Quantum Lattice Enumeration and Tweaking Discrete Pruning

Yoshinori Aono[1], Phong Q. Nguyen[2,3], and Yixin Shen[4,3]

[1] National Institute of Information and Communications Technology, Japan
[2] Inria Paris, France
[3] CNRS, JFLI, University of Tokyo, Japan
[4] IRIF, Univ Paris Diderot, CNRS, France

Abstract. Enumeration is a fundamental lattice algorithm. We show how to speed up enumeration on a quantum computer, which affects the security estimates of several lattice-based submissions to NIST: if T is the number of operations of enumeration, our quantum enumeration runs in roughly $\sqrt{T}$ operations. This applies to the two most efficient forms of enumeration known in the extreme pruning setting: cylinder pruning but also discrete pruning introduced at Eurocrypt '17. Our results are based on recent quantum tree algorithms by Montanaro and Ambainis-Kokainis. The discrete pruning case requires a crucial tweak: we modify the preprocessing so that the running time can be rigorously proved to be essentially optimal, which was the main open problem in discrete pruning. We also introduce another tweak to solve the more general problem of finding close lattice vectors.

1 Introduction

The main two hard lattice problems are finding short lattice vectors (SVP) and close lattice vectors (CVP), either exactly or approximately. Both have been widely used in cryptographic design for the past twenty years: Ajtai's SIS [2] and Regev's LWE [39] are randomized variants of respectively SVP and CVP.

With the NIST standardization of post-quantum cryptography and the development of fully-homomorphic encryption, there is a need for convincing security estimates for lattice-based cryptosystems. Yet, in the past ten years, there has been regular progress in the design of lattice algorithms, both in theory (*e.g.* [21,32,1]) and practice (*e.g.* [36,22,33,18,20,26,10]), which makes security estimates tricky. Lattice-based NIST submissions use varying cost models, which gives rise to a wide range of security estimates [5]. The biggest source of divergence is the cost assessment of a subroutine to find nearly shortest lattice vectors in certain dimensions (typically the blocksize of reduction algorithms), which is chosen among two families: sieving [3,36,33,26,14] and enumeration.

Enumeration is the simplest algorithm to solve SVP/CVP: it outputs $L \cap B$, given a lattice L and an n-dimensional ball $B \subseteq \mathbb{R}^n$. Dating back to the early 1980s [38,25], it has been significantly improved in practice in the past twenty years, thanks to pruning methods introduced by Schnorr *et al.* [42,43,41], and later revisited and generalized as cylinder pruning [22] and discrete pruning [10]: these methods offer a trade-off by enumerating over a subset $S \subseteq B$, at the expense of missing solutions. One may only be interested in finding one point in $L \cap S$ (provided it exists), or the 'best' point in $L \cap S$, *i.e.* a point minimizing the distance to a target. Enumeration and cylinder pruning compute $L \cap S$ by a depth-first search of a tree with super-exponentially many nodes. Discrete pruning is different, but the computation of S uses special enumerations.

The choice between sieving and enumeration for security estimates is not straightforward. On the one hand, sieving methods run in time $2^{O(n)}$ much lower than enumeration's $2^{O(n \log n)}$, but require exponential space. On the other hand, until very recently [6], the largest lattice numerical challenges had all been solved by pruned enumeration, either directly or as a subroutine: cylinder pruning [22] for NTRU challenges [44] (solved by Ducas-Nguyen) and Darmstadt's lattice challenges [29] (solved by Aono-Nguyen), and discrete pruning [20,10] for Darmstadt's SVP challenges [40] (solved by Kashiwabara-Teruya). Among all lattice-based submissions [37,5] to NIST, the majority chose sieving over enumeration based on the analysis of NewHope [8, Sect. 6], which states that sieving is more efficient than enumeration in dimension ≥ 250 for both classical and quantum computers. But this analysis is debatable: [8] estimates the cost of sieving by a lower bound (ignoring sub-exponential terms) and that of enumeration by an upper bound (either [18, Table 4] or [17, Table 5.2]), thereby ignoring the lower bound of [18] (see [11] for improved bounds).

The picture looks even more blurry when considering the impact of quantum computers, which is especially relevant to NIST standardization. The quantum speed-up is rather limited for sieving: the best quantum sieve algorithm runs in heuristic time $2^{0.265n+o(n)}$, only slighty less than the best classical (heuristic) time $2^{0.292n+o(n)}$ [26,14]. And the quantum speed-up for enumeration is unclear, as confirmed by recent discussions on the NIST mailing-list [4]. In 2015, Laarhoven *et al.* [27, Sect. 9.1] noticed that quantum search algorithms do not apply to enumeration: indeed, Grover's algorithm assumes that the possible solutions in the search space can be indexed and that one can find the i-th possible solution efficiently, whereas lattice enumeration explores a search tree of an unknown structure which can only be explored locally. Three recent papers [8,19,7] mention in a short paragraph that Montanaro's quantum backtracking algorithm [34] can speed up enumeration, by decreasing the number T of operations to $\sqrt{T}$. However, no formal statement nor details are given in [8,19,7]. Furthermore, none of the lattice-based submissions to NIST cite Montanaro's algorithm [34]: the only submission that mentions enumeration in a quantum setting is NTRU-HSS-KEM [24], where it is speculated that enumeration might have a $\sqrt{T}$ quantum variant.

Our results. We show that lattice enumeration and its cylinder and discrete pruning variants can all be quadratically sped up on a quantum computer, unlike sieving. This is done by a careful interpretation and analysis of enumeration as tree algorithms. Interestingly, we show that this speedup also applies to extreme pruning [22] where one repeats enumeration over many reduced bases: a naive approach would only decrease the classical cost mt (where m is the number of bases and t is the number of operations of a single enumeration) to $m\sqrt{t}$ quantum operations, but we bring it down to $\sqrt{mt}$.

First, we clarify the application of Montanaro's algorithm [34] to enumeration with cylinder pruning: the analysis of [34] assumes that the degree of the tree is bounded by a constant, which is tailored for constraint satisfaction problems, but is not the setting of lattice enumeration. To tackle enumeration, we add basic tools such as binary tree conversion and dichotomy: we obtain that if a lattice enumeration (with or without cylinder pruning) searches over a tree with T nodes, the best solution can be found by a quantum algorithm using roughly $\sqrt{T}$ poly-time operations, where there is a polynomial overhead, which can be decreased if one is only interested in finding one solution. This formalizes earlier brief remarks of [8,19,7], and applies to both SVP and CVP.

Our main result is that the quantum quadratic speed-up also applies to the recent discrete pruning enumeration introduced by Aono and Nguyen [10] as a generalization of Schnorr's sampling algorithm [41]. To do so, we tweak discrete pruning and use an additional quantum algorithm, namely that of Ambainis and Kokainis [9] from STOC '17 to estimate the size of trees. Roughly speaking, given a parameter T, discrete pruning selects T branches (optimizing a certain metric) in a larger tree, and derives T candidate short lattice vectors from them. Our quantum variant directly finds the best candidate in roughly $\sqrt{T}$ operations.

As mentioned previously, we show that the quadratic speed-up of both enumerations also applies to the extreme pruning setting (required to exploit the full power of pruning): if one runs cylinder pruning over m trees, a quantum enumeration can run in $\sqrt{T}$ poly-time operations where T is the sum of the m numbers of nodes, rather than $\sqrt{mT}$ naively; and there is a similar phenomenon for discrete pruning.

As a side result, we present two tweaks to discrete pruning [10], to make it more powerful and more efficient. The first tweak enables to solve CVP in such a way that most of the technical tools introduced in [10] can be reused. This works for the approximation form of CVP, but also its exact version formalized by the *Bounded Distance Decoding* problem (BDD), which appears in many cryptographic applications such as LWE. In BDD, the input is a lattice basis and a lattice point shifted by some small noise whose distribution is crucial. We show how to handle the most important noise distributions, such as LWE's Gaussian distribution and finite distributions used in GGH [23] and lattice attacks on DSA [35]. Enumeration, which was historically only described for SVP, can trivially be adapted to CVP, and so does [22]'s cylinder pruning [30]. However, discrete pruning [10] appears to be less simple.

The second tweak deals with the selection of optimal discrete pruning parameters, and is crucial for our quantum variant. Intuitively, given an integer $T > 0$, the problem is to find the T "best" integral vectors $\boldsymbol{t} \in \mathbb{N}^n$ which minimize some objective function $f(\boldsymbol{t})$. Aono and Nguyen [10] introduced a fast practical algorithm to do so for a very special useful choice of f, but the algorithm was heuristic: no good bound on the running time was known. We show that their algorithm can actually behave badly in the worst case, *i.e.* it may take exponential time. But we also show that by a careful modification, the algorithm becomes provably efficient and even optimal for that f, and heuristically for more general choices of f: the running time becomes essentially T operations.

Our theoretical analysis has been validated by experiments, which show that in practical BDD situations, discrete pruning is as efficient as cylinder pruning. Since discrete pruning has interesting features (such as an easier parallelization and an easier generation of parameters), it might become the method of choice for large-scale blockwise lattice reduction.

Impact. Fig. 1 illustrates the impact of our quantum enumeration on security estimates: the red and yellow curves show $\sqrt{\#bases * N}$ where N is an upper bound cost, i.e., number of nodes of enumeration with extreme pruning with probability $1/\#bases$. The upper bounds for HKZ/Rankin bases are computed by the method of [11]. Here, we omitted the polynomial overhead factor because small factors in quantum sieve have also never been investigated. Note that the estimate $2^{(0.187\beta\log\beta-1.019\beta+16.1)/2}$ (called Q-Enum in [5]) of a hypothetical quantum enumeration in NTRU-HSS-KEM [24], which is the square-root of a numerical interpolation of the upper bound of [18,17], is higher than our HKZ estimate: however, both are less than 2^{128} until blocksize roughly 400.

Quantum enumeration with extreme pruning would be faster than quantum sieve up to higher dimensions than previously thought, around 300 if we assume that 10^{10} quasi-HKZ-bases can be obtained for a cost similar as enumeration, or beyond 400 if 10^{10} Rankin-bases (see [18]) can be used instead. Such ranges

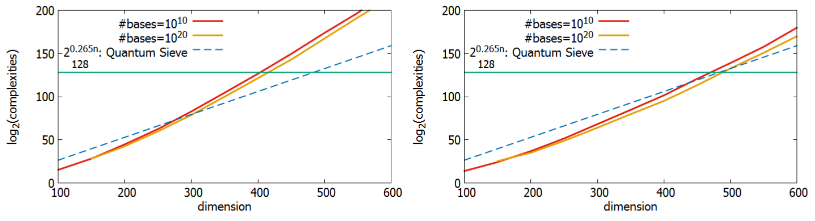

Fig. 1. Q-sieve vs Q-enum: (Left) Using HKZ bases (Right) Using Rankin bases

would affect the security estimates of between 11 and 17 NIST submissions (see Fig. 2), depending on which basis model is considered: these submissions state that the best attack runs BKZ with a blocksize seemingly lower than our threshold between quantum enumeration and quantum sieving, except in the case of S/L NTRU Prime, for which the blocksize 528 corresponds to less than 2^{200} in Fig. 1, whereas the target NIST category is 5.

Furthermore, we note that our quantum speedup might actually be more than quadratic. Indeed, the number T of enumeration nodes is actually a random variable: the average quantum running time is $\mathbb{E}(\sqrt{T})$, which is $\leq \sqrt{\mathbb{E}(T)}$ and potentially much less (*e.g.* a log-normal distribution). It would be useful

Name	NIST category	Blocksize
EMBLEM	1	260/337
uRound2	1	286/302/304
Ding Key Exchange	1	330-366
R EMBLEM	1	345/383
CRYSTALSDilithium	1	347
uRound2	2	355/358/386/397
CRYSTALSKyber	1	386
NewHope	1	386
uRound2	3	394/401/427/425
NTRUEncrypt	1	319
S/L NTRU Prime	5	528

Fig. 2. Lattice-based NIST submissions affected by quantum enumeration

to identify the distribution of T: it cannot be log-normal for LLL bases (unlike what seems to be suggested in [45]), because it would violate the provable running time $2^{O(n^2)}$ of enumeration with LLL bases.

On the other hand, 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 [34] and that of Ambainis and Kokainis [9].

Related work. Babai's nearest plane algorithm [13] can be viewed as the first form of BDD discrete pruning, using only a single cell. Lindner-Peikert's algorithm [28] generalizes it using exponentially many cells, and is the BDD analogue of Schnorr's random sampling [41] (see [30]). But for both [41,28], the selection of cells is far from being optimal. In 2003, Ludwig [31] applied Grover search to speed up [41] quantumly.

Roadmap. Sect. 2 provides background. Sect. 3 gives a general description of enumeration to find close lattice vectors. In Sect. 4, we speed up cylinder pruning enumeration on a quantum computer, using [34]. In Sect. 5, we adapt lattice enumeration with discrete pruning to CVP. In Sect. 6, we show how to efficiently select the best parameters for discrete pruning, by modifying the orthogonal enumeration of [10]. In Sect. 7, we speed up discrete pruning enumeration on a quantum computer, using [34,9]. Supplementary material is given in Appendix, including proofs and experimental results.

2 Preliminaries

We follow the notations of [10].

General. $\mathbb{N}$ is the set of integers ≥ 0. For any finite set U, its number of elements is $\#U$. For any measurable subset $S \subseteq \mathbb{R}^n$, its volume is $\mathrm{vol}(S)$. We use row representations of matrices. The Euclidean norm of a vector $\boldsymbol{v} \in \mathbb{R}^n$ is $\|\boldsymbol{v}\|$. We denote by $\mathrm{Ball}_n(\boldsymbol{c}, R)$ the n-dim Euclidean ball of radius R and center $\boldsymbol{c}$, whose volume is $\mathrm{vol}(\mathrm{Ball}_n(R)) = R^n \frac{\pi^{n/2}}{\Gamma(n/2+1)}$. If $\boldsymbol{c}$ is omitted, we mean $\boldsymbol{c} = 0$.

Lattices. A *lattice* L is a discrete subgroup of $\mathbb{R}^m$, or equivalently the set $L(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \{\sum_{i=1}^n x_i \boldsymbol{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of n linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^m$. Such $\boldsymbol{b}_i$'s form a *basis* of L. All the bases have the same number n of elements, called the dimension or rank of L, and the same n-dimensional volume of the parallelepiped $\{\sum_{i=1}^n a_i \boldsymbol{b}_i \ : \ a_i \in [0,1)\}$ they generate. We call this volume the co-volume of L, denoted by $\mathrm{covol}(L)$. The lattice L is said to be *full-rank* if $n = m$. The *shortest vector problem* (SVP) asks to find a non-zero lattice vector of minimal Euclidean norm. The *closest vector problem* (CVP) asks to find a lattice vector closest to a target vector.

Orthogonalization. For a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of a lattice L and $i \in \{1, \ldots, n\}$, we denote by π_i the orthogonal projection on $\mathrm{span}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1})^\perp$. The *Gram-Schmidt orthogonalization* of the basis B is defined as the sequence of orthogonal vectors $B^\star = (\boldsymbol{b}_1^\star, \ldots, \boldsymbol{b}_n^\star)$, where $\boldsymbol{b}_i^\star := \pi_i(\boldsymbol{b}_i)$. We can write each $\boldsymbol{b}_i$ as $\boldsymbol{b}_i^\star + \sum_{j=1}^{i-1} \mu_{i,j} \boldsymbol{b}_j^\star$ for some unique $\mu_{i,1}, \ldots, \mu_{i,i-1} \in \mathbb{R}$. Thus, we may represent the $\mu_{i,j}$'s by a lower-triangular matrix μ with unit diagonal. $\pi_i(L)$ is a lattice of rank $n+1-i$ generated by $\pi_i(\boldsymbol{b}_i), \ldots, \pi_i(\boldsymbol{b}_n)$, with $\mathrm{covol}(\pi_i(L)) = \prod_{j=i}^n \|\boldsymbol{b}_j^\star\|$.

Gaussian Heuristic. The classical Gaussian Heuristic provides an estimate on the number of lattice points inside a "nice enough" set:

Heuristic 1 *Given a full-rank lattice $L \subseteq \mathbb{R}^n$ and a measurable set $S \subseteq \mathbb{R}^n$, the number of points in $S \cap L$ is approximately* $\mathrm{vol}(S)/\mathrm{covol}(L)$.

Both rigorous results and counter-examples are known (see [10]). One should therefore experimentally verify its use, especially for pruned enumeration which relies on strong versions of the heuristic, where the set S is not fixed, depending on a basis of L.

Statistics. We denote by $\mathbb{E}()$ the expectation and $\mathbb{V}()$ the variance of a random variable. For discrete pruning, it is convenient to extend $\mathbb{E}()$ to any measurable set C of $\mathbb{R}^n$ by using the squared norm, that is $\mathbb{E}\{C\} := \mathbb{E}_{\boldsymbol{x} \in C}(\|\boldsymbol{x}\|^2)$.

Gaussian distribution. The CDF of the Gaussian distribution of expectation 0 and variance σ^2 is $\frac{1}{2}(1 + \mathrm{erf}(\frac{x}{\sigma\sqrt{2}}))$ where the error function is $\mathrm{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$. The multivariate Gaussian distribution over $\mathbb{R}^m$ of parameter σ selects each coordinate with Gaussian distribution.

