Concrete Analysis of Quantum Lattice Enumeration*

Shi Bai[1][0000−0002−0746−3054], Maya-Iggy van Hoof[2], Floyd B. Johnson[1], Tanja Lange[3], and Tran Ngo[1][0000−0001−9157−7822]

[1] Florida Atlantic University, United States.
shih.bai@gmail.com, johnsonf2017@fau.edu, ngotbtran@gmail.com
[2] Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany.
iggy.hoof@rub.de
[3] Eindhoven University of Technology, the Netherlands.
tanja@hyperelliptic.org

Abstract. Lattice reduction algorithms such as BKZ (Block-Korkine-Zolotarev) play a central role in estimating the security of lattice-based cryptography. The subroutine in BKZ which finds the shortest vector in a projected sublattice can be instantiated with enumeration algorithms. The enumeration procedure can be seen as a depth-first search on some "enumeration tree" whose nodes denote a partial assignment of the coefficients, corresponding to lattice points as a linear combination of the lattice basis with the coefficients. This work provides a concrete analysis for the cost of quantum lattice enumeration based on Montanaro's quantum tree backtracking algorithm. More precisely, we give a concrete implementation in the quantum circuit model. We also show how to optimize the circuit depth by parallelizing the components. Based on the circuit designed, we discuss the concrete quantum resource estimates required for lattice enumeration.

Keywords: Lattices · Quantum algorithms · Enumeration · Quantum backtracking.

* Author list in alphabetical order; see https://ams.org/profession/leaders/CultureStatement04.pdf. This research was funded in part by the U.S. National Science Foundation under Grant No. 2044855 & 2122229, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy–EXC 2092 CASA–390781972 "Cyber Security in the Age of Large-Scale Adversaries", and the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP). This work was done in part while Tanja Lange was visiting the Simons Institute for the Theory of Computing and in part when she was with Academia Sinica, Taiwan.

1 Introduction

A Euclidean lattice is the set of all integral linear combinations of n linearly independent basis vectors $\mathbf{b}_1, \cdots, \mathbf{b}_n \in \mathbb{Q}^m$. Lattices have attracted considerable interest in recent years as they can be used to construct cryptographic schemes which are conjectured to be quantum-resistant. The benefits of using lattices are reflected by three out of the four selected schemes (Kyber [ABD+21], Dilithium [BDK+21] and Falcon [FHK+20]) in NIST's Post-Quantum Cryptography Standardization process [NIS16] having their security rely on the presumed intractability of lattice problems.

A fundamental computational problem in lattice-based cryptography is the approximated shortest vector problem, denoted SVP_γ where $\gamma \geq 1$ is called the approximation factor. In SVP_γ, one is given a lattice basis as input, and asked to find a nonzero lattice point of length within a factor γ of the length of the shortest nonzero vector. Lattice reduction algorithms such as Block-Korkine-Zolotarev (BKZ) [SE94] and its variants [GNR10,CN11,MW16,AWHT16] are considered as the most practical algorithms for solving the approximated shortest vector problem, balancing the quality (e.g., approximation factor) and runtime. Thus they play a central role in estimating the security of lattice-based cryptography. A lattice reduction algorithm relies on subroutines to find shortest vectors in smaller dimensional projected sublattices. This smaller dimension is a parameter of the algorithm called the "block size". There are two main families of algorithms for instantiating this subroutine: algorithms based on sieving [AKS01,NV08,MV10b,BDGL16] and algorithms based on enumeration [Kan83,FP85,SE94]. Sieving algorithms have better asymptotic performance but require much more memory. This paper focuses on enumeration algorithms.

The enumeration procedure can be seen as a depth-first search on some "enumeration tree", where a tree path encodes a partial assignment of the coefficients. Classically, the enumeration algorithm explores the tree nodes in a depth-first way as far as possible along a branch before backtracking. During the search, the algorithm only needs to hold the information for the nodes in the current path (e.g., from the root to the node being searched), thus requiring polynomial memory w.r.t the lattice dimension. This is one of the main benefits of using enumeration for lattice reduction. On the other hand, such a tree backtracking algorithm requires local knowledge before each iteration and thus cannot be instantiated straightforwardly by a quantum search algorithm such as Grover [Gro96]. Recall that Grover's algorithm assumes black-box access to every point in the input domain. In a tree backtracking scenario, these points are the coefficients of the linear combinations, which need not to be known in advance. This motivates the study of quantum algorithms for lattice enumeration.

Prior and related work. Montanaro presented an interesting quantum algorithm [Mon18] to solve the tree backtracking problem. The approach is based on the use of a quantum walk algorithm of Belovs [Bel13]. Montanaro's algorithm was originally presented in the context of solving the constraint satisfaction problem (CSP). A standard approach to solve CSP is via tree backtracking. Let $\mathcal{T}_u$ be an upper bound on the number of nodes in the tree and n be the tree depth. Mon-

tanaro's first algorithm detects the existence of a solution in the tree in $O(\sqrt{\mathcal{T}_u n})$ steps using quantum phase estimation. This provides a quadratic speed-up to the classical tree backtracking algorithm. A second algorithm of Montanaro uses the aforementioned detection algorithm to locate the solution, by applying the first algorithm iteratively, on each child of a node, which contains the solution. This leads to, at most, a polynomial increase in the running-time, e.g., it finds a solution in time $O(\sqrt{\mathcal{T}} n^{1.5} \log n)$, where $\mathcal{T}$ is the number of nodes in the tree. In lattice enumeration, the bound $\mathcal{T}$ (or $\mathcal{T}_u$) is usually super-exponential in terms of the input dimension n. Both algorithms use poly(n) space.

In a classical tree backtracking algorithm, it often happens that the actual number of nodes visited $\mathcal{T}'$ is much smaller than the tree size $\mathcal{T}$, for example, if the classical algorithms are optimized to search the most promising branches first. Ambainis and Kokainis [AK17] described a quantum algorithm which is better for such cases. More precisely, their quantum algorithm solves the search tree backtracking problem in $\tilde{O}(\sqrt{\mathcal{T}'} n^{1.5})$ steps, i.e., achieving the same complexity (up to logarithmic factors) as before but now in $\mathcal{T}'$, the number of nodes *examined* in the classical algorithm, instead of in $\mathcal{T}$, the total number of nodes.

As lattice point enumeration can be phrased as a tree backtracking procedure, a natural question is whether one can apply Montanaro's algorithm to the problem of lattice point enumeration. It has been briefly mentioned in previous work [ADPS16,ABB+17,PLP16] that Montanaro's algorithm can be used to speed up enumeration. Applicability was later confirmed in the work by Aono, Nguyen and Shen [ANS18], with more details given. They also proposed methods to apply the quantum algorithm to the extreme pruned enumeration, including both cylinder pruning and discrete pruning. Their work targeted a higher level, focusing on asymptotic strategies for optimizing the extreme pruning in the quantum backtracking algorithm, without going into detail about the quantum circuit and resource estimates. Indeed, they left as an open question in [ANS18]: *"We stress that this is just a first assessment of quantum enumeration. If one is interested in more precise estimates, such as the number of quantum gates, one would need to assess the quantum cost of the algorithm of Montanaro and that of Ambainis and Kokainis"*. This is the research question that we are focusing on in this paper.

In terms of concrete estimates, Campbell, Khurana and Montanaro [CKM19] and Martiel and Remaud [MR20] have both considered the implementation of Montanaro's backtracking algorithm in the context of graph coloring and SAT (satisfiability) problems, where the first work focuses on optimizing the circuit depth and the second work focuses on optimizing memory.

Recently, Albrecht, Prokop, Shen and Wallden [APSW22] considered how Noisy Intermediate Scale Quantum (NISQ) devices can be used to solve lattice enumeration. More precisely, they describe a mapping that encodes the SVP problem into the ground state of a Hamiltonian operator and use variational quantum algorithms such as the Variational Quantum Eigensolver [PMS+14,MRBAG16] to solve the encoded optimization problem. Their simulated experiments show that between 1000 and 1600 qubits are sufficient to encode the SVP for a 180-

dimensional lattice, which matches the current record dimension [DSvW21a] in the "Darmstadt SVP Challenge". Compared to [APSW22], our implementation is based on the quantum circuit model, which assumes fault-tolerant components.

Very recently, Bindel, Bonnetain, Tiepelt and Virdia [BBTV23] proposed to use a hybrid classical-quantum enumeration algorithm, which starts with classic enumeration and then switches to quantum enumeration for subtrees at a certain level. They state that their "results for combined classical-quantum enumeration suggest that current quantum enumeration with cylinder pruning techniques are unlikely to provide practical speedups against cryptographic instances of lattice problems in a MaxDepth setting."

Finally, there is no doubt that estimating the quantum resources required plays an important role in the cryptanalysis for post-quantum era. Substantial progress has been made for various problems, including Grover's algorithm for AES [GLRS16,BNS19,JNRV20], Shor's algorithm for ECDLP [RNSL17,BBVL21], algorithms by Kuperberg, Regev and Childs-Jao-Soukharev for CSIDH [BLMP19,BS20], and quantum lattice sieving for SVP [AGPS20]. This work aims to fill the gap for resource estimates of quantum lattice enumeration.

Contribution. To the best of our knowledge, no prior work has considered the concrete design and resource estimates for lattice enumeration in the quantum circuit model. In this work, we provide a concrete implementation of Montanaro's algorithm for lattice enumeration, together with resource estimates, in the quantum circuit model.

Overall, Montanaro's algorithm conducts phase estimation for an operator $U := R_B R_A$ on the root of the enumeration tree. We will describe the implementation in a modular approach: starting with the general phase estimation circuit, and the implementation of operators R_A and R_B and their components, and then covering the implementation of some predicate function P, which is used as an oracle inside the implementation of the operators R_A and R_B.

Our implementation is based on the so-called "Clifford+T" approach, which forms a universal gate set. This choice of gate set is motivated by fault-tolerant computation. The non-Clifford gates in our implementation consist of Toffoli and T gates, where Toffoli gates can be constructed using T gates. Their fault-tolerant implementation is often more expensive compared to Clifford gates. Thus we will optimize the circuit by optimizing the T depth. We will also show how to parallelize the circuit components to optimize the T depth of the circuit. Such optimization is motivated by NIST's Post-Quantum Cryptography Standardization process [NIS16]. NIST suggests that quantum attacks are restricted to a fixed circuit depth, namely the "maxdepth" parameter. This parameter is derived from the difficulty of running extremely long serial computations on quantum computers. Example values for "maxdepth" range from 2^{40} logical gates to 2^{96} logical gates, making the circuit depth the crucial parameter.

For simplicity of discussion, our implementation is designed for Montanaro's detection algorithm (e.g., see Theorem 1). The solution finding algorithm (e.g., see Theorem 2) and the improved method by Ambainis and Kokainis [AK17] use the detection algorithm as a main computational component.

Based on the proposed circuit, we will also discuss the quantum resource estimates required for lattice enumeration. The complexity of quantum attacks can be measured in terms of circuit depth and size. Overall, our circuit has T depth and size bounded respectively by

$$32\sqrt{\mathcal{T}n}\cdot\left[16np(\log B + 2\log n + p^{0.158}) + O(n\log B) + 8d^2\log(d\sqrt{\mathcal{T}n}) + 4d^2\log d + O(d^2)\right];$$

$$32\sqrt{\mathcal{T}n}\cdot\left[8(d+1)(14pn^2(B+1) + O(n^2B)) + 8d^2\log(d\sqrt{\mathcal{T}n}) + 16d^2\log d + O(d^2)\right],$$

where n is the lattice dimension, d is the degree of the enumeration tree (maximum number of children for any node), p is the precision required in the arithmetic and B bounds the coefficients size. This is presented in parameterized form since one can adapt these parameters given a concrete input. We also keep some of the lower-order terms coming from different components of the estimate.

For cryptographic size lattices, it is reasonable to expect $d \approx n, B \approx n^2, p \approx 3n$ under common heuristics (e.g., see discussions in Subsections 4.4 & 4.5). Let $\log(\mathcal{T}) \approx c \cdot n\log n$ where c denotes the dominating constant and $c \leq \frac{1}{2e}$ [ABF+20,ABLR21]. The T-depth and size is about

$$(128cn^3\log n + O(n^{2.158}))\sqrt{\mathcal{T}n} \text{ and } (10752n^6 + O(n^5))\sqrt{\mathcal{T}n}.$$

It is tempting to plug-in the best tree size estimate $\log(\mathcal{T}) \approx 0.125n\log n - 0.654n + 25.84$ from [ABLR21] so $c \approx 0.125$. However, we stress that this is not accurate, as the classical simulation [ABLR21] involves extensive extreme pruning steps and preprocessing, whose implementation and impacts are left as future work (see discussions in the next subsection).

Comparison and discussion. The security of lattice-based schemes is often estimated by a lattice reduction algorithm like BKZ, instantiated with sieving as the SVP solver. Sieving algorithms have been shown to be faster than enumeration in the classical setting, both asymptotically and in practice [ADH+19]. Concrete experiments were also demonstrated in [ADH+19,DSvW21b]. However, the comparison of concrete resources in the quantum world is considerably more complicated and has not been thoroughly studied, to the best of our knowledge.

Let d be the lattice dimension. The best lattice sieving algorithm [BDGL16] has a running-time of $2^{0.292d+o(d)}$ and memory requirement of $2^{0.210d+o(d)}$, using locality sensitive hashing. This has been adapted to a quantum sieving algorithm [Laa15], which runs asymptotically in $2^{0.265d+o(d)}$ and also requires a quantum memory of $2^{0.210d+o(d)}$. This was improved in [CL21] and further in [BCSS23], by replacing the Grover oracle with quantum random walks. This results in the fastest quantum sieving algorithm known to date with a complexity of $2^{0.2563d+o(d)}$ [BCSS23]. All of these works focus on resource estimates in an asymptotic sense, i.e., without detailing the circuit design or concrete resources. One exception is [AGPS20], which provides concrete quantum resource estimates (with Grover-based search) for several dominant parts of lattice sieving algorithms: a primary optimization target of [AGPS20] is the implementation of the "popcount" operation in sieving. Quantum enumeration via backtracking algorithm has been studied in [ANS18] in the query model, where the concrete cost

for each query is not detailed. As a result, the actual cost may be significantly higher than the asymptotic estimate. This has also been stressed in [ABLR21] as an open question, e.g., *"This suggests an analogous investigation to [AGPS20] for quantum enumeration as a pressing research question"*. Our paper aims to take a step forward in understanding this question for quantum enumeration. Our work is similar to [AGPS20] in philosophy, providing a concrete implementation of the dominant arithmetic for lattice enumeration. Therefore, similar to [AGPS20], our estimates are neither upper bounds nor lower bounds.

Given the (limited) research results in quantum enumeration and sieving, it is premature to conclude the crossover point between quantum enumeration and quantum sieving at this stage. This work does not fully settle this question, but takes a step forward in better understanding of where the crossover points will be based on gate design and qubit resources for quantum enumeration. We highlight several open questions that will be most critical in answering this question.

