# Dual lattice attacks for closest vector problems (with preprocessing)

Thijs Laarhoven[1⋆] and Michael Walter[2⋆]

[1] Eindhoven University of Technology, The Netherlands
mail@thijs.com
[2] Institute of Science and Technology Austria, Austria
michael.walter@ist.ac.at

**Abstract.** The dual attack has long been considered a relevant attack on lattice-based cryptographic schemes relying on the hardness of learning with errors (LWE) and its structured variants. As solving LWE corresponds to finding a nearest point on a lattice, one may naturally wonder how efficient this dual approach is for solving more general closest vector problems, such as the classical closest vector problem (CVP), the variants bounded distance decoding (BDD) and approximate CVP, and preprocessing versions of these problems. While primal, sieving-based solutions to these problems (with preprocessing) were recently studied in a series of works on approximate Voronoi cells [Laa16b,DLdW19, Laa20,DLvW20], for the dual attack no such overview exists, especially for problems with preprocessing. With one of the take-away messages of the approximate Voronoi cell line of work being that primal attacks work well for approximate CVP(P) but scale poorly for BDD(P), one may wonder if the dual attack suffers the same drawbacks, or if it is a better method for solving BDD(P).

In this work we provide an overview of cost estimates for dual algorithms for solving these "classical" closest lattice vector problems. Heuristically we expect to solve the search version of average-case CVPP in time and space $2^{0.293d+o(d)}$. For the distinguishing version of average-case CVPP, where we wish to distinguish between random targets and targets planted at distance approximately the Gaussian heuristic from the lattice, we obtain the same complexity in the single-target model, and we obtain query time and space complexities of $2^{0.195d+o(d)}$ in the multi-target setting, where we are given a large number of targets from either target distribution. This suggests an inequivalence between distinguishing and searching, as we do not expect a similar improvement in the multi-target setting to hold for search-CVPP. We analyze three slightly different decoders, both for distinguishing and searching, and experimentally obtain concrete cost estimates for the dual attack in dimensions 50 to 80, which confirm our heuristic assumptions, and show that the hidden order terms in the asymptotic estimates are quite small.

Our main take-away message is that the dual attack appears to mirror the approximate Voronoi cell line of work – whereas using approximate Voronoi cells works well for approximate CVP(P) but scales poorly for BDD(P), the dual approach scales well for BDD(P) instances but performs poorly on approximate CVP(P).

**Keywords:** lattice-based cryptography, lattice algorithms, primal/dual attacks, closest vector problem (CVP), bounded distance decoding (BDD)

## 1 Introduction

*Post-quantum cryptography.* Ever since the breakthrough work of Shor in the 1990s [Sho94], revealing how quantum computers pose a major threat to currently deployed cryptographic primitives, researchers have been studying alternative approaches which have the potential to be resistant against quantum attacks. Over the past few decades, the field of "post-quantum cryptography" [BBD09] has gained in popularity, and the recent NIST standardization process [oSN17] has further focused the attention of the cryptographic community on preparing for a future where large-scale quantum computers are a reality.

⋆ Thijs Laarhoven is supported by an NWO Veni grant (016.Veni.192.005). Michael Walter is supported by the European Research Council, ERC consolidator grant (682815 – TOCNeT). Part of this work was done while both authors were visiting the Simons Institute for the Theory of Computing at the University of California, Berkeley, for the Spring 2020 semester on "Lattices: Algorithms, Complexity, and Cryptography".

*Lattice-based cryptography.* Out of all proposed alternatives for "classical" cryptography, lattice-based cryptography has emerged as a prime candidate for secure and efficient cryptography in the post-quantum era. Many basic primitives are simple and efficient to realize with e.g. learning with errors (LWE) and its ring variants [Reg05, SSTX09, Reg10], while more advanced cryptographic primitives (such as fully homomorphic encryption [Gen09]) can also be constructed using lattices. By far the most schemes submitted to the NIST competition base their security on the hardness of hard lattice problems.

*Closest vector problems.* Various hard lattice problems have been considered over time, with the two classical hard problems being the shortest vector problem (SVP) and the closest vector problem (CVP). The latter problem asks to find a nearest lattice point to an arbitrary target, and is arguably the hardest. CVP algorithms appear in various cryptographic contexts, both directly and indirectly (as a subroutine within another algorithm). Closely related to CVP are easier variants, such as bounded distance decoding (BDD) and approximate CVP, preprocessing versions of these problems, and modern related variants such as learning with errors (LWE) and structured variants.

*Primal attacks.* For solving CVP and its variants, various approaches have been studied to date. Lattice enumeration [Kan83, MW15, AN17] was long considered the most practical, with a low memory requirement and fast heuristics. Babai's polynomial time algorithms for the easiest CVP variants [Bab86] can be viewed as based on enumeration as well. Lattice sieving methods [AKS01, MV10, BDGL16, DLdW19, Laa20, DLvW20] make use of so-called approximate Voronoi cells, and achieve a superior asymptotic scaling of the time complexity. With recent advances in sieving [BDGL16, ADH$^+$19, DSvW21, svp20] this likely is the method of choice when assessing the hardness of these problems in cryptographically relevant dimensions. The line of work on approximate Voronoi cells showed that while results for preprocessing problems are promising both for average-case CVPP and approximate CVPP, the results were somewhat disappointing for BDDP. An open question was raised whether this is inherent to the primal approach, and if other approaches were more suitable for BDDP.

