times so that the failure chance becomes negligible.

Even more generally, consider any computable function f with a b-bit input. A more general algorithm in [6] finds a root using approximately $(2^b/r)^{1/2}$ quantum evaluations of f and a small amount of overhead, where r is the number of roots. If no root exists then the algorithm says so after approximately $2^{b/2}$ quantum evaluations of f. As above, the algorithm can fail (or take longer than expected), but only with negligible probability; and, as above, the same result can also be obtained from Grover's algorithm.

As a trivial application, take $b = n$ and $f = \Sigma - s$: finding a root of $\Sigma - s$ costs $2^{n/2}$. Some implementation details of this quantum subset-sum algorithm appeared in [9] in 2009. We emphasize, however, that the same operation count is achieved by well-known non-quantum algorithms, and is solidly beaten by recent non-quantum algorithms.

Review: Left-right split. Define $L_1 = \{(\Sigma(I_1), I_1) : I_1 \subseteq \{1, 2, \dots, n/2\}\}$ and $L_2 = \{(s - \Sigma(I_2), I_2) : I_2 \subseteq \{n/2 + 1, n/2 + 2, \dots, n\}\}$. Note that each of these sets has size just $2^{n/2}$.

Compute L_1 by enumerating sets I_1. Store the elements of L_1 in a table, and sort the table by its first coordinate. Compute L_2 by enumerating sets I_2. For each $(s - \Sigma(I_2), I_2) \in L_2$, look for $s - \Sigma(I_2)$ by binary search in the sorted table. If there is a collision $\Sigma(I_1) = s - \Sigma(I_2)$, print out $I_1 \cup I_2$ as a root of $\Sigma - s$ and stop. If there are no collisions, print "there is no subset-sum solution" and stop.

This algorithm costs $2^{n/2}$. It uses $2^{n/2}$ memory, and one can object that random access to memory is expensive, but we emphasize that this paper follows the tradition of simply counting operations. There are several standard variants of this algorithm: for example, one can sort L_1 and L_2 together, or one can store the elements of L_1 in a hash table.

Quantum left-right split. Redefine L_1 as $\{(\Sigma(I_1), I_1) : I_1 \subseteq \{1, 2, \dots, n/3\}\}$; note that $n/2$ has changed to $n/3$. Compute and sort L_1 as above; this costs $2^{n/3}$.

Consider the function f that maps a subset $I_2 \subseteq \{n/3 + 1, n/3 + 2, \dots, n\}$ to 0 if $s - \Sigma(I_2)$ is a first coordinate in L_1, otherwise to 1. Binary search in the sorted L_1 table costs only 1, so computing f costs only 1.

Now use quantum search to find a root of f, i.e., a subset $I_2 \subseteq \{n/3 + 1, \dots, n\}$ such that $s - \Sigma(I_2)$ is a first coordinate in L_1. There are $2n/3$ bits of input to f, so the quantum search costs $2^{n/3}$.

Finally, find an I_1 such that $s - \Sigma(I_2) = \Sigma(I_1)$, and print $I_1 \cup I_2$. Like the previous algorithm, this algorithm finds a root of $\Sigma - s$ if one exists; any root I of $\Sigma - s$ can be expressed as $I_1 \cup I_2$.

Note that, with the original split of $\{1, \dots, n\}$ into left and right halves, quantum search would not have reduced cost (modulo polynomial factors). Generalizing the original algorithm to allow an unbalanced split, and in particular a split into $n/3$ and $2n/3$, is pointless without quantum computers but essential for the quantum optimization. The split into $n/3$ and $2n/3$ imitates the approach used by Brassard, Høyer, and Tapp in [8] to search for hash-function collisions.

This algorithm uses $2^{n/3}$ memory, and as before one can object that random access to memory is expensive, especially when memory locations are quantum superpositions. See [5] for an extended discussion of the analogous objection to [8]. We again emphasize that this paper follows the tradition of simply counting operations; we do not claim that improved operation counts imply improvements in other cost models.

3 Walks

This section summarizes Ambainis's unique-collision-finding algorithm [2] (from the edge-walk perspective of [23]); introduces a new way to streamline Ambainis's algorithm; and applies the streamlined algorithm to the subset-sum context, obtaining cost $2^{n/3}$ in a different way from Section 2. This section's subset-sum algorithm uses collision finding as a black box, but the faster algorithms in Section 5 do not.

Review: Quantum walks for finding unique collisions. Consider any computable function f with b-bit inputs such that there is a unique pair of colliding inputs, i.e., exactly one pair (x, y) of b-bit strings such that $f(x) = f(y)$. The problem tackled in [2] is to find this pair (x, y).

The algorithm has a positive integer parameter $r < 2^b$, normally chosen on the scale of $2^{2b/3}$. At each step the algorithm is in a superposition of states of the form $(S, f(S), T, f(T))$. Here S and T are sets of b-bit strings such that $\#S = r$, $\#T = r$, and $\#(S \cap T) = r - 1$; i.e., S and T are adjacent vertices in the "Johnson graph" of r-subsets of the set of b-bit strings, where edges are between sets that differ in exactly one element. The notation $f(S)$ means $\{f(x) : x \in S\}$.

The algorithm begins in a uniform superposition of states; setting up this superposition uses $O(r)$ quantum evaluations of f. The algorithm then performs a "quantum walk" that alternates two types of steps: diffusing each state to a new choice of T while keeping S fixed, and diffusing each state to a new choice of S while keeping T fixed. Only one element of T changes when S is fixed (and vice versa), so each step uses only $O(1)$ quantum evaluations of f.

Periodically (e.g., after every $2\lceil\sqrt{r}\rceil$ steps) the algorithm negates the amplitude of every state in which S contains a colliding pair, i.e., in which $\#f(S) < r$. Because $f(S)$ has already been computed, checking whether $\#f(S) < r$ does not involve any evaluations of f. One can object that this check is nevertheless extremely expensive; this objection is discussed in the "data structures" subsection below.

Ambainis's analysis shows that after roughly $2^b/\sqrt{r}$ steps the algorithm has high probability of being in a state in which S contains a colliding pair. Observing this state and then sorting the pairs $(f(x), x)$ for $x \in S$ reveals the colliding pair. Overall the algorithm uses only $O(2^{2b/3})$ evaluations of f.

As in the case of Grover's algorithm, this algorithm is easily generalized to the case that there are p pairs of colliding inputs, and to the case that p is not known in advance. The algorithm is also easily generalized to functions of S more complicated than "contains a colliding pair".