Quantum Tree Algorithms. Like in [9], a tree $\mathcal{T}$ is locally accessed given:

1. the root r of $\mathcal{T}$
2. a black box which, given a node v, returns the number of children $d(v)$ for this node. If $d(v) = 0$, v is called a leaf.
3. a black box which, given a node v and $i \in [d(v)]$, returns the i-th child of v.

We denote by $V(\mathcal{T})$ its set of nodes, $L(\mathcal{T})$ its set of leaves, $d(\mathcal{T}) = \max_{v \in V(\mathcal{T})} d(v)$ its degree and $n(\mathcal{T})$ an upper-bound of its depth. When there is no ambiguity, we use d and n directly without the argument $\mathcal{T}$. We also denote by $\#\mathcal{T}$ the number of nodes of the tree $\mathcal{T}$.

Backtracking is a classical algorithm for solving problems such as constraint satisfaction problems, by performing a tree search in depth-first order. Each node represents a partial candidate and its children say how to extend a candidate. There is a black-box function $\mathcal{P} : V(\mathcal{T}) \to \{true, false, indeterminate\}$ such that $\mathcal{P}(v) \in \{true, false\}$ iff v is a leaf: a node $v \in V(\mathcal{T})$ is called marked if $\mathcal{P}(v) = true$. Backtracking determines whether $\mathcal{T}$ contains a marked node, or outputs one or all marked nodes. Classically, this can be done in $\#\mathcal{V}(\mathcal{T})$ queries. Montanaro [34] studied the quantum case:

Theorem 2 ([34]). *There is a quantum algorithm* **ExistSolution**$(\mathcal{T}, T, \mathcal{P}, n, \varepsilon)$ *which given* $\varepsilon > 0$, *a tree* $\mathcal{T}$ *such that* $d(\mathcal{T}) = O(1)$, *a black box function* $\mathcal{P}$, *and upper bounds* T *and* n *on the size and the depth of* $\mathcal{T}$, *determines if* $\mathcal{T}$ *contains a marked node by making* $O(\sqrt{Tn}\log(1/\varepsilon))$ *queries to* $\mathcal{T}$ *and to the black box function* $\mathcal{P}$, *with a probability of correct answer* $\geq 1 - \varepsilon$. *It uses* $O(1)$ *auxiliary operations per query and uses* `poly`(n) *qubits.*

Theorem 3 ([34]). *There is a quantum algorithm* **FindSolution**$(\mathcal{T}, \mathcal{P}, n, \varepsilon)$ *which, given* $\varepsilon > 0$, *a tree* $\mathcal{T}$ *such that* $d(\mathcal{T}) = O(1)$, *a black box function* $\mathcal{P}$, *and an upper bound* n *on the depth of* $\mathcal{T}$, *outputs* x *such that* $\mathcal{P}(x)$ *is true, or "not found" if no such* x *exists by making* $O(\sqrt{\#\mathcal{V}(\mathcal{T})}n^{3/2}\log(n)\log(1/\varepsilon))$ *queries to* $\mathcal{T}$ *and to the black box function* $\mathcal{P}$, *with correctness probability at least* $1 - \varepsilon$. *It uses* $O(1)$ *auxiliary operations per query and uses* `poly`(n) *qubits.*

Notice that Th. 3 does not require an upper-bound on $\#\mathcal{V}(\mathcal{T})$ as input.

Ambainis and Kokainis [9] gave a quantum algorithm to estimate the size of trees, with input a tree $\mathcal{T}$ and a candidate upper bound T_0 on $\#\mathcal{V}(\mathcal{T})$. The algorithm must output an estimate for $\#\mathcal{V}(\mathcal{T})$, *i.e.* either a number of $\hat{T} \in [T_0]$ or a claim "$\mathcal{T}$ contains more than T_0 vertices". The estimate is δ-correct if:

1. the estimate is $\hat{T} \in [T_0]$ which satisfies $|T - \hat{T}| \leq \delta T$ where T is the actual number of vertices;
2. the estimate is "$\mathcal{T}$ contains more than T_0 vertices" and the actual number of vertices T satisfies $(1+\delta)T > T_0$.

An algorithm solves the tree size estimation problem up to precision $1 \pm \delta$ with correctness probability at least $1 - \varepsilon$ if for any $\mathcal{T}$ and any T_0, the probability that it outputs a δ-correct estimate is at least $1 - \varepsilon$.

Theorem 4 ([9]). *There is a quantum algorithm* **TreeSizeEstimation**$(\mathcal{T}, T_0, \delta, \varepsilon)$ *which, given* $\varepsilon > 0$, *a tree* $\mathcal{T}$, *and upper bounds* d *and* n *on the degree and the depth of* $\mathcal{T}$, *solves tree size estimation up to precision* $1 \pm \delta$, *with correctness probability at least* $1 - \varepsilon$. *It makes* $O\left(\frac{\sqrt{nT_0}}{\delta^{1.5}} d\log^2(\frac{1}{\varepsilon})\right)$ *queries to* $\mathcal{T}$ *and* $O(\log(T_0))$ *non-query transformations per query. The algorithm uses* `poly`$(n, \log(d), \log(T_0), \log(\delta), \log(\log(1/\varepsilon)))$ *qubits.*

3 Enumeration with Pruning

We give an overview of lattice enumeration and pruning, for the case of finding close lattice vectors, rather than finding short lattice vectors: this revisits the analysis model of both [22] and [10].

3.1 Finding Close Vectors by Enumeration

Let L be a full-rank lattice in $\mathbb{R}^n$. Given a target $\boldsymbol{u} \in \mathbb{Q}^n$, a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of L and a radius $R > 0$, enumeration [38,25] outputs $L \cap S$ where $S = \mathrm{Ball}_n(\boldsymbol{u}, R)$: by comparing all the distances to $\boldsymbol{u}$, one extracts a lattice vector closest to $\boldsymbol{u}$. It performs a recursive search using projections, to reduce the dimension of the lattice: if $\|\boldsymbol{v}\| \leq R$, then $\|\pi_k(\boldsymbol{v})\| \leq R$ for all $1 \leq k \leq n$. One can easily enumerate $\pi_n(L) \cap S$. And if one enumerates $\pi_{k+1}(L) \cap S$ for some $k \geq 1$, one can derive $\pi_k(L) \cap S$ by enumerating the intersection of a one-dimensional lattice with a suitable ball, for each point in $\pi_{k+1}(L) \cap S$. Concretely, it can be viewed as a depth-first search of the enumeration tree $\mathcal{T}$ which is a target of the quantum speed-up: the nodes at depth $n+1-k$ are the points of $\pi_k(L) \cap S$. The classical/quantum running-times of enumeration depends on R and the quality of B, but is typically super-exponential in n, even if $L \cap S$ is small.

3.2 Finding Close Vectors by Enumeration with Pruning

We adapt the general form of enumeration with pruning introduced by [10]: pruned enumeration uses a pruning set $P \subseteq \mathbb{R}^n$, and outputs $L \cap (\boldsymbol{u} + P)$. The advantage is that for suitable choices of P, enumerating $L \cap (\boldsymbol{u} + P)$ is much cheaper than $L \cap S$, and if we further intersect $L \cap (\boldsymbol{u} + P)$ with S, we may have found non-trivial points of $L \cap S$. Note that we use $\boldsymbol{u} + P$ rather than P, because it is natural to make P independent of $\boldsymbol{u}$, and it is what happens when one uses the pruning of [22] to search for close vectors. Following [22], we view the pruning set P as a random variable: it depends on the choice of basis B.

We distinguish two cases, which were considered separately in [10,22]:

Approximation setting: This was studied in [10], but not in [22]. Here, we are interested in finding any point in $L \cap S \cap (\boldsymbol{u}+P)$ by enumerating $L \cap (\boldsymbol{u}+P)$ then intersect it with the ball S, so we define the success probability as:

$$\Pr_{\mathrm{succ}} = \Pr_{P,\boldsymbol{u}}(L \cap S \cap (\boldsymbol{u} + P) \neq \emptyset), \tag{1}$$

which is the probability that it outputs at least one point in $L \cap S$. By (slightly) adapting the reasoning of [10] based on the Gaussian heuristic, we estimate that (1) is heuristically

$$\Pr_{\mathrm{succ}} \approx \min(1, \mathrm{vol}(S \cap (\boldsymbol{u} + P))/\mathrm{covol}(L)), \tag{2}$$

and that the number of elements of $L \cap S \cap (\boldsymbol{u} + P)$ is roughly $\mathrm{vol}(S \cap (\boldsymbol{u} + P))/\mathrm{covol}(L)$. This corresponds to approximating the closest vector problem in a lattice, whose hardness is used in most lattice-based signature schemes.

Unique setting: Here, we know that the target $\boldsymbol{u}$ is unusually close to the lattice, that is $L \cap S$ is a singleton, and we want to find the closest lattice point to $\boldsymbol{u}$: this is the so-called *Bounded Distance Decoding* problem (BDD), whose hardness is used in most lattice-based encryption schemes. Thus, $\boldsymbol{u}$ is of the form $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{e}$ where $\boldsymbol{v} \in L$ and $\boldsymbol{e} \in \mathbb{R}^n$ is very short, and we want to recover $\boldsymbol{v}$. This was implicitly studied in [22], but not in [10]: [22] studied the exact SVP case, where one wants to recover a shortest lattice vector (in our setting, if the target $\boldsymbol{u} \in L$, the BDD solution would be $\boldsymbol{u}$, but one could alternatively ask for the closest distinct lattice point, which can be reduced to finding a shortest lattice vector). We are only interested in finding the closest lattice point $\boldsymbol{v} \in L$, so we define the success probability as:

$$\Pr_{\text{succ}} = \Pr_{P,\boldsymbol{u}}(\boldsymbol{v} \in L \cap (\boldsymbol{u} + P)), \tag{3}$$

because we are considering the probability that the solution $\boldsymbol{v}$ belongs to the enumerated set $L \cap (\boldsymbol{u} + P)$. Usually, the target $\boldsymbol{u}$ is derived from the noise $\boldsymbol{e}$, which has a known distribution, then we can rewrite (3) as:

$$\Pr_{\text{succ}} = \Pr_{P,\boldsymbol{e}}(0 \in \boldsymbol{e} + P) = \Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in P). \tag{4}$$

In the context of SVP, we would instead define $\Pr_{\text{succ}} = \Pr_P(\boldsymbol{v} \in P)$ where $\boldsymbol{v}$ is a shortest lattice vector. In general, it is always possible to make $\boldsymbol{u}$ depend solely on $\boldsymbol{e}$: one can take a canonical basis of L, like the HNF, and use it to reduce $\boldsymbol{u}$ modulo L, which only depends on $\boldsymbol{e}$. Whether $\Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in P)$ can be evaluated depends on the choice of P and the distribution of the noise $\boldsymbol{e}$. For instance, if the distribution of $-\boldsymbol{e}$ is uniform over some measurable set E, then:

$$\Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in P) = \frac{\text{vol}(E \cap P)}{\text{vol}(E)}.$$

We discuss other settings in Sect. 5.6. This can be adapted to a discrete distribution. If the distribution of $-\boldsymbol{e}$ is uniform over a finite set $E \cap \mathbb{Z}^n$, then:

$$\Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in P) = \frac{\#(E \cap P \cap \mathbb{Z}^n)}{\#(E \cap \mathbb{Z}^n)},$$

where $\#(E \cap P \cap \mathbb{Z}^n)$ is heuristically $\approx \text{vol}(E \cap P)$ by the Gaussian heuristic, and $\#(E \cap \mathbb{Z}^n)$ is usually given by the specific choice of E.

When it fails, we can simply repeat the process with many different P's until we solve the problem, in the approximation-setting or the unique-setting.

We have discussed ways to estimate the success probability of pruned enumeration. To estimate the running time of the full algorithm, we need more information, which depends on the choice of pruning:

- An estimate of the cost of enumerating $L \cap S \cap (\boldsymbol{u} + P)$.
- An estimate of the cost of computing the (random) reduced basis B.

3.3 Cylinder Pruning

The first pruning set P ever used is the following generalization [22] of pruned enumeration of [42,43]. There, P is defined by a function $f : \{1, \ldots, n\} \to [0, 1]$, a radius $R > 0$ and a lattice basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ as follows:

$$P_f(B, R) = \{\boldsymbol{x} \in \mathbb{R}^n \text{ s.t. } \|\pi_{n+1-i}(\boldsymbol{x})\| \leq f(i)R \text{ for all } 1 \leq i \leq n\}, \tag{5}$$

where the π_i's are the Gram-Schmidt projections defined by B. We call *cylinder pruning* this form of enumeration, because $P_f(B, R)$ is an intersection of cylinders: each inequality $\|\pi_{n+1-i}(\boldsymbol{x})\| \leq f(i)R$ defines a cylinder. Cylinder pruning was introduced in the SVP setting, but its adaptation to CVP is straightforward [30].

Gama *et al.* [22] showed how to efficiently compute tight lower and upper bounds for $\text{vol}(P_f(B, R))$, thanks to the Dirichlet distribution and special integrals. Then we can do the same for $\text{vol}(P_f(B, R) \cap S)$ if S is any zero-centered ball. Using the shape of $P_f(B, R)$, [22] also estimated of the cost of enumerating $L \cap S \cap P_f(B, R)$, using the Gaussian heuristic on projected lattices $\pi_i(L)$: these estimates are usually accurate in practice, and they can also be used in the CVP case [30]. To optimize the whole selection of parameters, one finally needs to take into account the cost of computing the (random) reduced basis B: for instance, this is done in [18,12].

4 Quantum speed-up of Cylinder Pruning

4.1 Tools

The analysis of quantum tree algorithms requires the tree to have constant degree $d = O(1)$. Without this assumption, there is an extra $\texttt{poly}(d)$ term in the complexity bound like in Th. 4. Instead, it is more efficient to first convert the tree into a binary tree, so that the overhead is limited to $\texttt{poly}(\log d)$. We will use the following conversion (illustrated by Fig. 3):

Theorem 5. *One can transform any tree $\mathcal{T}$ of depth n and degree d into a binary one $\mathcal{T}_2$ so that: $\mathcal{T}_2$ can be explored locally; $\mathcal{T}$ and $\mathcal{T}_2$ have roughly the same number of nodes, namely $\#\mathcal{T} \leq \#\mathcal{T}_2 \leq 2\#\mathcal{T}$; the leaves of $\mathcal{T}$ and $\mathcal{T}_2$ are identical; the depth of $\mathcal{T}_2$ is $\leq n \log d$. Moreover, a black-box function $\mathcal{P}$ over $\mathcal{T}$ can be adapted a black box $\mathcal{P}_2$ for $\mathcal{T}_2$, so that the marked nodes of $\mathcal{T}$ and $\mathcal{T}_2$ are the same. One query to $\mathcal{P}_2$ requires at most one query to $\mathcal{P}$ with additional $O(\log d)$ auxiliary operations.*

In the context of enumeration with pruning, instead of enumerating the whole set $L \cap S$, we may only be interested in the 'best' vector in $L \cap S$, *i.e.* which minimize some distance. In terms of tree, this means that given a tree $\mathcal{T}$ with marked leafs defined by a predicate $\mathcal{P}$, we want to find a marked leaf minimizing an integral function g which is defined on the marked leaves of $\mathcal{T}$. We know that $L(\mathcal{T}) = L(\mathcal{T}_2)$. g is thus also defined on the marked leaves of $\mathcal{T}_2$. We denote by

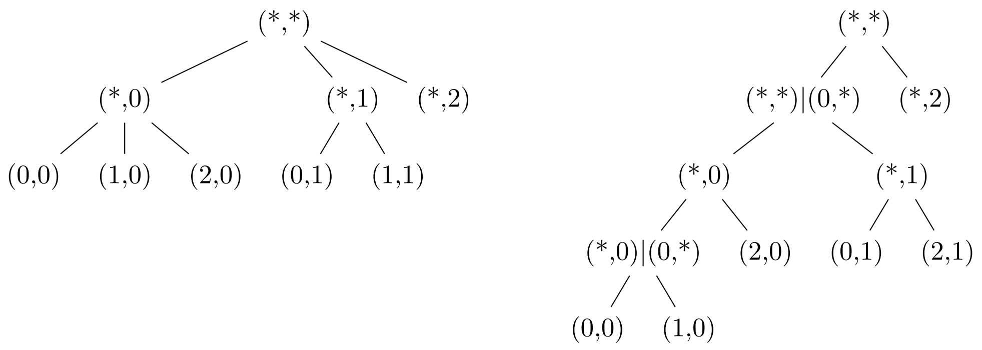

Fig. 3. An example of the transformation in Th. 5

g_V the predicate which returns true on a node $\mathcal{N}$ if and only if it is a marked leaf and $g(\mathcal{N}) \leq V$. We first find a parameter R such that there is at least one marked leaf $\mathcal{N}$ such that $g(\mathcal{N}) \leq R$. Then we decrease R by dichotomy using Th. 3 with different marking functions. We thus obtain **FindMin1**$(\mathcal{T}, \mathcal{P}, g, R, d, \varepsilon)$ (Alg. 1), which is a general algorithm to find a leaf minimizing the function g with error probability ε, using the binary tree $\mathcal{T}_2$.

