Subset Sum Quantumly in 1.17^n

Alexander Helm[1]
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany
alexander.helm@rub.de

Alexander May
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany
alex.may@rub.de

Abstract

We study the quantum complexity of solving the subset sum problem, where the elements $a_1, \ldots, a_n$ are randomly chosen from $\mathbb{Z}_{2^{\ell(n)}}$ and $t = \sum_i a_i \in \mathbb{Z}_{2^{\ell(n)}}$ is a sum of $n/2$ elements. In 2013, Bernstein, Jeffery, Lange and Meurer constructed a quantum subset sum algorithm with heuristic time complexity $2^{0.241n}$, by enhancing the classical subset sum algorithm of Howgrave-Graham and Joux with a quantum random walk technique. We improve on this by defining a quantum random walk for the classical subset sum algorithm of Becker, Coron and Joux. The new algorithm only needs heuristic running time and memory $2^{0.226n}$, for almost all random subset sum instances.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases Subset sum, Quantum walk, Representation technique

Digital Object Identifier 10.4230/LIPIcs.TQC.2018.5

Acknowledgements We are grateful to Stacey Jeffery for many insightful discussions.

1 Introduction

The subset sum (aka knapsack) problem is one of the most famous NP-hard problems. Due to its simpleness, it inspired many cryptographers to build cryptographic systems based on its hardness. In the 80s, many attempts for building secure subset sum based schemes failed [20], often because these schemes were built on subset sum instances $(a_1, \ldots, a_n, t)$ that turned out to be solvable efficiently.

Let $a_1, \ldots, a_n$ be randomly chosen from $\mathbb{Z}_{2^{\ell(n)}}$, $I \subset \{1, \ldots, n\}$ and $t \equiv \sum_{i \in I} a_i \bmod 2^{\ell(n)}$. The quotient $n/\ell(n)$ is usually called the *density* of a subset sum instance. In the *low density case* where $\ell(n) \gg n$, I is with high probability (over the randomness of the instance) a unique solution of the subset sum problem. Since unique solutions are often desirable for cryptographic constructions, most initial construction used low-density subset sums. However, Brickell [8] and Lagarias, Odlyzko [17] showed that low-density subset sums with $\ell(n) > 1.55n$ can be transformed into a lattice shortest vector problem that can be solved in practice in small dimension. This bound was later improved by Coster et al. [9] and Joux, Stern [15] to $\ell(n) > 1.06n$. Notice that this transformation does not rule out the hardness of subset sum in the low-density regime, since computing shortest vectors is in general also known to be NP-hard [2].

[1] Founded by NRW Research Training Group SecHuman.

In the high-density regime with $\ell = \mathcal{O}(\log n)$ dynamic programming solves subset sum efficiently, see [11]. However, for the case $\ell(n) \approx n$ only exponential time algorithms are known. Impagliazzo and Naor showed constructions of cryptographic primitives in AC^0 that can be proven as hard as solving random subset sums around density 1. Many efficient cryptographic constructions followed, see e.g. [18, 10] for some recent constructions – including a CCA-secure subset sum based encryption scheme – and further references.

Classical complexity of subset sum

Let us assume that $\ell = \mathrm{poly}(n)$ such that arithmetic in $\mathbb{Z}_{2^{\ell(n)}}$ can be performed in time $\mathrm{poly}(n)$. Throughout this paper, for ease of notation we omit polynomial factors in exponential running times or space consumptions.

For solving subset sum with $\mathbf{a} = (a_1, \ldots, a_n)$, one can naively enumerate all $\mathbf{e} \in \{0,1\}^n$ and check whether $\langle \mathbf{e}, \mathbf{a} \rangle \equiv t \bmod 2^{\ell(n)}$ in time 2^n.

Let $\mathbf{a}^{(1)} = (a_1, \ldots, a_{n/2})$ and $\mathbf{a}^{(2)} = (a_{n/2+1}, \ldots, a_n)$. In the Meet-in-the-Middle approach of Horowitz and Sahni [13], one enumerates all $\mathbf{e}^{(1)}, \mathbf{e}^{(2)} \in \{0,1\}^{n/2}$ and checks for an identity $\langle \mathbf{e}^{(1)}, \mathbf{a}^{(1)} \rangle \equiv t - \langle \mathbf{e}^{(2)}, \mathbf{a}^{(2)} \rangle \bmod 2^{\ell(n)}$. This improves the time complexity to $2^{n/2}$, albeit using also space $2^{n/2}$.

Schroeppel and Shamir [21] later improved this to time $2^{n/2}$ with only space $2^{n/4}$. It remains an open problem, whether time complexity $2^{n/2}$ can be improved in the worst case [4]. However, when studying the complexity of random subset sum instances in the average case, the algorithm of Howgrave-Graham and Joux [14] runs in time $2^{0.337n}$. This is achieved by representing $\mathbf{e} = \mathbf{e}^{(1)} + \mathbf{e}^{(2)}$ with $\mathbf{e}^{(1)}, \mathbf{e}^{(2)} \in \{0,1\}^n$ ambiguously, also called the representation technique. In 2011, Becker, Coron and Joux [5] showed that the choice $\mathbf{e}^{(1)}, \mathbf{e}^{(2)} \in \{-1,0,1\}^n$ leads to even more representations, which in turn decreases the running time on average case instances to $2^{0.291n}$, the best classical running time currently known.

Quantum complexity of subset sum

In 2013, Bernstein, Jeffery, Lange and Meurer [6] constructed quantum subset sum algorithms, inspired by the classical algorithms above. Namely, Bernstein et al. showed that quantum algorithms for the naive and Meet-in-the-Middle approach achieve run time $2^{n/2}$ and $2^{n/3}$, respectively. Moreover, a first quantum version of Schroeppel-Shamir with Grover search [12] runs in time $2^{3n/8}$ using only space $2^{n/8}$. A second quantum version of Schroeppel-Shamir using quantum walks [1, 3] achieves time $2^{0.3n}$. Eventually, Bernstein, Jeffery, Lange and Meurer used the quantum walk framework of Magniez et al. [19] to achieve a quantum version of the Howgrave-Graham, Joux algorithm with time and space complexity $2^{0.241n}$.

Our result

Interestingly, Bernstein et al. did not provide a quantum version of the best classical algorithm – the BCJ-algorithm by Becker, Coron and Joux [5] – that already classically has some quite tedious analysis. We fill this gap and complete the complexity landscape quantumly, by defining an appropriate quantum walk for the BCJ-algorithm within the framework of Magniez et al. [19]. Our run time analysis relies on some unproven conjecture that we make explicit in Section 4. Under this conjecture, we show that all but a negligible fraction of instances of subset sum can be solved quantumly in time and space $2^{0.226n}$, giving polynomial speedups over the best classical complexity $2^{0.291n}$ and the best quantum complexity $2^{0.241n}$.

In a nutshell, our conjecture states that in the run-time analysis we can replace in a quantum walk an update with expected constant cost by an update with polynomially upper-bounded cost (that might stop), without significantly affecting the error probability and the structure of the random walk graph. While it might be legitimate to use an unproven non-standard conjecture to say something reasonable on the quantum complexity of problems in post-quantum cryptography, especially in the context of the present NIST standardization process, our conjecture is somewhat unsatisfactory from a theoretical point of view. We hope that our work encourages people to base this conjecture on more solid theoretical foundations.

Apart from that our result holds for random subset sums with $\ell = \text{poly}(n)$, i.e. with polynomial density. However, our algorithm behaves worst for subset sum instances with unique solution, i.e. in the case $\ell(n) \geq n$. In the high-density case $\ell(n) < n$, our analysis is non-optimal and might be subject to improvements.