Data structures. The most obvious way to represent a set of b-bit strings is as a sorted array. The large overlap between S and T suggests storing the union $S \cup T$, together with a pointer to the element not in S and a pointer to the element not in T; similar comments apply to the multisets $f(S)$ and $f(T)$. Keeping a running tally of $\#f(S)$ allows easily checking whether $\#f(S) < r$.

To decide whether a b-bit string x is suitable as a new element of T, one must check whether $x \in S$. Actually, what the diffusion steps need is not merely knowing whether $x \in S$, but also knowing the number of elements of S smaller than x. ("Smaller" need not be defined lexicographically; the real objective is to compute a bijective map from b-bit strings to $\{1, 2, 3, \ldots, 2^b\}$ that maps S to $\{1, 2, 3, \ldots, r\}$.) The obvious sorted-array data structure allows these questions to be efficiently answered by binary search.

The big problem with this data structure is that inserting the new string into T requires, in the worst case, moving the other r elements of the array. This cost-r operation is performed at every step of the quantum walk, and dominates the total cost of the algorithm (unless evaluating f is very slow).

There is an extensive literature on classical data structures that support these operations much more efficiently. However, adapting a data structure to the quantum context raises three questions:

- Is the data-structure performance a random variable? Many data structures in the literature are randomized and provide good *average-case* performance but not good *worst-case* performance. The standard conversion of an algorithm to a quantum circuit requires first expressing the algorithm as a classical combinatorial circuit; the size of this circuit reflects the *worst-case* performance of the original algorithm.
- Does the performance of the data structure depend on S? For example, a standard hash table provides good performance for *most* sets S but not for *all* sets S.
- Is the data structure history-dependent? For most data structures, the representation of a set S depends on the order in which elements were added to and removed from the set. This spoils the analysis of the quantum walk through sets S, and presumably spoils the actual performance of the walk.

The first problem is usually minor: one can simply stop each algorithm after a constant time, where the constant is chosen so that the chance of an incorrect answer is negligible. The second problem can usually be converted into the first problem by some extra randomization: for example, one can choose a random hash function from a suitable family (as suggested by Wegman and Carter in [**33**]), or encrypt the b-bit strings before storing them. But the third problem is much more serious: it rules out balanced trees, red-black trees, most types of hash tables, etc.

Ambainis handles these issues in [**2**, Section 6] with an ad-hoc "combination of a hash table and a skip list", requiring several pages of analysis. We point out a much simpler solution: storing S etc. in a *radix tree*. Presumably this also saves time, although the speedup is not visible at the level of detail of our analysis.

The simplest type of radix tree is a binary tree in which the left subtree stores $\{x : (0, x) \in S\}$ and the right subtree stores $\{x : (1, x) \in S\}$; subtrees storing empty sets are omitted. To check whether $x \in S$ one starts from the root of the tree and follows pointers according to the bits of x in turn; the worst-case number of operations is proportional to the number of bits in x. Counting the number of elements of S smaller than x is just as easy if each tree node is augmented by a count of the number of elements below that node. A tree storing $f(S)$, with each leaf node accompanied by its multiplicity, allows an efficient running tally of the number $\#f(S)$ of distinct elements of $f(S)$, and in particular quickly checking whether $\#f(S) < r$.

Randomizing the memory layout of the nodes for the radix tree for S (inductively, by placing each new node at a uniform random free position) provides history-independence for the classical data structure: each possible representation of S has equal probability to appear. Similarly, creating a uniform superposition over all possible memory layouts of the nodes produces a unique quantum data structure representing S.

Subset-sum solutions via collisions. It is straightforward to recast the subset-sum problem as a collision-finding problem.

Consider the function f that maps $(1, I_1)$ to $\Sigma(I_1)$ for $I_1 \subseteq \{1, 2, \ldots, n/2\}$, and maps $(2, I_2)$ to $s - \Sigma(I_2)$ for $I_2 \subseteq \{n/2+1, n/2+2, \ldots, n\}$. Use the algorithm described above to find a collision in f. There are only $n/2+1$ bits of input to f, so the cost of this algorithm is only $2^{n/3}$.

In the "hard" cases of interest in this paper, there are not likely to be many collisions among inputs $(1, I_1)$, and there are not likely to be many collisions among inputs $(2, I_2)$, so the collision found has a good chance of having the form $\Sigma(I_1) = s - \Sigma(I_2)$, i.e., $\Sigma(I_1 \cup I_2) = s$. One can, alternatively, tweak the algorithm to ignore collisions among $(1, I_1)$ and collisions among $(2, I_2)$ even if such collisions exist.

4 Moduli

This section discusses the use of a "modulus" to partition the spaces being searched in Section 2. The traditional view is that this is merely a method to reduce memory consumption (which we do not measure in this paper); but moduli are also an essential building block for the faster algorithms of Section 5. This section reviews the traditional left-right split with a modulus, and then states a quantum algorithm with a modulus, as a warmup for the faster quantum algorithm of Section 5.

Schroeppel and Shamir in [29] introduced an algorithm with essentially the same reduction of memory consumption, but that algorithm does not use moduli and does not seem helpful in Section 5. Three decades later, Howgrave-Graham and Joux in [17, eprint version, Section 3.1] described the left-right split with a modulus as a "useful practical variant" of the Schroeppel–Shamir algorithm; Becker, Coron, and Joux stated in [3] that this was a "simpler but heuristic variant of Schroeppel–Shamir". A more general algorithm (with one minor restriction, namely a prime choice of modulus) had already been stated a few years earlier by Elsenhans and Jahnel; see [10, Section 4.2.1] and [11, page 2]. There are many earlier papers that used moduli to partition input spaces without stating the idea in enough generality to cover subset sums; we have not attempted to comprehensively trace the history of the idea.

Review: Left-right split with a modulus. Choose a positive integer $M \approx 2^{n/4}$, and choose $t \in \{0, 1, 2, \ldots, M-1\}$. Compute

$$L_1 = \{(\Sigma(I_1), I_1) : I_1 \subseteq \{1, 2, \ldots, n/2\}, \quad \Sigma(I_1) \equiv t \pmod{M}\}.$$

The problem of finding all I_1 here is a size-$n/2$ subset-sum problem modulo M. This problem can, in turn, be solved as a small number of size-$n/2$ subset-sum problems without moduli, namely searching for subsets of $x_1 \bmod M, x_2 \bmod M, \ldots, x_{n/2} \bmod M$ having sum t or $t + M$ or $\ldots$ or $t + (n/2 - 1)M$. Note, however, that it is important for this size-$n/2$ subroutine to find *all* solutions rather than just *one* solution.