First, the dominating term in our estimates is the $O(\sqrt{\mathcal{T}n})$ term due to the phase estimation. It would be interesting to investigate quantum algorithms for parallelizing the phase estimation step in order to reduce the depth even further. Second, the concrete cost of extreme pruning in quantum enumeration requires further study. It has been proposed [ANS18] that one can run Montanaro's algorithm on a combined tree with all re-randomized bases. This has the advantage of enclosing the number of trees within the square root in the complexity. In an intuitive manner, all re-randomized bases can be fed in, inevitably increasing the memory. By comparison, in the classical setting, each enumeration can occur sequentially after another, so memory is not increased. Alternatively, Montanaro's algorithm can be run on each basis separately to control the memory while sacrificing the running time. Third, a binary tree conversion oracle is used in [ANS18] with some overheads, whose concrete cost is unknown. Investigating the overhead introduced by the transformation will provide a more precise picture for quantum enumeration. This paper focuses on the 'backbone' implementation so we will leave these questions for future research.

2 Preliminaries

Notations. We let lower-case bold letters denote column vectors and upper-case bold letters denote matrices. A matrix $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n)$ is presented in a column-wise way. We let matrix indices start with index 1. For a vector $\mathbf{x}$, we use $\|\mathbf{x}\|$ to denote its ℓ_2-norm. For $n \geq 1$ and $r > 0$, we let $V_n(r)$ denote the volume of the n-dimensional ball of radius r. We also let v_n denote the volume of an n-dimensional unit ball where $v_n = \pi^{n/2}/\Gamma(1 + n/2) \approx \left(\frac{2\pi\,\mathrm{e}}{n}\right)^{n/2}/\sqrt{n\pi}$. We denote by log the logarithm to base 2 and by ln the natural logarithm

Euclidean lattices. A lattice $\mathcal{L}$ is an additive discrete subgroup of $\mathbb{Q}^m$. Equivalently, it can be described as the set of all integral linear combinations of n linearly independent basis vectors: Let $\mathbf{B} = (\mathbf{b}_1, \cdots, \mathbf{b}_n) \in \mathbb{Q}^{m \times n}$ be a full rank matrix. The lattice $\mathcal{L}$ generated by $\mathbf{B}$ is defined as $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} \mid \forall \mathbf{x} \in \mathbb{Z}^n\}$. Let n be the rank (or dimension) of the lattice $\mathcal{L}$. It is called a full rank lat-

tice when $m = n$. The matrix $\mathbf{B}$ is called a basis of $\mathcal{L}(\mathbf{B})$. Given a matrix $\mathbf{B}$ and a vector $\mathbf{v}$, we let $\pi_i(\mathbf{v})$ denote the orthogonal projection of $\mathbf{v}$ onto the linear subspace $(\mathbf{b}_1, \cdots, \mathbf{b}_{i-1})^\perp$. Further, we let $\pi_i(\mathbf{B}_{[i:j]})$ denote the block $(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_{j-1}))$, let $\pi_i(\mathcal{L}_{[i:j)})$ denote the corresponding lattice generated by $\pi_i(\mathbf{B}_{[i:j]})$, and let $\mathbf{B}^* = [\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*]$ denote the Gram–Schmidt orthogonalization of $\mathbf{B}$, thus $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$. The volume (or determinant) of $\mathcal{L}(\mathbf{B})$ is defined as $\mathrm{vol}(\mathcal{L}(\mathbf{B})) = \prod_{i \le n} \|\mathbf{b}_i^*\|$, which does not depend on the choice of basis of $\mathcal{L}$.

The norm of a shortest non-zero vector in $\mathcal{L}$ is denoted by $\lambda_1(\mathcal{L})$ which is called the minimum of the lattice $\mathcal{L}$. Let $\mathcal{L}$ be a rank-n lattice. Minkowski's convex body theorem states that $\lambda_1(\mathcal{L}) \le 2 \cdot v_n^{-1/n} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. The analysis of lattice algorithms often relies on heuristic assumptions such as the so-called Gaussian Heuristic (GH). Let $\mathcal{S}$ be a measurable set in the span of $\mathcal{L}$. The Gaussian Heuristic states that the number of lattice points in $\mathcal{S}$ is $|\mathcal{L} \cap \mathcal{S}| \approx \mathrm{vol}(\mathcal{S})/\mathrm{vol}(\mathcal{L})$. When $\mathcal{S}$ is an n-dimensional ball of radius r, the latter quantity is about $(v_n \cdot r^n)/\mathrm{vol}(\mathcal{L})$. Taking $v_n \cdot r^n \approx \mathrm{vol}(\mathcal{L})$, we see that $\lambda_1(\mathcal{L})$ is about $\mathrm{GH}(\mathcal{L}) \coloneqq v_n^{-1/n} \cdot \mathrm{vol}(\mathcal{L})^{1/n} \approx \sqrt{n/(2\pi\,\mathrm{e})} \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. The Gaussian heuristic holds for random lattices, see [BL21] for definitions of distributions. Let $\mathbf{B}$ be a lattice basis for $\mathcal{L}$. We define the Root Hermite Factor of the basis $\mathbf{B}$ as $\delta(\mathbf{B}) = (\|\mathbf{b}_1\|/\mathrm{vol}(\mathcal{L})^{1/n})^{1/(n-1)}$. Here, we use the normalization by the $(n-1)$-th root (sometimes it is defined by normalization with the n-th root).

Two fundamental average-case problems used in lattice-based cryptography are the short integer solution problem (SIS) [Ajt96,MR04] and the learning with errors problem (LWE) [Reg05]. Algorithms for solving the LWE and SIS problems often involve algorithms for solving worst-case problems such as the approximated shortest vector problem (SVP_γ). On input a lattice basis $\mathbf{B}$, SVP_γ asks to find a non-zero lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{v}\| \le \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$. When $\gamma = 1$, it is exact SVP.

Lattice algorithms. The difficulty of solving SVP_γ varies with respect to its approximation factor γ, which is usually a function of the rank n of the lattice. When $\gamma = 2^{\Omega(n)}$, the LLL algorithm [LLJL82] finds a SVP_γ solution in polynomial time. The time complexities of the best known algorithms that find the exact solutions (e.g. $\gamma = 1$) are at least exponential in the dimension of the lattice: representative exact algorithms include enumeration algorithms [Kan83,FP85,SE94], sieving algorithms [AKS01] and Voronoi cell algorithms [MV10a]. Most cryptographic constructions rely on lattice problems with approximation factors that are polynomial or slightly sub-exponential in n. In such a regime, the best practical algorithms are lattice reduction algorithms such as Block-Korkine-Zolotarev (BKZ) [Sch87,CN11,HPS11]. Given a input basis, these output a basis made of relatively short vectors.

For a lattice basis $\mathbf{B}$, let $\mu_{i,j} \coloneqq \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle$. It is size-reduced if $\forall i \ge j$, $|\mu_{i,j}| \le 1/2$, HKZ-reduced (Hermite-Korkine-Zolotarev) if it is size-reduced and satisfies: $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(\mathcal{L}_{[i:n)}))$, $\forall i \le n$, and BKZ-β reduced with blocksize β if it is size-reduced and satisfies $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(\mathcal{L}_{[i,\min(i+\beta-1,n)]}))$, $\forall i \le n$. BKZ algorithms, such as [SE94], take as inputs a block-size β and a basis $\mathbf{B}$ of a lattice $\mathcal{L}$, and output a basis which is close to being BKZ-β reduced. A typical BKZ

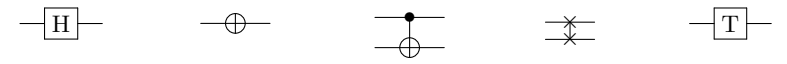
algorithm invokes SVP solvers on consecutive local blocks $\pi_k(\mathbf{B}_{[k,\min(k+\beta-1,n)]})$ for all $k < n$ (this is called a *BKZ tour*). After each run of the SVP-solver, if we find $\lambda_1(\pi_k(\mathbf{B}_{[k,\min(k+\beta-1,n)]})) < \alpha \cdot \|\mathbf{b}_k^*\|$ for some relax factor $\alpha \geq 1$, then BKZ updates this block by inserting the shorter vector found by the SVP-solver at index k. LLL reductions are used whenever there is an update on the basis. The BKZ tours are repeated and the whole algorithm terminates when no change occurs at all during a tour or some termination condition is met. A useful heuristic is the *Geometric Series Assumption* (GSA) introduced in [Sch03], which states that the Gram-Schmidt norms $\{\|\mathbf{b}_i^*\|\}_{i\leq n}$ of a BKZ-β reduced basis behave as a geometric series, i.e., there exists a constant $r > 1$ such that $\|\mathbf{b}_i^*\|/\|\mathbf{b}_{i+1}^*\| \approx r$ for all $i < n$, where $r \approx \beta^{1/\beta}$ asymptotically.

Lattice enumeration. Lattice reduction algorithms such as BKZ make calls to exact (or approximate) SVP oracles of smaller dimensions. These SVP solvers can be instantiated by the aforementioned exact SVP algorithms. In this work, we focus on exact SVP solvers instantiated by enumeration. In short, such algorithms proceed with an exhaustive search for all (or partial) lattice vectors within a certain region. We review the algorithms of [Kan83,FP85,SE94].

Given a basis $\mathbf{B}$ of $\mathcal{L}$ and a good upper bound R for the length of a shortest vector in $\mathcal{L}$, an enumeration algorithm outputs a shortest vector $\mathbf{v}$ with $\|\mathbf{v}\| \leq R$ if such a vector exists. To find $\mathbf{v}$, the enumeration algorithm searches over a tree formed by all vectors of norm bounded by R. This can be achieved by writing $\mathbf{v} = \sum_{j=1}^{n} v_j \mathbf{b}_j = \sum_{j=1}^{n} (v_j + \sum_{i=j+1}^{n} \mu_{i,j} \mathbf{v}_i)\mathbf{b}_j^*$ and bounding each of the projections:

$$\|\pi_{n+1-l}(\mathbf{v})\|^2 = \sum_{j=n+1-l}^{n} (v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i)^2 \|\mathbf{b}_j^*\|^2 \leq R^2, \ \forall\, 1 \leq l \leq n. \qquad (1)$$

The Schnorr-Euchner algorithm [SE94] conducts a depth first search of the enumeration tree. More precisely, the algorithm starts with $l = 1$ and fixes some v_n in a bounded range as determined by $\|\pi_n(\mathbf{v})\|^2 \leq R^2$. Inductively, assume the algorithm has proceeded to level $l \geq 2$ and hence all v_{n+2-i} for $2 \leq i \leq l$ are fixed. Using $\|\pi_{n+1-l}(\mathbf{v})\|^2 \leq R^2$, a permissible range for the coefficient v_{n+1-l} can be derived. The algorithm backtracks when the range becomes invalid and succeeds when it reaches a leaf. This procedure thus creates an enumeration tree of depth n and only requires poly(n) memory. The running time of an enumeration-based algorithm depends heavily on the quality of the input lattice basis $\mathbf{B}$. In Kannan's enumeration algorithm [Kan83], a strong preprocessing is performed before enumeration, to achieve a quasi-HKZ shape. Subsequent analysis [HS07] shows the Kannan's enumeration achieves a worst-case running-time of $n^{n/2e+o(n)}$. Heuristically the number of enumerated lattice vectors can be estimated using the Gaussian heuristic. To speed up the enumeration, Schnorr and Hörner [SH95] proposed a technique known as tree pruning. The idea is to reduce the search space by performing the search on a subset of all possible solutions, which are more likely to be short. An improved algorithm, namely extreme pruning, was proposed by Gama, Nguyen and Regev [GNR10]. The main idea is to heavily prune the enumeration tree and repeat the procedure by rerandomizing



(a) H-gate (b) NOT (c) CNOT (d) SWAP (e) T-gate

Fig. 1: Basic quantum gates.

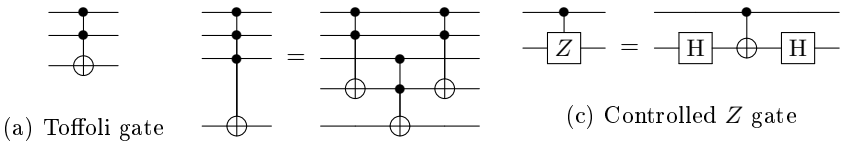
the basis. They also proposed methods for choosing the pruning parameters to optimize the running-time versus success probability. This leads to the current-state-of-the-art of enumeration-based lattice reduction algorithms, which has been adopted in BKZ 2.0 [CN11] and implemented in the FPLLL library [dt22] and the progressive preprocessing lattice reduction library [AWHT16]. In this work, for convenience, it is sufficient to assume the bound R in inequality (1) can be replaced by some number $R_l \coloneqq f(R, l, \mathbf{B})$ where f can be pre-computed.

Quantum circuits. In this work, we will formulate the quantum architecture based on the quantum circuit model. As stated before, the gate count uses the "Clifford+T" gate set. Such choice of gate set is motivated by fault-tolerant computation, as many quantum error correcting codes can naturally implement Clifford gates. The T-gate is non-Clifford and used to implement the useful Toffoli gate. As fault-tolerant implementation of T-gates is more involved than for Clifford gates, our resource estimates minimize T-depth and T-gate count

We assume the implementations of the following gates are the primitive building blocks and give their circuit notation in Figure 1.

The Hadamard-gate (denoted H) acts on a single qubit. In outer product notation, $\mathrm{H} = \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| - |1\rangle\langle 1|)$. We denote $|+\rangle \coloneqq \mathrm{H}|1\rangle$ and $|-\rangle \coloneqq \mathrm{H}|0\rangle$. The multi-bit H gate is defined as $\mathrm{H}^{\otimes n} \coloneqq \mathrm{H}\otimes\cdots\otimes\mathrm{H}$ with n such H gates. The NOT-gate (sometimes called the X-gate) is a single-qubit gate, which flips the bit when the inputs are from the standard basis. As outer product, $\mathrm{NOT} = |1\rangle\langle 0| + |0\rangle\langle 1|$. The CNOT-gate is a two-qubit operation, where the first qubit is usually referred to as the control qubit and the second qubit as the target qubit. As outer product, $\mathrm{CNOT} = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|$. The SWAP-gate is a two-qubit operation that swaps the state of the two input qubits. As outer product, $\mathrm{SWAP} = |00\rangle\langle 00| + |11\rangle\langle 11| + |01\rangle\langle 10| + |10\rangle\langle 01|$. The SWAP gate can be implemented for free by reassigning the labels, assuming distance between qubits does not matter. The T-gate is a single-qubit gate, which rotates the Z-axis of the Bloch sphere by 45' degree. As outer product, $\mathrm{T} = |0\rangle\langle 0| + e^{i\pi/4}|1\rangle\langle 1|$. The Eastin-Knill theorem [EK09] shows that not all the unitary operators in a universal operator set can be implemented transversally. Implementations often choose the T-gate as the non-transversal gate.