*Dual attacks.* An alternative approach for solving hard lattice problems (and in particular closest vector problems) is based on the dual lattice [AR04]. Using short vectors from the dual lattice, and computing dot products with a target vector, one can obtain probabilistic evidence indicating whether the target vector lies close to the lattice. Using many dual vectors, one can then construct both distinguishers and search algorithms using a gradient ascent approach [LLM06, DRS14]. The dual attack has mostly been considered in the context of LWE and BDD [LP11, APS15, HKM18, ADH$^+$19], which suggests the dual approach may work better for the BDD regime and worse for the approximate CVP regime. To this date several open questions remain however, such as a thorough heuristic overview of the costs of the dual attack for classical lattice problems (i.e. not LWE), as well as an experimental assessment of the practicality of the dual attack for solving such closest vector problems. Moreover, the preprocessing setting has not yet been studied from a heuristic point of view, in the way that the approximate Voronoi cell approach has been studied recently.

### 1.1 Contributions

*Revisiting the dual attack.* In this work we give a thorough, practical overview of the strengths and weaknesses of the dual attack for solving most closest vector problems. After covering preliminaries (Section 2), we describe a generalized model which covers most variants of CVP(P) (Section 3). We then study different algorithms both for distinguishing and searching for nearby lattice points (Section 4). We provide a heuristic complexity analysis of these different algorithms, both with and without preprocessing (Section 5), and we finally verify the heuristics and obtain more concrete cost estimates through experiments (Section 6).

*Decoders.* Concretely, in terms of algorithms we provide three different *decoders* for combining dot products of different dual vectors with the target vector. The Aharonov–Regev decoder was presented in [AR04] and finds its motivation in approximating Gaussians over the lattice via the Fourier transform. Our novel Neyman–Pearson decoder follows from applying the celebrated Neyman–Pearson lemma [NP33] to the considered problem, and results in a slightly different, but asymptotically equivalent decoder to the AR decoder. The third (new) decoder is a simpler alternative to the previous two decoders, which conveniently requires no knowledge of the estimated distance from the lattice, has the same asymptotic performance as the other two decoders, and also appears to perform well in practice.

*Algorithms.* Besides using these decoders for distinguishing, we present a heuristic search algorithm based on the classical gradient ascent approach, previously studied from a theoretical point of view in e.g. [LLM06, DRS14]. We implemented this search algorithm (as well as distinguishers) in dimensions 50 to 80, showing that the actual performance closely matches our predictions, with fast convergence for most BDD instances.

*Asymptotics.* On the theoretical side, we show optimality of the Neyman–Pearson decoder within our model, thereby showing optimality of the associated heuristics using the proposed decoders. For preprocessing problems, we can solve BDDP with radius $r \cdot g_d$ (with $g_d$ the Gaussian heuristic) in time $e^{dr^2(1+o(1))/e^2}$ for small $r$. For $r$ close to 1, distinguishing many targets as either all being BDD samples or all being random vectors can be done in time $e^{d(1+o(1))/e^2} \approx 2^{0.195d+o(d)}$. Actually finding a nearest vector, or distinguishing based on only one target vector requires time $2^{0.293d+o(d)}$ when $r \approx 1$. The results for the preprocessing setting are shown in Figures 1a–1b. For the setting where preprocessing is not free, taking into account basis reduction costs leads to Figures 1c–1d. These sketch the same picture as the preprocessing results: the dual attack seems complementary to the primal attack, with the dual attack scaling well for $r \leq 1$ and the primal attack scaling well for $r \geq 1$.

*Experiments.* To validate the heuristic, theoretical claims, as well as to get an idea how well the dual attack really works for solving these problems in practice, we implemented and tested our algorithms as well, focusing on the BDD(P) regime with radius slightly below the Gaussian heuristic. The performance of the distinguishers and search algorithms is remarkably close to our theoretical predictions, even when the preprocessing simply consists of a lattice sieve.

*Take-away message.* Summarizing, in practice the dual attack seems to perform as good as can be expected from the theoretical estimates. Whereas a primal approach works better for approximate CVP(P), the dual attack is arguably the right solution for solving BDD(P). With the approximate Voronoi cell approach and the dual attack working well in disjoint regimes, one could say the dual attack is a complementary solution to the primal attack.