A reasonable choice of subroutine here is the original left-right-split algorithm (without a modulus). This subroutine costs $2^{n/4}$ for a problem of size $n/2$, and is trivially adapted to find all solutions. For this adaptation one must add the number of solutions to the cost, but in this context one expects only about $2^{n/2}/M \approx 2^{n/4}$ subsets I_1 to satisfy $\Sigma(I_1) \equiv t \pmod{M}$, for a total cost of $2^{n/4}$. One can also tweak this subroutine to work directly with sums modulo M, rather than separately handling t, $t + M$, etc.

Similarly compute

$$L_2 = \{(s - \Sigma(I_2), I_2) : I_2 \subseteq \{n/2+1, \ldots, n\}, \quad \Sigma(I_2) \equiv s - t \pmod{M}\}.$$

Store L_1 in a sorted table, and for each $(s - \Sigma(I_2), I_2) \in L_2$ check whether $s - \Sigma(I_2)$ appears in the table. If there is a collision $\Sigma(I_1) = s - \Sigma(I_2)$, print $I_1 \cup I_2$ as a root of $\Sigma - s$ and stop. Otherwise try another value of t, repeating until all choices of $t \in \{0, 1, 2, \ldots, M-1\}$ are exhausted.

One expects each choice of t to cost $2^{n/4}$, as discussed above. There are $M \approx 2^{n/4}$ choices of t, for a total cost of $2^{n/2}$. If there is a subset-sum solution then it will be found for some choice of t.

Quantum search with a modulus. The algorithm above is a classical search for a root of the function that maps t to 0 if there is a collision $\Sigma(I_1) = s - \Sigma(I_2)$ satisfying $\Sigma(I_1) \equiv t \pmod{M}$ (and therefore also satisfying $\Sigma(I_2) \equiv s - t \pmod{M}$). One way to take advantage of quantum computers here is to instead search for t by Grover's algorithm, which finds the root with only $\sqrt{M} \approx 2^{n/8}$ quantum evaluations of the same function, for a total cost of $2^{n/8}2^{n/4} = 2^{3n/8}$.

Quantum walks with a modulus. A different way to take advantage of quantum computers is as follows.

Recall that the collision-finding algorithm of Section 3 walks through adjacent pairs of size-r sets S, searching for sets that contain collisions under a function f. Each set S is stored in a radix tree, as is the multiset $f(S)$. Each radix tree is augmented to record at each node the number of distinct elements below that node, allowing fast evaluation of the number of elements of S smaller than a specified input and fast evaluation of whether S contains a collision.

One can design this collision-finding algorithm in four steps:

- Start with a simple classical collision-finding algorithm that computes $f(S)$ where S is the set of *all* 2^b b-bit strings.

- Generalize to a lower-probability algorithm that computes $f(S)$ where S is a set of only r strings and that checks whether S contains the collision.
- Build a data structure that expresses the entire computation of the lower-probability algorithm. Observe that this data structure allows efficiently moving from S to an adjacent set: a single element of S has only a small impact on the computation.
- Apply a quantum walk, walking through adjacent pairs of size-r sets S while maintaining this data structure for each S. This takes $O(\sqrt{r}/\sqrt{p})$ steps where p is the success probability of the previous algorithm, plus cost r to set up the data structure in the first place.

We now imitate the same four-step approach, starting from the classical left-right split with a modulus and ending with a new quantum subset-sum algorithm. The following description assumes that the correct value of t is already known; the overhead of searching for t is discussed after the algorithm.

First step: Classical algorithm. Recall that the subroutine to find all I_1 with $\Sigma(I_1) \equiv t$ computes $\Sigma(I_{11}) \bmod M$ for all $I_{11} \subseteq \{1, 2, \ldots, n/4\}$; computes $t - \Sigma(I_{12}) \bmod M$ for all $I_{12} \subseteq \{n/4+1, n/4+2, \ldots, n/2\}$; and finds collisions between $\Sigma(I_{11}) \bmod M$ and $t - \Sigma(I_{12}) \bmod M$. Similarly, the subroutine to find all I_2 finds collisions between $\Sigma(I_{21}) \bmod M$ for $I_{21} \subseteq \{n/2+1, n/2+2, \ldots, 3n/4\}$ and $s - t - \Sigma(I_{22}) \bmod M$ for $I_{22} \subseteq \{3n/4+1, 3n/4+2, \ldots, n\}$. The high-level algorithm finishes by finding collisions between $\Sigma(I_1)$ and $s - \Sigma(I_2)$.

Second step: Generalize to a lower-probability computation by restricting the sets that contain collisions. Specifically, instead of enumerating *all* subsets I_{11}, take a random collection S_{11} containing exactly r such subsets; here $r \le 2^{n/4}$ is an algorithm parameter optimized below. Similarly take a random collection S_{12} of exactly r subsets I_{12}. Find collisions between $\Sigma(I_{11}) \bmod M$ and $t - \Sigma(I_{12}) \bmod M$; one expects about r^2/M collisions, producing r^2/M sets $I_1 = I_{11} \cup I_{12}$ satisfying $\Sigma(I_1) \equiv t \pmod{M}$. Similarly take random size-r sets S_{21} and S_{22} consisting of, respectively, subsets I_{21} and I_{22}; find collisions between $\Sigma(I_{21}) \bmod M$ and $s-t-\Sigma(I_{22}) \bmod M$, obtaining about r^2/M sets I_2 satisfying $\Sigma(I_2) \equiv s-t \pmod{M}$. Finally check for collisions between $\Sigma(I_1)$ and $s-\Sigma(I_2)$. One can visualize the construction of $I = I_1 \cup I_2$ as a three-level binary tree with I as the root, I_1 and I_2 as its left and right children, and $I_{11}, I_{12}, I_{21}, I_{22}$ as the leaves.

Recall that we are assuming that the correct value of t is known, i.e., that the desired subset-sum solution is expressible as $I_1 \cup I_2$ with $\Sigma(I_1) \equiv t \pmod{M}$ and $\Sigma(I_2) \equiv s - t \pmod{M}$. Then S_{11} has probability $r/2^{n/4}$ of containing the set $I_{11} = I_1 \cap \{1, 2, \ldots, n/4\}$. Similar comments apply to S_{12}, S_{21}, and S_{22}, for an overall success probability of $(r/2^{n/4})^4$.

The optimal choice of r is discussed later, and is far below the classical extreme $2^{n/4}$. This drop is analogous to the drop in list sizes from a simple left-right split (list size $2^{n/2}$) to the quantum-walk subset-sum algorithm of Section 3 (list size $2^{n/3}$). This drop has an increasingly large impact on subsequent levels of the tree: the number of sets $I_{11}, I_{12}, I_{21}, I_{22}$ is reduced by a factor of $2^{n/4}/r$, and the number of sets I_1, I_2 is reduced by a factor of $(2^{n/4}/r)^2$.