The complexity $2^{0.226n}$ is achieved for subset sum solutions $t \equiv \sum_{i \in I} a_i \bmod 2^{\ell(n)}$ with worst case $|I| = n/2$. We also analyze the complexity for $|I| = \beta n$ with arbitrary $\beta \in [0,1]$. For instance for $\beta = 0.2$, our quantum-BCJ algorithm runs in time and space $2^{0.175n}$.

The paper is organized as follows. Section 2 defines some notation. In Section 3, we repeat the BCJ algorithm and its classical complexity analysis that we later adapt to the quantum case. In Section 4, we analyze the cost of a random walk on the search space defined by the BCJ algorithm and define an appropriate data structure. In Section 5, we put things together and analyze the complexity of the BCJ algorithm, enhanced by a quantum walk technique.

2 Preliminaries

Let $D = \{-1, 0, 1\}$ be a digit set, and let $\alpha, \beta \in \mathbb{Q} \cap [0,1]$ with $2\alpha + \beta \leq 1$. We use the notation $\mathbf{e} \in D^n[\alpha, \beta]$ to denote that $\mathbf{e} \in D^n$ has αn (-1)-entries, $(\alpha+\beta)n$ 1-entries and $(1 - 2\alpha - \beta)n$ 0-entries. Especially, $\mathbf{e} \in D^n[0, \beta]$ is a binary vector with βn 1-entries. Throughout the paper we ignore rounding issues and assume that αn and $(\alpha + \beta)n$ take on integer values.

We naturally extend the binomial coefficient notation $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ to a multinomial coefficient notation

$$\binom{n}{k_1, \ldots, k_r} = \frac{n!}{k_1! \ldots k_r!(n - k_1 - \ldots - k_r)!}.$$

Let $H(x) = -x \cdot \log_2(x) - (1-x) \cdot \log_2(1-x)$ denote the binary entropy function. From Stirling's formula one easily derives

$$\binom{\alpha n}{\beta n} \approx 2^{\alpha \cdot H(\frac{\beta}{\alpha})n},$$

where the $\approx$-notation suppresses polynomial factors.

Analogous, let $g(x,y) := -x \cdot \log_2(x) - y \cdot \log_2(y) - (1-x-y) \cdot \log_2(1-x-y)$. Then

$$\binom{\alpha n}{\beta n, \gamma n} \approx 2^{\alpha \cdot g(\frac{\beta}{\alpha}, \frac{\gamma}{\alpha})n}.$$

Let $\mathbb{Z}_{2^{\ell(n)}}$ be the ring of integers modulo $2^{\ell(n)}$. For the n-dimensional vectors $\mathbf{a} = (a_1, \ldots, a_n) \in (\mathbb{Z}_{2^{\ell(n)}})^n$, $\mathbf{e} = (e_1, \ldots, e_n) \in D^n[\alpha, \beta]$ the inner product is denoted

$$\langle \mathbf{a}, \mathbf{e} \rangle = \sum_{i=1}^{n} a_i e_i \bmod 2^{\ell(n)}.$$

We define a random weight-β (solvable) subset sum instance as follows.

▶ **Definition 1** (Random Subset Sum). Let $\mathbf{a}$ be chosen uniformly at random from $(\mathbb{Z}_{2^{\ell(n)}})^n$. For $\beta \in [0,1]$, choose a random $\mathbf{e} \in D^n[0,\beta]$ and compute $t = \langle \mathbf{a}, \mathbf{e} \rangle \in \mathbb{Z}_{2^{\ell(n)}}$. Then $(\mathbf{a}, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}$ is a *random subset sum instance.* For $(\mathbf{a}, t)$, any $\mathbf{e}' \in \{0,1\}^n$ with $\langle \mathbf{a}, \mathbf{e}' \rangle \equiv t \bmod 2^{\ell(n)}$ is called a *solution.*

3 Subset Sum Classically – The BCJ Algorithm

Let $D = \{-1, 0, 1\}$ and let $(\mathbf{a}, t) = (a_1, \ldots, a_n, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}$ be a subset sum instance with solution $\mathbf{e} \in D^n[0, \frac{1}{2}]$. That is $\langle \mathbf{e}, \mathbf{a} \rangle \equiv t \bmod 2^{\ell(n)}$, where $n/2$ of the coefficients of $\mathbf{e}$ are 1 and $n/2$ coefficients are 0.

Representations

The core idea of the Becker-Coron-Joux (BCJ) algorithm is to represent the solution $\mathbf{e}$ *ambiguously* as a sum

$$\mathbf{e} = \mathbf{e}_1^{(1)} + \mathbf{e}_1^{(2)} \text{ with } \mathbf{e}_1^{(1)}, \mathbf{e}_1^{(2)} \in D^n[\alpha_1, 1/4].$$

This means that we represent $\mathbf{e} \in D^n[0, 1/2]$ as a sum of vectors with $\alpha_1 n$ (-1)-entries, $(1/4 + \alpha_1)n$ 1-entries and $(3/4 - 2\alpha_1)n$ 0-entries. We call $(\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(2)})$ a *representation* of $\mathbf{e}$.

Thus, every 1-coordinate e_i of $\mathbf{e}$ can be represented as either $1 + 0$ or $0 + 1$ via the i^{th}-coordinates of $\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(2)}$. Since we have $n/2$ 1-coordinates in $\mathbf{e}$, we can fix among these $n/4$ 0-coordinates and $n/4$ 1-coordinates in $\mathbf{e}_1^{(1)}$, determining the corresponding entries in $\mathbf{e}_2^{(1)}$. This can be done in $\binom{n/2}{n/4}$ ways.

Analogously, the 0-coordinates in $\mathbf{e}$ can be represented as either $(-1)+1$, $1+(-1)$ or $0+0$. Again, we can fix among these $n/2$ coordinates $\alpha_1 n$ (-1)-coordinates, $\alpha_1 n$ 1-coordinates and $n/2 - 2\alpha_1 n$ 0-coordinates in $\mathbf{e}_1^{(1)}$. This can be done in $\binom{n/2}{\alpha_1 n, \alpha_1 n}$ ways.

Thus, in total we represent our desired solution $\mathbf{e}$ in

$$R_1 = \binom{n/2}{n/4} \binom{n/2}{\alpha_1 n, \alpha_1 n} \text{ ways.}$$

However, notice that constructing a *single representation* of $\mathbf{e}$ is sufficient for solving subset sum. Thus, the main idea of the BCJ algorithm is to compute only a $1/R_1$-fraction of all representations such that on expectation a single representation survives.

This is done by computing only those representations $(\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(2)})$ such that the partial sums

$$\langle \mathbf{e}_1^{(1)}, \mathbf{a} \rangle \text{ and } t - \langle \mathbf{e}_1^{(2)}, \mathbf{a} \rangle$$

attain a fixed value modulo 2^{r_1}, where $r_1 = \lfloor \log R_1 \rfloor$. This value can be chosen randomly, but for simplicity of notation we assume in the following that both partial sums are 0 modulo 2^r.