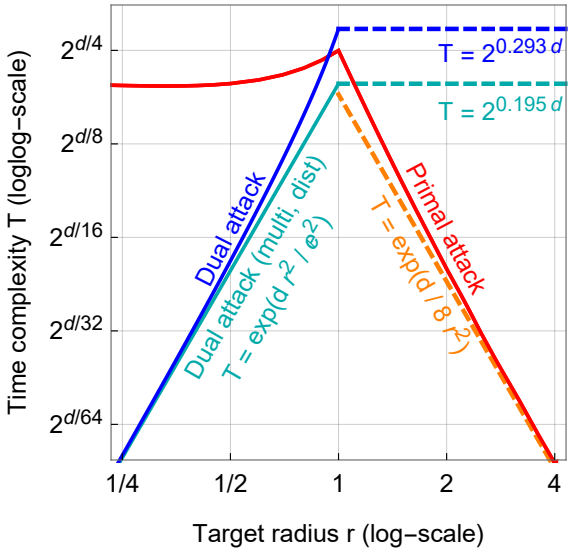
### 1.2 Open problems

Recently, algorithms for CVP(P) have found further applications, besides for studying the hardness of lattice problems [BKV19, PMHS19]. One application that may be of interest is in a hybrid with lattice enumeration [DLdW20]. Within the enumeration tree, various CVP instances appear on the same lattice, one of which leads to the solution. Rather than solving CVP for each of these targets, a distinguisher may be sufficient for discarding the majority of targets, and depending upon the depth in the tree we may in fact be faced with BDD instances, for which the dual attack may work better than the primal attack used in [DLdW20]. We leave a study of this hybrid for future work.

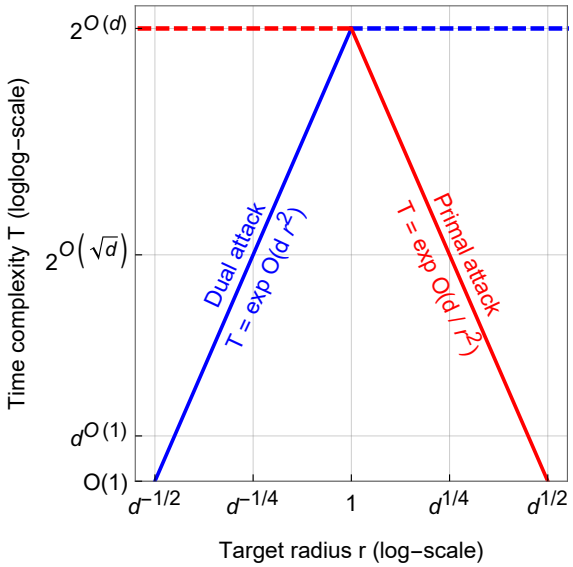
Throughout the paper some other open problems arose. Can the multi-target distinguisher be turned into a search algorithm with the same complexity $2^{0.195d+o(d)}$ for "average-case" search CVPP? Which of the presented decoders is best in practice? How should one choose the step size for the gradient ascent method, and how does the step size influence the number of steps and the probability of converging to the correct solution? On the implementation-side, since the dual attack scales particularly well across many cores running in parallel, how does the dual attack compare to primal attacks on large-scale experiments? And finally, are extrapolations based on our preliminary experimental results in moderate dimensions representative for the true complexities in high dimensions?

### Acknowledgments

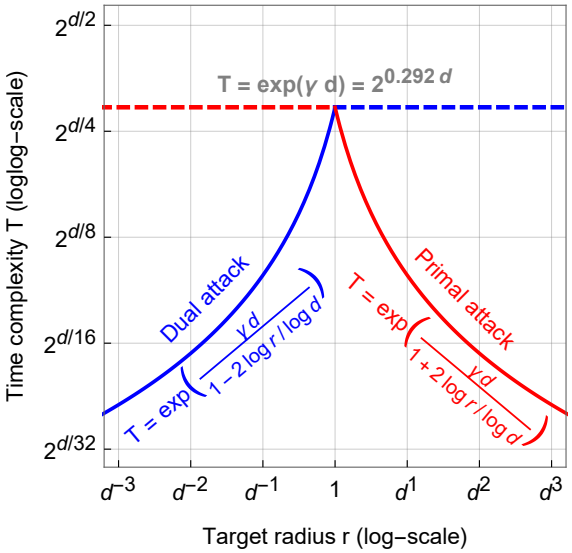
The authors thank Sauvik Bhattacharya, Léo Ducas, Rachel Player, and Christine van Vredendaal for early discussions on this topic and on preliminary results. The authors further thank the reviewers of CT-RSA 2021 for their valuable feedback.



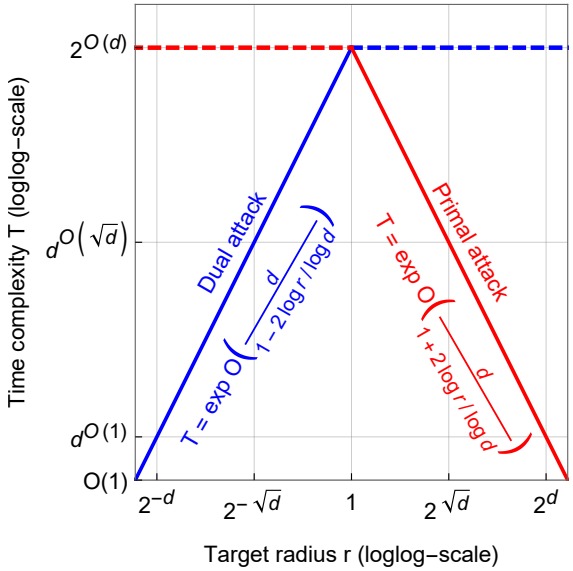

(a) Complexities for problems with preprocessing for small radii $r = O(1)$.