An interesting consequence of this drop is that one can reduce M without creating bottlenecks at subsequent levels of the tree. Specifically, taking $M \approx r$ means that one expects about r sets I_1 and about r sets I_2.

Third step: Data structure. This lower-probability computation is captured by a data structure that contains the following sets in augmented radix trees:

- The size-r set S_{11} of subsets $I_{11} \subseteq \{1, 2, \ldots, n/4\}$.
- The set $\{(\Sigma(I_{11}) \bmod M, I_{11}) : I_{11} \in S_{11}\}$.
- The size-r set S_{12} of subsets $I_{12} \subseteq \{n/4+1, n/4+2, \ldots, n/2\}$.
- The set $\{(t - \Sigma(I_{12}) \bmod M, I_{12}) : I_{12} \in S_{12}\}$.
- The set S_1 of $I_{11} \cup I_{12}$ for all pairs $(I_{11}, I_{12}) \in S_{11} \times S_{12}$ such that $\Sigma(I_{11}) \equiv t - \Sigma(I_{12}) \pmod M$, subject to the limit discussed below.
- The size-r set S_{21} of subsets $I_{21} \subseteq \{n/2+1, n/2+2, \ldots, 3n/4\}$.
- The set $\{(\Sigma(I_{21}) \bmod M, I_{21}) : I_{21} \in S_{21}\}$.
- The size-r set S_{22} of subsets $I_{22} \subseteq \{3n/4+1, 3n/4+2, \ldots, n\}$.
- The set $\{(s - t - \Sigma(I_{22}) \bmod M, I_{22}) : I_{22} \in S_{22}\}$.
- The set S_2 of $I_{21} \cup I_{22}$ for all pairs $(I_{21}, I_{22}) \in S_{21} \times S_{22}$ such that $\Sigma(I_{21}) \equiv s - t - \Sigma(I_{22}) \pmod M$.
- The set $\{(\Sigma(I_1), I_1) : I_1 \in S_1\}$.
- The set $\{(s - \Sigma(I_2), I_2) : I_2 \in S_2\}$.
- The set S of $I_1 \cup I_2$ for all pairs $(I_1, I_2) \in S_1 \times S_2$ such that $\Sigma(I_1) = s - \Sigma(I_2)$, subject to the limit discussed below.

Note that this data structure supports, e.g., fast removal of an element I_{11} from S_{11} followed by fast insertion of a replacement element I'_{11}. Checking for $\Sigma(I_{11}) \bmod M$ in the stored set $\{(t - \Sigma(I_{12}) \bmod M, I_{12})\}$ efficiently shows which elements have to be removed from S_1, and then a similar check shows which elements have to be removed from S.

Each element I_{11} of S_{11} affects very few elements of S_1; on average one expects "very few" to be $r/M \approx 1$. To control the time taken by each step of the algorithm we put a polynomial limit on the number of elements of S_1 involving any particular I_{11}. If this limit is reached then (to ensure history-independence) we use a random selection of elements, but this limit has negligible chance of affecting the algorithm output. Similar comments apply to I_{12}, I_{21}, and I_{22}.

Fourth step: Walk through adjacent pairs of 4-tuples $(S_{11}, S_{12}, S_{21}, S_{22})$ of size-r sets, maintaining the data structure above and searching for tuples for which the final set S is nonempty. Amplifying the $(r/2^{n/4})^4$ success probability mentioned above to a high probability requires a quantum walk consisting of $O(\sqrt{r}(2^{n/4}/r)^2)$ steps. Setting up the data structure in the first place costs $O(r)$.

For r on the scale of $2^{0.2n}$ these costs are balanced at $2^{0.2n}$; but recall that this assumes that t is already known. A classical search for t means repeating this algorithm $M \approx r \approx 2^{0.2n}$ times, for a total cost of $2^{0.4n}$. We do better by using amplitude amplification [7], repeating the quantum walk only $2^{0.1n}$ times, for a total cost of $2^{0.3n}$. We do not describe amplitude amplification in detail; this subset-sum algorithm is superseded by the approach of Section 5.

5 Representations

"Representations" are a technique to improve the "left-right split with a modulus" algorithm of Section 4. Howgrave-Graham and Joux introduced this technique in [17] and obtained a subset-sum algorithm that costs just $2^{(0.337\ldots+o(1))n}$. Beware that [17] incorrectly claimed a cost of $2^{(0.311\ldots+o(1))n}$; the underlying flaw in the analysis was corrected in [3] with credit to May and Meurer.

This section reviews the Howgrave-Graham–Joux algorithm, and then presents a new quantum subset-sum algorithm with cost only $2^{(0.241\ldots+o(1))n}$. The new quantum algorithm requires the quantum-walk machinery discussed in Section 3.

We simplify the algorithm statements in this section by considering only half-weight sets I; i.e., we search only for sets I with $\#I = n/2$ and $\Sigma(I) = s$. We comment, however, that straightforward generalizations of these algorithms, still within the same cost bound, handle smaller known weights (adjusting the set sizes shown below, at the expense of some complications in notation); also handle larger known weights (replacing I and s with their complements); and handle arbitrary unknown weights (trying all $n + 1$ possible weights).

Review: the basic idea of representations. Recall that the original left-right split partitions I as $I_1 \cup I_2$ where $I_1 \subseteq \{1, \ldots, n/2\}$ and $I_2 \subseteq \{n/2 + 1, \ldots, n\}$. The main idea of representations is to partition I in a different, ambiguous way as $I_1 \cup I_2$ with $I_1, I_2 \subseteq \{1, 2, \ldots, n\}$ and $\#I_1 = \#I_2 = n/4$. Note that there are $\binom{n/2}{n/4} \approx 2^{n/2}$ such partitions. The key observation is that finding only one out of these exponentially many *representations* (I_1, I_2) of I is sufficient to solve the subset-sum problem.

Recall also the idea of moduli: pick $t \in \{0, 1, 2, \ldots, M - 1\}$ and hope that $\Sigma(I_1) \equiv t \pmod M$. In Section 4, there was only one choice of I_1, so one expects each choice of t to work with probability only about $1/M$, forcing a search through choices of t. In this section, there are $\approx 2^{n/2}$ choices of I_1, so one expects a single choice of t to work with high probability for M as large as $2^{n/2}$.