Algorithm 1 Finding a minimum: **FindMin1**$(\mathcal{T}, \mathcal{P}, g, R, d, \varepsilon)$

```
Input: A tree $\mathcal{T}$ with marked leaves defined by the predicate $\mathcal{P}$. An integral
    function $g$ defined on the marked leaves of $\mathcal{T}$. A parameter $R$, such that
    $g(\mathcal{N}) \leq R$ has at least one solution over all of the marked leaves. An upper-
    bound $d$ of the number of children of a node in $\mathcal{T}$.
Output: A marked leaf $\mathcal{N}$ such that $g$ takes its minimum on $\mathcal{N}$ among all the
    marked leaves explored by the backtracking algorithm.
 1: $\mathcal{T}_2 \leftarrow$ the corresponding binary tree of $\mathcal{T}$.[5]
 2: $N \leftarrow R$, $N' \leftarrow 0$, $Round \leftarrow \lceil \log_2 R \rceil$, $\mathbf{v} \leftarrow (0, \cdots, 0)$
 3: while $N' < N - 1$ do
 4:     Call FindSolution$(\mathcal{T}_2, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon / Round)$
 5:     if FindSolution$(\mathcal{T}_2, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon / Round)$ returns $\mathbf{x}$ then
 6:         $\mathbf{v} \leftarrow \mathbf{x}$, $N \leftarrow \lfloor (N + N')/2 \rceil$
 7:     else
 8:         $N' \leftarrow \lfloor (N + N')/2 \rceil$
 9:     end if
10: end while
11: return $\mathbf{v}$
```

[5] The access to $\mathcal{T}_2$ is guaranteed by Th. 5 via the access to $\mathcal{T}$.

Theorem 6. *Let $\varepsilon > 0$. Let $\mathcal{T}$ be a tree with its marked leaves defined by a predicate $\mathcal{P}$. Let d be an upper-bound on the degree of $\mathcal{T}$. Let g be an integral function defined on the marked leaves such that $g(\mathcal{N}) \leq R$ has at least one solution over all of the marked leaves. Then Alg. 1 outputs $\mathcal{N} \in \mathcal{T}$ such that g takes its minimum on $\mathcal{N}$ among all of the marked leaves of $\mathcal{T}$, with probability at least $1-\varepsilon$. It requires $O(\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log(\lceil\log_2 R\rceil/\varepsilon)\lceil\log_2 R\rceil)$ queries on $\mathcal{T}$ and on $\mathcal{P}$, where $T = \#\mathcal{T}$. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations. The algorithm needs $\texttt{poly}(n\log d, \log R)$ qubits.*

Proof. Correctness is trivial. Regarding the query complexity, there are in total *Round* $= \lceil\log_2 R\rceil$ calls to **FindSolution**. According to Th. 3, each call requires $O(\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log(Round/\varepsilon))$ queries on the local structure of $\mathcal{T}_2$ and on g. Thus according to Th. 5, in total, we need $O(\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log(\lceil\log_2 R\rceil/\varepsilon)\lceil\log_2 R\rceil)$ queries on the local structure of $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations. For each call, we need $\texttt{poly}(n\log d)$ qubits. In total, we need $\texttt{poly}(n\log d, \log R)$ qubits. □

If we know an upper-bound of T of the number of nodes in the tree $\mathcal{T}$, we can speed up the algorithm by replacing **FindSolution** by **ExistSolution** in lines 4, 5: the new algorithm is given and analyzed in Appendix as Alg. 8 (**FindMin2**$(\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon)$).

4.2 Application to Cylinder Pruning

Lemma 1. *Let $(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ be an LLL-reduced basis. Let $\mathcal{T}$ be the backtracking tree corresponding to the cylinder pruning algorithm for SVP with radius $R \leq \|\boldsymbol{b}_1\|$ and bounding function f. Then the degree of the tree satisfies: $d(\mathcal{T}) \leq 2^n$.*

Proof. In $\mathcal{T}$, the number of children of a node $\mathcal{N}$ of depth k can be upper-bounded by $d_k = 2f(k)\frac{\|\boldsymbol{b}_1\|}{\|\boldsymbol{b}^\star_{n-k+1}\|} + 1 \leq 2^{(n-k)/2+1} + 1$. The result follows from the fact that an LLL-reduced basis satisfies: $\frac{\|\boldsymbol{b}_1\|^2}{\|\boldsymbol{b}^\star_i\|^2} \leq 2^{i-1}$ for all $1 \leq i \leq n$. □

Theorem 7. *There is a quantum algorithm which, given $\varepsilon > 0$, an LLL-reduced basis $B = (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ of a lattice L in $\mathbb{Z}^n$, a radius $R \leq \|\boldsymbol{b}_1\|$ and a bounding function $f : \{1, \cdots, n\} \to [0,1]$, outputs with correctness probability $\geq 1-\varepsilon$:*

1. *a non-zero vector $\boldsymbol{v}$ in $L \cap P_f(B,R)$, in time $O(\sqrt{T}n^3\texttt{poly}(\log(n), \log(1/\varepsilon))))$, if $L \cap P_f(B,R) \not\subseteq \{0\}$.*
2. *all vectors in $L \cap P_f(B,R)$, in time $O(\#(L \cap P_f(B,R))\sqrt{T}n^3\log(n)\texttt{poly}(\log(\#(L\cap P_f(B,R)), \log(1/\varepsilon)))$.*
3. *a shortest non-zero vector $\boldsymbol{v}$ in $L \cap P_f(B,R)$, in time $O(\sqrt{T}n^3\beta\texttt{poly}(\log(n), \log(1/\varepsilon), \log(\beta)))$, if $L \cap P_f(B,R) \not\subseteq \{0\}$. Here β is the bitsize of the vectors of B.*

Here T is the total number of nodes in the enumeration tree $\mathcal{T}$ searched by the cylinder pruning algorithm over $P_f(B,R)$.

Proof. Let $\mathcal{T}$ be the enumeration tree searched by the cylinder pruning algorithm in which a node of depth i, where $1 \leq i \leq n$, is encoded as $(*, \cdots, *, x_{n-i+1}, \cdots, \cdots, x_n)$ and where the root is encoded as $(*, \cdots, *)$. Let $\mathcal{T}_2$ be the corresponding binary tree. Let $\mathcal{P}$ be a predicate which returns true only on the nodes encoded as $(x_1, \cdots, x_n)$ in $\mathcal{T}_2$ (*i.e.* the leaves of $\mathcal{T}_2$, where all the variables are assigned), such that $\| \sum_{i=1}^{n} x_i \boldsymbol{b}_i \|^2 \leq R^2$ and $(x_1, \cdots, x_n) \neq (0, \cdots, 0)$.

For 1, if $L \cap P_f(B, R) \neq \emptyset$, we apply **FindSolution**$(\mathcal{T}_2, \mathcal{P}, n \log d, \varepsilon)$. For 2, we find all marked nodes by simply repeating the algorithm **FindSolution**, modifying the oracle operator to strike out previously seen marked elements, which requires space complexity $O(\#(L \cap P_f(B, R)))$.

For 3, if $L \cap P_f(B, R) \neq \emptyset$, we apply Th. 6 to **FindMin1**$(\mathcal{T}, \mathcal{P}, \| \cdot \|^2, R^2, 2^n + 1, \varepsilon)$. In $\mathcal{T}_2$, the height of the tree can be upper-bounded by $n \log d = O(n^2)$. We also have $Round = O(\beta)$. The time complexity is $O(\sqrt{T} n^3 \beta \texttt{poly}(\log(n), \log(1/\varepsilon), \log(\beta)))$. $\square$

As corollary, we obtain the following quantum speed-up of Kannan's algorithm for the shortest vector problem:

Theorem 8. *There is a quantum algorithm which, given $\varepsilon > 0$, and a basis B of a full-rank lattice L in $\mathbb{Z}^n$, with entries of bitlength$\leq \beta$, outputs a shortest non-zero vector of L, with error probability at most ε, in time $(n^{\frac{n}{4e}} + o(n)) \cdot \texttt{poly}(\log(n), \log(1/\varepsilon), \beta)$ using $\texttt{poly}(\texttt{n}, \beta)$ qubits.*

We can also apply the quantum tree algorithms to extreme pruning. If we run cylinder pruning over m trees, we can combine these trees into a global one and apply the quantum tree algorithms on it.

Theorem 9 (Quantum speed-up for SVP extreme pruning). *There is a quantum algorithm which, given $\varepsilon > 0$, m LLL-reduced bases $B_1, \cdots B_m$ of a lattice L in $\mathbb{Z}^n$,a radius $R \leq \min_i \|\boldsymbol{b}_{1,i}\|$ where $\boldsymbol{b}_{1,i}$ is the first vector of B_i and a bounding function $f : \{1, \cdots, n\} \to [0, 1]$, outputs with correctness probability $\geq 1 - \varepsilon$ a shortest non-zero vector $\boldsymbol{v}$ in $L \cap (\cup P_f(B_i, R))$, in time $O(\sqrt{T} n^3 \beta \texttt{poly}(\log(n), \log(1/\varepsilon), \log(\beta), \log(m)))$, if $L \cap (\cup P_f(B_i, R) \not\subseteq \{0\}$. Here β is a bound on the bitsize of vectors of B_i's, T is the sum of number of nodes in the enumeration trees $\mathcal{T}_i$ searched by cylinder pruning over $P_f(B_i, R)$ for all $1 \leq i \leq m$.*

In the case of CVP with target vector $\boldsymbol{u}$, we use the cylinder pruning algorithm with radius $R \leq \sqrt{\sum_{i=1}^{n} \|\boldsymbol{b}_i^\star\|^2/2}$ and bounding function f. The degree of the tree is now upper-bounded by $d = \max \sqrt{\sum_{i=1}^{n} \|\boldsymbol{b}_i^\star\|^2} / \|\boldsymbol{b}_j^\star\| + 1$. We have $\log d = O(\beta + n)$ where β is the bitsize of the vectors of the basis B. We can obtain a similar theorem as Th. 7 with different overheads. For exemple for the first case, the time complexity becomes $O(\sqrt{T} n^{3/2} (n + \beta)^{3/2} \texttt{poly}(\log(n), \log(1/\varepsilon), \log(\beta))))$.

For the extreme pruning for CVP the time complexity is $O(\sqrt{T} n^{3/2} (n + \beta)^{3/2} \beta \texttt{poly}(\log(n), \log(1/\varepsilon), \log(\beta), \log(m)))$

5 BDD Enumeration with Discrete Pruning

We adapt Aono-Nguyen's discrete pruning [10] to the BDD case. First, we recall discrete pruning, then we modify it.

5.1 Discrete Pruning for the Enumeration of Short Vectors

Discrete pruning is based on lattice partitions defined as follows. Let L be a full-rank lattice in $\mathbb{Q}^n$. An L-partition is a partition $\mathcal{C}$ of $\mathbb{R}^n$ such that:

- The partition is countable: $\mathbb{R}^n = \cup_{t \in T} \mathcal{C}(t)$ where T is a countable set, and $\mathcal{C}(t) \cap \mathcal{C}(t') = \emptyset$ whenever $t \neq t'$.
- Each cell $\mathcal{C}(t)$ contains a single lattice point, which can be found efficiently: given any $t \in T$, one can "open" the cell $\mathcal{C}(t)$, *i.e.* compute $\mathcal{C}(t) \cap L$ in polynomial time. In other words, the partition defines a function $g : T \to L$ where $\mathcal{C}(t) \cap L = \{g(t)\}$, and one can compute g in polynomial time.

Discrete pruning is obtained by selecting the pruning set P as the union of finitely many cells $\mathcal{C}(t)$, namely $P = \cup_{t \in U} \mathcal{C}(t)$ for some finite $U \subseteq T$. Then $L \cap P = \cup_{t \in U}(L \cap \mathcal{C}(t))$ can be enumerated by opening each cell $\mathcal{C}(t)$ for $t \in U$.

[10] presented two useful L-partitions: Babai's partition where $T = \mathbb{Z}^n$ and each cell $\mathcal{C}(t)$ is a box of volume covol(L); and the natural partition where $T = \mathbb{N}^n$ and each cell $\mathcal{C}(t)$ is a union of non-overlapping boxes, with total volume covol(L). The natural partition is preferable, and [10] explained how to select good cells for the natural partition. In theory, one would like to select the cells $\mathcal{C}(t)$ which maximize vol$(\mathcal{C}(t) \cap S)$: [10] shows how to compute vol$(\mathcal{C}(t) \cap S)$, but an exhaustive search to derive the best vol$(\mathcal{C}(t) \cap S)$ exactly would be too expensive. Instead, [10] shows how to approximate efficiently the optimal selection, by selecting the cells $\mathcal{C}(t)$ minimizing $\mathbb{E}(\mathcal{C}(t))$: given m, it is possible to compute in practice the m cells which minimize $\mathbb{E}(\mathcal{C}(t))$.

5.2 Universal Lattice Partitions

Unfortunately, in the worst case, L-partitions are not sufficient for our framework: if $P = \cup_{t \in U} \mathcal{C}(t)$, then $L \cap (P + \boldsymbol{u}) = \cup_{t \in U}(L \cap (\mathcal{C}(t) + \boldsymbol{u}))$ but the number of elements in $L \cap (\mathcal{C}(t) + \boldsymbol{u})$ is unclear, and it is also unclear how to compute in $L \cap (\mathcal{C}(t) + \boldsymbol{u})$ efficiently. To fix this, we could compute instead $L \cap P \cap S = \cup_{t \in U}(L \cap \mathcal{C}(t)) \cap S$, but that creates two issues:

- In the unique setting, it is unclear how we would evaluate success probabilities. Given a tag t and a target $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{e}$ where $\boldsymbol{e}$ is the noise and $\boldsymbol{v} \in L$, we would need to estimate the probability that $\boldsymbol{v} \in \mathcal{C}(t)$, *i.e.* $\boldsymbol{u} - \boldsymbol{e} \in \mathcal{C}(t)$.
- We would need to select the tag set U depending on the target $\boldsymbol{u}$, without knowing how to evaluate success probabilities.

BDD asks to find the lattice point $\boldsymbol{v} \in L$ closest to some target vector $\boldsymbol{u} \in \mathbb{Q}^n$, unusually close to L. To adapt discrete pruning to BDD, the most natural solution would be to subtract $\boldsymbol{u}$ to the lattice L as follows.

Definition 1. *Let L be a full-rank lattice in $\mathbb{Q}^n$. An L-partition $\mathcal{C}$ is* universal *if for all $\boldsymbol{u} \in \mathbb{Q}^n$, the shifted partition $\mathcal{C} + \boldsymbol{u}$ is an L-partition,* i.e.*:*

- *The partition is countable: $\mathbb{R}^n = \cup_{t \in T} \mathcal{C}(t)$ where T is a countable set, and $\mathcal{C}(t) \cap \mathcal{C}(t') = \emptyset$ whenever $t \neq t'$.*
- *For any $\boldsymbol{u} \in \mathbb{Q}^n$, each cell $\mathcal{C}(t)$ contains a single point in $L - \boldsymbol{u} = \{\boldsymbol{v} - \boldsymbol{u}, \boldsymbol{v} \in L\}$, which can be found efficiently: given any $t \in T$ and $\boldsymbol{u} \in \mathbb{Q}^n$, one can "open" the cell $\boldsymbol{u} + \mathcal{C}(t)$,* i.e. *compute $(\boldsymbol{u} + \mathcal{C}(t)) \cap L$ in polynomial time.*

Unfortunately, an L-partition is not necessarily universal, even in dimension one. Indeed, consider the L-partition $\mathcal{C}$ with $T = \mathbb{Z}$ defined as follows: $\mathcal{C}(0) = [-1/2, 1/2]$; For any $k > 0$, $\mathcal{C}(k) = (k - 1/2, k + 1/2]$; For any $k < 0$, $\mathcal{C}(k) = [k - 1/2, k + 1/2)$. It can be checked that $\mathcal{C}$ is not universal: the shifted cell $\mathcal{C}(0) + 1/2$ contains two lattice points, namely 0 and 1. Fortunately, we show that the two L-partitions (related to Gram-Schmidt orthogonalization) introduced in [10] for discrete pruning are actually universal:

Lemma 2. *Let B be a basis of a full-rank lattice L in $\mathbb{Z}^n$. Let $T = \mathbb{Z}^n$ and for any $\boldsymbol{t} \in T$, $\mathcal{C}_{\mathbb{Z}}(\boldsymbol{t}) = \boldsymbol{t}B^\star + \mathcal{D}$ where $\mathcal{D} = \{\sum_{i=1}^n x_i \boldsymbol{b}_i^\star$ s.t. $-1/2 \leq x_i < 1/2\}$. Then Babai's L-partition $(\mathcal{C}_{\mathbb{Z}}(), T)$ with Alg. 9 (in App.) is universal.*

Lemma 3. *Let B be a basis of a full-rank lattice L in $\mathbb{Z}^n$. Let $T = \mathbb{N}^n$ and for any $\boldsymbol{t} = (t_1, \ldots, t_n) \in T$, $\mathcal{C}_{\mathbb{N}}(\boldsymbol{t}) = \{\sum_{i=1}^n x_i \boldsymbol{b}_i^\star$ s.t. $-(t_i + 1)/2 < x_i \leq -t_i/2$ or $t_i/2 < x_i \leq (t_i + 1)/2\}$. Then the natural partition $(\mathcal{C}_{\mathbb{N}}(), T)$ with Alg. 10 (in App.) is universal.*

5.3 BDD Discrete Pruning from Universal Lattice Partitions

Any universal L-partition $(\mathcal{C}, T)$ and any vector $\boldsymbol{u} \in \mathbb{Q}^n$ define a partition $\mathbb{R}^n = \cup_{t \in T} (\boldsymbol{u} + \mathcal{C}(t))$. Following the SVP case, discrete pruning opens finitely many cells $\boldsymbol{u} + \mathcal{C}(t)$, as done by Alg. 2: discrete pruning is parametrized by a finite set $U \subseteq T$ of tags, specifying which cells $\boldsymbol{u} + \mathcal{C}(t)$ to open. It is therefore a pruned CVP enumeration with pruning set $P = \cup_{\boldsymbol{t} \in U} \mathcal{C}(\boldsymbol{t})$.