We will also use several controlled gates that can be implemented from the above primitive quantum gates. This includes the Toffoli gate and controlled-Z gate, with their circuit notation given in Figure 2. The Toffoli gate can be decomposed into smaller quantum gates, which can be made fault-tolerant. Among



(a) Toffoli gate

(b) 3-Toffoli gate, using 3 Toffoli gates and 1 ancillary qubit

(c) Controlled Z gate

Fig. 2: Synthesized quantum gates.

them, the most expensive gate is the T-gate. A standard decomposition of the Toffoli gate into the Clifford+T set can be found in [NC11], which has a T-count of 7 with a T-depth of 6. This has been improved in the design of [AMMR13] which has a T-count of 7 and a T-depth of 3. This appears to be the best T-depth without ancillas. With ancillas, the T-depth can be reduced to 1, as described in [Sel13]. This is mostly useful when the T-gate is expensive and ancillas are cheap. In terms of T-gate count, [Jon13] describes a probabilistic circuit which has a T-count of 4 using one ancilla bit.

The Toffoli gate can be extended into a multi-Toffoli gate. The simplest design for an n-Toffoli gate (where $n > 2$) starts by applying a Toffoli gate from the first 2 control qubits onto an ancillary qubit, the intermediate result. Then it repeatedly applies a Toffoli gate with the next control qubit and the current intermediate result onto a new ancillary qubit to get a new intermediate result. Finally, it copies the result onto the target qubit with a CNOT-gate, or replaces the last ancillary qubit with the target qubit. It also needs to clean up the intermediate results with Toffoli gates, using in total $2n - 2$ Toffoli gates, $n - 1$ ancillary qubits and a CNOT gate. If the last ancillary qubit is replaced with the target, $2n - 3$ Toffoli gates and $n - 2$ ancillary qubits suffice.

The Z gate is a gate to flip the phase of the $|1\rangle$ state. In outer product form, $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$. The Z gate can be implemented by putting a NOT gate between 2 Hadamard gates. It can be controlled by replacing the NOT gate with a CNOT, Toffoli or multi-Toffoli gate.

The $R_y(\theta)$ gate rotates by $\theta/2$ degrees around the y-axis of the Bloch sphere. In outer product form, $R_y(\theta) = \cos(\theta/2)\,|0\rangle\langle 0| - \sin(\theta/2)\,|0\rangle\langle 1| + \sin(\theta/2)\,|1\rangle\langle 0| + \cos(\theta/2)\,|1\rangle\langle 1|$. To get it, we first need to introduce the $R_x(\theta)$ and $R_z(\theta)$ gates, rotating around the x- and z-axes respectively. The $R_z(\theta)$ gate can be made from the base set using $4\log(\frac{1}{\epsilon}) + O(\log(\log(\frac{1}{\epsilon}))$ T-gates for ϵ accuracy [RS16]. The $R_x(\theta)$ gate can be made using $R_x(\theta) = HR_z(\theta)H$, which then can be turned into the $R_y(\theta)$ gate using $R_y(\theta) = T^2R_x(\theta)T^6$. So to get $R_y(\theta)$ we need 8 T gates, 2 Hadamard gates, and a $R_z(\theta)$-gate. The $R_y(\theta)$ gate can be decomposed into $\mathrm{NOT}R_y(-\theta/2)\mathrm{NOT}R_y(\theta/2)$, meaning it can be made controllable by replacing the NOT gates with CNOT, Toffoli or multi-Toffoli gates. Finally, the

Hadamard gate can be decomposed into $R_y(\frac{\pi}{4})ZR_y(-\frac{\pi}{4})$. By making the Z gate controllable, we can make a controllable Hadamard gate.

Cost model. We make a number of assumptions for our quantum resource cost model. We assume a full capacity of parallelism, e.g., the circuit can run any number of gates simultaneously as long as these gates act independently on different qubits. The time complexity of the circuit depends on its depth, which is why we mainly focus on Toffoli and T gates. We use Toffoli and T gates simultaneously, and as noted above, the Toffoli gate can be implemented using a circuit of T gates with T-depth 1. Some component may involve a polynomial number of ancillas, which we sometimes trade off in favor of a smaller circuit depth. Overall we aim to optimize the circuit depth and then the circuit size.

3 Quantum Tree Backtracking

Our implementation focuses on the quantum tree backtracking algorithm of Montanaro [Mon18], based on the electric network framework [Bel13], for detecting a solution. Montanaro's algorithm was presented for solving the so-called constraint satisfaction problem (CSP). Some further background on tree backtracking for solving the CSP is given in Appendix B. In this section, we provide some details about Montanaro's algorithm, to facilitate our concrete implementation and circuit presented in Section 4.

It can be seen that lattice enumeration allows for tree backtracking, which we will denote as "enumeration trees". Indeed, inequality (1) suggests that, at the l-th ($l \geq 2$) level of the tree, $v_{n+2-l}, \ldots, v_n$ are already determined. To go down the tree, it remains to bound and select a value, if it exists, for v_{n+1-l} according to Inequality (1). We let d denote the maximal number of choices of v_i. Thus this can be represented by a tree with up to n layers and degree d.

3.1 Montanaro's algorithms

A first algorithm of Montanaro detects whether the tree contains a solution for some predicate P in $O(\sqrt{\mathcal{T}n})$ steps where $\mathcal{T}$ is an upper bound on the tree size and n is the height of the tree. This is given in Theorem 1.

Theorem 1 ([Mon18, Theorem 1.1]). *Let $\mathcal{T}$ be an upper bound on the number of vertices in a tree formed by some constraint. Then for any $0 < \delta < 1$ there is a quantum algorithm which, given $\mathcal{T}$, evaluates a predicate P and a function h (which determines how to extend a given partial assignment) for $O(\sqrt{\mathcal{T}n}\log(1/\delta))$ times each, outputs true if there exists x such that $P(x)$ is true, and outputs false otherwise. The algorithm uses* $\mathrm{poly}(n)$ *qubits, $O(1)$ auxiliary operations per use of P and h, and fails with probability at most δ.*

In the lattice enumeration context, an upper bound $\mathcal{T}$ is often super-exponential in n and the tree height n can be the lattice dimension. An upper bound $\mathcal{T}$ on the tree size is an input in the above theorem. This detection algorithm can be modified to actually locate the solution, via repeated applications on the subtrees. This leads to Montanaro's second algorithm:

Theorem 2 ([Mon18, Theorem 1.2]). *Let $\mathcal{T}$ be the number of vertices in a tree formed by some constraint. Then for any $0 < \delta < 1$ there is a quantum algorithm which makes $O(\sqrt{\mathcal{T}} n^{1.5} \log n \log(1/\delta))$ evaluations of each of P and h, and outputs x such that $P(x)$ is true, or "not found" if no such x exists. If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, there is a quantum algorithm which outputs x_0 making $O(\sqrt{\mathcal{T}} n \log^3 n \log(1/\delta))$ evaluations of each of P and h. In both cases the algorithm uses $\mathrm{poly}(n)$ space, $O(1)$ auxiliary operations per use of P and h, and fails with probability at most δ.*

In the second algorithm in Theorem 2, $\mathcal{T}$ denotes the actual number of vertices in the tree, which needs not to be given as an input to the algorithm. In both algorithms, the tree degree d is assumed to be $O(1)$ in the analysis.

The main essence of Theorem 1 is an algorithm to determine the presence of a solution given a tree root. This detection algorithm also serves as the main computational component in Theorem 2. We will thus focus on the algorithm in Theorem 1 for detecting a solution. During the algorithm, the state is a superposition of all the possible paths in the tree, and the primary tool we need from quantum computing is phase estimation, as described in Theorem 3.

Theorem 3 ([Kit96,CEMM98]). *Assume a unitary U is given as a black box. There exists a quantum algorithm that, given an eigenvector ψ of U with eigenvalue $e^{i\phi}$, outputs a real number w such that $|w - \phi| \leq \delta$ with probability at least 9/10. Moreover, the algorithm uses $O(1/\delta)$ controlled applications of U and $\frac{1}{\delta}\,\mathrm{poly}(\log(1/\delta))$ other elementary operations.*

To get a precision of $\delta = 2^{-s}$ for some positive integer s, we need to apply controlled-U 2^s times and use $O(s^2)$ more gates to perform a quantum Fourier transform on s qubits. Also in practice, we end up only caring about being able to distinguish the eigenvalue $1 = e^{i0}$ from other eigenvalues, so we only need to check if the output w is close to 0 or not. We then construct a single 'walk step' U, that reflects over superpositions of nodes and their children, except for marked leaves. We construct this walk step such that it keeps the sign of the uniform superposition over the vertices and the path from the root to a leaf that contains a solution. If we had a path, we could use phase estimation to detect if the eigenvalue of this path was one. However, if we had a path, we could just test if the leaf contains a solution. Instead, we use that the root is 'close enough' to a path and test the eigenvalue of the root. If a marked vertex (one that contains a solution) exists, it is likely that phase estimation would output eigenvalue 1. On the other hand, if the tree does not contain a solution, it is less likely that phase estimation would output eigenvalue 1, since 1 is 'not very close' to the uniform superposition of vertices. Thus, by repeating the phase estimation with high precision, we can eventually get the desired confidence.

The walk steps Montanaro introduced use a diffusion operator D_x which acts on the Hilbert space spanned by a node $|x\rangle$ and its children. Let d_x denote the degree of the node $|x\rangle$ and $x \to y$ mean y is a child of x. D_x is defined as:

- If x is marked, D_x is the identity.

- If x is not marked and not the root, it changes the sign of the uniform superposition of x and its children y and leaves any orthogonal superpositions alone. This can be described by a Householder transformation, e.g.:

$$D_x \coloneqq I - 2\left|\psi_x\right\rangle\left\langle\psi_x\right|, \text{ with } \left|\psi_x\right\rangle \coloneqq (1/\sqrt{d_x})(\left|x\right\rangle + \sum_{y,x\to y}\left|y\right\rangle).$$

- At the root, $D_r \coloneqq I - 2\left|\psi_r\right\rangle\left\langle\psi_r\right|$ where

$$\left|\psi_r\right\rangle = (1/\sqrt{1+d_r n})(\left|r\right\rangle + \sqrt{n}\sum_{y,r\to y}\left|y\right\rangle).$$

It is worthwhile to note the D_x operator affects the node x and its children simultaneously. Montanaro's algorithm splits the tree into vertices of even and odd distance from the root, called set A for even and set B for odd, using that trees are bipartite. Each step of the 'walk' consists of applying $R_B R_A$ where $R_A = \bigoplus_{x\in A} D_x$ and $R_B = \left|r\right\rangle\left\langle r\right| + \bigoplus_{x\in B} D_x$. The operator R_A applies the diffusion operator to all vertices of even distance from the root (including the root) and their children, changing the sign of any ψ_x (or ψ_r), but leaving orthogonal states untouched. The operator R_B acts similarly on vertices of odd distance together with the root, and the $\left|r\right\rangle\left\langle r\right|$ in R_B actually leaves the root untouched. Now this operator has been constructed such that we have two possible eigenvectors $\left|\phi\right\rangle, \left|\eta\right\rangle$ not orthogonal to $\left|r\right\rangle$ with eigenvalue 1, where

$$\left|\phi\right\rangle = \sqrt{n}\left|r\right\rangle + \sum_{x\neq r, x\rightsquigarrow x_0}(-1)^{l(x)}\left|x\right\rangle \text{ and } \left|\eta\right\rangle = \left|r\right\rangle + \sqrt{n}\sum_{x\neq r}\left|x\right\rangle,$$

where $l(x)$ is the distance of x to the root and $x \rightsquigarrow x_0$ means every x on the path to a marked vertex x_0 (including x_0). If no marked vertex exists $\left|\eta\right\rangle$ has eigenvalue 1 else $\left|\phi\right\rangle$ has eigenvalue 1. Finally we apply quantum phase estimation of $R_B R_A$ to $\left|r\right\rangle$, and check if it has eigenvalue 1. This returns 1 with probability greater than $1/2$ if a marked vertex exists and smaller than $1/4$ if it does not [Mon18, Lemma 2.4]. To conclude, Montanaro's tree backtracking algorithm for detecting a solution is described in Algorithm 1. Montanaro showed that this detection algorithm fails with probability at most δ when the phase estimation is invoked for $K = \lceil \gamma \log(1/\delta)\rceil$ times for some universal constant γ.

3.2 Quantum lattice enumeration

Aono, Nguyen and Shen [ANS18] adapted the algorithms by Montanaro [Mon18] and Ambainis and Kokainis's [AK17] to the context of lattice point enumeration. They obtain the following theorems:

Theorem 4 ([ANS18, Theorem 7 & 8]). *For any $\delta > 0$, given an LLL-reduced basis $\boldsymbol{B}$ and a radius R together with a pruning function f, there is a quantum algorithm that outputs a shortest non-zero vector $\boldsymbol{v}$ in $\mathcal{L}(\boldsymbol{B})\cap P_f(B,R)$, with correctness probability $\geq 1-\delta$, in time $O(\sqrt{\mathcal{T}}n^3\beta\,\mathrm{poly}(\log(n),\log(1/\epsilon),\log(\beta)))$, where β is the bit-size of input vectors. Applying this to Kannan's algorithm leads to a quantum enumeration algorithm of $n^{n/4e+o(n)}\cdot\mathrm{poly}(\log(n),\log(1/\epsilon),\beta)$.*

Algorithm 1 Detecting a solution (Alg. 2 of [Mon18]), universal constants β, γ.

Input: Operators R_A, R_B, a failure probability δ, upper bounds on the depth n and the number of vertices $\mathcal{T}$.

Output: Solution exists or not.

Repeat the following subroutine $K = \lceil \gamma \log(1/\delta) \rceil$ times:
(a) Apply phase estimation to the operator $R_B R_A$ on $|r\rangle$ with precision $\beta/\sqrt{\mathcal{T}n}$.
(b) If the eigenvalue is 1, accept; otherwise, reject.
If the number of acceptances is at least $3K/8$, return "solution exists"; otherwise, return "no solution".

There are two interesting modifications in [ANS18]. First, one can transform a tree of depth n and degree d to a binary tree of depth $n \log d$. The access to the predicate of the original tree can be modified efficiently to a predicate access to the transformed tree. For an LLL-reduced basis, the enumeration tree has $d \leq 2^n$ and thus leads to a binary tree of depth $O(n^2)$. The main idea in the proof of Theorem 4 is to apply Montanaro's [Mon18] second algorithm iteratively on the transformed binary tree. Second, [ANS18] proposes to run the quantum algorithm on a combined tree, consisting of m rerandomized bases, in the extreme pruning setup. This leads to a factor of $O(\sqrt{m})$ savings on the quantum enumeration, compared to calling the procedure m times sequentially. Our work focuses on the implementation aspects. Thus we do not repeat the details on extreme pruning and the binary tree transform, as already given in [ANS18].