These observations motivate the following strategy. Pick a modulus $M \approx 2^{n/2}$ and choose a random target value $t \in \{0, 1, \ldots, M - 1\}$. Compute

$$L_1 = \{(\Sigma(I_1), I_1) : I_1 \subseteq \{1, \ldots, n\},\ \#I_1 = n/4,\ \Sigma(I_1) \equiv t \pmod M\}$$

and

$$L_2 = \{(s - \Sigma(I_2), I_2) : I_2 \subseteq \{1, \ldots, n\},\ \#I_2 = n/4,\ \Sigma(I_2) \equiv s - t \pmod M\}.$$

If there is a collision $\Sigma(I_1) = s - \Sigma(I_2)$ satisfying $I_1 \cap I_2 = \{\}$, print $I_1 \cup I_2$ and stop. If there are no such collisions, repeat with another choice of t. One expects a negligible failure probability after a polynomial number of repetitions.

(We point out that if $t \equiv s - t \pmod M$ then computing L_1 immediately produces L_2. One can arrange for this by choosing a random odd M and taking t to be half of s modulo M; one can also choose a random even M if s is even. If other speed constraints prevent M from being chosen randomly then one can still try these special values of t first. Similar comments apply to the next level of the Howgrave-Graham–Joux algorithm described below. Of course, the resulting speedup is not visible at the level of detail of our analysis.)

One expects $\#L_1 \approx \binom{n}{n/4}/2^{n/2} \approx 2^{0.311\ldots n}$: there are $\binom{n}{n/4}$ sets $I_1 \subseteq \{1,\ldots,n\}$ with $\#I_1 = n/4$, and one expects each I_1 to satisfy $\Sigma(I_1) \equiv t \pmod M$ with probability $1/M \approx 1/2^{n/2}$. (The calculation of $0.311\ldots$ relies on the standard approximation $\binom{n}{\alpha n} \approx 2^{H(\alpha)n}$, where $H(\alpha) = -\alpha\log_2\alpha - (1-\alpha)\log_2(1-\alpha)$.) The same comment applies to L_2. One also expects the number of collisions between L_1 and L_2 to be exponentially large, about $\#L_1\#L_2/2^{n/2} \approx 2^{0.122\ldots n}$, since each $\Sigma(I_1)$ is already known to match each $s - \Sigma(I_2)$ modulo M; but the only collisions satisfying $I_1 \cap I_2 = \{\}$ are collisions arising from subset-sum solutions.

The remaining task is to compute L_1 and L_2 in the first place. Howgrave-Graham and Joux solve these two weight-$n/4$ modular subset-sum problems by first applying another level of representations (using a smaller modulus that divides M), obtaining four weight-$n/8$ modular subset-sum problems; they solve each of those problems with a weight-$n/16$ left-right split. The details appear below.

Review: The complete Howgrave-Graham–Joux algorithm. Choose a positive integer $M_1 \approx 2^{n/4}$. Choose a positive integer $M \approx 2^{n/2}$ divisible by M_1. Choose randomly $s_1 \in \{0,1,\ldots,M-1\}$ and define $s_2 = s - s_1$. Choose randomly $s_{11} \in \{0,1,\ldots,M_1-1\}$ and define $s_{12} = s_1 - s_{11}$. Choose randomly $s_{21} \in \{0,1,\ldots,M_1-1\}$ and define $s_{22} = s_2 - s_{21}$. Also choose random subsets $R_{111}, R_{121}, R_{211}, R_{221}$ of $\{1,2,\ldots,n\}$, each of size $n/2$, and define R_{ij2} as the complement of R_{ij1}.

The following algorithm searches for a weight-$n/2$ subset-sum solution I decomposed as follows: $I = I_1 \cup I_2$ with $\#I_i = n/4$ and $\Sigma(I_i) \equiv s_i \pmod M$; furthermore $I_i = I_{i1} \cup I_{i2}$ with $\#I_{ij} = n/8$ and $\Sigma(I_{ij}) \equiv s_{ij} \pmod{M_1}$; furthermore $I_{ij} = I_{ij1} \cup I_{ij2}$ with $\#I_{ijk} = n/16$ and $I_{ijk} \subseteq R_{ijk}$. These constraints are shown as a tree in Figure 5.1. One expects a weight-$n/2$ subset-sum solution to decompose in this way with high probability (inverse polynomial in n), as discussed later, and if it does decompose in this way then it is in fact found by this algorithm.

Start with, for each $(i,j,k) \in \{1,2\}\times\{1,2\}\times\{1,2\}$, the set S_{ijk} of all subsets $I_{ijk} \subseteq R_{ijk}$ with $\#I_{ijk} = n/16$. Compute the sets

$$L_{ij1} = \{(\Sigma(I_{ij1}) \bmod M_1, I_{ij1}) : I_{ij1} \in S_{ij1}\},$$
$$L_{ij2} = \{(s_{ij} - \Sigma(I_{ij2}) \bmod M_1, I_{ij2}) : I_{ij2} \in S_{ij2}\}.$$

Merge L_{ij1} and L_{ij2} to obtain the set S_{ij} of $I_{ij1} \cup I_{ij2}$ for all pairs $(I_{ij1}, I_{ij2}) \in S_{ij1} \times S_{ij2}$ such that $\Sigma(I_{ij1}) \equiv s_{ij} - \Sigma(I_{ij2}) \pmod{M_1}$. Note that each $I_{ij} \in S_{ij}$ has $\Sigma(I_{ij}) \equiv s_{ij} \pmod{M_1}$ and $\#I_{ij} = n/8$. Next compute the sets

$$L_{i1} = \{(\Sigma(I_{i1}) \bmod M, I_{i1}) : I_{i1} \in S_{i1}\},$$
$$L_{i2} = \{(s_i - \Sigma(I_{i2}) \bmod M, I_{i2}) : I_{i2} \in S_{i2}\}.$$

Merge L_{i1} and L_{i2} to obtain the set S_i of $I_{i1} \cup I_{i2}$ for all pairs $(I_{i1}, I_{i2}) \in S_{i1} \times S_{i2}$ such that $\Sigma(I_{i1}) \equiv s_i - \Sigma(I_{i2}) \pmod M$ and $I_{i1} \cap I_{i2} = \{\}$. Note that each $I_i \in S_i$