More precisely, we construct the lists

$$\begin{aligned} L_1^{(1)} &= \{(\mathbf{e}_1^{(1)}, \langle \mathbf{e}_1^{(1)}, \mathbf{a} \rangle) \in D^n[\alpha_1, 1/4] \times \mathbb{Z}_{2^{\ell(n)}} \mid \langle \mathbf{e}_1^{(1)}, \mathbf{a} \rangle \equiv 0 \bmod 2^{r_1}\} \text{ and} \\ L_1^{(2)} &= \{(\mathbf{e}_1^{(2)}, \langle \mathbf{e}_1^{(2)}, \mathbf{a} \rangle) \in D^n[\alpha_1, 1/4] \times \mathbb{Z}_{2^{\ell(n)}} \mid t - \langle \mathbf{e}_1^{(2)}, \mathbf{a} \rangle \equiv 0 \bmod 2^{r_1}\}. \end{aligned}$$

Hence, $L_1^{(1)}, L_1^{(2)}$ have the same expected list length, which we denote shortly by

$$\mathbb{E}[|L_1|] = \frac{\binom{n}{\alpha_1 n, (1/4 + \alpha_1)n}}{2^{r_1}}.$$

Constructing the lists

$L_1^{(1)}, L_1^{(2)}$ are constructed recursively, see also Fig. 1. Let us first explain the construction of $L_1^{(1)}$. We represent $\mathbf{e}_1^{(1)} \in D^n[\alpha_1, 1/4]$ as

$$\mathbf{e}_1^{(1)} = \mathbf{e}_2^{(1)} + \mathbf{e}_2^{(2)} \text{ with } \mathbf{e}_2^{(1)}, \mathbf{e}_2^{(2)} \in D^n[\alpha_2, 1/8], \text{ where } \alpha_2 \geq \alpha_1/2.$$

By the same reasoning as before, the number of representations is

$$R_2 = \binom{\alpha_1 n}{\alpha_1/2n}\binom{(1/4+\alpha_1)n}{(1/8+\alpha_1/2)n}\binom{(3/4-2\alpha_1)n}{(\alpha_2-\alpha_1/2)n,(\alpha_2-\alpha_1/2)n},$$

where the three factors stand for the number of ways of representing (-1)-, 1- and 0-coordinates of $\mathbf{e}_1^{(1)}$. Let $r_2 = \lfloor \log R_2 \rfloor$. We define

$$\begin{aligned} L_2^{(j)} &= \{(\mathbf{e}_2^{(j)}, \langle \mathbf{e}_2^{(j)}, \mathbf{a}\rangle) \in D^n[\alpha_2, 1/8] \times \mathbb{Z}_{2^{\ell(n)}} \mid \langle \mathbf{e}_2^{(j)}, \mathbf{a}\rangle \equiv 0 \bmod 2^{r_2}\} \text{ for } j = 1,2,3, \\ L_2^{(4)} &= \{(\mathbf{e}_2^{(4)}, \langle \mathbf{e}_2^{(4)}, \mathbf{a}\rangle) \in D^n[\alpha_2, 1/8] \times \mathbb{Z}_{2^{\ell(n)}} \mid t - \langle \mathbf{e}_2^{(4)}, \mathbf{a}\rangle \equiv 0 \bmod 2^{r_2}\}. \end{aligned}$$

Thus, we obtain on level 2 of our search tree in Fig. 1 expected list sizes

$$\mathbb{E}[|L_2|] = \frac{\binom{n}{\alpha_2 n, (1/8+\alpha_2)n}}{2^{r_2}}.$$

An analogous recursive construction of level-3 lists $L_3^{(j)}$ from our level-2 lists yields

$$\mathbb{E}[|L_3|] = \frac{\binom{n}{\alpha_3 n, (1/16+\alpha_3)n}}{2^{r_3}},$$

where $r_3 = \lfloor \log R_3 \rfloor$ with

$$R_3 = \binom{\alpha_2 n}{\alpha_2/2n}\binom{(1/8+\alpha_2)n}{(1/16+\alpha_2/2)n}\binom{(7/8-2\alpha_2)n}{(\alpha_3-\alpha_2/2)n,(\alpha_3-\alpha_2/2)n}.$$

The level-3 lists are eventually constructed by a standard Meet-in-the-Middle approach from the following level-4 lists (where we omit the definition of $L_4^{(15)}, L_4^{(16)}$ that is analogous with $\mathbf{t} - \langle \mathbf{e}_4^{(\cdot)}, \mathbf{a}\rangle$)

$$\begin{aligned} L_4^{(2j-1)} &= \{(\mathbf{e}_4^{(2j-1)}, \langle \mathbf{e}_4^{(2j-1)}, \mathbf{a}\rangle) \in D^{n/2}[\alpha_3/2, 1/32] \times 0^{n/2} \times \mathbb{Z}_{2^{\ell(n)}}\} \text{ and} \\ L_4^{(2j)} &= \{(\mathbf{e}_4^{(2j)}, \langle \mathbf{e}_4^{(2j)}, \mathbf{a}\rangle) \in 0^{n/2} \times D^{n/2}[\alpha_3/2, 1/32] \times \mathbb{Z}_{2^{\ell(n)}}\} \text{ for } j = 1, \dots, 7 \end{aligned}$$

of size

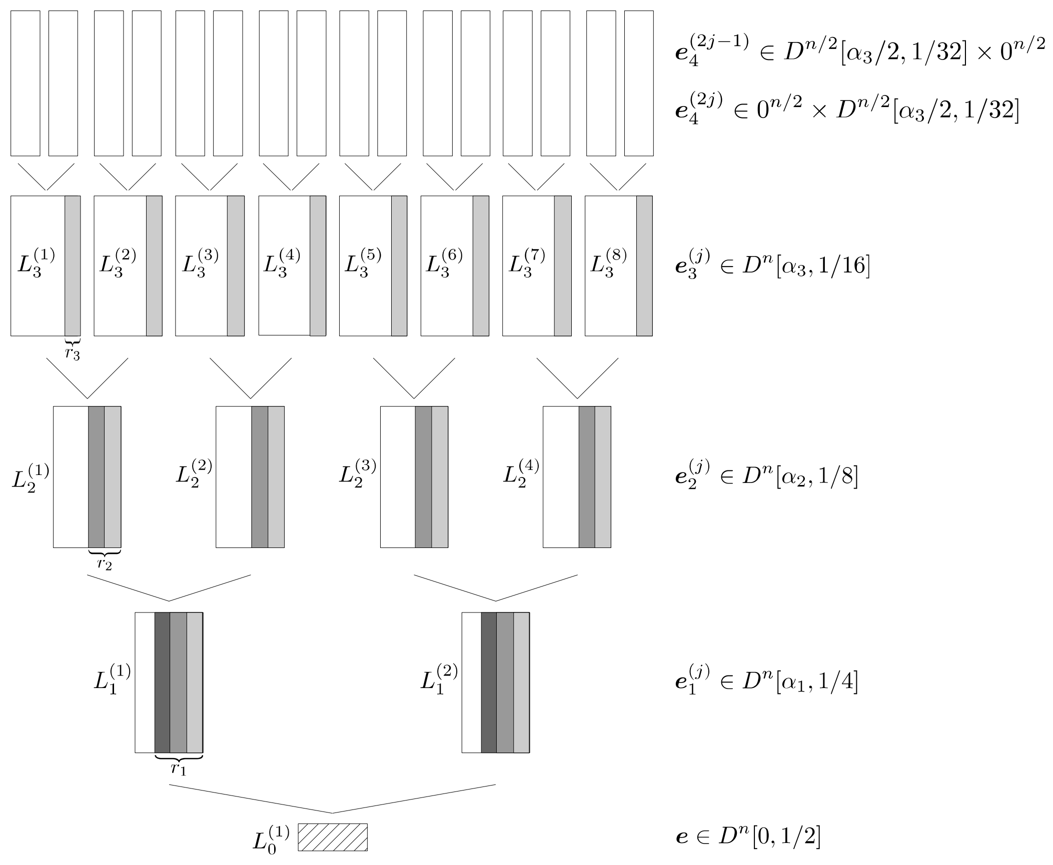
$$|L_4| = \binom{n/2}{(\alpha_3/2)n, (1/32+\alpha_3/2)n}.$$