4 Using Backtracking for Enumeration

In this section, we describe the circuit for the quantum lattice enumeration, based on Montanaro's tree backtracking [Mon18]. We also give the resource estimate, which is based on a design optimized w.r.t the T-depth. Overall, the algorithm conducts phase estimation of some operator $R_B R_A$ on the root of subtrees. We make the implementation modular, starting with the phase estimation, and the implementation of R_A or R_B, and then the implementation of the predicate which is used as an oracle inside the implementation of R_A and R_B.

4.1 Phase estimation

To start with, the main algorithm of Montanaro [Mon18] (see Algorithm 1) detects if there exists a solution in a given subtree. Note that Step (a) in Algorithm 1 can run in parallel, thus the main computation is the phase estimation step of the operator $R_B R_A$ on the input $|r\rangle$ where $|r\rangle$ denotes the root of the tree. For completeness, Figure 3 shows a general phase estimation circuit. Step (a) in Algorithm 1 implies that, for the correctness of phase estimation, one requires the precision to be $\approx \beta/\sqrt{\mathcal{T}n}$ (note β is some universal constant independent of the input size). This means that controlled-U operators show up $O(\sqrt{\mathcal{T}n})$ times.

In [CKM19], Campbell, Khurana and Montanaro have given a more precise estimate on the hidden constant. For example, to achieve a failure probability

$\delta \leq 0.1$ (see Algorithm 1 for δ), one can take the number of repetitions $K = 79$ and then the number of controlled-U operators can be bounded by $32\sqrt{\mathcal{T}n}$. Note that the input $|0\rangle^{\otimes m}$ takes $m = O(\log(\mathcal{T}n))$ qubits. Asymptotically, $\mathcal{T}$ upper-bounds the tree size which is about $n^{cn+o(n)}$ where $c \leq \frac{1}{2e}$ [ABF+20,ABLR21]. Thus m is about $O(n \log n)$. The quantum Fourier transform (QFT) step can be constructed in $O(\log^2(\mathcal{T}n))$ primitive gates. We will see in Subsection 4.2 that the input to the U operators requires about $\Theta(n \log d)$ qubits.

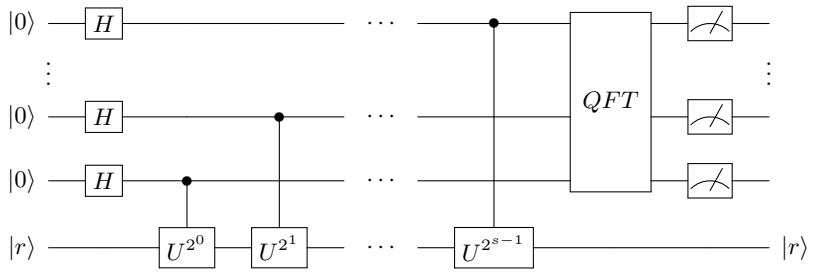


Fig. 3: Circuit for phase estimation where $U := R_B R_A$.

4.2 Implementation of R_A and R_B

In the phase estimation algorithm, e.g., Figure 3, a key step is the controlled-U operator. We consider implementing a single step of the operator $U = R_B R_A$. Montanaro provides a high level description to implement $R_A = \bigoplus_{x \in A} D_x$, with R_B being analogous, in Algorithm 3 of [Mon18]. We implement this algorithm for lattice enumeration step by step and give the number of gates and ancillary qubits required.

Implementation of R_A (and R_B). The input of the algorithm is:

- An integer $|l\rangle$ which is the depth we are at, ranging from 0 (indicating we are at the root) to n (indicating we are at a leaf). This corresponds to the value l as in Inequality (1) for lattice enumeration.
- An array $|v\rangle = |v_1\rangle \dots |v_n\rangle$ which is the path in the tree, and each $v_i \in [d] \cup \{*\}$, where $[d] = \{1, \dots, d\}$, and $*$ denotes unassigned. Each element of $[d]$ corresponds to one child of the node. The indices i correspond to the values i in Inequality (1).
- Access to an oracle P, which given a path in the tree $|l\rangle\,|v\rangle$ returns true, false or indeterminate.
- Access to a heuristic h to determine the next index to branch on. In the lattice enumeration context, this simply decreases from n to 1.

The input thus requires $\lceil \log n \rceil + n\lceil \log d + 1 \rceil$ qubits where the coefficients v_i are the dominating terms. The algorithm also needs some ancillary qubits: (1) $\mathcal{H}_{\mathsf{anc}}$ which stores $a \in [d] \cup \{*\}$ and starts in $|*\rangle$. (2) $\mathcal{H}_{\mathsf{children}}$ which stores an array $S \subseteq [d]$ and starts empty. Note $\mathcal{H}_{\mathsf{anc}}, \mathcal{H}_{\mathsf{children}}$ take up $\lceil \log d + 1 \rceil$ and d qubits respectively. The algorithm also uses some extra qubits to do intermediate computations that we will describe later.

We now describe the algorithm (Algorithm 3 of Montanaro [Mon18]) and its implementation. Note, to be consistent with the description of Algorithm 3 of Montanaro [Mon18], the indices are from 1 to n. This should be reversed outside this subsection when considering our application as in Inequality (1).

Step 1. If $P(|l\rangle |v\rangle)$ is true, return. To implement this step, we call P and save the output of $P(|l\rangle |v\rangle)$ in an ancillary qubit $|c_1\rangle$ where $c_1 = 1$ when P returned NOT true and 0 otherwise, which we will use later.

Step 2. If l is odd, subtract $h((i_1, v_1), \ldots, (i_{l-1}, v_{l-1}))$ from i_l and swap a with v_l. To implement this step, we swap a and v_l controlled by the least significant qubit of l. This uses $\lceil \log d + 1 \rceil$ Toffoli gates and twice that many CNOT gates. The subtraction is not relevant for lattice enumeration context.

Step 3. If $a \neq *$, subtract 1 from l. To implement this step, we need to first find out whether $a = *$, using a $\lceil \log d + 1 \rceil$-Toffoli gate onto an ancillary qubit $c_3 = 1(a \neq *)$ and some NOT gates depending on how we represent $*$. As described in Section 2, we can implement the i-Toffoli gate trivially using $2i-3$ Toffoli gates and $i-2$ ancillary qubits, starting and ending in $|0\rangle$. Then we do a controlled reverse incrementer circuit on l controlled by c_3, using Gidney's design [Gid15]. The incrementer circuit (using the $n-2$ zeroed bit design) uses $2\lceil \log n \rceil - 4$ Toffoli gates, $\lceil \log n \rceil - 1$ CNOT gates and 1 NOT gate using $\lceil \log n \rceil - 2$ ancillary qubits. To make it controllable, we convert the CNOT gates into Toffoli gates, additionally controlled by c_3, and the NOT gate into a CNOT gate also controlled by c_3.

Step 4. This is skipped.

Step 5. For each $w \in [d]$: If $P(|l+1\rangle |v_1\rangle \ldots |v_l\rangle |w\rangle)$ is not false, set $S = S \cup \{w\}$. $|S\rangle$ is implemented as a string of d qubits, each representing one entry of $[d]$. For each $w \in [d]$ we access P, check whether the result is false or not, and change the appropriate qubit of $|S\rangle$, which starts as the $|0\rangle^d$ state. We repeat this step d times for a total of d uses of the oracle P and CNOT gates. We leave d ancillary qubits with the intermediate results to decrease calls to P.

Step 6. If $l = 0$, perform the operation $I - 2|\phi_{n,S}\rangle\langle\phi_{n,S}|$ on a. Otherwise, perform the operation $I - 2|\phi_{1,S}\rangle\langle\phi_{1,S}|$ on a. This is the step where we perform D_x. To do this, we need the function $U_{\alpha,S}$:

$$|*\rangle \mapsto \frac{1}{\sqrt{\alpha|S|+1}} \left(|*\rangle + \sqrt{\alpha} \sum_{i \in S} |i\rangle \right).$$

$U_{\alpha,S}$ maps $|*\rangle$ into $|\phi_{\alpha,S}\rangle$. We can then do

$$U_{\alpha,S} \left(I - 2|*\rangle\langle *| \right) U_{\alpha,S}^{-1} = I - 2|\phi_{\alpha,S}\rangle\langle\phi_{\alpha,S}|.$$

The operator $I - 2|*\rangle\langle *|$ requires a check whether $|a\rangle = |*\rangle$ and applies a conditional Z gate if $|a\rangle = |*\rangle$, using a $\lceil \log d+1 \rceil$+1-Toffoli gate. Additionally condition the Z gate on c_1 as well as the relevant qubit used in the Fourier transform. No other step has to be controlled to make $R_B R_A$ controllable.

Step 7. Uncompute S and j by reversing Steps 5 and 4, making d more uses of P.

Step 8. If $a \neq *$, add 1 to l. If l is now odd, add $h((i_1, v_1), \ldots, (i_{l-1}, v_{l-1}))$ to i_l and swap v_l with a. (Now $a = *$ again.) Another uncomputation step, reverse Steps 3, 2 & 1 with the same cost.

The number of gates required and the number of additional qubits needed is summarized in Table 1.

Table 1: Summary of circuit and gates required. Ancillary qubits are required until they are subtracted, zeroed qubits are required only for one step and are returned to zero in that step.

Step	TOF	CNOT	P calls	Ancillas	Zeroed qubits
1	0	0	1	1	0
2	$\lceil \log d + 1 \rceil$	$2\lceil \log d + 1 \rceil$	0	0	0
3	$5\lceil \log d + 1 \rceil - 8$	2	0	1	$\lceil \log d + 1 \rceil - 2$
5	0	d	d	d	0
6	$2U_{\alpha,S}^{-1} + 2\lceil \log d + 1 \rceil - 1$	$2U_{\alpha,S}^{-1}$	0	0	$U_{\alpha,S}^{-1}$
7,8	$6\lceil \log d + 1 \rceil - 8$	$d + 2\lceil \log d + 1 \rceil + 2$	$d+1$	$-(d+2)$	$\lceil \log d + 1 \rceil - 2$

As described in Subsection 4.3, the cost of $U_{\alpha,S}^{-1}$ is:

$$2\lceil \log n \rceil + \lceil \log d \rceil (2d^2 + 8d + 2) + \frac{5}{2}d^2 + \frac{11}{2}d + 1$$

TOF gates, $\frac{d}{2}\lceil \log d + 1 \rceil + d^2 + 3d$ CNOT gates, $2d + 1 + \lceil \log d \rceil$ zeroed qubits taking into account the cost saving of doing $U_{\alpha,S}^{-1}(I - 2|\phi_{1,S}\rangle\langle\phi_{1,S}|)U_{\alpha,S}$ and an additional T-depth of

$$4d^2(\log(d/\epsilon) + 5/2) - 12d(\log(d/\epsilon) + 5/2) + 8\log(d/\epsilon) + 20$$

due to error correction for maximum error ϵ.

Parallelizing Steps 5 & 7. In the case where calls to P dominate the cost, which we will see is our case, Step 5 and its inverse Step 7 can easily be parallelized by making d copies of $|l\rangle|v\rangle$ and then calling P to $|l+1\rangle|v_1\rangle \ldots |v_l\rangle|w\rangle$ for each $w \in [d]$ in parallel. This reduces the depth of R_A from $2d+2$ calls to P to 4 calls to P. The additional cost is $2d(\lceil \log n \rceil + n\lceil \log d + 1 \rceil)$ CNOT gates.

4.3 The function $U_{\alpha,S}$

Performing the map $U_{\alpha,S}$ for variable degree requires us to implement the map

$$|S\rangle|0\rangle \mapsto |S\rangle \frac{1}{\sqrt{|S|}} \sum_{S_i \in S} |i\rangle .$$

For this we will need $d+1$ ancillary qubits for the transformation starting in $|1\rangle$, which we will call $|\sigma\rangle$, and $|s\rangle$ of size $\lceil \log d \rceil$ to store the size of S. To calculate the size of S we calculate the Hamming weight of the string $|S\rangle$, which we will put into ancillary qubit array $|s\rangle$. To implement the Hamming weight function, we need at most $2d-2$ Toffoli gates, using the design from [BP05] (lower bounding the Hamming weight of d to 1). The classical design can be turned into a quantum circuit by replacing the conjunction gates with Toffoli gates and XOR gates with CNOT gates, with an additional $d-1$ Toffoli gates to clean up zeroed ancillary space. The CNOT complexity of this is bounded by $4d-2$, modifying adder design which is described in [BP05]. We also need to distinguish the cases: use a $\lceil \log n \rceil$-Toffoli gate to get ancillary qubits $|\beta\rangle = 1 (l = 0)$. Next we fix the state of $|*\rangle$ by applying a rotation. To do this, we have to repeat for $i = 0, \ldots, d$:

1. Check if $s = i$ using some NOT gates and one $\lceil \log d \rceil$-Toffoli gate to add everything together, putting the result in ancillary qubit $|c_6\rangle$.
2. Controlled by $|c_6\rangle$ and $|\beta\rangle$ perform the rotation $|1\rangle \to \frac{1}{\sqrt{ni+1}} |0\rangle + \sqrt{\frac{ni}{ni+1}} |1\rangle$ on the last qubit of $|\sigma\rangle$ using $R_{\sin^{-1}(\frac{1}{\sqrt{ni+1}})}$, $R_{-\sin^{-1}(\frac{1}{\sqrt{ni+1}})}$ and 2 TOF gates.
3. Controlled by $|c_6\rangle$ and NOT $|\beta\rangle$ perform the rotation $|1\rangle \to \frac{1}{\sqrt{i+1}} |0\rangle + \sqrt{\frac{i}{i+1}} |1\rangle$ on the last qubit of $|\sigma\rangle$ using $R_{\sin^{-1}(\frac{1}{\sqrt{i+1}})}$, $R_{-\sin^{-1}(\frac{1}{\sqrt{i+1}})}$ and 2 TOF gates.
4. Uncompute $|c_6\rangle$. This can be done with only 1 Toffoli gate by ignoring the cleanup of the $\lceil \log d \rceil$-Toffoli gate in Step 1 and keeping the ancillary qubits around, cleaning them up now.

This also provides a sketch for creating the rest of the desired superposition. Figure 4 shows an example of how to create a uniform superposition over 3 qubits. We will do the same over a variable number of qubits.

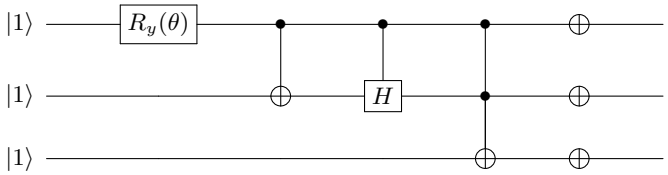


Fig. 4: Circuit to turn $|111\rangle$ into $(|100\rangle + |010\rangle + |001\rangle)/\sqrt{3}$. $\theta = 2\cos^{-1}(1/\sqrt{3})$.