$$\begin{tikzcd}[column sep=tiny]
 & & & I \subseteq \{1,\dots,n\};\quad \#I = n/2;\quad \Sigma(I) = s & & & & \\
 & \begin{array}{c} I_1 \subseteq \{1,\dots,n\};\quad \#I_1 = n/4; \\ \Sigma(I_1) \equiv s_1 \pmod{M} \end{array} \arrow[urr] & & & & \begin{array}{c} I_2 \subseteq \{1,\dots,n\};\quad \#I_2 = n/4; \\ \Sigma(I_2) \equiv s_2 \pmod{M} \end{array} \arrow[ull] & & \\
\begin{array}{c} I_{11} \subseteq \{1,\dots,n\}; \\ \#I_{11} = n/8; \\ \Sigma(I_{11}) \equiv s_{11} \\ \pmod{M_1} \end{array} \arrow[ur] & & \begin{array}{c} I_{12} \subseteq \{1,\dots,n\}; \\ \#I_{12} = n/8; \\ \Sigma(I_{12}) \equiv s_{12} \\ \pmod{M_1} \end{array} \arrow[ul] & & \begin{array}{c} I_{21} \subseteq \{1,\dots,n\}; \\ \#I_{21} = n/8; \\ \Sigma(I_{21}) \equiv s_{21} \\ \pmod{M_1} \end{array} \arrow[ur] & & \begin{array}{c} I_{22} \subseteq \{1,\dots,n\}; \\ \#I_{22} = n/8; \\ \Sigma(I_{22}) \equiv s_{22} \\ \pmod{M_1} \end{array} \arrow[ul] & \\
\begin{array}{c} I_{111} \\ \subseteq R_{111}; \\ \#I_{111} \\ = n/16 \end{array} \arrow[u] & \begin{array}{c} I_{112} \\ \subseteq R_{112}; \\ \#I_{112} \\ = n/16 \end{array} \arrow[ul] & \begin{array}{c} I_{121} \\ \subseteq R_{121}; \\ \#I_{121} \\ = n/16 \end{array} \arrow[u] & \begin{array}{c} I_{122} \\ \subseteq R_{122}; \\ \#I_{122} \\ = n/16 \end{array} \arrow[ul] & \begin{array}{c} I_{211} \\ \subseteq R_{211}; \\ \#I_{211} \\ = n/16 \end{array} \arrow[u] & \begin{array}{c} I_{212} \\ \subseteq R_{212}; \\ \#I_{212} \\ = n/16 \end{array} \arrow[ul] & \begin{array}{c} I_{221} \\ \subseteq R_{221}; \\ \#I_{221} \\ = n/16 \end{array} \arrow[u] & \begin{array}{c} I_{222} \\ \subseteq R_{222}; \\ \#I_{222} \\ = n/16 \end{array} \arrow[ul]
\end{tikzcd}$$

Fig. 5.1. Decomposition of a weight-$n/2$ subset-sum solution $I \subseteq \{1, 2, \dots, n\}$.

has $\Sigma(I_i) \equiv s_i \pmod{M}$ and $\#I_i = n/4$. Next compute the sets

$$L_1 = \{(\Sigma(I_1), I_1) : I_1 \in S_1\},$$
$$L_2 = \{(s - \Sigma(I_2), I_2) : I_2 \in S_2\}.$$

Merge L_1 and L_2 to obtain the set S of $I_1 \cup I_2$ for all pairs $(I_1, I_2) \in S_1 \times S_2$ such that $\Sigma(I_1) = s - \Sigma(I_2)$ and $I_1 \cap I_2 = \{\}$. Note that each $I \in S$ has $\Sigma(I) = s$ and $\#I = n/2$. If S is nonempty, print its elements and stop.

Review: Success probability of the algorithm. Consider any weight-$n/2$ subset $I \subseteq \{1, \dots, n\}$ with $\Sigma(I) = s$. There are $\binom{n/2}{n/4} \approx 2^{n/2} \approx M$ ways to partition I as $I_1 \cup I_2$ with $\#I_1 = n/4$ and $\#I_2 = n/4$, and as discussed earlier one expects that with high probability at least one of these ways will satisfy $\Sigma(I_1) \equiv s_1 \pmod{M}$, implying $\Sigma(I_2) \equiv s_2 \pmod{M}$.

Similarly, there are $\binom{n/4}{n/8} \approx 2^{n/4} \approx M_1$ ways to partition I_1 as $I_{11} \cup I_{12}$ with $\#I_{11} = n/8$ and $\#I_{12} = n/8$. One expects that with high probability at least one of these ways will satisfy $\Sigma(I_{11}) \equiv s_{11} \pmod{M_1}$, implying $\Sigma(I_{12}) \equiv s_{12} \pmod{M_1}$ (since M_1 divides M and $s_{11} + s_{12} = s_1$). Analogous comments apply to I_2, I_{21}, I_{22}.

A uniform random subset of a set of size $n/8$ has size exactly $n/16$ with probability $\Theta(1/\sqrt{n})$, so with probability $\Theta(1/n^2)$ each of the four sets $I_{ij1} = I_{ij} \cap R_{ij1}$ has size exactly $n/16$, implying that each of the four complementary sets $I_{ij2} = I_{ij} \cap R_{ij2}$ also has size exactly $n/16$. (Experiments indicate that the probability is somewhat worse, although still inverse polynomial in n, if all R_{ij1}

are chosen to be identical, even if this set is randomized as discussed in [17]. The idea of choosing independent sets appeared in [4] with credit to Bernstein.)

Overall the probability of I being decomposed in this way, i.e., of I being found by this algorithm, is inverse polynomial in n. As above, one expects a negligible failure probability after a polynomial number of repetitions of the algorithm.

Review: Cost of the algorithm. Each set S_{ijk} has size $\binom{n/2}{n/16} \approx 2^{0.271\ldots n}$. One expects $\#S_{ij1}\#S_{ij2}/M_1$ collisions $\Sigma(I_{ij1}) \equiv s_{ij} - \Sigma(I_{ij2}) \pmod{M_1}$, and therefore $\#S_{ij1}\#S_{ij2}/M_1 \approx 2^{0.293\ldots n}$ elements in S_{ij}.

Each $\Sigma(I_{i1})$ is already known by construction to be the same as each $s_i - \Sigma(I_{i2})$ modulo M_1. One expects it to be the same modulo M with probability M_1/M, for a total of $\#S_{i1}\#S_{i2}M_1/M \approx 2^{0.337\ldots n}$ collisions modulo M. This also dominates the algorithm's overall running time.

Relatively few of these collisions modulo M have $I_{i1} \cap I_{i2} = \{\}$. The only possible elements of S_1 are sets I_1 with $\Sigma(I_1) \equiv s_1 \pmod{M}$ and $\#I_1 = n/4$; one expects the number of such sets to be $\binom{n}{n/4}/2^{n/2} \approx 2^{0.311\ldots n}$. Furthermore, as discussed earlier, each $\Sigma(I_1)$ is already known by construction to be the same as each $s - \Sigma(I_2)$ modulo M, so one expects it to be the same integer with probability about $M/2^n$, for a total of $\#S_1\#S_2M/2^n \approx 2^{0.122\ldots n}$ collisions.

New: quantum walks with moduli and representations. We now combine quantum walks with the idea of representations. The reader is assumed to be familiar with the simpler quantum algorithm of Section 4.