Let us define indicator variables

$$X_{i,j} = \langle \mathbf{e}_i^{(2j-1)}, \mathbf{a}\rangle \text{ and } X_{i,j}^+ = \langle \mathbf{e}_i^{(2j)}, \mathbf{a}\rangle \text{ for } i = 1,2,3,4 \text{ and } j = 1, \dots, 2^{i-1}.$$

By the randomness of $\mathbf{a}$, we have $\Pr[X_{i,j} = c] = \Pr[X_{i,j}^+ = c] = \frac{1}{2^{\ell(n)}}$ for all $c \in \mathbb{Z}_{2^{\ell(n)}}$. Thus, all $X_{i,j}$, $X_{i,j}^+$ are uniformly distributed in $\mathbb{Z}_{2^{\ell(n)}}$, and therefore also uniformly distributed modulo 2^{r_i} for any $r_i \leq \ell(n)$. Unfortunately, for fixed i,j the pair $X_{i,j}, X_{i,j}^+$ is not independent. We assume in the following that this (mild) dependence does not affect the run time analysis.

▶ **Heuristic 1.** *For the BCJ runtime analysis, we can treat all pairs $X_{i,j}, X_{i,j}^+$ as independent.*



$\boldsymbol{e}_4^{(2j-1)} \in D^{n/2}[\alpha_3/2, 1/32] \times 0^{n/2}$
$\boldsymbol{e}_4^{(2j)} \in 0^{n/2} \times D^{n/2}[\alpha_3/2, 1/32]$
$L_3^{(1)}$ $L_3^{(2)}$ $L_3^{(3)}$ $L_3^{(4)}$ $L_3^{(5)}$ $L_3^{(6)}$ $L_3^{(7)}$ $L_3^{(8)}$
r_3
$\boldsymbol{e}_3^{(j)} \in D^n[\alpha_3, 1/16]$
$L_2^{(1)}$ $L_2^{(2)}$ $L_2^{(3)}$ $L_2^{(4)}$
r_2
$\boldsymbol{e}_2^{(j)} \in D^n[\alpha_2, 1/8]$
$L_1^{(1)}$ $L_1^{(2)}$
r_1
$\boldsymbol{e}_1^{(j)} \in D^n[\alpha_1, 1/4]$
$L_0^{(1)}$
$\boldsymbol{e} \in D^n[0, 1/2]$

Figure 1 Tree structure of the BCJ-Algorithm.

Under Heuristic 1 it can easily be shown that for all but a negligible fraction of random subset sum instances the lists sizes are sharply concentrated around their expectation. More precisely, a standard Chernoff bound shows that for all but a negligible fraction of instances the list size of $L_i^{(j)}$ lies in the interval

$$\mathbb{E}(|L_i|) - \mathbb{E}(|L_i|)^{1/2} \leq |L_i| \leq \mathbb{E}(|L_i|) + \mathbb{E}(|L_i|)^{1/2} \text{ for } i = 1, 2, 3. \tag{1}$$

In other words, for all but some pathological instances we have $|L_i| = \mathcal{O}(\mathbb{E}(|L_i|)$.

We give a description of the BCJ algorithm in Algorithm 1. Here we assume in more generality that a subset sum instance $(\mathbf{a}, t)$ has a solution $\mathbf{e} \in D^n[0, \beta]$. As one would expect, Algorithm 1 achieves its worst-case complexity for $\beta = \frac{1}{2}$ with a balanced number of zeros and ones in $\mathbf{e}$. However, one can also analyze the complexity for arbitrary β, as we will do for our quantum version of BCJ.

For generalizing our description from before to arbitrary β, we have to simply replace $\mathbf{e}_i^{(j)} \in D^n[\alpha_i, \frac{1}{2}2^{-i}]$ by $\mathbf{e}_i^{(j)} \in D^n[\alpha_i, \beta 2^{-i}]$.

By the discussion before, the final condition $|L_0^{(1)}| > 0$ in Algorithm 1 implies that we succeed in constructing a representation $(\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(2)}) \in (D^n[\alpha_1, \beta n/2])^2$ of $\mathbf{e} \in D^n[0, \beta]$, where the $\mathbf{e}_1^{(j)}$ recursively admit representations $(\mathbf{e}_2^{(2j-1)}, \mathbf{e}_2^{(2j-1)}) \in (D^n[\alpha_2, \beta n/4])^2)$, and so forth.

Algorithm 1: Becker-Coron-Joux (BCJ) algorithm.

```
Input      : $(\mathbf{a}, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}, \beta \in [0,1]$
Output     : $\mathbf{e} \in D^n[0, \beta]$
Parameters : Optimize $\alpha_1, \alpha_2, \alpha_3$.
Construct all level-4 lists $L_4^{(j)}$ for $j = 1, \ldots, 16$.
for $i = 3$ down to $0$ do
    Compute $L_i^{(j)}$ from $L_{i+1}^{(2j-1)}$, $L_{i+1}^{(2j)}$ for $j = 1, \ldots, 2^i$.
end
if $|L_0^{(1)}| > 0$ then output an arbitrary element from $L_0^{(1)}$.
```

Thus, one can eventually express

$$\mathbf{e} = \mathbf{e}_4^{(1)} + \mathbf{e}_4^{(2)} + \ldots + \mathbf{e}_4^{(16)}.$$

However, notice that we constructed all lists in such a way that on expectation at least one representation survives for every list $L_i^{(j)}$ from the for-loop of Algorithm 1. This implies that the BCJ algorithm succeeds in finding the desired solution $\mathbf{e}$, and therefore the leaves of our search tree in Fig. 1 contain elements that sum up to $\mathbf{e}$. The following theorem and its proof show how to optimize the parameters α_i, $i = 1, 2, 3$ such that BCJ's running time is minimized while still guaranteeing a solution.

▶ **Theorem 2** (BCJ 2011). *Under Heuristic 1 Algorithm 1 solves all but a negligible fraction of random subset sum instances* $(\mathbf{a}, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}$ *(Definition 1) in time and memory* $2^{0.291n}$.

Proof. Numerical optimization yields the parameters

$$\alpha_1 = 0.0267, \ \alpha_2 = 0.0302, \ \alpha_3 = 0.0180.$$

This leads to

$$R_3 = 2^{0.241n}, \ R_2 = 2^{0.532n}, \ R_1 = 2^{0.799n} \text{ representations,}$$

which in turn yield expected list sizes

$$|L_4| = 2^{0.266n}, \ \mathbb{E}(|L_3|) = 2^{0.2909n}, \ \mathbb{E}(|L_2|) = 2^{0.279n}, \ \mathbb{E}(|L_1|) = 2^{0.217n}, \ \mathbb{E}(|L_0|) = 1.$$

For $i = 1, 2, 3$ the level-i lists $L_i^{(j)}$ can be constructed in time $2^{0.2909n}$ by looking at all pairs in $L_{i-1}^{(2j-1)} \times L_{i-1}^{(2j)}$. Under Heuristic 1, we conclude by Eq. (1) that for all but a negligible fraction of instance we have $|L_i| = \mathcal{O}(\mathbb{E}(|L_i|)$ for $i = 1, 2, 3$. Thus, the total running time and memory complexity can be bounded by $2^{0.291n}$. ◀

4 From Trees to Random Walks to Quantum Walks

In Section 3, we showed how the BCJ algorithm builds a search tree t whose root contains a solution $\mathbf{e}$ to the subset sum problem. More precisely, the analysis of the BCJ algorithm in the proof of Theorem 2 shows that the leaves of t contain a representation $(\mathbf{e}_4^{(1)}, \ldots, \mathbf{e}_4^{(16)}) \in L_4^{(1)} \times \ldots \times L_4^{(16)}$ of $\mathbf{e}$, i.e. $\mathbf{e} = \mathbf{e}_4^{(1)} + \ldots + \mathbf{e}_4^{(16)}$.