(b) Complexities for problems with preprocessing, for medium radii $r = d^{\pm O(1)}$.




(c) Complexities for problems without preprocessing for medium radii $r = d^{\pm O(1)}$.




(d) Complexities for problems without preprocessing, for large radii $r = 2^{d^{\pm O(1)}}$.

**Fig. 1.** Asymptotic complexities for primal and dual attacks, with (a,b) and without preprocessing (c,d), in the regimes of exponential (a,c) and arbitrary time complexity scalings (b,d). The radius $r$ denotes the multiplicative factor in front of the Gaussian heuristic for the distance from the target to the planted nearby lattice point. The dual attack (blue) represents distinguishing and searching costs for one target. The primal attack (red) covers approaches based on approximate Voronoi cells. When distinguishing a large number of targets, and using only one dual vector, we obtain slightly improved results for the preprocessing regime (cyan). The dual attack works well for BDD(P) (the regime $r \leq 1$), while the primal attack works well for approximate CVP(P) (the regime $r \geq 1$). The dual attack for BDDP becomes polynomial-time for radius $r \cdot g_d$ when $r = O(\sqrt{\log d / d})$. The primal attack for approximate CVPP becomes polynomial-time when $r = O(\sqrt{d / \log d})$.

## 2 Preliminaries

*Notation.* Reals and integers are denoted by lower case letters and distributions by upper case letter. Bold letters are reserved for vectors (lower case) and matrices (upper case). The $i$-th entry of a vector $\boldsymbol{v}$ is denoted by $v_i$ and the $j$-th column vector of a matrix $\mathbf{B}$ by $\boldsymbol{b}_j$. We denote the $d$-dimensional ball of radius $r$ by $B_d(r)$ and the unit-sphere by $\mathcal{S}^{d-1}$, which are always centered at $\mathbf{0}$ unless stated otherwise. We may omit the dimension $d$ if clear from context.

### 2.1 Lattices

*Basics.* Given a set $\mathbf{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m\} \subset \mathbb{R}^d$ of linearly independent basis vectors (which can equivalently be represented as a matrix $\mathbf{B} \in \mathbb{R}^{d \times m}$ with the vectors $\boldsymbol{b}_i$ as columns), the lattice generated by $\mathbf{B}$ is defined as $\mathcal{L} = \mathcal{L}(\mathbf{B}) := \{\mathbf{B}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{Z}^d\}$. In case $m = d$ we say the lattice is full-rank, and unless otherwise stated, throughout the paper we will implicitly assume $m = d$. We write $\mathrm{Vol}(\mathcal{L}) := \det(\mathbf{B}^T\mathbf{B})^{1/2}$ for the volume of a lattice $\mathcal{L}$, and w.l.o.g. throughout the paper we will assume that $\mathcal{L}$ is normalized to have volume one (i.e. by multiplying $\mathbf{B}$ by the appropriate scalar multiple). Given a basis $\mathbf{B}$, we write $\mathbf{B}^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_m^*\}$ for its Gram–Schmidt orthogonalization. We write $D_{\boldsymbol{t}+\mathcal{L},s}$ for the discrete Gaussian distribution on $\boldsymbol{t}+\mathcal{L}$ with probability mass function satisfying $\Pr[\boldsymbol{X} = \boldsymbol{x}] \propto \rho_s(\boldsymbol{x}) := \exp(-\pi\|\boldsymbol{x}\|^2/s^2)$, normalized such that $\sum_{\boldsymbol{x}\in\boldsymbol{t}+\mathcal{L}} \Pr[\boldsymbol{X} = \boldsymbol{x}] = 1$. We define $\lambda_1(\mathcal{L}) := \min_{\boldsymbol{v}\in\mathcal{L}\setminus\{\mathbf{0}\}} \|\boldsymbol{v}\|$ and for $\boldsymbol{t} \in \mathbb{R}^d$ we define $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) := \min_{\boldsymbol{v}\in\mathcal{L}} \|\boldsymbol{t} - \boldsymbol{v}\|$, where all norms are Euclidean norms.

*Dual lattices.* Given a lattice $\mathcal{L}$, its dual lattice $\mathcal{L}^*$ contains all vectors $\boldsymbol{w} \in \mathbb{R}^d$ such that $\langle \boldsymbol{v}, \boldsymbol{w}\rangle \in \mathbb{Z}$ for all primal lattice vectors $\boldsymbol{v} \in \mathcal{L}$. This set of vectors again forms a lattice, and for full-rank primal lattices $\mathcal{L}$ a basis of this dual lattice is given by $\mathbf{B}^{-T}$. Since $\mathrm{Vol}(\mathcal{L}^*) = 1/\mathrm{Vol}(\mathcal{L})$, if $\mathcal{L}$ is normalized to have volume 1, then also $\mathcal{L}^*$ has volume 1. We commonly denote primal lattice vectors by $\boldsymbol{v}$, and dual vectors by $\boldsymbol{w}$.