We introduce a parameter $r \le \binom{n/2}{n/16}$ into the Howgrave-Graham–Joux algorithm, and take each S_{ijk} as a random collection of exactly r weight-$n/16$ subsets I_{ijk} of R_{ijk}. The extreme case $r = \binom{n/2}{n/16}$ is the same as the original Howgrave-Graham–Joux algorithm: in this case S_{ijk} is the set of all weight-$n/16$ subsets of R_{ijk}. For smaller r this generalized algorithm has lower success probability, as discussed below, but is also faster. The resulting computation is captured by the following 29 sets, which we store in augmented radix trees:

- For each i, j, k, a set S_{ijk} consisting of exactly r weight-$n/16$ subsets of R_{ijk}.
- For each i, j, the set $L_{ij1} = \{(\Sigma(I_{ij1}) \bmod M_1, I_{ij1}) : I_{ij1} \in S_{ij1}\}$.
- For each i, j, the set $L_{ij2} = \{(s_{ij} - \Sigma(I_{ij2}) \bmod M_1, I_{ij2}) : I_{ij2} \in S_{ij2}\}$.
- For each i, j, the set S_{ij} of $I_{ij1} \cup I_{ij2}$ for all pairs $(I_{ij1}, I_{ij2}) \in S_{ij1} \times S_{ij2}$ such that $\Sigma(I_{ij1}) \equiv s_{ij} - \Sigma(I_{ij2}) \pmod{M_1}$, subject to the limit discussed below.
- For each i, the set $L_{i1} = \{(\Sigma(I_{i1}) \bmod M, I_{i1}) : I_{i1} \in S_{i1}\}$.
- For each i, the set $L_{i2} = \{(s_i - \Sigma(I_{i2}) \bmod M, I_{i2}) : I_{i2} \in S_{i2}\}$.
- For each i, the set S_i of $I_{i1} \cup I_{i2}$ for all pairs $(I_{i1}, I_{i2}) \in S_{i1} \times S_{i2}$ such that $\Sigma(I_{i1}) \equiv s_i - \Sigma(I_{i2}) \pmod{M}$ and $I_{i1} \cap I_{i2} = \{\}$, subject to the limit discussed below.
- The set $L_1 = \{(\Sigma(I_1), I_1) : I_1 \in S_1\}$.
- The set $L_2 = \{(s - \Sigma(I_2), I_2) : I_2 \in S_2\}$.
- The set S of $I_1 \cup I_2$ for all pairs $(I_1, I_2) \in S_1 \times S_2$ such that $\Sigma(I_1) = s - \Sigma(I_2)$ and $I_1 \cap I_2 = \{\}$, subject to the limit discussed below.

Like the data structure in the quantum walk of Section 4, this data structure supports, e.g., fast removal of an element I_{111} from S_{111} followed by fast insertion of a replacement element I'_{111}.

The optimal choice of r is discussed later; it is far below the starting list size $\binom{n/2}{n/16} \approx 2^{0.272n}$ used by Howgrave-Graham and Joux, and is even below $2^{n/4}$. One expects the number of collisions modulo M_1 to be $r^2/2^{n/4}$, which is smaller than r, and the list sizes on subsequent levels to be even smaller. Consequently this quantum algorithm ends up being bottlenecked at the bottom level of the tree, while the original algorithm is bottlenecked at a higher level.

Furthermore, one expects each element I_{ijk} of S_{ijk} to affect, on average, approximately $r/2^{n/4}$ elements of the set of collisions modulo M, and therefore to affect 0 elements of S_{ij} in almost all cases, 1 element of S_{ij} with exponentially small probability, 2 elements of S_{ij} with far smaller probability, etc. As in Section 4, to control the time taken by each step of the algorithm we put a polynomial limit on the number of elements of S_{ij} involving any particular I_{ijk}, a polynomial limit on the number of elements of S_i involving any particular I_{ij}, and a polynomial limit on the number of elements of S involving any particular I_i. A constant limit of 100 seems ample, and there is no obvious problem with a limit of 1.

Compared to the original Howgrave-Graham–Joux algorithm, the success probability of the generalized algorithm drops by a factor of $(r/\binom{n/2}{n/16})^8$, since each target I_{ijk} has chance only $r/\binom{n/2}{n/16}$ of being in S_{ijk}. Amplifying this to a high probability requires a quantum walk consisting of $O(\sqrt{r}(\binom{n/2}{n/16}/r)^4)$ steps. Setting up the data structure in the first place costs $O(r)$. For r on the scale of $\binom{n/2}{n/16}^{4/4.5}$ these costs are balanced at $\binom{n/2}{n/16}^{4/4.5}$, i.e., at $2^{(0.241\ldots+o(1))n}$.

References

[1] — (no editor), *20th annual symposium on foundations of computer science*, IEEE Computer Society, New York, 1979. MR 82a:68004. See [32].

[2] Andris Ambainis, *Quantum walk algorithm for element distinctness*, SIAM Journal on Computing **37** (2007), 210–239. URL: http://arxiv.org/abs/quant-ph/0311001. Citations in this document: §3, §3, §3.

[3] Anja Becker, Jean-Sebastien Coron, Antoine Joux, *Improved generic algorithms for hard knapsacks*, in Eurocrypt 2011 [**27**] (2011). URL: http://eprint.iacr.org/2011/474. Citations in this document: §1, §1, §1, §4, §5.

[4] Anja Becker, Antoine Joux, Alexander May, Alexander Meurer, *Decoding random binary linear codes in $2^{n/20}$: how $1 + 1 = 0$ improves information set decoding*, in Eurocrypt 2012 [**28**] (2012). URL: http://eprint.iacr.org/2012/026. Citations in this document: §1, §1, §1, §1, §5.

[5] Daniel J. Bernstein, *Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?*, in Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems (2009). URL: http://cr.yp.to/papers.html#collisioncost. Citations in this document: §2.

[6] Michel Boyer, Gilles Brassard, Peter Høyer, Alain Tapp, *Tight bounds on quantum searching*, Fortschritte Der Physik **46** (1998), 493–505. URL: http://arxiv.org/abs/quant-ph/9605034v1. Citations in this document: §2, §2.

[7] Gilles Brassard, Peter Høyer, Michele Mosca, Alain Tapp, *Quantum amplitude amplification and estimation*, in [**20**] (2002), 53–74. URL: http://arxiv.org/abs/quant-ph/0005055. Citations in this document: §4.

[8] Gilles Brassard, Peter Høyer, Alain Tapp, *Quantum cryptanalysis of hash and claw-free functions*, in LATIN'98 [**21**] (1998), 163–169. MR 99g:94013. Citations in this document: §2, §2.