The remaining d qubits of $|\sigma\rangle$ will function as the d elements of S. We will also need d more ancillary qubits to save some multi-Toffoli gates, $|t_1\rangle, \ldots, |t_d\rangle$ starting in $|0\rangle$. For $i = 1, \ldots, d$:

1. Apply a Toffoli gate to $|t_i\rangle$ controlled by $|t_{i-1}\rangle$ and $|\sigma_{i-1}\rangle$ to make sure $|\sigma\rangle$ is 1 up to this point. At $i = 1$ use a CNOT gate controlled by $|\sigma_{d+1}\rangle$ instead.
2. Now, for $j = 1, \ldots, d+1-i$:

(a) Check if $|s\rangle = j$ using some NOT gates and one $\lceil \log d \rceil$-Toffoli gate, putting the result in $|T_{i,j}\rangle$.
(b) If $j > 2$: controlled by $|T_{i,j}\rangle$, $|t_i\rangle$ and $|S_i\rangle$ perform the rotation $|1\rangle \to \frac{1}{\sqrt{j}}|0\rangle + \sqrt{\frac{j-1}{j}}|1\rangle$ on $|\sigma_i\rangle$ using $R_{\sin^{-1}(\frac{1}{\sqrt{j}})}$, $R_{-\sin^{-1}(\frac{j-1}{\sqrt{j}})}$, 4 TOF gates and 2 CNOT gates.
(c) If $j = 2$: controlled by $|T_{i,j}\rangle$, $|t_i\rangle$ and $|S_i\rangle$ perform a NOT gate and a conditional Hadamard gate on $|\sigma\rangle_i$ to perform the rotation $|1\rangle \to |+\rangle$, using 4 TOF gates and 3 CNOT gates.
(d) If $j = 1$: controlled by $|T_{i,j}\rangle$, $|t_i\rangle$ and $|S_i\rangle$ perform a 3-Toffoli gate on $|\sigma_i\rangle$ to perform $|1\rangle \to |0\rangle$.
(e) Uncompute $|T_{i,j}\rangle$, using the same strategy as for c_6.

3. Perform a conditional decrementer gate on $|s\rangle$ conditioned by S_i so it is equal to the number of remaining entries of $|S\rangle$.

We can optionally uncompute $|t_1\rangle, \ldots, |t_d\rangle$ at the end using half of a d-Toffoli gate (Step 1 is the first half of this d-Toffoli gate). $|s\rangle$ is always $|0\rangle$ at the final step of the algorithm. Apply a NOT gate to every qubit of $|\sigma\rangle$ to see that we are close to the desired superposition:

$$|\beta\rangle|\sigma\rangle = |0\rangle\left(\frac{1}{\sqrt{|S|+1}}\left(|0^d 1\rangle + \sum_{S_i \in S}|0^{i-1}10^{d-i}0\rangle\right)\right) + |1\rangle\left(\frac{1}{\sqrt{n|S|+1}}\left(|0^d 1\rangle + \sqrt{n}\sum_{S_i \in S}|0^{i-1}10^{d-i}0\rangle\right)\right).$$

Next, apply NOT to qubits of $|a\rangle$ so $|*\rangle \to |0\rangle$. Then, for each $i = 1, \ldots, d+1$, apply CNOTs controlled by $|\sigma_i\rangle$ to $|a\rangle$ to set it to $|S_i\rangle$ for $i = 1, \ldots, d$ and $|*\rangle$ for $i = d+1$. Finally, uncompute $|\sigma\rangle$ with $d+1$ $\lceil \log d + 1 \rceil$-Toffoli gates and $|\beta\rangle$ with a $\lceil \log n \rceil$-Toffoli gate. The number of Toffoli gates in $U_{\alpha,S}$ is

$$2\lceil \log n \rceil + \lceil \log d \rceil (2d^2 + 8d + 2) + \frac{5}{2}d^2 + \frac{15}{2}d - 1,$$

the number of CNOT gates is $\frac{d}{2}\lceil \log d + 1 \rceil + d^2 + 7d - 2$ and the number of ancillary qubits is $2d + 1 + \lceil \log d \rceil$, skipping uncomputation of $|\beta\rangle$ and $|t_1\rangle, \ldots, |t_d\rangle$. However, since we do $U_{\alpha,S}^{-1}$ first, some operation on $|a\rangle$ and then $U_{\alpha,S}$, we can skip the computation of $|s\rangle$ as well, saving $2d-2$ Toffoli gates and $4d-2$ CNOT gates, and only uncompute $|\beta\rangle, |t_1\rangle, \ldots, |t_d\rangle$ instead of computing them.

Dealing with errors from the rotations. When applying the rotation gates, we introduce an error [RS16]. Because the final superposition will have up to d rotations[1] applied, the total maximum error is $\max(\epsilon_i, 1 - \Pi_{i=1,\ldots,d}(1-\epsilon_i))$ where ϵ_i is the error at layer i. If we want to bound the error by some target ϵ, we can set ϵ_i to $1 - \sqrt[d]{1-\epsilon} \approx \epsilon/d$. Since each rotation gate has worst case T-depth $4\log(1/\epsilon_i) + 10$ [Sel15,RS16, Section 8.3] and we do a total of $d(d+1) - 4d + 2$ rotations, we get a total T-depth of $4d^2(\log(d/\epsilon) + 5/2) - 12d(\log(d/\epsilon) + 5/2) + 8\log(d/\epsilon) + 20$, which is dominated by $4d^2\log(d/\epsilon)$ and cannot be parallelized.

[1] At each $i = 1, \ldots, d$, only 0 or 1 rotation gates actually introduce this error. Conditional rotation gates where the condition is not met cancel the error introduced, since $\mathrm{NOT}R_z(-\theta)\mathrm{NOT} = R_z(\theta)$, meaning they have the same error.

4.4 Bounds and arithmetic in the predicate

A predicate function P is used in Step 1, 5, 7 & 8 for the implementation of R_A (and R_B) in Subsection 4.2. In this subsection, we will discuss some bounds and the main arithmetic used in the predicate. The implementation of the predicate and resources are presented in Subsection 4.5.

The predicate takes inputs of the form $|l\rangle |v_n\rangle \cdots |v_{n-l+1}\rangle |*\rangle$ for $1 \leq l \leq n$ and checks whether it satisfies Inequality (1). It returns one of false, satisfied or true. Note that the true is triggered when the inequality is satisfied and the level is $l = n$. Thus we may simply assume $P(\bullet)$ returns either false or satisfied. Observe that here we denote the index i in reverse order to be consistent with the notation in Inequality (1). Thus the main computational step at the l-th level (e.g., with input $|l\rangle$) in the predicate is to check if v_j satisfies:

$$\underbrace{\sum_{j \geq n+1-l}}_{(III)} (\underbrace{\sum_{i \geq j}}_{(II)} \underbrace{\mu_{i,j} \cdot v_i}_{(I)})^2 \cdot \underbrace{\|\mathbf{b}_j^*\|^2}_{(IV)} \leq R^2, \tag{2}$$

where R is the enumeration radius (e.g., $R = \|\mathbf{b}_1\|$ or $R = \mathrm{GH}(\mathcal{L})$). In the case of pruning, R can be replaced by a pruning function R_l that can be pre-computed classically. For convenience, we mark the steps of the computation by $(I) - (IV)$ that we will refer to later.

Simplified model. We make a few simplifications to the description, to avoid the implementation becoming overly complicated and unreadable. First, Step 5 of Algorithm 3 of Montanaro [Mon18] (see Subsection 4.2) asks to check $\forall w \in [d]$, if $P(|l+1\rangle |v_n\rangle \ldots |v_{n-l+1}\rangle |w\rangle)$ is not false. These superpositions $|v_i\rangle$ and $|w\rangle$ stand for indices of the nodes (say, as the children). However, to compute the predicate, one requires the superpositions corresponding to the actual values of the v_i. It is not efficient to store these superpositions, but one can compute them on-demand given the inputs $|l+1\rangle |v_n\rangle \ldots |v_{n-l+1}\rangle$. We denote such a procedure as a *"translator"* which translates the input indices into the concrete values. The algorithm will start with the first index $|v_n\rangle$ (indexing to children) and then compute the concrete value corresponding to the index. One does this iteratively, and similarly for the new input $|w\rangle$, it determines which (indices of) children $|w\rangle$ to keep in the walk (e.g., set $S = S \cup \{w\}$ in Step 5). This procedure invokes at most n executions of the predicate P in serial with inputs $\{|v_n\rangle \cdots |v_{n-i+1}\rangle\}$ for $i = 1$ to l. We will factor such depth/size increment into the final estimate. Note that Step 5 for different inputs $|w\rangle$ can be parallelized. Thus, we may abuse the notation for v_i, which denotes both the index of the node and the actual value.

In the predicate, we will need to use the elements $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$, which are classic data. The address indices (e.g., i, j) are also classic, so this can be done in a quantum circuit model using a universal quantum gate set. The Gram-Schmidt coefficients μ_{ij} for $i > j$ take about $n^2/2$ registers, each of p bits. We will need the squared vector norm $\|\mathbf{b}_i^*\|^2$ for all i, which takes $n \cdot p$ bits. We will omit this circuit for simplicity. There might be a need to use the qRAM for the case of extreme pruning, but in this work, we only need to store/access the classical information so a plain quantum circuit would work.

Let d denote the maximum degree of the enumeration tree, B be a bound for $\max_i |v_i|$ and n denote the tree depth (e.g., the lattice dimension or the transformed binary tree height as in [ANS18]). In enumeration, floating-point arithmetic is also needed, thus we denote p as the precision required. We will discuss p, d, B as functions of n.

Precision requirement. In a standard implementation of lattice enumeration and lattice reduction, one uses floating-point numbers to speed up the computation (instead of working in $\mathbb{Q}$). For example, this is the approach used in the FPLLL library [dt22] and the progressive lattice reduction library [AWHT16].

To guarantee the correctness of lattice enumeration, one would require sufficient precision working over floating point numbers. Equivalently, this is the required precision for the enumeration tree to contain the solution node. The precision required in lattice enumeration over floating point numbers has been studied in [PS08]. Theorem 3 of [PS08] shows that to solve SVP-γ where $\gamma = 1.01$ the precision of the floating point numbers used needs to be $\Theta(n)$ for an n-dimensional lattice to guarantee correctness. Similarly, the precision required in an LLL reduction over floating point numbers (one can think of this as the starting step of a lattice reduction algorithm) cannot be too small. This has been studied in Theorem 1 of [NS05]. Asymptotically, a floating-point precision $\log_2(3)n \approx 1.6n + o(n)$ suffices. In practice, however, the constant 1.6 can be reduced to about 0.3 in folklore. Heuristically, the enumeration works on a more reduced basis than LLL-reduced and it requires smaller precision than that of LLL. Thus we will use the folklore value and assume the precision p' required for a correct enumeration is $p' \approx 0.3n$.

Our targeted dimension n for lattice enumeration would be about 400. For example, Kyber-512 [ABD+21] requires BKZ blocksize 406, Dilithium (security level 2) [BDK+21] requires blocksize 423 and Falcon-512 requires [FHK+20] blocksize 411. This heuristically requires a precision of $p' \approx 120$ bits. For larger dimension, the precision can be increased as needed.

Bounds on tree degree. The enumeration tree size $\mathcal{T}$ can be bounded by $n^{n/2e+o(n)}$, by the analysis in [HS07]. In practice, the tree size can be estimated using the Gaussian heuristic, given the lengths of Gram-Schmidt vectors. In an ideal case, $d = O(n)$ by taking $d^n \approx n^{n/2e+o(n)}$ if one assumes the nodes are uniformly distributed over a perfect d-ary tree. This argument is not true. For example, in a LLL-reduced basis, $\mathbf{b}_n^*$ could already be about 2^{n-1} times smaller than the enumeration radius R (say, $\|\mathbf{b}_1\|$). Thus the number of v_n can be already huge. The situation is much better if the input basis is preprocessed. It is conceptually simpler to assume the basis is already well-preprocessed: e.g., we assume the input basis is already HKZ reduced and revisit the enumeration. Alternatively, one can preprocess the basis with blocksize $n - o(n)$ which amortizes the running-time and leads to a similar quality (e.g., see Theorem 5 of [ABF+20] for a similar approach). We discuss several methods to bound d. For convenience, we let d_i denote the maximum degree for nodes at the i-th level in the tree. This corresponds to the maximum number of choices for v_i over fixed $\{v_{i+1}, \ldots, v_n\}$'s.

To begin, we bound the number of choices for v_n. One can model the geometry of an HKZ-reduced basis with the following (logarithmic) Gram–Schmidt length profile, following the same approach as in [HPS11,ABF+20].

$$u_k = (\ln k)/2 + (u_1 + u_2 + \cdots + u_k)/k, \ \forall k \leq n, \ \text{and} \ u_1 = 0. \tag{3}$$

In this equation, we let $u_1, \ldots, u_n$ be the Gram–Schmidt log-norms in reversed order of an HKZ-reduced basis. The low-order terms are omitted. Lemma 2 of in [ABF+20] bounds the u_i by

$$(\ln^2 k)/4 + (\ln k)/2 \leq u_k < (\ln^2 k)/4 + (\ln k)/2 + 1.$$

Thus the number of choices for v_n can be bounded by $2\|\mathbf{b}_1\|/\|\mathbf{b}_n^*\| + 1 \approx 2e^{u_n - u_1} + 1 \approx n^{(\ln n)/4}$. We have assumed the HKZ Gram–Schmidt profile fitting the exact form in Equation (3). Alternatively, if one can bear with the Geometric Series Assumption (though it seems not plausible for the HKZ case) of ratio $r \approx n^{1/n}$, one can obtain a much better bound for v_n which is about n. We can then bound degrees for the remaining v_i. Observe that, when the Gram-Schmidt vectors have a non-increasing length profile (e.g., $\forall i, \|\mathbf{b}_i\|^* \geq \|\mathbf{b}_{i+1}\|^*$), the number of choices for v_i is no larger than the number of choices for v_{i+1}.

A better method, in practice, is to use the concrete values of each $R/\|\mathbf{b}_i^*\|$ to bound the degree d_i. This also works when the Gram-Schmidt vectors' lengths are not monotonic. Note the maximum number of choices d_i for v_i can be solely bounded by R and $\|\mathbf{b}_i^*\|$. To facilitate a running example discussion, we take $d \approx n^2$ (the exponent 2 is motivated by plugging $n \approx 400$ into the asymptotic $n^{(\ln n)/4}$). The final estimates will be given in a parameterized form and can be adapted to other parameters.

Bounds on coefficients. We will also need to bound $B = \max_i |v_i|$. This will be used to determine the arithmetic discussed in the next paragraph. If a basis satisfies the so-called dual Korkine-Zolotarev condition, then B can be bounded by $n^{1.5}$ [HR14]. However, a dual Korkine-Zolotarev reduction requires to solve the SVP problem. Albrecht, Prokop, Shen and Wallden [APSW22] have bounded each $|v_i| \leq R \cdot \|\mathbf{b}_i^\dagger\|$, where $\mathbf{b}_i^\dagger$ denotes the i-th vector in the dual basis of $\mathbf{B}$ (see Lemma 1 of [APSW22]). This gives a useful method to compute the bound in practice, e.g., one can set some radius R and then take the largest $\|\mathbf{b}_i^\dagger\|$ to bound B. This can be done classically before the start of quantum tree backtracking. We can also bound B by Inequality (2). Following our running example, v_n can be bounded by n^2. Working backwards on Inequality (2) and using that $\|\mu_{i,j}\| \leq 1/2$, the largest v_i can be bounded by $O(n^4)$ in our example.