Idea of Random Walk

In a random walk, we no longer enumerate the lists $L_4^{(j)}$ completely, but only a random subset $U_4^{(j)} \subseteq L_4^{(j)}$ of some fixed size $|U_4| := |U_4^{(j)}|$, that has to be optimized. We run on these projected leaves the original BCJ algorithm, but with parameters $\alpha_1, \alpha_2, \alpha_3$ that have to be optimized anew. On the one hand, a small $|U_4|$ yields small list sizes, which in turn speeds up the BCJ algorithm. On the other hand, a small $|U_4|$ reduces the probability that BCJ succeeds. Namely, BCJ outputs the desired solution $\mathbf{e}$ iff $(\mathbf{e}_4^{(1)}, \ldots, \mathbf{e}_4^{(16)}) \in U_4^{(1)} \times \ldots \times U_4^{(16)}$, which happens with probability

$$\epsilon = \left(\frac{|U_4|}{|L_4|}\right)^{16}. \tag{2}$$

The graph $G = (V, E)$ of our Random Walk

We define vertices V with labels $U_4^{(1)} \times \ldots \times U_4^{(16)}$. Each vertex $v \in V$ contains the complete BCJ search tree with leaf lists defined by its label. Two vertices with labels $\ell = U_4^{(1)} \times \ldots \times U_4^{(16)}$ and $\ell' = V_4^{(1)} \times \ldots \times V_4^{(16)}$ are adjacent iff their symmetric difference is $|\Delta(\ell, \ell')| = 1$. I.e., we have $U_4^{(j)} = V_4^{(j)}$ for all j but one $V_4^{(i)} \neq V_4^{(i)}$ for which $U_4^{(i)}, V_4^{(i)}$ differ by only one element.

▶ **Definition 3** (Johnson graph)**.** Given an N-size set L the Johnson graph $J(N, r)$ is an undirected graph $G_J = (V_J, E_J)$ with vertices labeled by all r-size subsets of L. An edge between two vertices $v, v' \in V_J$ with labels ℓ, ℓ' exists iff $|\Delta(\ell, \ell')| = 1$.

In our case, we define $N = |L_4|, r = |U_4|$ and for each of our 16 lists $L_4^{(j)}$ its corresponding Johnson graph $J_j(N, r)$. However, by our construction above we want that two vertices are adjacent iff they differ in only one element throughout all 16 lists.

Let us therefore first define the Cartesian product of graphs. We will then show that our graph $G = (V, E)$ is exactly the Cartesian product

$$J^{16}(N, r) := J_1(N, r) \times \ldots \times J_{16}(N, r).$$

▶ **Definition 4.** Let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be undirected graphs. The Cartesian product $G_1 \times G_2 = (V, E)$ is defined via

$$\begin{aligned} V &= V_1 \times V_2 = \{v_1 v_2 \mid v_1 \in V_1,\ v_2 \in V_2\} \text{ and} \\ E &= \{(u_1 u_2, v_1 v_2) \mid (u_1 = v_1 \wedge (u_2, v_2) \in E_2) \vee ((u_1, v_1) \in E_1 \wedge u_2 = v_2)\} \end{aligned}$$

Thus, in $J_1(n, r) \times J_2(n, r)$ the labels $v_1 v_2$ are Cartesian products of the labels $U_4^{(1)}, U_4^{(2)}$. An edge in $J_1(n, r) \times J_2(n, r)$ is set between two vertices with labels $U_4^{(1)} \times U_4^{(2)}$, $V_4^{(1)} \times V_4^{(2)}$ iff $U_4^{(1)} = V_4^{(1)}$ and $U_4^{(2)}$, $V_4^{(2)}$ differ by exactly one element or vice versa, as desired.

Mixing time

The mixing time of a random walk depends on its so-called spectral gap.

▶ **Definition 5** (Spectral gap)**.** Let G be an undirected graph. Let λ_1, λ_2 be the eigenvalues with largest absolute value of the transition matrix of the random walk on G. Then the *spectral gap* of a random walk on G is defined as $\delta(G) := |\lambda_1| - |\lambda_2|$.

For Johnson graphs it is well-known that $\delta(J(N,r)) = \frac{N}{r(N-r)} = \Omega(\frac{1}{r})$. The following lemma shows that for our graph $J^{16}(N,r)$ we have as well

$$\delta(J^{16}(N,r)) = \Omega\left(\frac{1}{r}\right) = \Omega\left(\frac{1}{|U_4|}\right). \tag{3}$$

▶ **Lemma 6** (Kachigar, Tillich [16])**.** *Let $J(N,r)$ be a Johnson graph, and let $J^m(N,r) := \bigtimes_{i=1}^{m} J(n,r)$. Then $\delta(J^m) \geq \frac{1}{m}\delta(J)$.*

Walking on G

We start our random walk on a random vertex $v \in V$, i.e. we choose random $U_4^{(j)} \subseteq L_4^{(j)}$ for $j = 1, \ldots, 16$ and compute the corresponding BCJ tree t_v on these sets. This computation of the starting vertex v defines the *setup cost* T_S of our random walk.

Let us quickly compute T_S for the BCJ algorithm, neglecting all polynomial factors. The level-4 lists $U_4^{(j)}$ can be computed and sorted with respect to the inner products $\langle \mathbf{e}_4^{(j)}, \mathbf{a}\rangle \bmod 2^{r_3}$ in time $|U_4|$. The level-3 lists contain all elements from their two level-4 children lists that match on the inner products. Thus we expect $\mathbb{E}(|U_3|) = |U_4|^2/2^{r_3}$ elements that match on their inner products. Analogous, we compute level-2 lists in expected time $|U_3|^2/2^{r_2-r_3}$. However, now we have to filter out all $\mathbf{e}_2^{(j)}$ that do not possess the correct weight distribution, i.e. the desired number of (-1)s, 0s, and 1s. Let us call any level-i $\mathbf{e}_i^{(j)}$ *consistent* if $\mathbf{e}_i^{(j)}$ has the correct weight distribution on level i. Let $p_{3,2}$ denote the probability that a level-2 vector constructed as a sum of two level-3 vectors is consistent. From Section 3 we have

$$\frac{|L_3|^2}{2^{r_2-r_3}} \cdot p_{3,2} = \mathbb{E}(|L_2|),$$

which implies

$$p_{3,2} := \frac{\binom{n}{\alpha_2 n, (1/8+\alpha_2)n}}{\binom{n}{\alpha_3 n, (1/16+\alpha_3)n}^2} \cdot 2^{r_2-r_3}.$$

Thus, after filtering for the correct weight distribution we obtain an expected level-2 list size of $\mathbb{E}(|U_2|) = |U_3|^2/2^{r_2-r_3} \cdot p_{3,2}$. Analogous, on level 1 we obtain expected list size $\mathbb{E}(|U_1|) = |U_2|^2/2^{r_1-r_2} \cdot p_{2,1}$ with

$$p_{2,1} := \frac{\binom{n}{\alpha_1 n, (1/4+\alpha_1)n}}{\binom{n}{\alpha_2 n, (1/8+\alpha_2)n}^2} \cdot 2^{r_1-r_2}.$$