*Lattice problems.* Given a description of a lattice $\mathcal{L}$, the shortest vector problem (SVP) asks to find a shortest non-zero lattice vector $\boldsymbol{s} \in \mathcal{L}$, satisfying $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L})$. For approximate versions of this problem ($\mathrm{SVP}_r$ with $r \geq 1$), returning any non-zero lattice vector $\boldsymbol{v} \in \mathcal{L}$ of norm $\|\boldsymbol{v}\| \leq r \cdot \lambda_1(\mathcal{L})$ suffices as a solution. A different relaxation of SVP is unique SVP ($\mathrm{uSVP}_r$ with $r \leq 1$), where one is tasked to find the shortest non-zero vector in a lattice with the guarantee that there is one particularly short vector $\boldsymbol{s}$ in the lattice: $\|\boldsymbol{s}\| \leq r \cdot \min_{\boldsymbol{v}\in\mathcal{L},\boldsymbol{v}\neq\lambda\cdot\boldsymbol{s}} \|\boldsymbol{v}\|$. Given a description of a lattice $\mathcal{L}$ and a target vector $\boldsymbol{t} \in \mathbb{R}^d$, the closest vector problem (CVP) is to find a lattice vector $\boldsymbol{s} \in \mathcal{L}$ satisfying $\|\boldsymbol{t} - \boldsymbol{s}\| = \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$. Similarly, we may define approximate CVP ($\mathrm{CVP}_r$ with $r \geq 1$) as the approximate version of CVP, where any vector $\boldsymbol{s} \in \mathcal{L}$ with $\|\boldsymbol{t} - \boldsymbol{s}\| \leq \gamma \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$ qualifies as a solution. The analogue of uSVP for the inhomogeneous setting is often called bounded distance decoding ($\mathrm{BDD}_r$ with $r \leq 1$), where one is tasked to find the closest vector with the guarantee that it lies within radius $r \cdot \lambda_1(\mathcal{L})$ of the lattice.

### 2.2 Heuristic assumptions

Before describing and analyzing dual lattice attacks, let us first describe the heuristic assumptions we will use to model the problems. Under this heuristic model we will be able to obtain sharp bounds on the costs of dual attacks, at the cost of no longer having a proof of correctness – for exotic, non-random lattices these bounds may well be too optimistic, and even for average-case lattices we can only "prove" the resulting complexities under these additional assumptions.

*The Gaussian heuristic.* The determinant $\det(\mathcal{L}) = \mathrm{Vol}(\mathcal{L})$ describes the volume of the fundamental domain, as well as the density of lattice points in space. This metric indicates that if we have a random, large region $\mathcal{R} \subset \mathbb{R}^d$ of volume $\mathrm{Vol}(\mathcal{R})$, then the number of lattice points contained in $\mathcal{R}$ can be estimated as $\mathrm{Vol}(\mathcal{R})/\mathrm{Vol}(\mathcal{L})$. Since a ball of radius $r$ has volume $\mathrm{Vol}(B(r)) = r^d V_d(1)$, with $V_d(1) = (2\pi e/d)^{d/2+o(d)}$ the volume of the ball with unit radius, we expect for any constant $\varepsilon > 0$ with overwhelming probability the ball of radius $r$ around a point in space to contain no lattice points if $r < (1-\varepsilon)g_d$, and to contain many lattice points if $r > (1+\varepsilon)g_d$. Here, $g_d = \sqrt{d/(2\pi e)} \cdot \left(1 + O\left(\frac{1}{d}\right)\right)$ denotes the Gaussian heuristic in dimension $d$. Putting such a ball around the origin, we expect a shortest non-zero vector in the lattice to have norm $\lambda_1(\mathcal{L}) = g_d(1 + o(1))$, and similarly for random target vectors $\boldsymbol{t} \in \mathbb{R}^d$ we expect the closest vector in the lattice to $\boldsymbol{t}$ to lie at distance $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) = g_d(1 + o(1))$.

*Distributions of lattice points.* Building upon the Gaussian heuristic, we further estimate the distribution of lattice points in space as follows: there is one lattice point $\mathbf{0}$ of norm 0, and for any $\alpha \geq 1$ we expect there to be $\alpha^{d+o(d)}$ lattice vectors of norm $\alpha g_d(1+o(1))$. The Euclidean norm of the $n$-th shortest vector in a lattice of volume 1 can therefore be estimated as $n^{1/d} g_d(1+o(1))$.

*The geometric series assumption.* In the context of lattice basis reduction, we rely on the geometric series assumption [SE94], which states that the norms of the Gram–Schmidt vectors of a block reduced basis form a geometric sequence, i.e. $\|\boldsymbol{b}_i^*\| = \delta^{d-2i+1}$. Here, $\delta > 1$ indicates the quality of the basis: the smaller $\delta$, the better the quality of the basis, and the harder these bases generally are to obtain. For block reduced bases with block size $k$, we obtain the commonly known estimate $\delta = \delta_k = g_k^{1/(k-1)} = 1 + O(\log k/k)$ (see e.g. [APS15, MW16]). This estimate is mostly accurate when $k \ll d$, as for $k \approx d$ the shape of the basis is expected to be an average-case HKZ-shape for which the Gram–Schmidt norms do not exactly follow a geometric sequence anymore.