Fixed-point arithmetic. To compute the predicate, Inequality (2) involves floating point addition and multiplications of p'-bits. To the best of our knowledge, the quantum circuits for floating point arithmetic are much more expensive than those for integer arithmetic. To deal with the lack of efficient circuits, we will normalize the numbers to fixed-point numbers, with adequate precision.

For example, Haener, Soeken, Roetteler and Svore [HSRS18] have studied floating-point addition and multiplication circuits based on reversible networks

that are obtained from a LUT-Based hierarchical reversible logic synthesis approach [SRWDM17]. A hand-optimized circuit is also proposed in the same work, that improves the automatically generated circuits. In general, floating-point circuits require shifting and re-normalization gates, which are expensive to construct in general. For example, the floating-point circuit described in [SRWDM17] for a 64-bit adder has T-count 26348 and T-depth 7224.

The method of [HSRS18] works for general floating-point arithmetic. We observe that the computation in Inequality (2) is often simpler than generic floating-point arithmetic. For example, Step (I) in Inequality (2) involves integer-floating point multiplication $\mu_{i,j} \cdot v_i$. One can use repeated additions $\sum_{\#v_i} \mu_{i,j}$ and thus this boils down to an addition of two floating points. This floating point arithmetic can be emulated by integer arithmetic with the same precision/mantissa, as long as no overflow happens. We can use the quantum circuit for integer addition by Takahashi, Tani and Kunihiro [TTK10]. Their construction has size $7p+O(1)$, with circuit depth $5p+O(1)$ and Toffoli depth $2p+O(1)$.

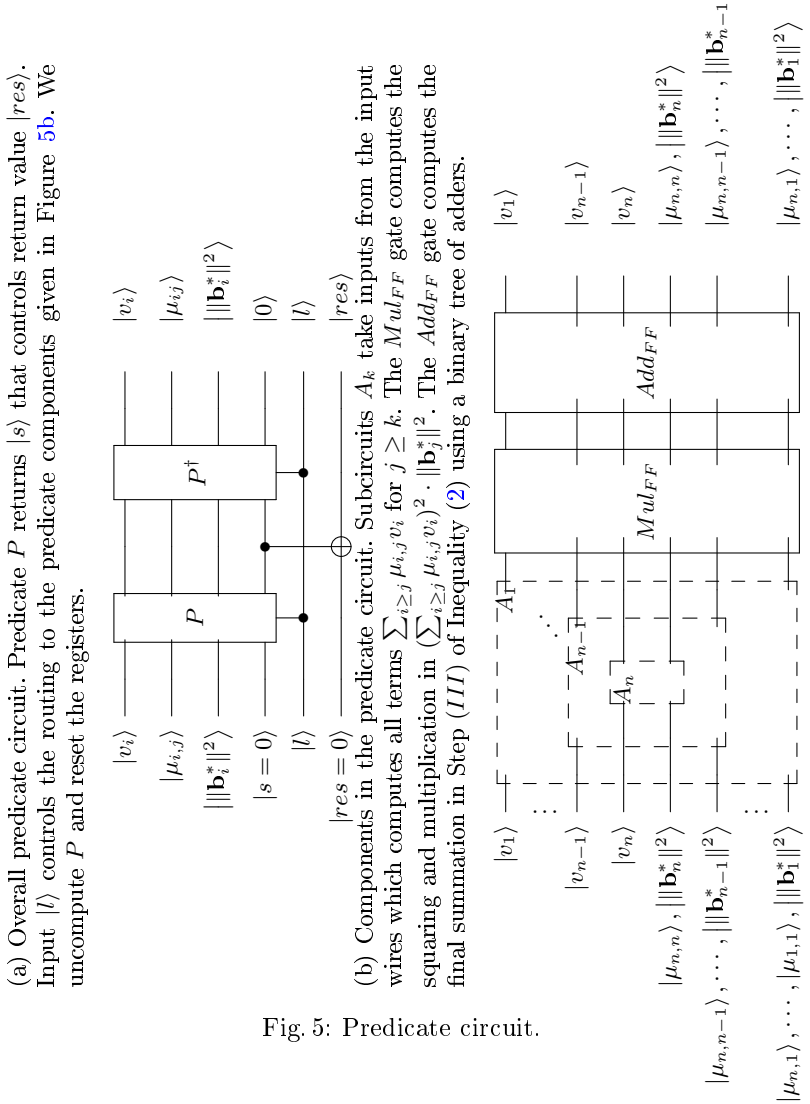
We will need to set the precision to accommodate sufficient mantissa and to avoid overflow. Starting with the precision p' (say, ≈ 120 in our running-example) for the floating-point mantissa. Step (I) of Inequality (2) increases the bit size by at most $\log B$. Step (II) of Inequality (2) increases the bit size by at most $\log n$. The squaring and multiplication after Step (II) increase the bit size by at most 4 times. In the end, Step (III) has a summation of at most n items. Therefore, one can set the precision $p = 4(p' + \log(B) + \log(n)) + \log(n)$. This is $\approx 4p' + 21 \log n$ for our running example. Thus one can use fixed-point arithmetic such as the integer arithmetic with precision p.

To do this, we can normalize the $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$ before starting the quantum step (e.g., similar normalization on the exponent has been used in the FPLLL library [dt22] for a different context). When the $\mu_{i,j}$'s (and $\|\mathbf{b}_i^*\|^2$) have similar magnitudes among themselves (e.g., similar exponents in the floating-point representation), the normalization will not cancel out any values. We have assumed the $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$ have similar magnitudes (e.g., see Appendix A for experiments on this). If this is not the case, one can use more precision to take care of this. For values $\|\mathbf{b}_i^*\|^2$, the maximal difference on the magnitude heuristically happens on $\|\mathbf{b}_1\|^2/\|\mathbf{b}_n^*\|^2$. This is asymptotically $n^{\ln n/2}$ for an HKZ reduced basis and thus adds $\approx \ln^2 n$ to the precision. The distribution of $\mu_{i,j}$ has been studied in the context of random sampling methods in [Sch03], and assumed to be uniform in $[-1/2, 1/2]$ in those scenarios. If we assume so, an additional precision of $O(\log n)$ with some small hidden constant is sufficient. In experiments, they seem to concentrate towards the boundary for those $i \approx j$ for LLL-reduced bases (see [NS06] for the distribution of $\mu_{i,j}$ in experiments). In Appendix A, we demonstrate further experiments on this. To summarize, one can take $p \approx 4p' + 4 \log B + \log^2 n + 7 \log n \in \Theta(n)$.

For our running example, e.g., $p \approx 10p'$ is sufficient for our targeted dimension. The value 10 here is chosen for convenience and can be substituted by a practical value given the input, which is likely to be much smaller.

4.5 Implementation of the predicate

The circuit of predicate P is given in Figure 5a which consists of several components as plotted in Figure 5b. We start with the description of the registers. First, the inputs of the predicate are superpositions $|l\rangle$ and $|v_i\rangle$. We need two input registers: for $|l\rangle$ this is $\log n$ bits, and for $|v_i\rangle$ this is $n \log d$ bits. We also need to access classic data $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$, and this can be done by using universal quantum gate set. We need an addressing register of $\log n$ bits and an output register of $(n^2/2+n)\cdot p$ bits. In total, the input size is $\log n + n \log d + (n^2/2+n)\cdot p$ bits. We also need poly(n) ancillary qubits for intermediate results that we will omit in the discussion. The circuit starts by looking at the input $|l\rangle$ and decides the routing to the arithmetic components. We will thus focus on the arithmetic gates which are the dominating components.



(a) Overall predicate circuit. Predicate P returns $|s\rangle$ that controls return value $|res\rangle$. Input $|l\rangle$ controls the routing to the predicate components given in Figure 5b. We uncompute P and reset the registers.

(b) Components in the predicate circuit. Subcircuits A_k take inputs from the input wires which computes all terms $\sum_{i\geq j} \mu_{i,j} v_i$ for $j \geq k$. The Mul_{FF} gate computes the squaring and multiplication in $(\sum_{i\geq j} \mu_{i,j} v_i)^2 \cdot \|\mathbf{b}_j^*\|^2$. The Add_{FF} gate computes the final summation in Step (III) of Inequality (2) using a binary tree of adders.

Fig. 5: Predicate circuit.

For fixed l and v_i, the main task is to compute the left-hand side of Inequality (2). We describe the implementation and resource estimate. In Figure 5b, each of the boxes A_{n-l+1} denotes a subcircuit, which computes the terms $s_j := \left(\sum_{i \geq j} \mu_{i,j} v_i\right)$ for $j \geq n-l+1$. For a fixed input l, there are l such terms s_j to compute in the circuit A_{n-l+1}. The computation of s_j, for different indices j, can be parallelized by making at most n copies of each $|v_i\rangle$ (observe that the same v_i's are used at most n times in different indices j). One can also note that $A_k \subset A_{k-1}$ for $k \in (1, n]$ as A_k computes all the terms required by A_{k-1} except s_{k-1}. Thus A_l can be regarded as a subcircuit of A_{l-1}. The predicate circuit starts by looking at the input $|l\rangle$ and deciding the routing to the required A_{n-l+1}. We discuss the computation in A_{n-l+1}.

Step 1. First, the circuit computes each of the terms $\mu_{i,j} \cdot v_i$ (for fixed i, j), one can use repeated addition of at most B numbers at precision p. A serial implementation requires a depth $O(B)$ of adders. One can parallelize this using a binary tree of adders. This has a depth of $\lceil \log B \rceil$ in terms of adders. Using [TTK10], each circuit (for computing a single $\mu_{i,j} v_i$) has a Toffoli depth of $\approx (2p + O(1)) \log(B)$ and circuit size bounded by about $(7p + O(1))B$.

Step 2. Within the computation of s_j for a fixed j, the computation of at most n terms $\mu_{i,j} v_i$ for different i can be parallelized. Thus the circuit has about the same depth as in Step 1, but with an increment of n times on the size.

Step 3. To compute $s_j = \left(\sum_{i \geq j} \mu_{i,j} v_i\right)$ for a single j, one can again use a binary tree of adders. This adds the Toffoli depth by $(2p + O(1)) \log(n)$ and circuit size by $(7p + O(1))n$.

Step 4. As discussed above, the computation of s_j for different indices j can be made in parallel by making at most n copies of the input. This process requires a poly number of the CNOT gates which we will omit. Therefore the circuit depth remains similar and size increases by at most n times. The circuit discussed so far has Toffoli depth of $2p(\log B + \log n) + O(\log B) + O(\log n)$ and size about $n^2(7p(B+1) + O(B) + O(1))$. This completes all the computation in A_{n-l+1} for the input l.

Step 5. Given all s_j's for $j \geq n-l+1$, we need to compute $\sum_{j \geq n-l+1} s_j^2 \|\mathbf{b}_j^*\|^2$. This involves squaring, multiplication and a final summation.

- The squaring and multiplication are done by the subcircuit Mul_{FF} in Figure 5a. Computing $s_j^2 \|\mathbf{b}_j^*\|^2$ for different j can be done in parallel. For integer squaring/multiplication, we use the quantum circuit designed by Parent, Roetteler and Mosca [PRM17]. This is a Karatsuba based circuit which fits our input size. Their construction has (Toffoli) size and depth bounded by $98p^{1.585}$ and $p^{1.158}$ respectively.
- The final summation, over $s_j^2 \|\mathbf{b}_j^*\|^2$ for all $j \geq n-l+1$, can be implemented by a binary tree of adders, which has a Toffoli depth $(2p + O(1)) \log n$ and circuit size $(7p + O(1))n$.

Step 6. There is a final comparison to decide the returned value, which we ignore.

Taking into account the $P^\dagger$, the predicate circuit has Toffoli depth about $4p(\log B + 2\log n + p^{0.158}) + O(\log B) + O(\log n)$. For usual parameters, $O(\log n)$ subsumes in $O(\log B)$ where the hidden constants are small integers. Thus it becomes $4p(\log B + 2\log n + p^{0.158}) + O(\log B)$. The precision $p \approx 1.2n + 4\log B + \log^2 n + 7\log n \in \Theta(n)$ for most inputs of interest. If the bound B is large, the term $(4p \log B)$ (e.g., this term corresponds to the repeated addition of $O(B)$ terms) dominates the running-time. If the bound $B = \text{poly}(n)$, the multiplication $4p^{1.158}$ could dominate the running-time. The Toffoli size is bounded by $14pn^2B + n^2O(B) + 14n^2p + O(n^2)$. Note the constant in $O(B)$ is small, but the constant in $O(n^2)$ subsumes terms like $196p^{1.585}$.

We discuss several usual cases that are relevant in either theory or practice, depending on the relation of B with respect to dimension.

(i) In an LLL-reduced basis, $B \approx 2^n$ which will dominate the size. The Toffoli depth is about $4p(n + O(n^{0.158}))$ and size is about $(14p + O(1))n^2 2^n$.
(ii) For an HKZ-reduced basis, $B \approx n^{\ln n}$. The Toffoli depth is bounded by $4p(p^{0.158} + O(\log^2 n))$ and size is about $(14p + O(1))n^2 n^{\ln n}$.
(iii) In our running-example, we took $B \approx n^4$ and $p = 3n$, the Toffoli depth and size have order $12n(6\log n + (3n)^{0.158}) + O(\log n)$ and $42n^7 + O(n^6)$).
(iv) In practice, for cryptographic size lattices, it is reasonable to have $B \approx n^2$. The Toffoli depth and size have order $4p(4\log n + p^{0.158}) + O(\log n)$ and $14pn^4 + O(n^4)$.

Finally, we note this is a single evaluation of the predicate. As discussed in the "Simplified Model" paragraph of Subsection 4.4, one needs to iteratively compute the values, which scales the depth/size by a factor of at most n.

4.6 Summary of resource estimates

We summarize the resource estimates developed from the previous subsections. In a single controlled-$R_B R_A$ operator, there are two expensive operations. First, the predicate is called in Steps 1 and 5, where the latter can be parallelized. One can scale the depth by 2 and size by $d + 1$. In addition, the uncomputation in Steps 7 and 8 scale the depth by another factor of 2 and size by $d + 1$. Second, the operation $I - 2\left|\phi_{1,S}\right\rangle\left\langle\phi_{1,S}\right|$ is performed on $|a\rangle$ in Step 6. We factor out the Toffoli gates and give the resource estimates in terms of T size and depth. We use the Toffoli gate as implemented in [Sel13] where Toffoli uses four T-gates and has a T-depth of one. Overall, the circuit has T-depth bounded by

$$16np(\log B + 2\log n + p^{0.158}) + O(n\log B) + 8d^2\log(d/\epsilon) + 4d^2\log d + O(d^2)$$

and T-size bounded by

$$8(d+1)(14pn^2(B+1) + O(n^2B)) + 8d^2\log(d/\epsilon) + 16d^2\log d + O(d^2)$$

In practice, for cryptographic size lattices where $n \gtrapprox 400$, it is reasonable to expect $d \approx n, B \approx n^2$ and $p \leq 3n$ (see Appendix A for experiments on this),

where the T-depth and size have order $48n^{2.158} + 8n^2 \log(n/\epsilon) + O(n^2 \log n)$ and $336n^6 + 8n^2 \log(n/\epsilon) + O(n^5)$.