The level-0 list can be computed in expected time $|U_1|^2/2^{n-r_1}$. In total we obtain

$$\mathbb{E}[T_S] = \max\left\{|U_4|, \frac{|U_4|^2}{2^{r_3}}, \frac{|U_3|^2}{2^{r_2-r_3}}, \frac{|U_2|^2}{2^{r_1-r_2}}, \frac{|U_1|^2}{2^{n-r_1}}\right\}$$

Analogous to the reasoning in Section 3 (see Eq. 1), for all but a negligible fraction of random subset sum instances we have $|U_i| = \mathcal{O}(\mathbb{E}(|U_i|))$. Thus, for all but a negligible fraction of instances and neglecting constants we have

$$\begin{aligned} T_S = \quad & \max\left\{|U_4|, \tfrac{|U_4|^2}{2^{r_3}}, \tfrac{\mathbb{E}(|U_3|)^2}{2^{r_2-r_3}}, \tfrac{\mathbb{E}(|U_2|)^2}{2^{r_1-r_2}}, \tfrac{\mathbb{E}(|U_1|)^2}{2^{n-r_1}}\right\} & (4)\\ \leq \quad & \max\left\{|U_4|, \tfrac{|U_4|^2}{2^{r_3}}, \tfrac{|U_4|^4}{2^{r_2+r_3}}, \tfrac{|U_4|^8}{2^{r_1+r_2+2r_3}}, \tfrac{|U_4|^{16}}{2^{n+r_1+2r_2+4r_3}}\right\} := \tilde{T}_S. & (5) \end{aligned}$$

If t_v contains a non-empty root with subset-sum solution $\mathbf{e}$, we denote v *marked*. Hence, we walk our graph $G = J_1(|L_4|, |U_4|) \times \ldots \times J_{16}(|L_4|, |U_4|)$ until we hit a marked vertex, which solves subset sum.

The cost for checking whether a vertex v is marked is denoted *checking cost* T_C. In our case checking can be done easily by looking at t_v's root. Thus, we obtain (neglecting polynomials)

$$T_C = 1. \tag{6}$$

Since any neighboring vertices v, v' in G only differ by one element in some leaf $U_4^{(j)}$, when walking from v to v' we do not have to compute the whole tree $t_{v'}$ anew, but instead we update t_v to $t_{v'}$ by changing the nodes on the path from list $U_4^{(j)}$ to its root accordingly. The cost of this step is therefore called *update cost* T_U. Our cost T_U heavily depends on the way we internally represent t_v. In the following, we define a data structure that allows for optimal *update cost* per operation.

4.1 Data Structure for Updates

Let us assume that we have a data structure that allows the three operations search, insertion and deletion in time logarithmic in the number of stored elements. In Bernstein et al. [7], it is e.g. suggested to use radix trees. Since our lists have exponential size and we ignore polynomials in the run time analysis, every operation has cost 1. This data structure also ensures the uniqueness of quantum states $|U_4^{(1)}, \ldots, U_4^{(16)}\rangle$, which in turn guarantees correct interference of quantum states with identical lists.

Definition of data structure

Recall from Section 3, that BCJ level-4 lists are of the form $L_4^{(j)} = \{(\mathbf{e}_4^{(j)}, \langle \mathbf{e}_4^{(j)}, \mathbf{a}\rangle)\}$. For our $U_4^{(j)} \subset L_4^{(j)}$ we store in our data structure the $\mathbf{e}_4^{(j)}$ and their inner products with $\mathbf{a}$ separately in

$$E_4^{(j)} = \{\mathbf{e}_4^{(j)} \mid \mathbf{e}_4^{(j)} \in U_4^{(j)}\} \text{ and } S_4^{(j)} = \{(\langle \mathbf{e}_4^{(j)}, \mathbf{a}\rangle, \mathbf{e}_4^{(j)}) \mid \mathbf{e}_4^{(j)} \in U_4^{(j)}\}, \tag{7}$$

where in $S_4^{(j)}$ elements are addressed via their first datum $\langle \mathbf{e}_4^{(j)}, \mathbf{a}\rangle$. Analogous, for $U_i^{(j)}$, $i = 3, 2, 1$ we also build separate $E_i^{(j)}$ and $S_i^{(j)}$. For the root list $U_0^{(1)}$, it suffices to build $E_0^{(1)}$.

We denote the operations on our data structure as follows. Insert$(E_i^{(j)}, \mathbf{e})$ inserts $\mathbf{e}$ into $E_i^{(j)}$, whereas Delete$(E_i^{(j)}, \mathbf{e})$ deletes one entry $\mathbf{e}$ from $E_i^{(j)}$. Furthermore, $\{\mathbf{e}_i\} \leftarrow$ Search$(S_i^{(j)}, \langle \mathbf{e}_i^{(j)}, \mathbf{a}\rangle)$ returns the list of all $\mathbf{e}_i$ with first datum $\langle \mathbf{e}_i^{(j)}, \mathbf{a}\rangle$.

Deletion/Insertion of an element

Our random walk replaces a list element in exactly one of the leaf lists $U_4^{(j)}$. We can perform the update by first deleting the replaced element and update the path to the root accordingly, and second adding the new element and again updating the path to the root.

Let us look more closely at the deletion process. On every level we delete a value, and then compute via the sibbling vertex, which values we have to be deleted recursively on the parent level. For illustration, deletion of $\mathbf{e}$ in $U_4^{(3)}$ triggers the following actions.

```
- Delete $(E_4^{(3)}, \mathbf{e})$.
- $\{\mathbf{e}_4^{(4)}\} \leftarrow \text{Search}(S_4^{(4)}, \langle \mathbf{e}, \mathbf{a}\rangle \bmod 2^{r_3})$        // $\mathbb{E}(|\{\mathbf{e}_4^{(4)}\}|) = \frac{|U_4|}{2^{r_3}}$
- For all $\mathbf{e}_3^{(2)} = \mathbf{e} + \mathbf{e}'$ with $\mathbf{e}' \in \{\mathbf{e}_4^{(4)}\}$
    - Delete $(E_3^{(2)}, \mathbf{e}_3^{(2)})$
    - $\{\mathbf{e}_3^{(1)}\} \leftarrow \text{Search}(S_3^{(1)}, \langle \mathbf{e}_3^{(2)}, \mathbf{a}\rangle \bmod 2^{r_2})$        // $\mathbb{E}(|\{\mathbf{e}_3^{(1)}\}|) = \frac{|U_3|}{2^{r_2-r_3}}$
    - For all $\mathbf{e}_2^{(1)} = \mathbf{e}_3^{(2)} + \mathbf{e}'$ with $\mathbf{e}' \in \{\mathbf{e}_3^{(1)}\}$
        * Delete $(E_2^{(1)}, \mathbf{e}_2^{(1)})$
        * $\{\mathbf{e}_2^{(2)}\} \leftarrow \text{Search}(S_2^{(2)}, \langle \mathbf{e}_2^{(1)}, \mathbf{a}\rangle \bmod 2^{r_1})$        // $\mathbb{E}(|\{\mathbf{e}_2^{(2)}\}|) = \frac{|U_2|}{2^{r_1-r_2}}$
        * For all $\mathbf{e}_1^{(1)} = \mathbf{e}_2^{(1)} + \mathbf{e}'$ with $\mathbf{e}' \in \{\mathbf{e}_2^{(2)}\}$
            · Delete $(E_1^{(1)}, \mathbf{e}_1^{(1)})$.
            · $\{\mathbf{e}_1^{(2)}\} \leftarrow \text{Search}(S_1^{(2)}, \langle \mathbf{e}_1^{(1)}, \mathbf{a}\rangle \bmod 2^{n})$        // $\mathbb{E}(|\{\mathbf{e}_1^{(2)}\}|) = \frac{|U_1|}{2^{n-r_1}}$
            · For all $\mathbf{e}_0^{(1)} = \mathbf{e}_1^{(1)} + \mathbf{e}'$ with $\mathbf{e}' \in \{\mathbf{e}_1^{(2)}\}$
              o Delete $(E_0^{(1)}, \mathbf{e}_0^{(1)})$.
```