### 2.3 Lattice algorithms and cost models

*The cost of SVP.* To solve SVP, several heuristic methods are known, and most belong to either the class of enumeration algorithms (running in super-exponential time and polynomial space) [Kan83, MW15, AN17] or sieving algorithms (running in $2^{O(d)}$ time and space) [AKS01, Laa16a, BDGL16]. As in our analysis we are interested in minimizing the asymptotic time complexity, we will model the cost of solving SVP by the cost of the best heuristic sieve in dimension $d$ [BDGL16], which classically runs in time $\mathrm{T}_{\mathrm{SVP}}(d) = (3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$. Apart from solving SVP, sieving algorithms are actually expected to return all the $(4/3)^{d/2} \approx 2^{0.208d+o(d)}$ shortest non-zero vectors in the lattice; compared to the best sieving algorithms for finding only one short vector, the overhead in the time complexity is only a factor $2^{o(d)}$ [Duc18]. To find even more short lattice vectors, one may use a *relaxed sieve* [Laa16b, Algorithm 5].

*The cost of approximate SVP.* To approximate the shortest vector in a lattice and obtain a basis following the GSA, one commonly uses block reduction with the block size $k$ determining the parameter $\delta = \delta_k$ [Sch87, SE94, GN08, MW16]. The cost of block reduction is usually modeled as the cost of solving SVP in dimension $k$, via $\mathrm{T}_{\mathrm{BKZ}}(k) = d^{O(1)} \cdot \mathrm{T}_{\mathrm{SVP}}(k)$, with some overhead which is polynomial in $d$. Together with the geometric series assumption, this describes a trade-off between the time complexity for the basis reduction and the quality of the reduced basis.

### 2.4 The cost of BDD(P)

In this section we describe the state of the art in terms of heuristic asymptotic time complexities for solving BDD and BDDP.

**Solving BDD.** To the best of our knowledge, the most efficient way to solve BDD is through Kannan's embedding [Kan87]. In the following we briefly sketch the analysis. (We do not claim novelty; this is just for convenience of the reader.)

Let $\mathbf{B}$ be a lattice basis (we assume w.l.o.g. that $\mathrm{Vol}(\mathcal{L}(\mathbf{B})) = 1$) and $\boldsymbol{t} = \boldsymbol{v} + \boldsymbol{e}$ be a target vector, where $\boldsymbol{v}$ is a lattice vector and $\|\boldsymbol{e}\| \leq \alpha g_d$. The embedding attack constructs the new basis

$$\begin{pmatrix} \mathbf{B} & \boldsymbol{t} \\ \mathbf{0} & c \end{pmatrix}$$

for some constant $c$. It is easy to see that the new lattice contains the vector $[\boldsymbol{e} \mid c]$ and has determinant $c$. One can now optimize over $c$ to minimize the ratio $\|[\boldsymbol{e} \mid c]\|/c^{\frac{1}{d+1}}$, which shows that the optimum is at $c = \|\boldsymbol{e}\|/\sqrt{d}$. It can be verified that under the Gaussian heuristic, a BDD instance with $\|\boldsymbol{e}\| = \alpha g_d$ leads to a uSVP instance with a gap of $(2\pi e \alpha^d)^{-\frac{1}{d+1}} \approx 1/\alpha$ between the first and second minimum. We can now run our favorite SVP approximation algorithm to approximate the shortest vector in this lattice up to a factor $1/\alpha$, which heuristically yields the solution. The algorithm of choice is usually lattice reduction, where the SVP oracle is instantiated with sieving. We note that it was suggested in [ADPS16], and verified

in [AGVW17], that lattice reduction actually behaves somewhat better in finding the solution than suggested by the approximation factor, but this does not impact the asymptotic running time of the approach so we focus on the above estimate for simplicity.

**Lemma 1 (Approximation factor of BKZ).** *Let $\alpha = d^a$ for some $a > 0$. Under suitable heuristics, lattice block reduction with sieving achieves an approximation factor $\alpha$ in time $2^{\frac{0.292}{1+2a}d+o(d)}$.*

*Proof.* Recall that lattice reduction with block size $k$ approximates the shortest vector by a factor about $g_k^{d/k}/g_d$. Substituting $k = \beta d$ for some $\beta \in (0,1)$ and noticing that $g_{\beta d} = \sqrt{\beta} g_d$ this yields

$$\frac{g_k^{d/k}}{g_d} = \beta^{\frac{1}{2\beta}} g_d^{\frac{1-\beta}{\beta}} = d^{\frac{1-\beta}{2\beta}+o(1)} \tag{1}$$

as long as $\beta$ is at least a constant. (If $\beta = o(1)$, the algorithm runs in sub-exponential time and the corresponding approximation factor is too large to be relevant for our work). For block reduction to find a short enough solution we need $\beta$ to be such that $d^{\frac{1-\beta}{2\beta}+o(1)} \leq \alpha$. In other words, we need $\beta \geq (1+2a-o(1))^{-1}$. Recall that block reduction in dimension $k$ runs in time

$$2^{0.292k+o(k)} = 2^{0.292\beta d+o(d)} = 2^{\frac{0.292}{1+2a}d+o(d)}.$$

□

**Corollary 1 (BDD via Kannan embedding).** *Let $r = d^a$ be a radius for some $a < \frac{1}{2}$. Solving BDD with radius $r$ using Kannan's embedding and lattice reduction with sieving has complexity $2^{\frac{0.292}{2(1-a)}d+o(d)}$ under suitable heuristics.*