[9] Weng-Long Chang, Ting-Ting Ren, Mang Feng, Lai Chin Lu, Kawuu Weicheng Lin, Minyi Guo, *Quantum algorithms of the subset-sum problem on a quantum computer*, International Conference on Information Engineering **2** (2009), 54–57. Citations in this document: §2.

[10] Andreas-Stephan Elsenhans, Jörg Jahnel, *The diophantine equation $x^4 + 2y^4 = z^4 + 4w^4$*, Mathematics of Computation **75** (2006), 935–940. URL: http://www.uni-math.gwdg.de/jahnel/linkstopaperse.html. Citations in this document: §4.

[11] Andreas-Stephan Elsenhans, Jörg Jahnel, *The Diophantine equation $x^4 + 2y^4 = z^4 + 4w^4$—a number of improvements* (2006). URL: http://www.uni-math.gwdg.de/jahnel/linkstopaperse.html. Citations in this document: §4.

[12] Henri Gilbert (editor), *Advances in cryptology — EUROCRYPT 2010, 29th annual international conference on the theory and applications of cryptographic techniques, French Riviera, May 30–June 3, 2010, proceedings*, Lecture Notes in Computer Science, 6110, Springer, 2010. See [17].

[13] Shafi Goldwasser (editor), *35th annual IEEE symposium on the foundations of computer science. Proceedings of the IEEE symposium held in Santa Fe, NM, November 20–22, 1994*, IEEE, 1994. ISBN 0-8186-6580-7. MR 98h:68008. See [30].

[14] Lov K. Grover, *A fast quantum mechanical algorithm for database search*, in [**26**] (1996), 212–219. MR 1427516. URL: http://arxiv.org/abs/quant-ph/9605043.

[15] Lov K. Grover, *Quantum mechanics helps in searching for a needle in a haystack*, Physical Review Letters **79** (1997), 325–328. URL: http://arxiv.org/abs/quant-ph/9706033. Citations in this document: §2, §2.

[16] Ellis Horowitz, Sartaj Sahni, *Computing partitions with applications to the knapsack problem*, Journal of the ACM **21** (1974), 277–292. Citations in this document: §2.

[17] Nicholas Howgrave-Graham, Antoine Joux, *New generic algorithms for hard knapsacks*, in Eurocrypt 2010 [**12**] (2010). URL: http://eprint.iacr.org/2010/189. Citations in this document: §1, §1, §1, §1, §4, §5, §5, §5.

[18] David S. Johnson, Uriel Feige (editors), *Proceedings of the 39th annual ACM symposium on the theory of computing, San Diego, California, USA, June 11–13, 2007*, Association for Computing Machinery, 2007. ISBN 978-1-59593-631-8. See [23].

[19] Dong Hoon Lee, Xiaoyun Wang (editors), *Advances in cryptology — ASIACRYPT 2011, 17th international conference on the theory and application of cryptology and information security, Seoul, South Korea, December 4–8, 2011, proceedings*, Lecture Notes in Computer Science, 7073, Springer, 2011. ISBN 978-3-642-25384-3. See [24].

[20] Samuel J. Lomonaco, Jr., Howard E. Brandt (editors), *Quantum computation and information. Papers from the AMS Special Session held in Washington, DC,*

January 19–21, 2000, Contemporary Mathematics, 305, American Mathematical Society, 2002. ISBN 0-8218-2140-7. MR 2003g:81006. See [7].

[21] Claudio L. Lucchesi, Arnaldo V. Moura (editors), *LATIN'98: theoretical informatics. Proceedings of the 3rd Latin American symposium held in Campinas, April 20–24, 1998*, Lecture Notes in Computer Science, 1380, Springer, 1998. ISBN ISBN 3-540-64275-7. MR 99d:68007. See [8].

[22] Vadim Lyubashevsky, Adriana Palacio, Gil Segev, *Public-key cryptographic primitives provably as secure as subset sum*, in TCC 2010 [**25**] (2010), 382–400. URL: http://eprint.iacr.org/2009/576. Citations in this document: §1, §1.

[23] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, Miklos Santha, *Search via quantum walk*, in STOC 2007 [**18**] (2007), 575–584. URL: http://arxiv.org/abs/quant-ph/0608026. Citations in this document: §3.

[24] Alexander May, Alexander Meurer, Enrico Thomae, *Decoding random linear codes in $\tilde{O}(2^{0.054n})$*, in Asiacrypt 2011 [**19**] (2011). URL: http://www.cits.rub.de/imperia/md/content/may/paper/decoding.pdf. Citations in this document: §1, §1, §1.

[25] Daniele Micciancio (editor), *Theory of cryptography, 7th theory of cryptography conference, TCC 2010, Zurich, Switzerland, February 9–11, 2010, proceedings*, Lecture Notes in Computer Science, 5978, Springer, 2010. ISBN 978-3-642-11798-5. See [22].

[26] Gary L. Miller (editor), *Proceedings of the twenty-eighth annual ACM symposium on the theory of computing, Philadelphia, PA, May 22–24, 1996*, Association for Computing Machinery, 1996. ISBN 0-89791-785-5. MR 97g:68005. See [14].

[27] Kenneth G. Paterson (editor), *Advances in cryptology — EUROCRYPT 2011, 30th annual international conference on the theory and applications of cryptographic techniques, Tallinn, Estonia, May 15–19, 2011, proceedings*, Lecture Notes in Computer Science, 6632, Springer, 2011. ISBN 978-3-642-20464-7. See [3].

[28] David Pointcheval, Thomas Johansson (editors), *Advances in cryptology — EUROCRYPT 2012 — 31st annual international conference on the theory and applications of cryptographic techniques, Cambridge, UK, April 15–19, 2012, proceedings*, Lecture Notes in Computer Science, 7237, Springer, 2012. ISBN 978-3-642-29010-7. See [4].

[29] Richard Schroeppel, Adi Shamir, *A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems*, SIAM Journal on Computing **10** (1981), 456–464. Citations in this document: §4.

[30] Peter W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring.*, in [**13**] (1994), 124–134; see also newer version [**31**]. MR 1489242.

[31] Peter W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing **26** (1997), 1484–1509; see also older version [**30**]. MR 98i:11108. URL: http://arxiv.org/abs/quant-ph/9508027.

[32] Mark N. Wegman, J. Lawrence Carter, *New classes and applications of hash functions*, in [**1**] (1979), 175–182; see also newer version [**33**].

[33] Mark N. Wegman, J. Lawrence Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Sciences **22** (1981), 265–279; see also older version [**32**]. ISSN 0022-0000. MR 82i:68017. Citations in this document: §3.