Overall, we aim for a constant failure probability of $\delta \approx 0.1$. The number of controlled-U operators applied can be bounded by about $32\sqrt{\mathcal{T}n}$ (see [CKM19]). We factor out the total error into ϵ by a scalar of $\sqrt{\mathcal{T}n}$ and the T-depth and size have order $O(\sqrt{\mathcal{T}n}\,(n^{2.158} + n^2 \log \mathcal{T}))$ and $O(\sqrt{\mathcal{T}n}\,(n^6 + n^2 \log(\mathcal{T})))$.

One can also apply the resource estimates on the transformed binary tree, following the method proposed in [ANS18]. In such a case, one should also modify the tree height from n to $n \log d$ in the analysis, as well as investigate the overheads brought in due to the transformation. We leave this as future work.

A Experiments supporting the heuristics

In this section, we provide some additional experiments to verify some heuristics we have made in the previous section for the resource estimates. These heuristic arguments are not strictly required for the quantum resource estimates as the estimates were presented in parameterized form (in terms of tree depth, degree, bound for v_i and precision). However, readers may find them useful for a more precise estimate given practical parameters of cryptographic relevance.

A.1 Experiments on the bounds for d and B

First, Subsection 4.4 uses a running example where $d \approx n^2, B \approx n^4$. It was also used that: "in practice, for cryptographic size lattices, it is reasonable to expect $d \approx n, B \approx n^2, p \approx 3n$" for a well-preprocessed basis. Notice that such heuristics are not asymptotically correct, even on a HKZ reduced basis. However, we are mostly interested in practical parameters for cryptographic size lattices. We conduct some experiments to verify that it is reasonable to bound $d \approx n$ and $B \approx n^2$.

In the first experiment (Figure 6), we took the top-25 solved SVP problems from the SVP Challenge project[2], and checked the bound for B with respect to the actually found short vector $\mathbf{v}$ as well as the bound for d (given a preprocessed basis). The dimensions of these solved SVP problems are $\{140 - 158, 160, 162, 170, 176, 178, 180\}$. In case there are multiple solutions, we take the solution $\mathbf{v}$ of the smallest norm. For each input basis of dimension n, we preprocess the basis by BKZ$-(\lceil \frac{n}{3} \rceil)$ and then compute the coefficients of $\mathbf{v}$ with respect to the reduced basis $\{\mathbf{b}_i\}_i$. The BKZ preprocessing is conducted using FPYLLL[3] with early abort and the default extreme pruning profile.

In the end, we compute two values: the maximum size of v_i's as in $\mathbf{v} = \sum_i v_i \mathbf{b}_i$ and $\|\mathbf{v}\|/\|b_n^*\|$. The first value can be used as B and the second value can be used to bound d (assuming a non-decreasing Gram–Schmidt length profile). We also scale the first value by $1/n^2$ and the second value by $1/n$ to give an idea

[2] https://www.latticechallenge.org/svp-challenge/
[3] https://github.com/fplll/fpylll

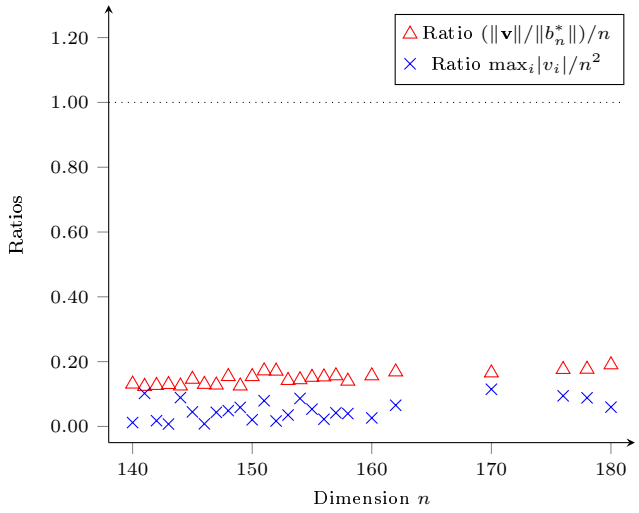


Fig. 6: Bounds d and B, based on solved SVP Challenges.

about B/n^2 and d/n (other scaling factors could be chosen). The results are plotted in Figure 6, where the x-axis denotes the input dimension and the y-axis denotes the two ratios (red/blue). One can observe that, in these experiments, the ratios $\max_i|v_i|/n^2$ and $(\|\mathbf{v}\|/\|b_n^*\|)/n$ are always smaller than 0.2. Thus it seems reasonable to assume the bounds $d \approx n, B \approx n^2$.

In the second experiment (Figure 7), we further check the bound for B in both experiments and simulations, using the Gaussian heuristic. In the simulation, we generate lattices of rank $n = \{60, 68, 76, \cdots, 500\}$ using the SVP Challenge generator. Each lattice is preprocessed by BKZ$-(\lceil \frac{n}{3} \rceil)$ in simulation. In the end of simulation, the ratio $(GH/\|b_n^*\|)/n$ is recorded, where GH denotes the expected length of shortest vectors indicated by the Gaussian heuristic. It seems that the ratio follows a slightly quadratic function. Fitting using NumPy[4] gives a curve of $1.8886 \times 10^{-6} \cdot n^2 + 0.0003 \cdot n + 0.0454$ which is plotted in Figure 7. One can see this grows very slowly with n. Furthermore, we also conduct BKZ experiments: we generate lattices of rank $n = \{60, 64, 68, \cdots, 192\}$ using the SVP Challenge generator. For each rank, we generate 32 random instances with different seeds. The lattices are then preprocessed by BKZ$-(\lceil \frac{n}{3} \rceil)$. The BKZ preprocessing is conducted using FPYLLL with early abort and the default extreme pruning profile. In the end, the average ratio $(GH/\|b_n^*\|)/n$ is recorded. One can see the experimental results follow very closely with the simulated results (but slightly larger, possibly due to the use of early abort in experiments). One can also notice these bases are weakly reduced, e.g., compared to a HKZ reduced basis. Using heavier preprocessing will lead to even smaller values on bounds B and d. We leave it for future work to develop better practical bounds.

[4] https://numpy.org/

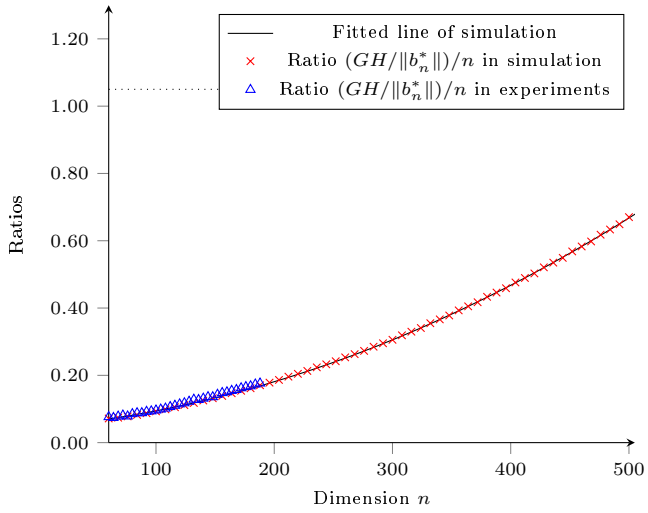


Fig. 7: Bound B, based on BKZ experiments and simulations.

A.2 Experiments on the bounds for $\mu_{i,j}$

Subsection 4.4 also assumed that the $\mu_{i,j}$'s have similar magnitude (see the "Precision requirement" paragraph) in a somewhat reduced basis, so one does not need to increase the precision too much to take care of this.

The distribution of $\mu_{i,j}$ has been studied in [Sch03] in the context of random sampling methods, which is assumed to be uniform in $[-1/2, 1/2]$. By such assumption, the maximal ratio on the magnitude of various $\mu_{i,j}$ is roughly $O(n^2)$. Thus an additional precision of $O(\log n)$ with some small hidden constant is sufficient. In experiments, they seem to concentrated towards the boundary for those $i \approx j$ for LLL-reduced bases, e.g., see [NS06] for the distribution of $\mu_{i,j}$ in experiments. We conduct further experiments: we generate random q-ary lattices using the "IntegerMatrix.random" function in FPYLLL whose dimensions range from $n = 140$ to 320 with a stepping of 2. We process the basis with LLL. In the end, we compute the ratio $(1/\min_{i,j} \mu_{i,j})/n^2$ for $i > j$. For each dimension n, we generated and ran 32 random instances, and calculated the average ratios. We also verified the values of $\mu_{i,j}$ using SageMath's "Matrix" object[5] with rational entries to calculate the Gram-Schmidt coefficient for very high precision (e.g., to avoid potential overflow in floating-point numbers). The results are plotted in Figure 8, where the x-axis denotes the input dimension and the y-axis records the logarithm (base 2) of the averaged ratios. One can observe that the logarithm of the averaged ratios $\log_2((1/\min_{i,j} \mu_{i,j})/n^2)$ is bounded from above by some small constant (≈ 6.1 in this experiment).

Heuristically, the variance in the size of $\mu_{i,j}$ can grow for more reduced bases. Thus we conduct further experiments to check the magnitude of the $\mu_{i,j}$'s for

[5] https://www.sagemath.org/

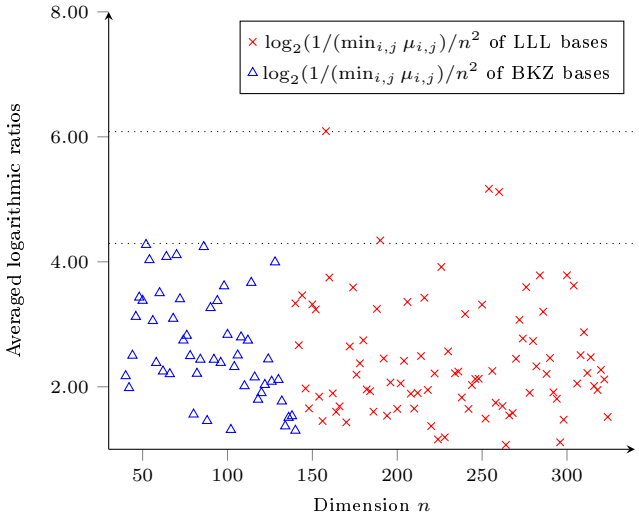


Fig. 8: Experiments on the bound for $\mu_{i,j}$ in LLL/BKZ reduced basis.

reduced bases preprocessed by larger blocksizes. We generate random q-ary lattices ranging from $n = 40$ to 140 and preprocess the basis with BKZ-$\frac{n}{2}$. In the end, we compute the same ratio $(1/\min_{i,j} \mu_{i,j})/n^2$. For each dimension n, we generated and ran 128 random instances, and calculated the average ratios. The results are also given in Figure 8. One can observe that the logarithm of the averaged ratios is also bounded from above by some small constant (≈ 4.3 in this experiment). Thus it seems reasonable to assume an additional precision of $O(\log n)$ with some small hidden constant in the fixed precision arithmetic.

B Constraint satisfaction problem and tree backtracking

A common use of tree backtracking is for solving the so-called constraint satisfaction problem (CSP). This is also the context for which the original algorithm of Montanaro [Mon18] focused on. We give some background on this.

In CSP, we assign values from a fixed list to a fixed number of variables, thus using a tree structure is a natural way to go through every possible answer. For example the boolean satisfiability problem, or SAT, gives us n variables we can assign as either true or false. The question is, given a series of logic statements, can we make the final result true? As a concrete example for $n = 3$, we consider statement $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. To solve this, one can construct a full tree with $n + 1$ layers, each node having 2 children (denoting, true or false), which would allow us to see all possible answers and see if we have a solution. The root corresponds to having no assigned values, with layer i in the full tree assigning true or false to variable x_i, as seen in Figure 9.

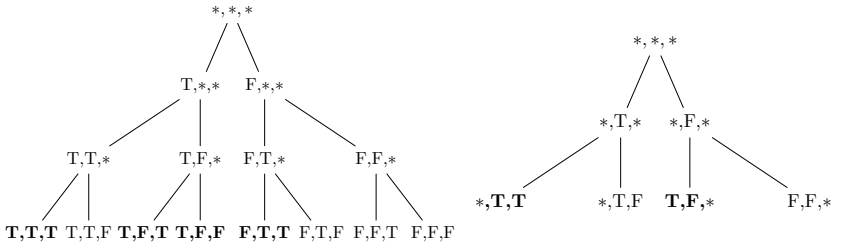


Fig. 9: Full tree (left) and backtracking tree (right) to solve $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$.

Continuing to explore the tree, we can see that if we assign false to both x_1 and x_2, it is not sensible to continue down this path in the tree. We would prefer to cut our losses, go back a bit and continue to the next possibility. This strategy is called backtracking, see also Figure 9. Instead of only checking at the leaves whether our assignment is correct, at each node we check some predicate function, that checks the partial assignment and returns true, false or indeterminate to see how our current assignment is doing. If this predicate returns false, we stop the path there and try to make different choices, if it returns indeterminate or true we continue. Furthermore, assigning a value to x_2 first will solve either $(x_1 \vee x_2)$ or $(\neg x_2 \vee x_3)$. To decide what to assign, we need some heuristic to decide what is the best variable to look at next, in this case looking at the variable that appears in the most parts that are not true yet.

Lattice enumeration constructs such trees as well, which we will denote as "enumeration trees". Inequality (1) suggests that, at the l-th ($l \geq 2$) level of the tree, $v_{n+2-l}, \ldots, v_n$ are already determined. To go down the tree, it remains to bound and select a value (if it exists) for v_{n+1-l} according to Inequality (1). We let d denote the maximal number of choices of v_i. Thus this can be represented by a tree with up to n layers and degree d.

References

ABB+17. Erdem Alkim, Nina Bindel, Johannes A. Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega, *Revisiting TESLA in the quantum random oracle model*, Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017 (Tanja Lange and Tsuyoshi Takagi, eds.), Springer, Heidelberg, 2017, pp. 143–162.

ABD+21. Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé, *CRYSTALS-KYBER: algorithm specifications and supporting documentation*, 2021, Submission to the NIST's post-quantum cryptography standardization process.

ABF+20. Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen, *Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$*, CRYPTO 2020, Part II

(Daniele Micciancio and Thomas Ristenpart, eds.), LNCS, vol. 12171, Springer, Heidelberg, August 2020, pp. 186–212.