Algorithm 2 Close-Vector Discrete Pruning from Universal Lattice Partitions

```
Input: A target vector $\boldsymbol{u} \in \mathbb{Q}^n$, a universal lattice partition $(\mathcal{C}(), T)$, a finite
    subset $U \subseteq T$ and if we are in the approximation setting, a radius $R$.
Output: $L \cap (\boldsymbol{u} + (S \cap P))$ where $S = \mathrm{Ball}_n(R)$ and $P = \cup_{t \in U} \mathcal{C}(t)$.
1: $\mathcal{R} = \emptyset$
2: for $t \in U$ do
3:     Compute $L \cap (\boldsymbol{u} + \mathcal{C}(t))$ by opening $\boldsymbol{u} + \mathcal{C}(t)$: in the approx setting, check
       if the output vector is within distance $\leq R$ to $\boldsymbol{u}$, then add the vector to
       the set $\mathcal{R}$. In the unique setting, check if the output vector is the solution.
4: end for
5: Return $\mathcal{R}$.
```

The algorithm performs exactly k cell openings, where $k = \#U$ is the number of cells, and each cell opening runs in polynomial time. So the running time is $\#U$ poly-time operations: one can decide how much time should be spent.

Since the running time is easy to evaluate like in the SVP case, there are only two issues: how to estimate the success probability and how to select U (which defines the pruning set $P = \cup_{t \in U} \mathcal{C}(t)$), in order to maximize the success probability.

5.4 Success Probability

Following Sect. 3.2, we distinguish two cases:

Approximation setting: Based on (2), the success probability can be derived from:

$$\mathrm{vol}(S \cap (\boldsymbol{u} + P)) = \sum_{\boldsymbol{t} \in U} \mathrm{vol}(\mathrm{Ball}_n(R) \cap \mathcal{C}(\boldsymbol{t})). \tag{6}$$

This is exactly the same situation as in the SVP case already tackled by [10]. They showed how to compute $\mathrm{vol}(\mathrm{Ball}_n(R) \cap \mathcal{C}(\boldsymbol{t}))$ for Babai's partition and the natural partition by focusing on the intersection of a ball with a box $H = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \text{ s.t. } \alpha_i \leq x_i \leq \beta_i\}$:

- In the case of Babai's partition, each cell $\mathcal{C}_{\mathbb{Z}}(\boldsymbol{t})$ is a box.
- In the case of the natural partition, each cell $\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})$ is the union of 2^j symmetric (non-overlapping) boxes, where j is the number of non-zero coefficients of $\boldsymbol{t}$. It follows that $\mathrm{vol}(\mathcal{C}_{\mathbb{N}}(\boldsymbol{t}) \cap \mathrm{Ball}_n(\mathbb{R})) = 2^j \mathrm{vol}(H \cap S)$, where H is any of these 2^j boxes.

And they also showed to approximate a sum $\sum_{\boldsymbol{t} \in U} \mathrm{vol}(\mathrm{Ball}_n(R) \cap \mathcal{C}(\boldsymbol{t}))$ in practice, without having to compute separately each volume.

Unique setting: Based on (4), if the noise vector is $\boldsymbol{e}$, then the success probability is

$$\Pr_{\mathrm{succ}} = \Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in P) = \sum_{\boldsymbol{t} \in U} \Pr_{P,\boldsymbol{e}}(-\boldsymbol{e} \in \mathcal{C}(\boldsymbol{t})) \tag{7}$$

It therefore suffices to compute the cell probability $\Pr_{P,\boldsymbol{e}}(\boldsymbol{e} \in \mathcal{C}(\boldsymbol{t}))$, instead of an intersection volume. Similarly to the approximation setting, we might be able to approximate the sum $\sum_{\boldsymbol{t} \in U} \Pr_{P,\boldsymbol{e}}(\boldsymbol{e} \in \mathcal{C}(\boldsymbol{t}))$ without having to compute individually each probability. In Sect. 5.6, we focus on the natural partition: we discuss ways to compute the cell probability $\Pr_{P,\boldsymbol{e}}(\boldsymbol{e} \in \mathcal{C}(\boldsymbol{t}))$ depending on the distribution of the noise $\boldsymbol{e}$.

In both cases, we see that the success probability is of the form:

$$\Pr_{\mathrm{succ}} = \sum_{\boldsymbol{t} \in U} f(\boldsymbol{t}), \tag{8}$$

for some function $f() : T \to [0, 1]$ such that $\sum_{\boldsymbol{t} \in T} f(\boldsymbol{t}) = 1$, where the formula (8) is rigorous for the unique setting, and heuristic for the approximation setting due to the Gaussian heuristic. If ever the computation of $f()$ is too slow to compute individually each term of $\sum_{\boldsymbol{t}\in U} f(\boldsymbol{t})$, we can use the statistical inference techniques of [10] to approximate (8) from the computation of a small number of $f(\boldsymbol{t})$. Note that if we know that the probability is reasonably large, say > 0.01, we can alternatively use Monte-Carlo sampling to approximate it.

5.5 Selecting Parameters

We would like to select the finite set U of tags to maximize $\Pr_{\text{succ}}$ given by (8). Let us assume that we have a function $g : T \to \mathbb{R}^+$ such that $\sum_{\boldsymbol{t}\in T} g(t)$ converges. If (8) provably holds, then $\sum_{\boldsymbol{t}\in T} f(t) = 1$, so the sum indeed converges. Since T is infinite, this implies that for any $B > 0$, the set $\{\boldsymbol{t} \in T \text{ s.t. } f(\boldsymbol{t}) > B\}$ is finite, which proves the following elementary result:

Lemma 4. *Let T be an infinite countable set. Let $f : T \to \mathbb{R}^+$ be a function such that $\sum_{\boldsymbol{t}\in T} f(t)$ converges. Then for any integer $m > 0$, there is a finite subset $U \subseteq T$ of cardinal m such that $f(\boldsymbol{t}) \leq \min_{\boldsymbol{u}\in U} f(\boldsymbol{u})$ for all $\boldsymbol{t} \in T \setminus U$. Such a subset U maximizes $\sum_{\boldsymbol{u}\in U} f(\boldsymbol{u})$ among all m-size subsets of T.*

Any such subset U would maximize $\Pr_{\text{succ}}$ among all m-size subsets of T, so we would ideally want to select such a U for any given m. And m quantifies the effort we want to spend on discrete pruning, since the bit-complexity of discrete pruning is exactly m poly-time operations.

Now that we know that optimal subsets U exist, we discuss how to find such subsets U efficiently. In the approximation setting of [10], the actual function $f()$ is related to volumes: we want to select the k cells which maximize $\text{vol}(\text{Ball}_n(R) \cap \mathcal{C}(\boldsymbol{t}))$ among all the cells. This is too expensive to do exactly, but [10] provides a fast heuristic method for the natural partition, by selecting the cells $\mathcal{C}(t)$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$: given as input m, it is possible to compute efficiently in practice the tags of the m cells which minimize

$$\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} = \sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12} \right) \|\boldsymbol{b}_i^\star\|^2.$$

In other words, this is the same as replacing the function $f()$ related to volumes by the function

$$h(\boldsymbol{t}) = e^{-\sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12} \right) \|\boldsymbol{b}_i^\star\|^2},$$

and it can be verified that $\sum_{\boldsymbol{t}\in\mathbb{N}^n} h(\boldsymbol{t})$ converges. In practice (see [10]), the m cells maximizing $h(\boldsymbol{t})$ (*i.e.* minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$) are almost the same as the cells maximizing $\text{vol}(\text{Ball}_n(R) \cap \mathcal{C}(\boldsymbol{t}))$.

However, the method of [10] was only heuristic. In Sect. 6, we modify that method to make it fully provable: for any integer $m > 0$, we can provably find the best m cells in essentially m polynomial-time operations and polynomial space (the m solutions are output as a stream).

5.6 Noise Distributions in the Unique Setting

We discuss how to evaluate the success probability of BDD discrete pruning in the unique setting for the natural partition. This can easily be adapted to Babai's partition, because it also relies on boxes. Following (7), it suffices to evaluate:

$$p(\boldsymbol{t}) = \Pr_{P,\boldsymbol{e}}(\boldsymbol{e} \in -\mathcal{C}(\boldsymbol{t})), \tag{9}$$

where P is the (random) pruning set, $\boldsymbol{e}$ is the BDD noise and $\mathcal{C}(\boldsymbol{t})$ is the cell of the tag $\boldsymbol{t}$. We now analyze the most frequent distributions for $\boldsymbol{e}$.

LWE and Gaussian Noise. The most important BDD case is LWE [39]. However, there are many variants of LWE using different distributions of the noise $\boldsymbol{e}$. We will use the continuous Gaussian distribution over $\mathbb{R}^n$, like in [39]. Many schemes actually use a discrete distribution, such as some discrete Gaussian distribution over $\mathbb{Z}^n$ (or something easier to implement): because this is harder to analyze, cryptanalysis papers such as [28,30] prefer to ignore this difference, and perform experiments to check if it matches with the theoretical analysis. The main benefit of the Gaussian distribution over $\mathbb{R}^n$ is that for any basis, each coordinate is a one-dimensional Gaussian.

Lemma 5. *Let $\boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{N}^n$ be a tag of the natural partition $\mathcal{C}_{\mathbb{N}}()$ with basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$. If the noise $\boldsymbol{e}$ follows the multivariate Gaussian distribution over $\mathbb{R}^n$ with parameter σ, then:*

$$p(\boldsymbol{t}) = \prod_{i=1}^{n} \left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i + 1}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) \right) \tag{10}$$

Spherical Noise. If the noise $\boldsymbol{e}$ is uniformly distributed over a centered ball, we can reuse the analysis of [10]:

Lemma 6. *Let $(\mathcal{C}, T)$ be a universal L-partition. Let $\boldsymbol{t} \in T$ be a tag. If the noise $\boldsymbol{e}$ is uniformly distributed over the n-dimensional centered ball of radius R, then:*

$$p(\boldsymbol{t}) = \frac{\operatorname{vol}(\mathcal{C}(\boldsymbol{t}) \cap \mathrm{Ball}_n(R))}{\operatorname{vol}(\mathrm{Ball}_n(R))} \tag{11}$$

For both Babai's partition $\mathcal{C}_{\mathbb{Z}}$ and the natural partition $\mathcal{C}_{\mathbb{N}}$, $\mathcal{C}(\boldsymbol{t})$ is the union of disjoint symmetric boxes, so the evaluation of (11) is reduced to the computation of the volume of a ball-box intersection, which was done in [10].

Product of Finite Distributions. We now consider a general distribution $\mathcal{D}$ for the noise $\boldsymbol{e}$ where each coordinate e_i is independently sampled from the uniform distribution over some finite set. This includes the box distribution, which is the uniform distribution over a set of the form $\prod_{i=1}^{n}\{a_i,\ldots,b_i\}$. The continuous Gaussian distribution and the uniform distribution over a ball are both invariant by rotation. But if the noise distribution $\mathcal{D}$ is not invariant by rotation, the tag probability $p(\boldsymbol{t})$ may take different values for the same $(\|\boldsymbol{b}_1^\star\|,\ldots,\|\boldsymbol{b}_n^\star\|)$, which is problematic for analysing the success probability. To tackle this issue, we reuse the following heuristic assumption introduced in [22]:

Heuristic 10 *([22, Heuristic 3]) The distribution of the normalized Gram-Schmidt orthogonalization $(\boldsymbol{b}_1^\star/\|\boldsymbol{b}_1^\star\|,\ldots,\boldsymbol{b}_n^\star/\|\boldsymbol{b}_n^\star\|)$ of a random reduced basis $(\boldsymbol{b}_1,\ldots,\boldsymbol{b}_n)$ looks like that of a uniformly distributed orthogonal matrix.*

We obtain:

Lemma 7. *Let $\mathcal{C}_\mathbb{N}$ be the natural partition. Let $\boldsymbol{t}\in\mathbb{N}^n$ be a tag. If the distribution of the noise $\boldsymbol{e}$ has finite support, then under Heuristic 10:*

$$p(\boldsymbol{t}) = \sum_{r\in E}\Pr_{\boldsymbol{e}}(\|\boldsymbol{e}\| = r)\times \Pr_{\boldsymbol{x}\leftarrow S_n}(\boldsymbol{x}\in\mathcal{C}(\boldsymbol{t})/r) \tag{12}$$

where $E\subseteq\mathbb{R}_{\geq 0}$ denotes the finite set formed by all possible values of $\|\boldsymbol{e}\|$ and S_n denotes the n-dimensional unit sphere.

6 Linear Optimization for Discrete Pruning

We saw in Sect. 5.6 how to compute or approximate the probability $p(\boldsymbol{t})$ that the cell of the tag $\boldsymbol{t}$ contains the BDD solution. From Lemma 4, we know that for any integer $m>0$, there are m tags which maximize $p(\boldsymbol{t})$ in the sense that any other tag must have a lower $p(\boldsymbol{t})$. To select optimal parameters for BDD discrete pruning, we want to find these m tags as fast as possible, possibly in m operations and polynomial-space (by outputting the result as a stream).

6.1 Reduction to Linear Optimization

We distinguish two cases:

– Selection based on expectation. Experiments performed in [10] show that in practice, the m tags $\boldsymbol{t}$ which maximize $\mathrm{vol}(\mathcal{C}_\mathbb{N}(\boldsymbol{t})\cap\mathrm{Ball}_n(R))$ are essentially the ones which minimize the expectation $\mathbb{E}\{\mathcal{C}_\mathbb{N}(\boldsymbol{t})\}$ where $\mathbb{E}\{C\} := \mathbb{E}_{\boldsymbol{x}\in C}(\|\boldsymbol{x}\|^2)$ over the uniform distribution. Cor. 3 in [10] shows that this expectation is:

$$\mathbb{E}\{\mathcal{C}_\mathbb{N}(\boldsymbol{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4}+\frac{t_i}{4}+\frac{1}{12}\right)\|\boldsymbol{b}_i^\star\|^2.$$

So we can assume that for a noise uniformly distributed over a ball (see Lemma 6), the m tags $\boldsymbol{t}$ which maximize $p(\boldsymbol{t})$ are the the tags which minimize $\mathbb{E}\{\mathcal{C}_\mathbb{N}(\boldsymbol{t})\}$.

– Gaussian noise. If the noise distribution is the continuous multivariate Gaussian distribution, Lemma 5 shows that $p(\boldsymbol{t})$ is given by (10). This implies that the m tags $\boldsymbol{t}$ which maximize $p(\boldsymbol{t})$ are the ones which minimize $-\log p(\boldsymbol{t})$

In both cases, we want to find the m tags $\boldsymbol{t} \in \mathbb{N}^n$ which minimize an objective function g of the form $g(\boldsymbol{t}) = \sum_{i=1}^{n} f(i, t_i)$, where $f(i, t_i) \geq 0$. The fact that the objective function can be decomposed as a sum of individual positive functions in each coordinate allows us to view this problem as a *linear optimization*. We will see that in the case that g has integral outputs, it is possible to provably find the best m tags which minimize such a function g in essentially m operations. If g is not integral, it is nevertheless possible to enumerate all solutions such that $g(\boldsymbol{t}) \leq R$ where R is an input, in time linear in the number of solutions. A special case is the problem of enumerating smooth numbers below a given number.

In practice, it is more efficient to rely on the expectation, because it is faster to evaluate. Fig. 4 shows how similar are the best tags with respect to one indicator compared to another: to compare two sets A and B formed by the best M tags, the graph displays $\#(A \cap B)/M$. For instance, the top curve confirms the ex-

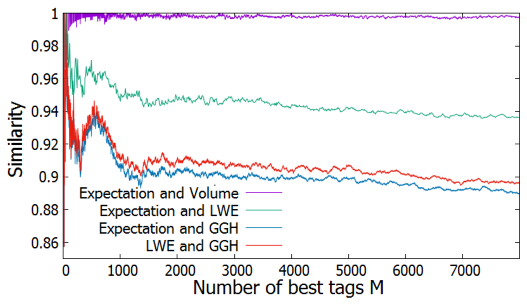

Fig. 4. Similarity between optimal sets of tags, depending on the objective function.

perimental result of [10] that the m tags $\boldsymbol{t}$ which maximize $\mathrm{vol}(\mathcal{C}_{\mathbb{N}}(\boldsymbol{t}) \cap \mathrm{Ball}_n(R))$ are almost the same as the ones which minimize the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$. The top second curve shows that the best tags that maximize the LWE probability are very close to those minimizing the expectation. The bottom two curves compare with the finite noise distribution arising in GGH challenges [23] (see Sect. B for details). In all cases, at most 10% of the best tags are different, and more importantly, we report that the global success probabilities are always very close, with a relative error typically $\leq 1\%$.

We conclude that in practice, the expectation is a very good indicator to select the best tags for the distributions studied in Sect. 5.6.

6.2 Limits of Orthogonal Enumeration

Aono and Nguyen [10, Sect. 6] presented a heuristic method to solve this linear optimization problem in the special case: $g(\boldsymbol{t}) = \mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} =$

$\sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \|\boldsymbol{b}_i^\star\|^2$, by noticing that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$ was the squared distance between a target point and a special lattice with a known orthogonal basis. This allowed to find all $\boldsymbol{t} \in \mathbb{N}^n$ such that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} \leq R$ for any R, using a variant [10, Alg. 6] of enumeration. And by using a binary search based on an early-abort variant, it was also possible to find an R yielding slightly more than m solutions.