*Proof.* Following the above outline of Kannan's approach, we require an SVP algorithm with approximation factor $\alpha = d^{\frac{1}{2}-a+o(1)}$. Lemma 1 shows that block reduction is such an algorithm with the claimed running time. □

**Solving BDDP.** The situation is less clear in the preprocessing setting. The above embedding approach is unlikely to be able to make much use of the free preprocessing. The obvious approach would be to strongly reduce the basis before embedding the target vector. But since the goal is to apply block reduction using sieving to the final basis, the preprocessing might as well directly precompute many short vectors for the individual blocks that will be considered during the reduction. Once the target is embedded, block reduction will successively pass it through the blocks and attempt to shorten it using the precomputed vectors. This is essentially an instance of the approximate Voronoi cell algorithm for which we know that its complexity does not scale well with the radius in the BDD parameter range (see Figure 1a and Section 2.5).

Another approach for BDDP is to strongly reduce the basis and perform enumeration to decode the target. While this might be an efficient way to tackle the problem for small instances in practice, it is asymptotically inefficient for the parameter ranges considered in this work. As shown by [HKM18], there is a very narrow range in the target distance parameter where the running time of enumeration switches from polynomial to super-exponential. (The analysis of enumeration in [HKM18] is in a somewhat different setting but carries to ours as well.) The parameters for which enumeration solves the problem in polynomial time are very small ($r = d^a$ for constant $a < 0$), which is outside the scope of this work for BDDP. Super-exponential running times are clearly asymptotically worse than what can already be achieved in the non-preprocessing setting.

### 2.5 The cost of approximate CVP(P)

*Closest vector problems with preprocessing.* To compare our results with the approximate Voronoi cells approach, we recall the state of the art in this direction. For a fair comparison, in the preprocessing setting we optimized for the time complexity, conditioned on the time complexity not exceeding the memory. This matches the dual attack's time–space trade-off (both the time and memory are $|\mathcal{W}|$), and prohibits costly

nearest neighbor searching which can make the time complexity arbitrarily small at the cost of more memory [DLdW19, Laa20, DLvW20]. We also do not cover batch-CVPP improvements as outlined in [DLvW20]; this would only slightly change the plot in Figure 1a.

For the red curve in Figure 1a, we used the formulas from [DLdW19], solving for an equal time and memory complexity, and then minimizing over all combinations of parameters that achieve an equal time and memory complexity. For BDDP, as explained in e.g. [DLdW19] one can achieve slightly better complexities than for average-case CVPP, but Figure 1a shows that this improvement is almost negligible when viewed on a loglog-scale. Note that the orange line $T = \exp(d/8r^2)$ denotes the asymptotic scaling for approximate CVPP for large approximation factors, as described in [Laa16b, Section 4.2] – the lower bound on the list size parameter $\alpha$ for approximate CVPP with approximation factor $r$ scales as $\alpha = 1 + 1/(8r^2) + O(r^{-4}) = \exp(1/(8r^2) + O(r^{-4}))$, so that the time complexity $T = \alpha^{d+o(d)}$ indeed scales as $T = \exp(d/(8r^2))$. This is almost the mirror image of the dual attack's complexity for small radius $r \to 0$, where the leading constant is $1/e^2 \approx 1/7.4$ rather than $1/8$.

*Closest vector problems, without preprocessing.* For problems without preprocessing, we can do a similar analysis as for the dual attack without preprocessing to analyze which block size is needed to make the attacks work. Similar to the analysis in Section 5, using BKZ with a block size $k = \beta d$ for constant $\beta$ results in an approximation factor $g_k^{d/k}/g_d = d^{1/(2\beta)-1/2}$ for the norms of the vectors output by the sieve on the top block. For constant $\beta$ the number of vectors remains $\exp O(d)$. With an exponential number of vectors from a larger sphere, using e.g. [Laa20, Theorem 3.1] shows that the volume of the resulting approximate Voronoi cell is of the order $d^{d/(2\beta)-d/2}$ larger than the exact Voronoi cell, defined by $\exp O(d)$ vectors of length with a constant approximation factor. To end up with a solution, we need to end up within a radius $r \cdot g_d$ from the origin, which is a region whose volume is a factor $r^d$ larger than the exact Voronoi cell. This yields the inequality $r^d \geq d^{d/(2\beta)-d/2}$, or equivalently $\beta \geq 1/(1 + 2\ln r/\ln d)$. Substituting the cost of BKZ with this block size then yields the red curves for approximate CVPP of Figures 1c and 1d.

## 3 Model

Before describing and analyzing dual attacks from a practical point of view, we first carefully describe the problem setting we consider in this paper, and the heuristic assumptions we make for studying the performance of these algorithms. For simplicity, w.l.o.g. throughout the remainder of the paper we will assume that lattices are *normalized*, so that $\mathrm{Vol}(\mathcal{L}) = \mathrm{Vol}(\mathcal{L}^*) = 1$.