ABLR21. Martin R. Albrecht, Shi Bai, Jianwei Li, and Joe Rowell, *Lattice reduction with approximate enumeration oracles - practical algorithms and concrete performance*, CRYPTO 2021, Part II (Virtual Event) (Tal Malkin and Chris Peikert, eds.), LNCS, vol. 12826, Springer, Heidelberg, August 2021, pp. 732–759.

ADH+19. Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens, *The general sieve kernel and new records in lattice reduction*, in Ishai and Rijmen [IR19], pp. 717–746.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe, *Post-quantum key exchange: A New Hope*, Proceedings of the 25th USENIX Conference on Security Symposium (USA), SEC'16, USENIX Association, 2016, p. 327–343.

AGPS20. Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck, *Estimating quantum speedups for lattice sieves*, ASIACRYPT 2020, Part II (Shiho Moriai and Huaxiong Wang, eds.), LNCS, vol. 12492, Springer, Heidelberg, December 2020, pp. 583–613.

Ajt96. Miklós Ajtai, *Generating hard instances of lattice problems (extended abstract)*, in STOC 1996 [STO96], pp. 99–108.

AK17. Andris Ambainis and Martins Kokainis, *Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games*, Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (New York, NY, USA), STOC 2017, Association for Computing Machinery, 2017, p. 989–1002.

AKS01. Miklós Ajtai, Ravi Kumar, and D. Sivakumar, *A sieve algorithm for the shortest lattice vector problem*, 33rd ACM STOC, ACM Press, July 2001, pp. 601–610.

AMMR13. Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler, *A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits*, Trans. Comp.-Aided Des. Integ. Cir. Sys. **32** (2013), no. 6, 818–830.

ANS18. Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen, *Quantum lattice enumeration and tweaking discrete pruning*, ASIACRYPT 2018, Part I (Thomas Peyrin and Steven Galbraith, eds.), LNCS, vol. 11272, Springer, Heidelberg, December 2018, pp. 405–434.

APSW22. Martin R. Albrecht, Miloš Prokop, Yixin Shen, and Petros Wallden, *Variational quantum solutions to the shortest vector problem*, Cryptology ePrint Archive, Paper 2022/233, 2022, https://eprint.iacr.org/2022/233.

AWHT16. Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi, *Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator*, in Fischlin and Coron [FC16], pp. 789–819.

BBTV23. Nina Bindel, Xavier Bonnetain, Marcel Tiepelt, and Fernando Virdia, *Quantum lattice enumeration in limited depth*, Cryptology ePrint Archive, Paper 2023/1423, 2023, https://eprint.iacr.org/2023/1423.

BBVL21. Gustavo Banegas, Daniel J. Bernstein, Iggy Van Hoof, and Tanja Lange, *Concrete quantum cryptanalysis of binary elliptic curves*, IACR TCHES **2021** (2021), no. 1, 451–472, https://tches.iacr.org/index.php/TCHES/article/view/8741.

BCSS23. Xavier Bonnetain, André Chailloux, André Schrottenloher, and Yixin Shen, *Finding many collisions via reusable quantum walks: Application to lattice sieving*, EUROCRYPT 2023, Part V (Carmit Hazay and Martijn Stam, eds.), LNCS, vol. 14008, Springer, Heidelberg, April 2023, pp. 221–251.

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven, *New directions in nearest neighbor searching with applications to lattice sieving*, 27th SODA (Robert Krauthgamer, ed.), ACM-SIAM, January 2016, pp. 10–24.

BDK+21. Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé, *CRYSTALS-Dilithium: algorithm specifications and supporting documentation*, 2021, Submission to the NIST's post-quantum cryptography standardization process.

Bel13. Aleksandrs Belovs, *Quantum walks and electric networks*, arXiv e-prints (2013), arXiv:1302.3143.

BL21. Daniel J. Bernstein and Tanja Lange, *Non-randomness of S-unit lattices*, Cryptology ePrint Archive, Report 2021/1428, 2021, https://eprint.iacr.org/2021/1428.

BLMP19. Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny, *Quantum circuits for the CSIDH: Optimizing quantum evaluation of isogenies*, in Ishai and Rijmen [IR19], pp. 409–441.

BNS19. Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher, *Quantum security analysis of AES*, IACR Trans. Symm. Cryptol. **2019** (2019), no. 2, 55–93.

BP05. Joan Boyar and René Peralta, *The exact multiplicative complexity of the hamming weight function*, Electron. Colloquium Comput. Complex. **TR05-049** (2005).

BS20. Xavier Bonnetain and André Schrottenloher, *Quantum security analysis of CSIDH*, in Canteaut and Ishai [CI20], pp. 493–522.

CEMM98. Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca, *Quantum algorithms revisited*, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **454** (1998), no. 1969, 339–354.

CI20. Anne Canteaut and Yuval Ishai (eds.), *Eurocrypt 2020, part ii*, LNCS, vol. 12106, Springer, Heidelberg, May 2020.

CKM19. Earl Campbell, Ankur Khurana, and Ashley Montanaro, *Applying quantum algorithms to constraint satisfaction problems*, Quantum **3** (2019), 167.

CL21. André Chailloux and Johanna Loyer, *Lattice sieving via quantum random walks*, ASIACRYPT 2021, Part IV (Mehdi Tibouchi and Huaxiong Wang, eds.), LNCS, vol. 13093, Springer, Heidelberg, December 2021, pp. 63–91.

CN11. Yuanmi Chen and Phong Q. Nguyen, *BKZ 2.0: Better lattice security estimates*, ASIACRYPT 2011 (Dong Hoon Lee and Xiaoyun Wang, eds.), LNCS, vol. 7073, Springer, Heidelberg, December 2011, pp. 1–20.

DSvW21a. Léo Ducas, Marc Stevens, and Wessel van Woerden, *Advanced lattice sieving on gpus, with tensor cores*, Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II (Berlin, Heidelberg), Springer-Verlag, 2021, p. 249–279.

DSvW21b. Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden, *Advanced lattice sieving on GPUs, with tensor cores*, EUROCRYPT 2021, Part II (Anne Canteaut and François-Xavier Standaert, eds.), LNCS, vol. 12697, Springer, Heidelberg, October 2021, pp. 249–279.

dt22. The FPLLL development team, *fplll, a lattice reduction library, Version: 5.4.2*, Available at https://github.com/fplll/fplll, 2022.

EK09. Bryan Eastin and Emanuel Knill, *Restrictions on transversal encoded quantum gate sets*, Phys. Rev. Lett. **102** (2009), 110502.

FC16. Marc Fischlin and Jean-Sébastien Coron (eds.), *Eurocrypt 2016, part i*, LNCS, vol. 9665, Springer, Heidelberg, May 2016.

FHK+20. Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang, *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. specification v1.2*, 2020, Submission to the NIST's post-quantum cryptography standardization process.

FP85. Ulrich Fincke and Michael Pohst, *Improved methods for calculating vectors of short length in a lattice, including a complexity analysis*, Mathematics of Computation **44** (1985), no. 170, 463–463.

Gid15. Craig Gidney, *Constructing large increment gates*, 2015, Last retrieved 18 Oct 2022 at https://algassert.com/circuits/2015/06/12/Constructing-Large-Increment-Gates.html.

GLRS16. Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt, *Applying grover's algorithm to AES: Quantum resource estimates*, Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016 (Tsuyoshi Takagi, ed.), Springer, Heidelberg, 2016, pp. 29–43.

GNR10. Nicolas Gama, Phong Q. Nguyen, and Oded Regev, *Lattice enumeration using extreme pruning*, EUROCRYPT 2010 (Henri Gilbert, ed.), LNCS, vol. 6110, Springer, Heidelberg, May / June 2010, pp. 257–278.

Gro96. Lov K. Grover, *A fast quantum mechanical algorithm for database search*, in STOC 1996 [STO96], pp. 212–219.

HPS11. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé, *Analyzing blockwise lattice algorithms using dynamical systems*, CRYPTO 2011 (Phillip Rogaway, ed.), LNCS, vol. 6841, Springer, Heidelberg, August 2011, pp. 447–464.

HR14. Ishay Haviv and Oded Regev, *On the lattice isomorphism problem*, 25th SODA (Chandra Chekuri, ed.), ACM-SIAM, January 2014, pp. 391–404.

HS07. Guillaume Hanrot and Damien Stehlé, *Improved analysis of kannan's shortest lattice vector algorithm*, CRYPTO 2007 (Alfred Menezes, ed.), LNCS, vol. 4622, Springer, Heidelberg, August 2007, pp. 170–186.

HSRS18. Thomas Haener, Mathias Soeken, Martin Roetteler, and Krysta M. Svore, *Quantum circuits for floating-point arithmetic*, Reversible Computation (Cham) (Jarkko Kari and Irek Ulidowski, eds.), Springer International Publishing, 2018, pp. 162–174.

IR19. Yuval Ishai and Vincent Rijmen (eds.), *Eurocrypt 2019, part ii*, LNCS, vol. 11477, Springer, Heidelberg, May 2019.

JNRV20. Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia, *Implementing grover oracles for quantum key search on AES and LowMC*, in Canteaut and Ishai [CI20], pp. 280–310.

Jon13. Cody Jones, *Low-overhead constructions for the fault-tolerant toffoli gate*, Phys. Rev. A **87** (2013), 022328.

Kan83. Ravi Kannan, *Improved algorithms for integer programming and related lattice problems*, 15th ACM STOC, ACM Press, April 1983, pp. 193–206.

Kit96. Alexei Y. Kitaev, *Quantum measurements and the abelian stabilizer problem*, Electron. Colloquium Comput. Complex. **TR96-003** (1996).

Laa15. Thijs Laarhoven, *Search problems in cryptography*, Ph.D. thesis, Eindhoven University of Technology, 2015.

LLJL82. Arjen K. Lenstra, Hendrik W. Lenstra Jr., and László Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen **261** (1982), 515–534.

Mon18. Ashley Montanaro, *Quantum-walk speedup of backtracking algorithms*, Theory of Computing **14** (2018), no. 15, 1–24.

MR04. Daniele Micciancio and Oded Regev, *Worst-case to average-case reductions based on Gaussian measures*, 45th FOCS, IEEE Computer Society Press, October 2004, pp. 372–381.

MR20. Simon Martiel and Maxime Remaud, *Practical implementation of a quantum backtracking algorithm*, SOFSEM 2020: Theory and Practice of Computer Science (Cham) (Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, eds.), Springer International Publishing, 2020, pp. 597–606.

MRBAG16. Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik, *The theory of variational hybrid quantum-classical algorithms*, New Journal of Physics **18** (2016), no. 2, 023023.

MV10a. Daniele Micciancio and Panagiotis Voulgaris, *A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations*, 42nd ACM STOC (Leonard J. Schulman, ed.), ACM Press, June 2010, pp. 351–358.

MV10b. ———, *Faster exponential time algorithms for the shortest vector problem*, 21st SODA (Moses Charika, ed.), ACM-SIAM, January 2010, pp. 1468–1480.

MW16. Daniele Micciancio and Michael Walter, *Practical, predictable lattice basis reduction*, in Fischlin and Coron [FC16], pp. 820–849.

NC11. Michael A. Nielsen and Isaac L. Chuang, *Quantum computation and quantum information: 10th anniversary edition*, Cambridge University Press, 2011.

NIS16. NIST, *National institute of standards and technology's Post-Quantum Cryptography Standardization*, 2016, https://csrc.nist.gov/projects/post-quantum-cryptography.

NS05. Phong Q. Nguyen and Damien Stehlé, *Floating-point LLL revisited*, EUROCRYPT 2005 (Ronald Cramer, ed.), LNCS, vol. 3494, Springer, Heidelberg, May 2005, pp. 215–233.

NS06. Phong Q. Nguyen and Damien Stehlé, *LLL on the average*, Proceedings of the 7th International Conference on Algorithmic Number Theory (Berlin, Heidelberg), ANTS'06, Springer-Verlag, 2006, p. 238–256.

NV08. P. Q. Nguyen and T. Vidick, *Sieve algorithms for the shortest vector problem are practical*, Journal of Mathematical Cryptology **2** (2008), no. 2.

PLP16. Rafael Pino, Vadim Lyubashevsky, and David Pointcheval, *The whole is less than the sum of its parts: Constructing more efficient lattice-based akes*, Proceedings of the 10th International Conference on Security and Cryptography for Networks - Volume 9841 (Berlin, Heidelberg), Springer-Verlag, 2016, p. 273–291.

PMS+14. Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien, *A variational eigenvalue solver on a photonic quantum processor*, Nature Communications **5** (2014), no. 1.

PRM17. Alex Parent, Martin Roetteler, and Michele Mosca, *Improved reversible and quantum circuits for karatsuba-based integer multiplication*, 12th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2017, June 14-16, 2017, Paris, France (Mark M. Wilde, ed.), LIPIcs, vol. 73, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 7:1–7:15.

PS08. Xavier Pujol and Damien Stehlé, *Rigorous and efficient short lattice vectors enumeration*, ASIACRYPT 2008 (Josef Pieprzyk, ed.), LNCS, vol. 5350, Springer, Heidelberg, December 2008, pp. 390–405.

Reg05. Oded Regev, *On lattices, learning with errors, random linear codes, and cryptography*, 37th ACM STOC (Harold N. Gabow and Ronald Fagin, eds.), ACM Press, May 2005, pp. 84–93.

RNSL17. Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter, *Quantum resource estimates for computing elliptic curve discrete logarithms*, ASIACRYPT 2017, Part II (Tsuyoshi Takagi and Thomas Peyrin, eds.), LNCS, vol. 10625, Springer, Heidelberg, December 2017, pp. 241–270.

RS16. Neil J Ross and Peter Selinger, *Optimal ancilla-free Clifford+ T approximation of z-rotations.*, Quantum Inf. Comput. **16** (2016), no. 11&12, 901–953.

Sch87. Claus-Peter Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoretical Computer Science **53** (1987), no. 2-3, 201–224.

Sch03. Claus-Peter Schnorr, *Lattice reduction by random sampling and birthday methods*, STACS, Springer, 2003, pp. 145–156.

SE94. Claus-Peter Schnorr and Michael Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Mathematics of Programming **66** (1994), 181–199.

Sel13. Peter Selinger, *Quantum circuits of t-depth one*, Phys. Rev. A **87** (2013), 042302.

Sel15. Peter Selinger, *Efficient clifford+t approximation of single-qubit operators*, Quantum Inf. Comput. **15** (2015), no. 1-2, 159–180.

SH95. Claus-Peter Schnorr and Horst Helmut Hörner, *Attacking the Chor-Rivest cryptosystem by improved lattice reduction*, EUROCRYPT'95 (Louis C. Guillou and Jean-Jacques Quisquater, eds.), LNCS, vol. 921, Springer, Heidelberg, May 1995, pp. 1–12.

SRWDM17. Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli, *Hierarchical reversible logic synthesis using luts*, Proceedings of the 54th Annual Design Automation Conference 2017 (New York, NY, USA), DAC '17, Association for Computing Machinery, 2017.

STO96. *28th acm stoc*, ACM Press, May 1996.

TTK10. Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro, *Quantum addition circuits and unbounded fan-out*, Quantum Info. Comput. **10** (2010), no. 9, 872–890.