[10, Sect. 6] reported that this algorithm worked very well in practice: if ℓ is the number of $\boldsymbol{t} \in \mathbb{N}^n$ such that $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} \leq R$, the number of nodes L of the enumeration algorithm [10, Alg. 6] seemed to be bounded by $O(\ell n)$, perhaps even $\ell \times n$. This was in contrast with the usual situation where the number of nodes of the enumeration tree is exponentially larger than the number of solutions. However, no rigorous result could be proved in [10], leaving it as an open problem to show the efficiency of [10, Alg. 6].

Surprisingly, we solve this open problem of [10] in the negative. More precisely, we show that there are cases where the number of nodes L of enumeration [10, Alg. 6] is exponentially larger than the number of solutions ℓ. To see this, consider the orthogonal lattice $\mathbb{Z}^n$ with the canonical basis. Then: $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} = \sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right)$. But we have:

Lemma 8. *Let $R = \frac{n}{12} + \frac{1}{2}$ and $n' = \lfloor n/10 \rfloor$. Then the number ℓ of $\boldsymbol{t} \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \leq R$ is exactly $n+1$. But the number ℓ' of $(x_{n-n'+1}, \ldots, x_n) \in \mathbb{N}^{n'}$ such that $\sum_{i=n-n'+1}^{n} \left(\frac{x_i^2}{4} + \frac{x_i}{4} + \frac{1}{12}\right) \leq R$ is $\geq 2^{n'}$.*

Proof. For the choice $R = \frac{n}{12} + \frac{1}{2}$, we have $\sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12}\right) \leq R$ if and only if all the t_i's are equal to zero, except at most one, which must be equal to one.

Furthermore, for any $(x_{n-n'+1}, \ldots, x_n) \in \{0,1\}^{n'}$, we have:

$$\sum_{i=n-n'+1}^{n} \left(\frac{x_i^2}{4} + \frac{x_i}{4} + \frac{1}{12}\right) \leq n' \left(\frac{1}{2} + \frac{1}{12}\right) \leq \frac{n}{10}\frac{7}{12} = \frac{7n}{120} < R.$$

□

It follows in this case that the number of nodes L of the enumeration algorithm [10, Alg. 6] for that R is at least exponential in n, though the number of solutions is linear in n.

6.3 Solving Linear Optimization

We show that a slight modification of orthogonal enumeration can solve the more general problem of linear optimization essentially optimally. This is based on two key ideas. The first idea is that when solving linear optimization, we may assume without loss of generality that each function $f(i,)$ is sorted by increasing value, with a starting value equal to zero, which changes the tree: $f(i,0) = 0$ and $f(i,j) \leq f(i,j')$ whenever $j \leq j'$. Indeed, it suffices to sort the values of $f(i,)$ if necessary and subtract the minimal value: however, note that for both the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4}+\frac{t_i}{4}+\frac{1}{12}\right)\|\boldsymbol{b}_i^\star\|^2$ and for $-\sum_{i=1}^{n}\log\left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i+1}{2}\cdot\|\boldsymbol{b}_i^\star\|\right)-\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma}\cdot\frac{t_i}{2}\cdot\|\boldsymbol{b}_i^\star\|\right)\right)$, the values of $f(i,)$ are already sorted. For instance, $\frac{t_i^2}{4}+\frac{t_i}{4}+\frac{1}{12}$ is an increasing function of t_i.

The second idea is that we may assume to simplify that f has integral values, which allows us to bound the running time of dichotomy. This is not directly true for the expectation $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} = \sum_{i=1}^{n}\left(\frac{t_i^2}{4}+\frac{t_i}{4}+\frac{1}{12}\right)\|\boldsymbol{b}_i^\star\|^2$. However, because we deal with integer lattices, the basis B is integral, the $\|\boldsymbol{b}_i^\star\|^2$'s are rational numbers with denominator $\operatorname{covol}(L(\boldsymbol{b}_1,\ldots,\boldsymbol{b}_{i-1}))^2$, so we can transform the expectation into an integer, by multiplying with a suitable polynomial-size integer.

First, we present a slight modification Alg. 3 of [10, Alg. 6], whose running time is provably essentially proportional to the number of solutions:

Theorem 11. *Assume that $f : \{1,\ldots,n\}\times\mathbb{N}\to\mathbb{R}$ satisfies $f(i,0)=0$ and $f(i,j)\geq f(i,j')$ for all i and $j>j'$. Given as input a number $R>0$, Alg. 3 outputs all $(v_1,\ldots,v_n)\in\mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i,v_i)\leq R$ using $O(nN+1)$ arithmetic operations and $\leq(2n-1)N+1$ calls to the function $f()$, where the number N is the number of $(v_1,\ldots,v_n)\in\mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i,v_i)\leq R$.*

Proof. To analyze the complexity of Alg. 3, let n_k denote the number of times we enter Lines 3–18, depending on the value of k, which is ≥ 1 and $\leq n$ at each Line 3. Then n_k can be decomposed as $n_k = a_k + b_k$, where a_k (resp. b_k) denotes the number of times we enter Lines 5–10 (resp. Lines 12–17). Notice that $a_{n+1}=0$ and a_1 is exactly the number N of $(v_1,\ldots,v_n)\in\mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i,v_i)\leq R$. And if $1<i\leq n$, then a_i is the number of times that the variable k is decremented from i to $i-1$. Similarly, $b_n=1$, and if $1\leq i\leq n$, then b_i is the number of times that the variable k is incremented from i to $i+1$. By Line 1 (resp. 14), the initial (resp. final) value of k is n (resp. $n+1$). Therefore, for any $1\leq i\leq n-1$, the number of times k is incremented from i to $i+1$ must be equal to the number of times k is decremented from $i+1$ to i, in other words: $b_i = a_{i+1}$. Thus, the total number of loop iterations is:

$$\sum_{i=1}^{n} n_i = \sum_{i=1}^{n}(a_i+b_i) = N+1+2\sum_{i=2}^{n} a_i.$$

Note that because $f(i,0)=0$, any partial assignment $\sum_{i=i_0}^{n} f(i,v_i)\leq R$ can be extended to a larger partial assignment $\sum_{i=1}^{n} f(i,v_i)\leq R$, which implies that $a_1\geq a_2\geq\ldots a_n$. It follows that the total number of loop iterations is:

$$\sum_{i=1}^{n+1} n_i \leq N+1+2(n-1)N = (2n-1)N+1.$$

For each loop iteration (Lines 3–18), the number of arithmetic operations performed is $O(1)$ and the number of calls to $f()$ is exactly one. It follows that the total number of arithmetic operations is $O(nN+1)$ and the number of calls to $f()$ is $\leq(2n-1)N+1$. □

We showed that the number of nodes in the search tree is linear in the number of solutions. Next, we present Alg. 4, which is a counting version of Alg. 3:

Theorem 12. *Assume that $f : \{1,\ldots,n\} \times \mathbb{N} \to \mathbb{R}$ satisfies $f(i,0) = 0$ and $f(i,j) \geq f(i,j')$ for all i and $j > j'$. Given as input two numbers $R > 0$ and $M > 0$, Alg. 4 decides if is $N \geq M$ or $N < M$, where N is the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq R$. Furthermore, if $N \geq M$, the number of arithmetic operations is $O(N)$, and otherwise, the number of arithmetic operations is $O(nN+1)$, and the algorithms outputs N.*

Proof. Similarly to the proof of Th. 11, let n_k denote the number of times we enter Lines 3–17, depending on the value of k, which is ≥ 1 and $\leq n$ at each Line 3. Then n_k can be decomposed as $n_k = a_k + b_k$, where a_k (resp. b_k) denotes the number of times we enter Lines 5–9 (resp. Lines 11–16).

Let M be the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq R$. If $M \leq N$, then Alg. 4 will perform the same operations as Alg. 3 (except Line. 6), so the cost is $O(nM+1) \leq O(nN+1)$ arithmetic operations. Otherwise, $M > N$, which means that the while loop will stop after exactly N iterations, and the total number of operations is therefore $O(N)$. □

Our main result states that if the function f is integral, given any M, Alg. 5 finds the best N assignments in time M where $M \leq N \leq (n+1)M$:

Theorem 13. *Assume that $f : \{1,\ldots,n\} \times \mathbb{N} \to \mathbb{N}$ satisfies $f(i,0) = 0$ and $f(i,j) < f(i,j')$ for all i and $j > j'$. Assume that $f(i,j) \leq j^{O(1)} 2^{n^{O(1)}}$. Given as input a number $M > 1$, Alg. 5 outputs the N assignments $(v_1,\ldots,v_n) \in \mathbb{N}^n$ which minimize $\sum_{i=1}^n f(i,v_i)$ in time $O(n(n+1)M) + n^{O(1)} + O(\log_2 M)$, where the number N satisfies: $M \leq N \leq (n+1)M$.*

Proof. We have the following invariant at the beginning of each loop iteration: the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq R_0$ is $< M$, and the the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq R_1$ is $\geq M$. Initially, this holds because the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq 0$ is 1 and the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq \sum_{i=1}^n f(i,\lceil M^{1/n} \rceil)$ is $\geq (M^{1/n})^n = M$. Furthermore, the loop preserves the invariant by definition of the loop. Since the length $R_1 - R_0$ decreases by a factor two, it follows that the number of loop iterations is $\leq \log_2(\sum_{i=1}^n f(i,\lceil M^{1/n} \rceil))$.

After the loop, we must have $R_0 = R_1 - 1$. Let N_1 (resp. N_0) be the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) \leq R_1$ (resp. R_0) after the loop. By the invariant, we know that $N_0 < M \leq N_1$. We claim that $(N_1 - N_0) \leq nM$, which implies that $N_1 \leq (n+1)M$. Notice that $N_1 - N_0$ is the number of $(v_1,\ldots,v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^n f(i,v_i) = R_1$. For any such assignment, one of the v_i's must be ≥ 1: if we decrement that v_i, we get a cost $< R_1$, so it must be $\leq R_0$ because $R_0 = R_1 - 1$, which means that this assignment is counted by N_0. Since we have at most n possibilities for i, it follows that $N_1 - N_0 \leq nM$. □

Furthermore, Alg. 5 uses negligible space, except that the output is linear in M: the best tags are actually output as a stream. If we sort the N tags, which requires space, we could output exactly the best M tags.

Algorithm 3 Enumeration of low-cost assignments

```
Input: A function $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geq f(i, j')$ for all $i$ and $j > j'$; a bound $R > 0$.
Output: All $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leq R$.
$v_1 = v_2 = \dots = v_n = 0$ and $\rho_{n+1} = 0$ and $k = n$
while true do
    $\rho_k = \rho_{k+1} + f(k, v_k)$  // cost of the tag $(0, \dots, 0, v_k, \dots, v_n)$
    if $\rho_k \leq R$ then
        if $k = 1$ then
            return $(v_1, \dots, v_n)$; (solution found)
            $v_k \leftarrow v_k + 1$
        else
            $k \leftarrow k - 1$ and $v_k \leftarrow 0$ // going down the tree
        end if
    else
        $k \leftarrow k + 1$ // going up the tree
        if $k = n + 1$ then
            exit (no more solutions)
        else
            $v_k \leftarrow v_k + 1$
        end if
    end if
end while
```

7 Quantum Speed-up of Discrete Pruning

We present a quadratic quantum speed-up for discrete pruning, namely:

Theorem 14. *There is a quantum algorithm which, given $\varepsilon > 0$, a number $M > 0$, and an LLL-reduced basis B of a full-rank lattice L in $\mathbb{Z}^n$, outputs the shortest non-zero vector in $L \cap P$ in time $O(n^2\sqrt{M})\texttt{poly}(\log(n), \log(M), \log(1/\epsilon), \beta)$ with error probability ε. Here, β denotes the bitsize of the vectors of B, $P = \cup_{\boldsymbol{t} \in U} \mathcal{C}_{\mathbb{N}}(\boldsymbol{t})$ where $\mathcal{C}_{\mathbb{N}}()$ is the natural partition with respect to B, U is formed by the N tags $\boldsymbol{t}$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$, for some $M \leq N \leq 32n^2M$ with probability at least $1 - \varepsilon/2$. If the algorithm is further given a target $\boldsymbol{u} \in \mathbb{Z}^n$, it also outputs the shortest vector in $(L - \boldsymbol{u}) \cap P$.*

By comparison, opening all the cells returned by Alg. 5 of Sect. 6 does the same in $O(M)$ poly-time operations, except that the upper bound on N is slightly

Algorithm 4 Counting low-cost assignments

```
Input: A function $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geq f(i, j')$ for all $i$ and $j > j'$; a bound $R > 0$ and a number $M \geq 0$.
Output: Decide if the number of $(v_1, \dots, v_n) \in \mathbb{N}^n$ such that $\sum_{i=1}^{n} f(i, v_i) \leq R$ is $\geq M$ or $< M$.
 1: $v_1 = v_2 = \cdots = v_n = 0$ and $\rho_{n+1} = 0$ and $k = n$ and $m = 0$
 2: while $m < M$ do
 3:   $\rho_k = \rho_{k+1} + f(k, v_k)$  // cost of the tag $(0, \dots, 0, v_k, \dots, v_n)$
 4:   if $\rho_k \leq R$ then
 5:     if $k = 1$ then
 6:       $m \leftarrow m + 1$ and $v_k \leftarrow v_k + 1$ (one more solution)
 7:     else
 8:       $k \leftarrow k - 1$ and $v_k \leftarrow 0$ // going down the tree
 9:     end if
10:   else
11:     $k \leftarrow k + 1$ // going up the tree
12:     if $k = n + 1$ then
13:       return $m < M$ // no more solutions
14:     else
15:       $v_k \leftarrow v_k + 1$
16:     end if
17:   end if
18: end while
19: return $m \geq M$
```

Algorithm 5 Enumeration of lowest-cost assignments

```
Input: A function $f : \{1, \dots, n\} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that $f(i, 0) = 0$ and $f(i, j) \geq f(i, j')$ for all $i$ and $j > j'$; a number $M > 0$.
Output: Output the $N$ assignments $(v_1, \dots, v_n) \in \mathbb{N}^n$ that minimize $\sum_{i=1}^{n} f(i, v_i)$, where $M \leq N \leq nM$.
 1: $R_0 \leftarrow 0$ and $R_1 \leftarrow \sum_{i=1}^{n} f(i, \lceil M^{1/n} \rceil)$;
 2: while $R_0 < R_1 - 1$ do
 3:   Call Alg. 4 with $R = \lfloor (R_0 + R_1)/2 \rceil$ and $M$
 4:   if number of solutions $\geq M$ then
 5:     $R_1 \leftarrow R$
 6:   else
 7:     $R_0 \leftarrow R$
 8:   end if
 9: end while
10: Call Alg. 3 with $R_1$.
```

lower. The proof of Th. 14 has two parts: first, we show how to determine the best N cells without computing them, for some N close to M, with high probability; then we find the best candidate inside these N cells. Both rely on a tree interpretation. Alg. 3 can be seen as a backtracking algorithm on a tree $\mathcal{T}(R)$, where each node can be encoded as $(*, \cdots, *, v_k, \cdots, v_n)$. The root is encoded as $(*, \cdots, *)$. Given a node $(*, \cdots, *, v_k, \cdots, v_n)$, if $k = 1$, then it is a leaf. If $\sum_{i=k}^{n} f(i, v_i) > R$, then it is also a leaf. If $\sum_{i=k}^{n} f(i, v_i) \leq R$, then its children are $(*, \cdots, *, v_{k-1}, v_k, \cdots, v_n)$, where v_{k-1} can take all integer values between 0 and $\rho_{v_k,\cdots,v_n}$. Here $\rho_{v_k,\cdots,v_n}$ is the smallest integer such that $f(i-1, \rho_{v_k,\cdots,v_n}) + \sum_{i=k}^{n} f(i, v_i) > R$. In case of discrete pruning, f is quadratic. We can compute $\rho_{v_k,\cdots,v_n}$ and build the black-box on $\mathcal{T}(R)$.

7.1 Determining the best cells implicitly

Given a number $M > 0$, Alg. 5 finds (in time essentially M) the best N vectors $\boldsymbol{t} \in \mathbb{N}^n$ (for some N close to M) minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\} = \sum_{i=1}^{n} \left(\frac{t_i^2}{4} + \frac{t_i}{4} + \frac{1}{12} \right) \|\boldsymbol{b}_i^\star\|^2$ by minimizing instead the function:

$$g(v_1, \cdots, v_n) = \sum_{i=1}^{n} f(i, v_i) = \sum_{i=1}^{n} v_i(v_i + 1)\|\boldsymbol{b}_i^\star\|^2 = \sum_{i=1}^{n} \alpha_i v_i(v_i + 1).$$

This is done by finding a suitable radius R by dichotomy, based on logarithmically many calls to Alg. 4 until the number of solutions is close to M, and eventually enumerating the marked leaves of a search tree by Alg. 3. Both Alg. 3 and Alg. 4 can be viewed as algorithms exploring a tree $\mathcal{T}(R)$ depending on a radius $R > 0$: Alg. 4 decides if the number $\#S(\mathcal{T}(R))$ of marked leaves (*i.e.* the number of outputs returned by Alg. 3) is $\geq$ or $<$ than an input number; Alg. 3 returns all the marked leaves.