Insertion of an element is analogous to deletion. Hence, the expected *update cost* is

$$\mathbb{E}(T_U) = \max\left\{1, \frac{|U_4|}{2^{r_3}}, \frac{|U_4|\mathbb{E}(|U_3|)}{2^{r_2}}, \frac{|U_4|\mathbb{E}(|U_3|)\mathbb{E}(|U_2|)}{2^{r_1}}, \frac{|U_4|\mathbb{E}(|U_3|)\mathbb{E}(|U_2|)\mathbb{E}(|U_1|)}{2^{n}}\right\} \quad (8)$$

$$\leq \max\left\{1, \frac{|U_4|}{2^{r_3}}, \frac{|U_4|^3}{2^{r_2+r_3}}, \frac{|U_4|^7}{2^{r_1+r_2+2r_3}}, \frac{|U_4|^{15}}{2^{n}}\right\} := \tilde{T}_U. \quad (9)$$

Notice that for the upper bounds $\tilde{T}_S$, $\tilde{T}_U$ from Eq. (5) and (9) we have

$$\tilde{T}_S = |U_4| \cdot \tilde{T}_U. \quad (10)$$

Quantum Walk Framework

While random walks take time $T = T_S + \frac{1}{\epsilon}\left(T_C + \frac{1}{\delta}T_U\right)$, their quantum counterparts achieve some significant speedup due to their rapid mixing, as summarized in the following theorem.

▶ **Theorem 7** (Magniez et al. [19]). *Let $G = (V, E)$ be a regular graph with eigenvalue 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 T_S, T_U, T_C be the setup, update and checking cost. Then there exists a quantum algorithm that with high probability finds a marked vertex in time*

$$T = T_S + \frac{1}{\sqrt{\epsilon}}\left(T_C + \frac{1}{\sqrt{\delta}}T_U\right).$$

Stopping unusually long updates

Recall that for setup, we showed that all instances but an exponentially small fraction finish the construction of the desired data structure in time T_S. However, the update cost is determined by the maximum cost over all *superexponentially many* vertices in a superposition. So even one node with unusually slow update may ruin our run time.

Therefore, we modify our quantum walk algorithm QW by imposing an upper bound of $\kappa = \text{poly}(n)$ steps for the update. After κ steps, we simply stop the update of all nodes and proceed as if the update has been completed. We denote by Stop-QW the resulting algorithm.

A first drawback of stopping is that some nodes that would get marked in QW, might stay unmarked in Stop-QW. However, since the event of stopping should not dependent on whether a node is marked or not, the ratio between marked and unmarked nodes and thus the success probability ϵ should not change significantly between QW and STOP-QW. Moreover, under Heuristic 1 and a standard Chernoff argument the probability of a node not finishing his update properly after κ steps is exponentially small.

A second drawback of stopping is that unfinished nodes partially destroy the structure of the Johnson graph, since different (truncated) representations of the same node do no longer interfere properly in a quantum superposition. We conjecture that this only mildly affects the spectral gap of the graph. A possible direction to prove such a conjecture might be to allow some kind of *self-repairing process* for a node. If a node cannot finish its update in time in one step, it might postpone the remaining work to subsequent steps to amortize the cost of especially expensive updates. After the repair work, a node then again joins the correct Johnson graph data structure in quantum superposition.

In the following heuristic, we assume that the change from QW to STOP-QW changes the success probability ϵ and the bound δ for the spectral gap only by a polynomial factor. This in turn allows us to analyze STOP-QW with the known parameters ϵ, δ from QW.

▶ **Heuristic 2.** *Let ϵ be the fraction of marked states and δ be the spectral gap of the random walk in* QW*. Then the fraction of marked states in* STOP-QW *is at least $\epsilon_{stop} = \frac{\epsilon}{poly(n)}$, and the spectral gap of the random walk on the graph in* STOPQW *is at least $\delta_{stop} = \frac{\delta}{poly(n)}$. Moreover, the stationary distribution of* STOP-QW *is close to the distribution of its setup. Namely, we obtain with high probability a random node of the Johnson graph with correctly built data structure.*

With the upcoming NIST standardization for post-quantum cryptography, there is an even stronger need to analyze quantum algorithms for cryptographic problems. There is a strong need to provide more solid theoretical foundations that justify assumptions like Heuristic 2, since cryptographic parameter selections will be based on best quantum attacks. Hence, any progress in proving Heuristic 2 finds a broad spectrum of applications in the cryptographic community.

5 Results

In this section, we describe the BCJ algorithm enhanced by a quantum random walk, see Algorithm 2. Our following main theorem shows the correctness of our quantum version of the BCJ algorithm and how to optimize the parameters for achieving the stated complexity.

▶ **Theorem 8** (BCJ-QW Algorithm)**.** *Under Heuristic 1 and Heuristic 2, Algorithm 2 solves with high probability all but a negligible fraction of random subset sum instances $(\mathbf{a}, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}$ (as defined in Definition 1) in time and memory $2^{0.226n}$.*

Proof. By Theorem 7, the running time T of Algorithm 2 can be expressed as

$$T = T_S + \frac{1}{\sqrt{\epsilon_{stop}}} \left(T_C + \frac{1}{\sqrt{\delta_{stop}}} T_U \right).$$

We recall from Heuristic 2, Eq. (2), (3) and (6)

$$\epsilon_{stop} \approx \epsilon = \left(\frac{|U_4|}{|L_4|} \right)^{16}, \ \delta_{stop} \approx \delta = \Omega\left(\frac{1}{|U_4|} \right) \text{ and } T_C = 1,$$

where the $\approx$-notation suppresses polynomial factors.

Algorithm 2: BCJ-QW ALGORITHM.

```
Input       : $(\mathbf{a}, t) \in (\mathbb{Z}_{2^{\ell(n)}})^{n+1}, \beta \in [0,1]$
Output      : $\mathbf{e} \in D^n[0,\beta]$
Parameters  : Optimize $\alpha_1, \alpha_2, \alpha_3$.
Construct all level-4 lists $E_4^{(j)}$ and $S_4^{(j)}$ for $j = 1, \ldots, 16$.    ▷ SETUP (see Eq. (7))
Construct all level-3 lists $E_3^{(j)}$ and $S_3^{(j)}$ for $j = 1, \ldots, 8$.
Construct all level-2 lists $E_2^{(j)}$ and $S_2^{(j)}$ for $j = 1, \ldots, 4$.
Construct all level-1 lists $E_1^{(j)}$ and $S_1^{(j)}$ for $j = 1, 2$.
Construct level-0 list $E_0$.
while $E_0 \neq \emptyset$ do                                                       ▷ CHECK
    for $1/\sqrt{\delta}$ times (via phase estimation) do
        Take a quantum step of the walk.                                             ▷ UPDATE
        Update the data structure accordingly, stop after $\kappa = \mathrm{poly}(n)$ steps.
    end
end
Output $\mathbf{e} \in E_0$.
```