This tree interpretation gives rise to Alg. 6, which is our quantum analogue of Alg. 5 with the following differences: we are only interested in finding a suitable radius R such that $N = \#S(\mathcal{T}(R))$ is close to M up to a factor of $32n^2$, with correctness probability at least $1 - \varepsilon/2$, because enumerating all the marked leaves would prevent any quadratic speed up. We replace Alg. 4 by the quantum tree size estimation algorithm of [9]: this gives a quadratic speed up, but approximation errors slightly worsen the upper bound on N. The input $(\alpha_1, \cdots, \alpha_n)$ of Alg. 6 corresponds to $(\|\boldsymbol{b}_1^\star\|^2, \cdots, \|\boldsymbol{b}_n^\star\|^2)$, where $(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ is an integer basis. We know that $(\|\boldsymbol{b}_1^\star\|^2, \cdots, \|\boldsymbol{b}_n^\star\|^2) \in \mathbb{Q}^n$, but by suitable multiplication preserving polynomial sizes, we may assume that $(\|\boldsymbol{b}_1^\star\|^2, \cdots, \|\boldsymbol{b}_n^\star\|^2) \in \mathbb{N}^n$. The order between the $\|\boldsymbol{b}_i^\star\|^2$'s doesn't matter in our analysis. We can assume that $\|\boldsymbol{b}_1^\star\|^2 \leq \cdots \leq \|\boldsymbol{b}_n^\star\|^2$. We show that Alg. 6 finds a radius R corresponding to the best M cells in approximately $\sqrt{M}$ quantum operations:

Theorem 15. *The output R of Alg. 6 satisfies $M \leq \#S(\mathcal{T}(R)) \leq 32n^2 M$ with probability $\geq 1 - \varepsilon/2$. Alg. 6 runs in quantum time $O(n^2\sqrt{M}\texttt{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$ where β is the bitsize of the basis vectors $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$. The algorithm needs $O(\texttt{poly}(n, \log(M), \log(1/\epsilon)))$ qubits.*

Algorithm 6 Computing implicitly the best cells quantumly

```
Input: $\varepsilon, M > 0$ and $(\alpha_1, \cdots, \alpha_n) \in \mathbb{N}^n$ with $\alpha_1 \leq \cdots \leq \alpha_n$ such that the input
    $f : \{1, \cdots, n\} \times \mathbb{N} \to \mathbb{N}$ of Alg. 3 satisfies $f(i, x) = \alpha_i x(x+1)$
Output: $R$ such that $M \leq \#S(\mathcal{T}(R)) \leq 32n^2 M$ with probability $\geq 1 - \varepsilon$
 1: $r \leftarrow \lceil \log_2(\sum_{i=1}^n f(i, \lceil (4nM)^{1/n} \rceil)) \rceil$ and $R \leftarrow \sum_{i=1}^n f(i, \lceil (4nM)^{1/n} \rceil)$ and
    $R_0 \leftarrow 0$ and $R_1 \leftarrow R$
 2: while $R_1 - R_0 > 1$ do
 3:     Call TreeSizeEstimation$(\mathcal{T}_2(R), 16n^2M, 1/2, \varepsilon r/2, 2)$
 4:     if the answer is "$\mathcal{T}_2(R)$ contains more than $16n^2M$ vertices" then
 5:         $R_1 \leftarrow R$ and $R \leftarrow \lfloor (R_0 + R_1)/2 \rceil$
 6:     else if the answer is "$\mathcal{T}_2(R)$ contains $\hat{T}$ vertices" with $\hat{T} < 3(2n-1)M$
        then
 7:         $R_0 \leftarrow R$ and $R \leftarrow \lfloor (R_0 + R_1)/2 \rceil$
 8:     else
 9:         Return $R$
10:     end if
11: end while
12: Return $R_0$
```

7.2 Finding the best lattice vector

We now know R such that the number N of $(v_1, \cdots, v_n) \in \mathbb{N}^n$ which satisfies $\sum_{i=1}^n f(i, v_i) \leq R$ is in $[M, 32n^2M]$ with probability at least $1 - \varepsilon/2$. All these solutions are leaves of the tree $\mathcal{T}(R)$ and they form the set U of the best N tags minimizing $\boldsymbol{t}$ minimizing $\mathbb{E}\{\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})\}$. Let $P = \cup_{\boldsymbol{t} \in U} \mathcal{C}_{\mathbb{N}}(\boldsymbol{t})$ where $\mathcal{C}_{\mathbb{N}}()$ is the natural partition with respect to the input basis B. We would like to find a shortest non-zero vector in $L \cap P$ for the SVP setting, or the shortest vector in $(L - \boldsymbol{u}) \cap P$ in the CVP setting, when we are further given target $\boldsymbol{u} \in \mathbb{Z}^n$. To do this, we notice that it suffices to apply **FindMin2** (in App), provided that the basis $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ is LLL-reduced. More precisely, we call **FindMin2**$(\mathcal{T}(R), \mathcal{P}, h, \|\mathbf{b}_1\|^2, d, 32n^2M, \varepsilon/2)$. Here $\mathcal{P}$ is the predicate which returns true on a node iff it is a leaf encoded as $(x_1, \cdots, x_n)$ such that $g(x_1, \cdots, x_n) = \sum_{i=1}^n f(i, x_i) \leq R$. $h_V(x_1, \cdots, x_n)$ is the predicate which indicates if the square of the norm of the lattice vector in the cell of tag $(x_1, \cdots, x_n)$ is $\leq V$. The time complexity is $O(n^2\sqrt{M}\texttt{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$.

Since the subroutine of determining the best cells and the one of finding a shortest non-zero vector, both have an error probability $\varepsilon/2$, by union bound, the total error probability is ε. We thus have proved Th. 14.

7.3 The Case of Extreme Pruning

In this section, we explain how to tackle the extreme pruning case, where one wants to run discrete pruning over many reduced bases. Due to space limitations, we only give a proof sketch, but the main ideas are the same.

Given m LLL-reduced bases $(\mathbf{B}_1, \cdots, \mathbf{B}_m)$ of the same integer lattice L of rank n, we define for each basis $\mathbf{B}_i$ a function $g_i : \mathbb{N}^n \to \mathbb{Q}$ such that $g_i(x_1, \cdots, x_n) = \sum_{j=1}^n \|\boldsymbol{b}^\star_{i,j}\|^2 x_i(x_i + 1)$, where $(\boldsymbol{b}^\star_{i,1}, \cdots, \boldsymbol{b}^\star_{i,n})$ is the Gram-Schmidt orthogonalization of the basis $\mathbf{B}_i$. Here, we want to first find the $\texttt{poly}(n) * M$ best cells with respect to all of the functions g_i altogether, and then find the shortest vector in these cells. Both steps have complexity $O(\sqrt{M}\texttt{poly}(n, \log M, \log 1/\varepsilon, \beta))$, where ε is the total error probability and where β is the bitsize of the vectors of the input bases.

Theorem 16. *There is a quantum algorithm which, given $\varepsilon > 0$, a number $M > 0$, and m LLL-reduced bases $(\mathbf{B}_1, \cdots, \mathbf{B}_m)$ of an n-rank integer lattice L, outputs the shortest non-zero vector in $L \cap P$ in time $O(\sqrt{M}\texttt{poly}(n, \log M, \log 1/\varepsilon, \beta))$ with error probability ε. Here, β denotes the maximum bitsize of the vectors of all given bases, $P = \cup_{(i,\boldsymbol{t})\in U} \mathcal{C}_{\mathbb{N}}(i, \boldsymbol{t})$ where $\mathcal{C}_{\mathbb{N}}(i, \cdot)$ is the natural partition with respect to B_i, U is formed by the N tuples $(i, \boldsymbol{t}) \in \{1, \cdots, m\} \times \mathbb{N}^n$ minimizing $g_i(\boldsymbol{t})$ among all tuples, for some $N = \texttt{poly}(n) * M$ with probability at least $1 - \varepsilon/2$. If the algorithm is further given a target $\boldsymbol{u} \in \mathbb{Z}^n$, it also outputs the shortest vector in $(L - \boldsymbol{u}) \cap P$.*

The main idea of the proof is the following. For each basis $\mathbf{B}_i$, there is a backtracking tree with respect to the function g_i as we explained in the previous section. We put all these trees together and obtain one single tree. We first apply the **TreeSizeEstimation** algorithm several times to find a good common radius R for all functions g_i by dichotomy, such that the total number of good cells in all trees is $\texttt{poly}(n) * M$. After that, we apply **FindMin2** to find the shortest vector among all these cells. Remark that in the previous section, we required the function g to have integral values, and this was achieved by multiplying all $\|\boldsymbol{b}^\star_i\|^2$ by a common denominator. Instead, we here want to keep the output rational, which is proved sufficient by the following lemma:

Lemma 9. *Given a basis $(\mathbf{b}_1, \cdots, \mathbf{b}_n)$ of an integer lattice L, $g : \mathbb{N}^n \to \mathbb{Q}$ such that $g(x_1, \cdots, x_n) = \sum_{i=1}^n \|\boldsymbol{b}^\star_i\|^2 x_i(x_i+1)$, we denote $\mathcal{T}(R)$ the backtracking tree for finding all solutions of $g(x_1, \cdots, x_n) \leq R$, $\mathcal{T}_2(R)$ the corresponding binary tree. For all $R \in \mathbb{R}^+$, $\#S(\mathcal{T}_2(R + \delta)) \leq 2n\#S(\mathcal{T}_2(R))$, where $\delta = \frac{1}{\prod_{i=1}^n \Delta_i}$ and $\Delta_i = \mathrm{covol}(\mathbf{b}_1, \cdots, \mathbf{b}_i)^2 = \prod_{j=1}^i \|\boldsymbol{b}^\star_i\|^2$.*

The proof of this lemma is the same as the proof of Lemma 10 in the App. by noticing that $\prod_{i=1}^n \Delta_i$ is a common denominator of all $\|\boldsymbol{b}^\star_i\|^2$.

For each basis $\mathbf{B}_i$, we define δ_i as in Lemma 9. In the dichotomy step, we stop when the difference of the two terms is smaller than $\min_{j\in\{1,\cdots,m\}} \delta_j$. The other steps are the same as in the previous section.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers 16H02780 and 16H02830, and partially supported by a mobility scholarship of the third author at the University of Tokyo in the frame of the Erasmus Mundus Action 2 Project

TEAM Technologies for Information and Communication Technologies, funded by the European Commission. This publication reflects the view only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein. We thank Frédéric Magniez for helpful discussions.

References

1. D. Aggarwal, D. Dadush, O. Regev, and N. Stephens-Davidowitz. Solving the shortest vector problem in 2^{n} time using discrete gaussian sampling: Extended abstract. In *Proceedings of 47th ACM STOC*, pages 733–742, 2015.
2. M. Ajtai. Generating hard instances of lattice problems. In *Proc. 28th ACM STOC*, pages 99–108, 1996.
3. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. of 33rd STOC*, pages 601–610. ACM, 2001.
4. M. Albrecht, J. Schanck, and D. Bernstein. Messages on the NIST pqc mailing-list in May, 2018.
5. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the LWE, NTRU schemes! Posted on the pqc-forum on Feb. 1, 2018. Available at `https://estimate-all-the-lwe-ntru-schemes.github.io/paper.pdf`.
6. M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. Postlethwaite, and M. Stevens. New records for lattice SVP challenges, 2018.
7. E. Alkim, N. Bindel, J. A. Buchmann, Ö. Dagdelen, E. Eaton, G. Gutoski, J. Krämer, and F. Pawlega. Revisiting TESLA in the quantum random oracle model. In *Proc. PQCrypto 2017*, volume 10346 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2017.
8. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *Proc. 25th USENIX*, pages 327–343. USENIX, 2016.
9. A. Ambainis and M. Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proc. STOC '17*. ACM, 2017.
10. Y. Aono and P. Q. Nguyen. Random sampling revisited: Lattice enumeration with discrete pruning. In *Advances in cryptology—EUROCRYPT 2017 Part II*, volume 10211 of *LNCS*, pages 65–102. Springer, 2017.
11. Y. Aono, P. Q. Nguyen, T. Seito, and J. Shikata. Lower bounds on lattice enumeration with extreme pruning. In *Proc. of 38th CRYPTO, Part II*, volume 10992 of *LNCS*. Springer, 2018.
12. Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. *IACR Cryptology ePrint Archive*, 2016:146, 2016. Full version of EUROCRYPT 2016.
13. L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. In *Proc. STACS'85*, volume 182 of *LNCS*, pages 13–20. Springer, 1985.
14. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proc. 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 10–24, 2016.
15. C. H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, Aug. 1989.

16. J. Buchmann, N. Büscher, F. Göpfert, S. Katzenbeisser, J. Krämer, D. Micciancio, S. Siim, C. van Vredendaal, and M. Walter. Creating cryptographic challenges using multi-party computation: The lwe challenge. In *Proceedings of the 3rd ACM AsiaPKC*, pages 11–20, New York, NY, USA, 2016. ACM.
17. Y. Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Univ. Paris 7, 2013.
18. Y. Chen and P. Q. Nguyen. BKZ 2.0: better lattice security estimates. In *Proc. ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011.
19. R. del Pino, V. Lyubashevsky, and D. Pointcheval. The whole is less than the sum of its parts: Constructing more efficient lattice-based AKEs. In *Proc. SCN 2016*, volume 9841 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2016.
20. M. Fukase and K. Kashiwabara. An accelerated algorithm for solving SVP based on statistical analysis. *JIP*, 23(1):67–80, 2015.
21. N. Gama and P. Q. Nguyen. Predicting Lattice Reduction. In *Proc. of Eurocrypt'08, LNCS, Springer Verlag*, pages 31–51, 2008.
22. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, 2010.
23. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. CRYPTO 1997*, volume 1294 of *LNCS*, pages 112–131. Springer, 1997.
24. A. Hülsing, J. Rijneveld, J. M. Schanck, and P. Schwabe. Ntru-hrss-kem: Algorithm specifications and supporting documentation. NIST submission.
25. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. 15th ACM STOC*, pages 193–206, 1983.
26. T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Proc. CRYPTO 2015 - Part I*, volume 9215 of *LNCS*. Springer, 2015.
27. T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptography*, 77(2-3):375–400, 2015.
28. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011.
29. R. Lindner and M. Rückert. TU Darmstadt lattice challenge. Available at `http://www.latticechallenge.org/`.
30. M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In *Topics in Cryptology - Proc. CT-RSA 2013*, volume 7779 of *LNCS*. Springer, 2013.
31. C. Ludwig. A faster lattice reduction method using quantum search. In *ISAAC 2003, Kyoto, Japan, December 15-17, 2003, Proceedings*, volume 2906 of *LNCS*, pages 199–208. Springer, 2003.
32. D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, 2010.
33. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proc. ACM-SIAM SODA*, pages 1468–1480, 2010.
34. A. Montanaro. Quantum walk speedup of backtracking algorithms. *ArXiv*, 2015.
35. P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.
36. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2):181–207, 2008.
37. NIST. Round 1 submissions for post-quantum cryptography standardization. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.
38. M. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, 1981.

39. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. 37th ACM STOC*, pages 84–93, 2005.
40. M. Schneider and N. Gama. SVP challenge. Available at `http://www.latticechallenge.org/svp-challenge/`.
41. C. P. Schnorr. Lattice reduction by random sampling and birthday methods. In *Proc. STACS 2003*, volume 2607 of *LNCS*, pages 145–156. Springer, 2003.
42. C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–199, 1994.
43. C.-P. Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proc. of Eurocrypt '95*, volume 921 of *LNCS*, pages 1–12. IACR, Springer-Verlag, 1995.
44. Security Innovation. NTRU challenge. Available at `https://www.securityinnovation.com/products/ntru-crypto/ntru-challenge`.
45. Y. Yu and L. Ducas. Second order statistical behavior of LLL and BKZ. In *Proc. SAC 2017*, pages 3–22, 2017.

A Revisiting Lindner-Peikert and Liu-Nguyen

We show that Lindner-Peikert's Nearest Planes algorithm [28] as randomized by Liu-Nguyen [30] can be viewed as a special case of discrete pruning.

Let $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ be a basis of a full-rank lattice L in $\mathbb{Z}^n$: denote by $B^\star = (\boldsymbol{b}_1^\star, \ldots, \boldsymbol{b}_n^\star)$ its Gram-Schmidt orthogonalization. Lindner and Peikert [28] presented a generalization of Babai's nearest plane algorithm, by adding some exhaustive search to increase the success probability, at the expense of the running time. Instead of choosing the closest plane in every i-th level, the NearestPlanes algorithm (Alg. 7) enumerates d_i distinct planes. We note that Alg. 7 is actually enumerating the set $L \cap (P + \boldsymbol{u})$ where $P = \{\sum_{i=1}^n x_i \boldsymbol{b}_i^\star : -\frac{d_i}{2} \leq x_i < \frac{d_i}{2}\}$. This corresponds to BDD discrete pruning with the natural partition using all the $\prod_{i=1}^n d_i$ tags $(t_1, \ldots, t_n) \in \prod_{i=1}^n \{0, 1, \cdots, d_i - 1\}$.

Here, we see that the choice $\prod_{i=1}^n \{0, 1, \cdots, d_i - 1\}$ of tags is rather arbitrary, and intuitively explains why [28] is not optimal. Note that [28, Sect. 4] mentions as a footnote: "*One could further generalize the algorithm to search within an approximate ball made up of 'bricks', thus capturing even more of the Gaussian without adding much more to the search space. However, this would significantly complicate the analysis, and we find that the present approach is already very effective.* ", which seems to correspond to discrete pruning with Babai's partition. Our framework allows to handle this "significantly more complicated analysis".

Algorithm 7 The NearestPlanes Algorithm [28]

Input: A lattice basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, a vector $\boldsymbol{d} = (d_1, d_2, \ldots, d_n) \in \mathbb{N}^n$, a target point $\boldsymbol{u} \in \mathbb{Q}^n$.
Output: A set of $\prod_{i=1}^n d_i$ distinct lattice vectors in $\mathcal{L}(\mathbf{B})$ close to $\boldsymbol{u}$.

```
1: if $m = 0$ then
2:     Return $\mathbf{0}$
3: else
4:     Compute the $d_n$ integers $c_1, c_2, \ldots, c_{d_n} \in \mathbb{Z}$ closest to $\langle \boldsymbol{b}_n^\star, \boldsymbol{u} \rangle / \langle \boldsymbol{b}_n^\star, \boldsymbol{b}_n^\star \rangle$
5:     Return $\bigcup_{i \in [d_n]} (c_i \boldsymbol{b}_n + NearestPlanes(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, (d_1, \ldots, d_{n-1}), \boldsymbol{u} - c_i \boldsymbol{b}_n\})$
6: end if
```

B Experiments

Most of our experiments were performed by a standard server with two Intel Xeon E5-2660 CPUs and 256-GB RAMs. We used the following public instances.

Random Lattices from the SVP Challenge. Throughout this section, we use n-dim random lattices generated by the SVP challenge generator [40]; these lattices are uniformly distributed among integer lattices of co-volume a fixed random prime of bit-length $10n$. We sometimes use the words "random PBKZ-β basis" or "random LLL basis" to denote the basis generated by the generator and reduced using the progressive BKZ library with target reduction level β [12].

LWE Challenge. The LWE Challenge [16] provides instances parametrized by the dimension n and the error ratio α, and other parameters are derived from them: the number m of samples is n^2, the modulus q is the first prime number larger than n^2 and the noise distribution is a discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter $s = \sqrt{2\pi}\alpha q$, which we heuristically assume to be close to a multivariate Gaussian over $\mathbb{R}^m$ of parameter $\sigma = \alpha q$. Any LWE instance is a BDD instance of some m-dim lattice. However, in practice, it is more efficient to choose a random subset of $m' \leq m$ samples, in which case the BDD lattice has dimension m': this is the folklore sublattice attack, where the optimal choice of m' depends on the basis reduction quality.

B.1 Accuracy of the Success Probability Analysis

We report on experiments supporting the validity of the success probability analysis of Sect. 5.4.

Approximation Setting of CVP. For a random PBKZ-40 reduced 100-dim basis B, we selected the 1,000 best tags by Alg. 5 with respect to cell expectation. Based on (2) and (6), the theoretical success probability is $\sum_{\boldsymbol{t}\in T} \mathrm{vol}(\mathrm{Ball}_n(1.4GH(L)) \cap C(\boldsymbol{t}))/\mathrm{covol}(L)$. Then we generated random target points $\boldsymbol{u}$ as integer points uniformly distributed over $[0, \mathrm{covol}(L)-1]^n$. We computed the experimental success rate that $\boldsymbol{u} \in \bigcup C(\boldsymbol{t})$ over the 1,000 tags. Fig. 5 shows the experimental results over 1,000 random target vectors for each random lattice with 122 different seeds. Overall, the average theoretical success probability is 0.183, while the experimental success rate is 0.192.

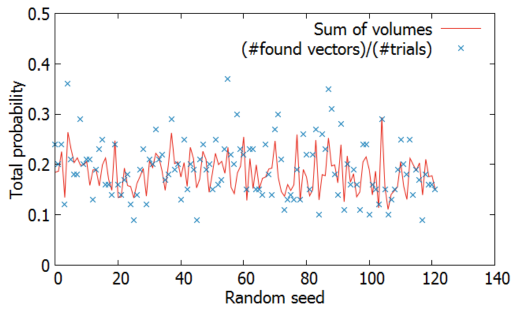

Fig. 5. Comparison between the theoretical and experimental success probability on randomly generated CVP instances in dimension 100.

Unique Setting: GGH-Key Recovery. We illustrate the analysis of Sect. 5.6 with GGH-key recovery in dimension 200: these instances were used by [30] to illustrate cylinder pruning, solutions correspond to the secret key in public challenges of the GGH cryptosystem [23]. An n-dim GGH key-recovery can be viewed as n BDD instance over a given basis B and n targets $\boldsymbol{u}$ of the form $\boldsymbol{w}_i = (0, \ldots, 0, z, 0, \ldots, 0)$ where $z = 4\lfloor\sqrt{n}+1\rceil$ [30]. By construction, we know that each coordinate of the noise $\boldsymbol{e}$ is uniformly distributed over $\{-4, -3, \ldots, 3\}$.

To compute the radius probability $\Pr_{\boldsymbol{e}}[\|\boldsymbol{e}\|^2 = r^2]$, we use the method of Sect. 5.6 starting with $F_{[i:i],0} = F_{[i:i],16} = 1/8$ and $F_{[i:i],1} = F_{[i:i],4} = F_{[i:i],9} = 1/4$ for all $i = 1, \ldots, n$: results are shown on the left-side of Figure 6. The success probability was derived from Sect. 5.4. The right-hand side of Fig. 6 shows the comparison between theoretical and experimental success probability for many PBKZ-60-reduced bases, using the $M = 1, 10, 100, 1000$ and 10000 best tags. This allows to compare the efficiency of discrete pruning with that

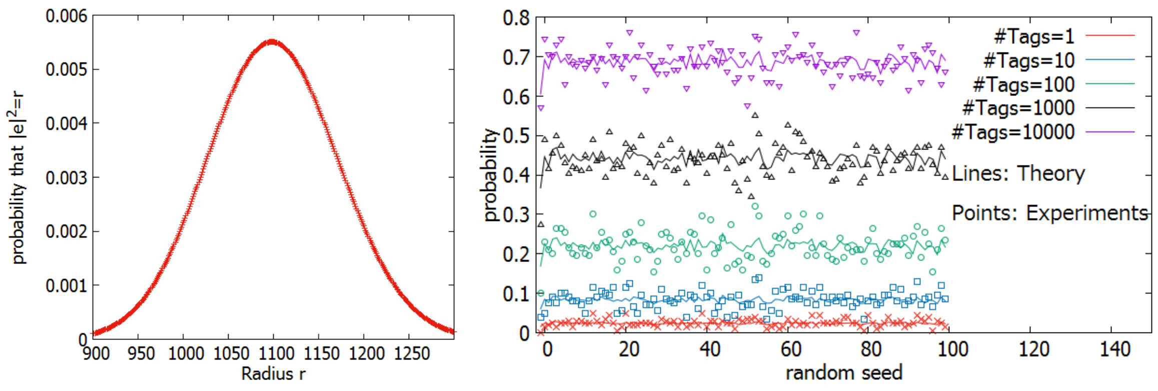

Fig. 6. (Left) The graph of $\Pr_{\boldsymbol{e}}[\|\boldsymbol{e}\|^2 = r^2]$ for GGH-200's noise; (Right) Theoretical and experimental success probability on PBKZ-60 bases of the GGH-200 lattice.

of cylinder pruning (as experimented in [30]): the leftmost points corresponding to $x = -1$ recall the success probabilities obtained by [30]. For instance, [30, Table 3] reported that 666 nodes achieve a success probability of 0.0418 on the average. Here, discrete pruning with just 10 tags achieve a success probability 0.0833 on the average. We conclude that in this setting, discrete pruning has similar performances as cylinder pruning.

B.2 Discrete vs Cylinder Pruning: the case of LWE

We compared the cost and success probability between discrete and cylinder pruning when solving LWE via BDD. For each LWE instance, we computed the cost and success probability for discrete pruning by running Alg. 5 and by computing the exact sum (8) with LWE probability (10). We also computed the upper bound of success probability and approximated cost by the methods in [22]. To optimize the bounding function, we used the method described in [11].

Fig. 7 shows the comparison in high and low probability areas. A very precise comparison is difficult: the complexity of discrete pruning is measured by the number of tags, whereas the complexity of cylinder pruning is measured by the number of nodes in the enumeration tree. There is no very accurate equivalence between tags and nodes: one tag costs at most m nodes in the enumeration tree, but it could be less, especially for an optimized implementation. Even if we take into account this factor m, it appears that discrete pruning is a bit faster than cylinder pruning in the low-probability regime, but the difference is limited.

From our experiments on LWE and GGH, we conclude that discrete pruning is at least as fast as cylinder pruning (which was considered to be the fastest method), in the BDD setting. However, the selection of parameters is easier with discrete pruning and can be done online, which should be helpful in blockwise lattice reduction like BKZ. Furthermore, there is room for improvement in terms of implementations, as noted in [10].

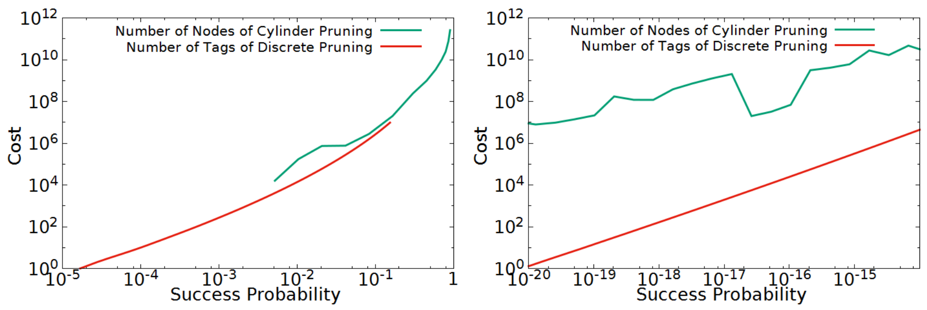

Fig. 7. Comparing costs of discrete and cylinder pruning for high (Left) and low probability (Right). Experiments done with PBKZ-60 on LWE challenges for $(n, \sigma) = (60, 0.005)$ and $m' = 3n = 180$ (Left) and $(80, 0.005)$ and $m' = 3n = 240$ (Right). The graph of cylinder pruning is irregular because it is difficult to find optimal costs.

B.3 Optimal Bases

Fukase and Kashiwabara [20] evaluated the quality of a basis for discrete pruning by the sum $\sum_{i=1}^{n} ||\boldsymbol{b}_i^\star||^2$, which we call the *energy* of the basis. It was reported in [20] that the lower this quantity, the higher the success probability.

However, we show that the energy is insufficient in the worst case to measure the success of discrete pruning. Fig. 8 shows that two bases with similar energy can have a completely different success probability: more precisely, it shows the evolution of three measurements (the energy, the value $C_0 = \mathrm{vol}(\mathcal{C}(\mathbf{0}))/\mathrm{covol}(L)$, and the value C_{1000} defined by the sum of $\mathrm{vol}(\mathcal{C}(\boldsymbol{t}_i))/\mathrm{covol}(L)$ over top 1,000 tags) when we modify the last k vectors (using some "random" k-dimensional unimodular matrix) of a reduced basis of the GGH-200 lattice, and reapply the LLL algorithm. The original basis has measurements $7.99 \cdot 10^6$, 0.00602 and 0.21148 respectively. For $k = 90$, the energy becomes $8.09 \cdot 10^6$, which is about 1% larger than the original value. However, the intersection volumes decrease to zero, which means that discrete pruning is much less likely to succeed.

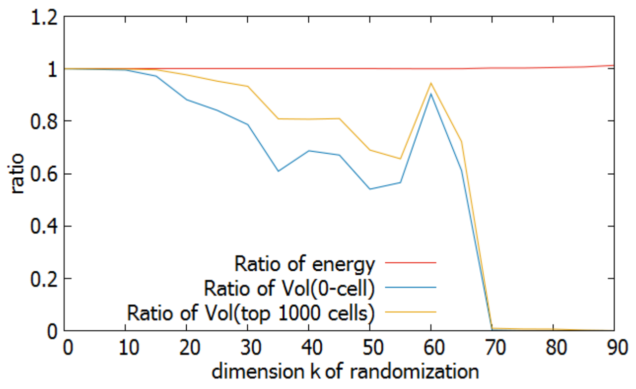

Fig. 8. Evolution of three measurements by randomly modifying the last k vectors.

C Proofs and missing materials of Section 4

C.1 Proof of Theorem 5

Proof (Proof of Theorem 5). For any node $\mathcal{N} \in \mathcal{T}$ which is not a leaf, we want to transform the subtree $\mathcal{N}$+its children into a binary subtree in $\mathcal{T}$. By making queries to the black box, we know the number of children $d(\mathcal{N})$ of $\mathcal{N}$. We also know the i-th child of $\mathcal{N}$ for all $1 \leq i \leq d(\mathcal{N})$. There is thus a bijection $f_{\mathcal{N}}$ between $[|0, \cdots, d(\mathcal{N})-1|]$ and the children of $\mathcal{N}$. We define $l_{\mathcal{N}} = \lceil \log_2(d(\mathcal{N})) \rceil$. For each node of the tree $\mathcal{N}$, we encode the corresponding node in $\mathcal{T}_2$ in the same way.

For those nodes which are in the local binary sub-tree in $\mathcal{T}_2$ corresponding to the local sub-tree $\mathcal{N}$+its children, and which does not correspond to $\mathcal{N}$ or its children, we can encode them as: $E(\mathcal{N})|((x_1 \in \{0,1\}, \cdots, x_i \in \{0,1\}, *, \cdots, *))$ where $E(\mathcal{N})$ is the encoding of the node $\mathcal{N}$, where $i \leq l_{\mathcal{N}}$ and where the number in base 10 corresponding to $(x_1, \cdots, x_i, 0, \cdots, 0)$ in base 2 is smaller than $d(\mathcal{N})-1$. Note that in the representation in base 2 $(x_1, \cdots, x_i, 0, \cdots, 0)$, the heaviest bit is on the left.

Given an encoding of a node $\mathcal{N}_2$ in $\mathcal{T}_2$ which is in the local binary sub-tree in $\mathcal{T}_2$ corresponding to the local sub-tree $\mathcal{N}$+its children in $\mathcal{T}$, we can easily build a black box which gives the children of this node by using the function $f_{\mathcal{N}}$ and the value $d(\mathcal{N})$. We omit the details here. Depending on if the node $\mathcal{N}_2$ corresponds to a node in $\mathcal{T}$ or not, a query on the node $\mathcal{N}_2$ requires a query on $\mathcal{T}$ or not. If $\mathcal{N}_2$ does not correspond to a node in $\mathcal{T}$, we need $O(\log d)$ auxiliary operations on the extra encoding to see if its children correspond to nodes in $\mathcal{T}$. These operations can then be quantized using standard techniques: one first transforms them to reversible maps using standard techniques [15], with potentially additional garbage of size polylogarithmic of the initial memory space.

According to our construction, the leaves of both trees are identical.

We will now prove that: $\#\mathcal{T} \leq \#\mathcal{T}_2 \leq 2\#\mathcal{T}$. The left-hand inequality is obvious. If a node $\mathcal{N}$ of $\mathcal{T}$ is a leaf or has a single child, the subtree of $\mathcal{N}$+its child (in case that it exists) will not change in $\mathcal{T}_2$. If $\mathcal{N}$ has at least two children, the subtree of $\mathcal{N}$+its children will be transformed into a binary subtree in $\mathcal{T}_2$. Assume that $\mathcal{N}$ has $k \geq 2$ children, the corresponding subtree in $\mathcal{T}_2$ has $2k-1$ nodes. It has $2k-2 < 2k$ nodes if we don't count the root corrsponding to $\mathcal{N}$ itself. Thus the k children of $\mathcal{N}$ are transformed into $2k-2$ nodes in $\mathcal{T}_2$ if $k \geq 2$. By combining the previous two cases, we obtain that $\#\mathcal{T}_2 \leq 2\#\mathcal{T}$. □

C.2 FindMin2($\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon$)

Theorem 17. *Let $\varepsilon > 0$. Let $\mathcal{T}$ be a tree with its marked leaves defined by a predicate $\mathcal{P}$ Let g be an integral function defined on the marked leaves such that $g(\mathcal{N}) \leq R$ has at least one solution over all of the marked leaves and an upper-bound d of the number of children of a node in $\mathcal{T}$. Then* **FindMin2**$(\mathcal{T}, \mathcal{P}, g, R, d, T, \varepsilon)$ *outputs a marked leaf $\mathcal{N}$, such that g takes its minimum on $\mathcal{N}$ among all of the marked leaves of $\mathcal{T}$, with probability at least $1-\varepsilon$. It requires*
$O(\sqrt{Tn\log d}\log((\lceil\log_2 R\rceil)/\varepsilon)\lceil\log_2 R\rceil+\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log((\lceil\log_2 R\rceil)/\varepsilon))$
queries on the tree $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations. The algorithm needs $\texttt{poly}(n\log d, \log R)$ qubits.

Proof. The correctness of the algorithm is easy to prove. We will compute the query complexity. There are in total $\mathit{Round} - 1 = \lceil\log_2 R\rceil$ calls on **ExistSolution** and one call on **FindSolution**. According to Theorem 2, for each call of **ExistSolution**, we need $O(\sqrt{Tn\log d}\log(\mathit{Round}/\varepsilon))$ queries on $\mathcal{T}_2$ and on g. According to Theorem 3, the call on **FindSolution** requires $O(\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log(\mathit{Round}/\varepsilon))$ queries on the local structure of the tree $\mathcal{T}_2$ and on g.