Let us first find an optimal size of $|U_4|$. Plugging ϵ, δ and T_C into T and neglecting constants yields run time

$$T = T_S + |L_4|^8 |U_4|^{-15/2} T_U.$$

Let us substitute T_U by its expectation $\mathbb{E}[T_U]$. We later show that T_U and $\mathbb{E}[T_U]$ differ by only a polynomial factor, and thus do not change the analysis. We can upper bound the right hand side using our bounds $\tilde{T}_S \geq T_S$, $\tilde{T}_U \geq \mathbb{E}[T_U]$ from Eq. (5) and (9). We minimize the resulting term by equating both summands

$$\tilde{T}_S = |L_4|^8 |U_4|^{-15/2} \tilde{T}_U.$$

Using the relation $\tilde{T}_S = |U_4| \cdot \tilde{T}_U$ from Eq. (10) results in

$$|U_4| = |L_4|^{16/17}.$$

Therefore, $|L_4|^8 |U_4|^{-15/2} \cdot \mathbb{E}[T_U] = |U_4| \cdot \mathbb{E}[T_U]$. Thus for minimizing the runtime T of Algorithm 2, we have to minimize the term $\max\{T_S, |U_4| \cdot \mathbb{E}[T_U]\}$, which equals T up to a factor of at most 2. Recall from Eq. (4), which holds under Heuristic 1 and for all but a negligible fraction of instances, and Eq. (8) that

$$\begin{aligned} T_S &= \max\left\{ |U_4|, \frac{|U_4|^2}{2^{r_3}}, \frac{\mathbb{E}(|U_3|)^2}{2^{r_2 - r_3}}, \frac{\mathbb{E}(|U_2|)^2}{2^{r_1 - r_2}}, \frac{\mathbb{E}(|U_1|)^2}{2^{n - r_1}} \right\}, \\ \mathbb{E}[T_U] &= \max\left\{ 1, \frac{|U_4|}{2^{r_3}}, \frac{|U_4|\mathbb{E}(|U_3|)}{2^{r_2}}, \frac{|U_4|\mathbb{E}(|U_3|)\mathbb{E}(|U_2|)}{2^{r_1}}, \frac{|U_4|\mathbb{E}(|U_3|)\mathbb{E}(|U_2|)\mathbb{E}(|U_1|)}{2^{n}} \right\}. \end{aligned}$$

Numerical optimization for minimizing $\max\{T_S, |U_4| \cdot \mathbb{E}[T_U]\}$ leads to parameters

$$\alpha_1 = 0.0120, \;\; \alpha_2 = 0.0181, \;\; \alpha_3 = 0.0125.$$

This gives

$$2^{r_3} = 2^{0.2259n}, \; 2^{r_2} = 2^{0.4518n}, \; 2^{r_1} = 2^{0.6627n} \text{ representations,}$$

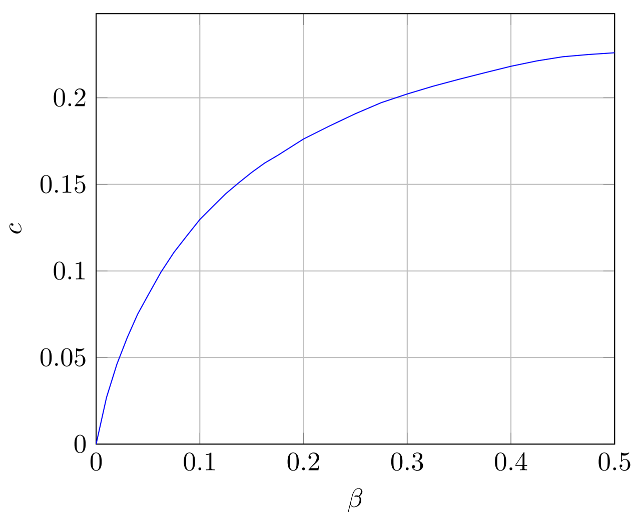


Figure 2 $c = \frac{\log T}{n}$ as a function of β for BCJ-QW.

which in turn yield expected list sizes

$$|U_4| = 2^{0.2259n},\ \mathbb{E}(|U_3|) = 2^{0.2259n},\ \mathbb{E}(|U_2|) = 2^{0.2109n},\ \mathbb{E}(|U_1|) = 2^{0.1424n}.$$

Plugging these values into our formulas for T_S, $\mathbb{E}[T_U]$ gives

$$\begin{aligned} T_S &= \max\{2^{0.2259n}, 2^{0.2259n}, 2^{0.2259n}, 2^{0.2109n}, 2^{-0.0524n}\} \text{ and} \\ |U_4| \cdot \mathbb{E}[T_U] &= \max\{2^{0.2259n}, 2^{0.2259n}, 2^{0.2259n}, 2^{0.2259n}, 2^{0.0310n}\}. \end{aligned}$$

It follows that $\mathbb{E}[T_U] = 1$. Since we have $T_U \leq \kappa = \text{poly}(n)$ by definition in Algorithm 2, the values T_U and $\mathbb{E}[T_U]$ differ by only a polynomial factor that we can safely ignore (by rounding up the runtime exponent). Thus, we conclude that Algorithm 2 runs in time $T = 2^{0.226n}$ using $|U_4| = 2^{0.226n}$ memory. ◀

▶ Remark. As in the classical BCJ case, a tree depth of 4 seems to be optimal for BCJ-QW. When analyzing varying depths, we could not improve over the run time from Theorem 8.

Complexity for the unbalanced case

We also analyzed subset sum instances with $t = \sum_{i \in I} a_i$, where $|I| = \beta n$ for arbitrary $\beta \in [0, 1]$. Notice that w.l.o.g. we can assume $\beta \leq 1/2$, since for $\beta > 1/2$ we can solve a subset sum instance with target $t' = \sum_{i=1}^{n} a_i - t$. Hence, the complexity graph is symmetric around $\beta = 1/2$. Fig. 2 shows the run time exponent c for our BCJ-QW algorithm with time $T = 2^{cn}$ as a function of β.

References

1 Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 50–59. ACM, 2001.

2 Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19. ACM, 1998.

3 Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007. doi:10.1137/S0097539705447311.

4 Per Austrin, Mikko Koivisto, Petteri Kaski, and Jesper Nederlof. Dense subset sum may be the hardest. *arXiv preprint arXiv:1508.06019*, 2015.

5 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 364–385. Springer, 2011.

6 Daniel J Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *International Workshop on Post-Quantum Cryptography*, pages 16–33. Springer, 2013.

7 Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. *Quantum Algorithms for the Subset-Sum Problem*, pages 16–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-38616-9_2.

8 Ernest F Brickell. Solving low density knapsacks. In *Advances in Cryptology*, pages 25–37. Springer, 1984.

9 Matthijs J Coster, Brian A LaMacchia, Andrew M Odlyzko, and Claus P Schnorr. An improved low-density subset sum algorithm. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 54–67. Springer, 1991.

10 Sebastian Faust, Daniel Masny, and Daniele Venturi. Chosen-ciphertext security from subset sum. In *Public-Key Cryptography–PKC 2016*, pages 35–46. Springer, 2016.

11 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM Journal on Computing*, 20(6):1157–1189, 1991.

12 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

13 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

14 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.

15 Antoine Joux and Jacques Stern. Improving the critical density of the lagarias-odlyzko attack against subset sum problems. In *International Symposium on Fundamentals of Computation Theory*, pages 258–264. Springer, 1991.

16 Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. *CoRR*, abs/1703.00263, 2017. URL: http://arxiv.org/abs/1703.00263, arXiv:1703.00263.

17 Jeffrey C Lagarias and Andrew M Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, 1985.

18 Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *Theory of Cryptography Conference*, pages 382–400. Springer, 2010.

19 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

20 Andrew M Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and computational number theory*, 42:75–88, 1990.

21 Richard Schroeppel and Adi Shamir. A T=O(2ˆn/2). *SIAM journal on Computing*, 10(3):456–464, 1981.