In total, the number of queries on $\mathcal{T}_2$ and on g is:

$$\begin{aligned}&O(\sqrt{Tn\log d}\log(\mathit{Round}/\varepsilon)*(\mathit{Round}-1))+\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log(\mathit{Round}/\varepsilon))\\ =\ &O(\sqrt{Tn\log d}\log((\lceil\log_2 R\rceil+1)/\varepsilon)\lceil\log_2 R\rceil+\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log((\lceil\log_2 R\rceil+1)/\varepsilon))\\ =\ &O(\sqrt{Tn\log d}\log((\lceil\log_2 R\rceil)/\varepsilon)\lceil\log_2 R\rceil+\sqrt{T}(n\log d)^{3/2}\log(n\log d)\log((\lceil\log_2 R\rceil)/\varepsilon))\end{aligned}$$

Algorithm 8 Finding a minimum, given an upper-bound of the tree-size

Input: A tree $\mathcal{T}$ with marked leaves defined by the predicate $\mathcal{P}$. An integral function g defined on the marked leaves of $\mathcal{T}$. A parameter R, such that $g(\mathcal{N}) \leq R$ has at least one solution over all of the marked leaves. An upper-bound d of the number of children of a node in $\mathcal{T}$.

Output: A marked leaf $\mathcal{N}$ such that g takes its minimum on $\mathcal{N}$ among all the marked leaves explored by the backtracking algorithm.

```
1: $\mathcal{T}_2 \leftarrow$ the corresponding binary tree of $\mathcal{T}$
2: $N \leftarrow R$, $N' \leftarrow 0$
3: $Round \leftarrow \lceil \log_2 R \rceil + 1$
4: $\mathbf{v} \leftarrow (0, \cdots, 0)$
5: while $N' < N - 1$ do
6:     Call ExistSolution($\mathcal{T}_2, T, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon/Round$)
7:     if ExistSolution($\mathcal{T}_2, T, g_{\lfloor (N+N')/2 \rceil}, n \log d, \varepsilon/Round$) returns "marked node exists" then
8:         $N \leftarrow \lfloor (N+N')/2 \rceil$
9:     else
10:        $N' \leftarrow \lfloor (N+N')/2 \rceil$
11:    end if
12: end while
13: Call FindSolution($\mathcal{T}_2, g_N, n \log d, \varepsilon/Round$)
14: if FindSolution($\mathcal{T}_2, g_N, n \log d, \varepsilon/Round$) returns $\mathbf{x}$ then
15:     $\mathbf{v} \leftarrow \mathbf{x}$
16:     return $\mathbf{v}$
17: else
18:     return
19: end if
```

Using Th. 5, this becomes $O(\sqrt{Tn \log d} \log((\lceil \log_2 R \rceil)/\varepsilon) \lceil \log_2 R \rceil + \sqrt{T}(n \log d)^{3/2} \log(n \log d) \log((\lceil \log_2 R \rceil)/\varepsilon))$ queries on $\mathcal{T}$ and on g. Each query on $\mathcal{T}$ requires $O(\log d)$ auxiliary operations.

Each call of **ExistSolution** and **FindSolution** requires $\texttt{poly}(n \log d)$ qubits. In total the algorithm needs $\texttt{poly}(n \log d, \log R)$ qubits.

D Proofs of Section 5

D.1 Proof of Lemma 2

Proof (Proof of 2). We already know from [10] that $(\mathcal{C}_{\mathbb{Z}}(), T)$ is a L-partition. To show that it is actually universal, it suffices to show that for all $\boldsymbol{u} \in \mathbb{Q}^n$, $(\boldsymbol{u} + \mathcal{C}_{\mathbb{Z}}(t)) \cap L$ is always a singleton, which can be found in polynomial time. To see this, note that Babai's nearest plane algorithm [13] implies that for any $\boldsymbol{t} \in \mathbb{Z}^n$ and any $\boldsymbol{u} \in \mathbb{R}^n$, there is a unique $\boldsymbol{v} \in L$ such that $\boldsymbol{v} - \boldsymbol{u} - \boldsymbol{t}B^\star \in \mathcal{D}$, and that $\boldsymbol{v}$ can be found in polynomial time when $\boldsymbol{u} \in \mathbb{Q}^n$. It follows that $(\boldsymbol{u} + \mathcal{C}_{\mathbb{Z}}(t)) \cap L = \{\boldsymbol{v}\}$. □

Algorithm 9 Universal cell opening for Babai's partition from Babai's Nearest Plane algorithm [13]

```
Input: A tag $\boldsymbol{t} \in \mathbb{Z}^n$, a target $\boldsymbol{u} \in \mathbb{Q}^n$, and a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{Q}^n$ of a lattice $L$, with Gram-Schmidt orthogonalization $B^\star$.
Output: $\boldsymbol{v} \in L$ such that $\{\boldsymbol{v}\} = L \cap (\boldsymbol{u} + \mathcal{C}_{\mathbb{Z}}(\boldsymbol{t}))$
1: $\boldsymbol{v} \leftarrow \boldsymbol{0}$ and $\boldsymbol{w} \leftarrow \boldsymbol{u} + \boldsymbol{t}B^\star$
2: for $i := n$ downto 1 do
3:     Compute the integer $c$ closest to $\langle \boldsymbol{b}_i^\star, \boldsymbol{w} \rangle / \langle \boldsymbol{b}_i^\star, \boldsymbol{b}_i^\star \rangle$
4:     $\boldsymbol{w} \leftarrow \boldsymbol{w} - c\boldsymbol{b}_i$ and $\boldsymbol{v} \leftarrow \boldsymbol{v} + c\boldsymbol{b}_i$
5: end for
6: Return $\boldsymbol{v}$
```

Algorithm 10 Universal cell opening for the natural partition: adaptation of [10, Alg. 3]

```
Input: A tag $\boldsymbol{t} \in \mathbb{N}^n$, a target $\boldsymbol{u} = \sum_{i=1}^n u_i \boldsymbol{b}_i^\star \in \mathbb{Q}^n$, and a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{Q}^n$ of a lattice $L$, with Gram-Schmidt matrix $\mu$.
Output: $\boldsymbol{v} \in L$ such that $\{\boldsymbol{v}\} = L \cap (\boldsymbol{u} + \mathcal{C}_{\mathbb{N}}(\boldsymbol{t}))$
1: for $i := n$ downto 1 do
2:     $y \leftarrow -\sum_{j=i+1}^n v_j \mu_{j,i}$ and $v_i \leftarrow \lfloor u_i + y + 0.5 \rfloor$
3:     if $v_i < u_i + y$ then
4:         $v_i \leftarrow v_i - (-1)^{t_i} \lceil t_i/2 \rceil$
5:     else
6:         $v_i \leftarrow v_i + (-1)^{t_i} \lceil t_i/2 \rceil$
7:     end if
8: end for
9: Return $\sum_{i=1}^n v_i \boldsymbol{b}_i$
```

D.2 Proof of Lemma 3

Proof (Proof of 3). We already know that $(\mathcal{C}_{\mathbb{N}}(), T)$ is an L-partition. Let $\boldsymbol{u} \in \mathbb{Q}^n$: we can write $\boldsymbol{u} = \sum_{i=1}^n u_i \boldsymbol{b}_i^\star$. Then we only need to show that the shifted cell $\boldsymbol{u} + \mathcal{C}_{\mathbb{N}}(\boldsymbol{t}) = \{\sum_{i=1}^n (u_i + x_i)\boldsymbol{b}_i^\star \text{ s.t. } -(t_i+1)/2 < x_i \leq -t_i/2 \text{ or } t_i/2 < x_i \leq (t_i+1)/2\}$ contains only one lattice point which can be found in polynomial time using Alg. 10. Consider the projection π over the orthogonal supplement to the subspace spanned by $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$. Then:

$$\pi(\boldsymbol{u}+\mathcal{C}_{\mathbb{N}}(\boldsymbol{t})) = \{(u_n+x_n)\boldsymbol{b}_n^\star \text{ s.t. } -(t_n+1)/2 < x_n \leq -t_n/2 \text{ or } t_n/2 < x_n \leq (t_n+1)/2\}.$$

Notice that the union $(u_n - (t_n+1)/2, u_n - t_n/2] \cup (u_n + t_n/2, u_n + (t_n+1)/2]$ only contains one integer: this is because the number of integers in an interval of the form $(x, y]$ is $\lfloor y \rfloor - \lfloor x \rfloor$ when $x \geq y$. And that integer can be found in polynomial time, as shown by Alg. 10. This shows that $\pi(\boldsymbol{u} + \mathcal{C}_{\mathbb{N}}(\boldsymbol{t})) \cap \pi(L)$ is a singleton, which can be found in polynomial time. Alg. 10 iterates this process using projections orthogonally to $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i$. □

D.3 Proof of Lemma 5

Proof (Proof of Lemma 5). Each cell is a product of 2^n boxes, so the CDF of the Gaussian distribution gives:

$$\begin{aligned} p(\boldsymbol{t}) &= 2^n \prod_{i=1}^{n} \frac{1}{2} \left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i + 1}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) \right) \\ &= \prod_{i=1}^{n} \left(\operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i + 1}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) - \operatorname{erf}\left(\frac{1}{\sqrt{2}\sigma} \cdot \frac{t_i}{2} \cdot \|\boldsymbol{b}_i^\star\| \right) \right) \end{aligned}$$

□

D.4 Proof of Lemma 7

Proof (Proof of Lemma 7). It suffices to decompose the noise $\boldsymbol{e}$ as $\boldsymbol{e} = \|\boldsymbol{e}\| \times \frac{\boldsymbol{e}}{\|\boldsymbol{e}\|}$. The variable $\|\boldsymbol{e}\|$ only takes finitely many values, and because of Heuristic 10,

$$\Pr_{\boldsymbol{e}} \left(-r \frac{\boldsymbol{e}}{\|\boldsymbol{e}\|} \in \mathcal{C}(\boldsymbol{t}) \right) = \Pr_{\boldsymbol{x} \leftarrow S_n} (\boldsymbol{x} \in \mathcal{C}(\boldsymbol{t})/r).$$

□

E Proofs of Th. 15 in Section 7

In order to prove the theorem, we will need the two following lemmas:

Lemma 10. *For all $R \in \mathbb{N}$, we have $\frac{\#\mathcal{T}_2(R)-2}{2(2n-1)} \leq \#S(\mathcal{T}_2(R)) \leq \#\mathcal{T}_2(R)$. We also have $\#\mathcal{T}_2(R+1) \leq 2n\#\mathcal{T}_2(R)$.*

Proof (Proof of Lemma 10). Under the transformation, the number of tags that we find in the tree with the parameter R won't change, *i.e.* $\#S(\mathcal{T}(R)) = \#S(\mathcal{T}_2(R))$.

Since we have: $\frac{\#\mathcal{T}(R)-1}{2n-1} \leq \#S(\mathcal{T}(R)) \leq \#\mathcal{T}(R)$ and we also know: $\#\mathcal{T}(R) \leq \#\mathcal{T}_2(R) \leq 2\#\mathcal{T}(R)$

We thus have $\frac{\#\mathcal{T}_2(R)-2}{2(2n-1)} \leq \#S(\mathcal{T}_2(R)) \leq \#\mathcal{T}_2(R)$

Now we will prove the second inequality. If there exists $(*, \cdots, *, v_k, \cdots, v_n) \in \mathcal{T}(R)$ where $v_k \neq 0$ such that $\sum_{j=k}^{n} f(j, v_j) = R + 1$, then $(*, \cdots, *, v_k + 1, \cdots, v_n) \in \mathcal{T}(R+1) \backslash \mathcal{T}(R)$. And for all $i \in [|1, k-1|]$, $(*, \cdots, *, v_i \in \{0, 1\}, 0, \cdots, 0, v_k, \cdots, v_n) \in \mathcal{T}(R+1) \backslash \mathcal{T}(R)$.

$(*, \cdots, *, v_k + 1, \cdots, v_n)$ generates two nodes in $\mathcal{T}_2(R+1) \backslash \mathcal{T}_2(R)$.

Each $(*, \cdots, *, v_i \in \{0, 1\}, 0, \cdots, 0, v_k, \cdots, v_n)$ generates one node in $\mathcal{T}_2(R+1) \backslash \mathcal{T}_2(R)$.

On the other hand, a node in $\mathcal{T}_2(R+1)\backslash\mathcal{T}_2(R)$ can only be derived from a node $(*, \cdots, *, v_k, \cdots, v_n) \in \mathcal{T}(R)$ (and thus from the equivalent node in $\mathcal{T}_2(R)$) such that $\sum_{j=k}^{n} f(j, v_j) = R+1$ and $v_k \neq 0$, by using the above processus.

Therefore, $\#\mathcal{T}_2(R+1) \leq 2n\#\mathcal{T}_2(R)$. $\square$

Lemma 11. *d can be upper-bounded by $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$.*

Proof (Proof of Lemma 11). At the beginning,

$$\begin{aligned}R = \sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil) &= (\sum_{i=1}^{n}\alpha_i)\lceil(4nM)^{1/n}\rceil)(\lceil(4nM)^{1/n}\rceil) + 1)\\ &< \sum_{i=1}^{n}\alpha_i\lceil(4nM)^{1/n}\rceil)(\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)) + 1)\\ &= f(1, \frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)) = g(\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)), 0, \cdots, 0)\end{aligned}$$

Since $\alpha_1 \leq \cdots \leq \alpha_n$, we also have: for all $1 \leq j \leq n$,
$R < g(0, \cdots, 0, \frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil), 0, \cdots, 0)$ where $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$ is on the j^{th} position.

Since R decreases during the execution of the algorithm, d can be upper-bounded by $\frac{\sum_{i=1}^{n}\alpha_i}{\alpha_1}\lceil(4nM)^{1/n}\rceil)$. $\square$

Proof (Proof of Th. 15).

We will prove that the output R satisfies $2(2n-1)M \leq \#\mathcal{T}_2(R) \leq 32n^2M$ with probability at least $1-\varepsilon/2$. Since we have $\frac{\#\mathcal{T}(R)-2}{2(2n-1)} \leq \#S(\mathcal{T}_2(R)) \leq \#\mathcal{T}_2(R)$, this proves that $M \leq \#S(\mathcal{T}(R)) \leq 32n^2M$ with probability at least $1-\varepsilon/2$.

Since the algorithm will end after at most *Round* $= \lceil\log_2(\sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil))\rceil$ calls of the tree size estimation algorithm with correctness probability at least
$1-\varepsilon/2(\lceil\log_2(\sum_{i=1}^{n} f(i, \lceil(4nM)^{1/n}\rceil))\rceil)$, by using the union bound, the correctness probability of our algorithm is at least $1-\varepsilon/2$. Now we can assume that all the answers of the tree size estimation algorithm are correct.

In the following we will omit the last four parameters in **TreeSizeEstimation** for the clarity of the proof.

In case that the algorithm returns R inside the **while** loop, the output of the first **TreeSizeEstimation**($\mathcal{T}_2(R)$) is "$\mathcal{T}_2(R)$ contains $\hat{T}$ vertices" with
$3(2n-1)M \leq \hat{T} \leq 16n^2M$. Since the tree size estimation is up to precision $1 \pm 1/2$, the real tree size should be in the interval $[\frac{3(2n-1)N}{1+1/2}, \frac{16n^2M}{1-1/2}] \subset [2(2n-1)M, 32n^2M]$.

In case that the algorithm returns R after the **while** loop, we have $R_1 = R_0 + 1$. The estimation of **TreeSizeEstimation**($\mathcal{T}(R_1)$) with the parameters as in the while loop is "$\mathcal{T}(R_1)$ contains more than $16n^2M$ vertices". Since the precision parameter is $1/2$, we have $\#\mathcal{T}_2(R_1) \geq \frac{16n^2M}{1+1/2} > 8n^2M$.

The estimation of **TreeSizeEstimation**($\mathcal{T}(R_0)$) with the parameters as in the while loop is "$\mathcal{T}(R_0)$ contains $\hat{T}$ vertices" with $\hat{T} < \lceil 3(2n-1)M \rceil$. Since the precision parameter is 1/2, we have $\#\mathcal{T}_2(R_0) \leq \frac{3(2n-1)M}{1-1/2} = 6(2n-1)M$.

By using Lemma 10, we know that for all $R \in \mathbb{N}$, $\#\mathcal{T}(R+1) \leq 2n\#\mathcal{T}(R)$.Thus, there exists $R > 0$ such that $2(2n-1)M \leq \#\mathcal{T}_2(R) \leq 4n(2n-1)M < 8n^2M$. This R should be R_0.

We proved that in each case, Alg. 6 outputs R such that $2(2n-1)M \leq \#\mathcal{T}(R) \leq 32n^2M$. Therefore, R satisfies $M \leq \#S(\mathcal{T}(R)) \leq 32n^2M$ with probability at least $1-\varepsilon/2$.

The number of queries to the trees is:

$$O(\sqrt{n \log d 16n^2M} \log^2(Round/\varepsilon)*Round) = O(n^2\sqrt{M}\texttt{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta)).$$

Since each query needs $O(\log(16n^2M)) = \texttt{poly}(\log(n), \log(M))$ non-query transformations, the time complexity of Alg. 6 is $O(n^2\sqrt{M}\texttt{poly}(\log(n), \log(M), \log(1/\varepsilon), \beta))$.

The algorithm needs $O(\texttt{poly}(n, \log(M), \log(1/\varepsilon)))$ qubits by using Th. 4.