### 3.1 Target distributions

To model closest vector problems accurately, and providing a tight analysis on the average-case performance of algorithms for solving them, we first describe two possible ways of modeling the distribution of target vectors given as input. The first is what may be described as the proper way of sampling uniformly random target vectors.

**Definition 1 (Random target distribution).** *Let $\mathcal{L} \subset \mathbb{R}^d$ be a full-rank lattice and let $\mathbf{B}$ be a basis of $\mathcal{L}$. The random target distribution for $\mathcal{L}$ corresponds to the distribution obtained by sampling target vectors $\boldsymbol{t}$ uniformly at random from the fundamental parallelepiped generated by the basis $\mathbf{B}$.*

This distribution is particularly convenient to work with when computing dot products with dual vectors, as the coefficients of $\boldsymbol{t}$ in terms of the basis $\mathbf{B}$ are uniform on $[-1/2, 1/2]$. For modeling the broader spectrum of closest vector problems, including variants where the target lies closer to the lattice than for the random case, we will mostly make use of the following target distribution.

**Definition 2 (Planted target distribution).** *Let $\mathcal{L} \subset \mathbb{R}^d$ be a full-rank lattice, and let $r > 0$ be given. In the planted target distribution with radius $r$, target vectors $\boldsymbol{t} \in \mathbb{R}^d$ are sampled as follows:*

- *Sample a lattice vector $\boldsymbol{c} \in \mathcal{L}$ from a sufficiently wide[3] Gaussian over $\mathcal{L}$.*
- *Sample an error vector $\boldsymbol{e} \in r \cdot g_d \cdot \mathcal{S}^{d-1}$ uniformly at random.*
- *Return the target vector $\boldsymbol{t} = \boldsymbol{c} + \boldsymbol{e}$.*

Note that in the limit of $r \to \infty$, the planted target distribution converges to the random target distribution. For $r \approx 1$ most of the generated targets will lie at distance approximately $g_d$ from the lattice (which matches the Gaussian heuristic prediction of the distance that random targets are expected to lie from the lattice), but we are given the extra guarantee that there is always at least one lattice vector which is close to our planted target vector. We stress that for any finite $r$, the planted target distribution does *not* exactly match the random target distribution: as we will see later, with a sufficiently large number of samples, one can distinguish between both distributions.

### 3.2 Planted closest vector problem

The closest vector problems we study in this paper can all be summarized as instances of what we define as the planted closest vector problem below, for a suitable choice of $r$.

**Definition 3 (Planted closest vector problem, searching).** *Let $\mathcal{L} \subset \mathbb{R}^d$ be a full-rank lattice. Given a sample $\boldsymbol{t}$ from the planted target distribution with radius $r > 0$, find a vector $\boldsymbol{v} \in \mathcal{L}$ at distance at most $r \cdot g_d$ from $\boldsymbol{t}$.*

With this parameterization, $r$ controls both the radius of the error vector $\boldsymbol{e}$ in the distribution of $\boldsymbol{t}$, as well as the desired distance from the returned lattice vector to the target point. The hardness of the above problem depends both on the lattice dimension (rank) $d$ and on this radius $r$, relative to the Gaussian heuristic $g_d \approx \sqrt{d/(2\pi e)}$ in dimension $d$. Roughly speaking, there are three distinct regimes to consider:

[3] Concretely, one may use a discrete Gaussian with a width sufficiently far above the smoothing parameter of the lattice. Note that for analyzing the dual attack, the exact distribution of $\boldsymbol{c}$ is irrelevant.

**The regime $r \ll 1$ — Bounded distance decoding.**
For error vectors sampled from a relatively small noise distribution, we obtain instances of the well-studied bounded distance decoding problem, with only one lattice vector $\boldsymbol{c}$ lying very close to the target $\boldsymbol{t}$. Heuristically, w.h.p. all other lattice vectors will lie at distance at least $g_d(1 + o(1))$ from $\boldsymbol{t}$, making the planted solution at distance $r \cdot g_d$ the only solution. However, unless $r < \frac{1}{2}$ there is no guarantee that the planted solution is the only solution. As is well known, bounded distance decoding becomes easier as the target lies closer and closer to the lattice; for sufficiently small $r$, the embedding approach via the approximate shortest vector problem becomes polynomial-time with a suitable form of lattice basis reduction (LLL [LLL82], BKZ [Sch87, SE94]).

**The regime $r \approx 1$ — "Average-case" closest vector problem.**
For error vectors of norm close to the Gaussian heuristic, we roughly have instances of the (average-case) closest vector problem, apart from the previous discussion on the difference between the planted target distribution and the random target distribution. We will likely find the hardest instances of this generalized problem in the regime where $r = 1 \pm o(1)$, as then the number of solutions within radius $r$ is small while the target does not lie much closer to the lattice than on average.

**The regime $r \gg 1$ — Approximate closest vector problem.**
For large $r$, the error vectors $\boldsymbol{e}$ are so large that we no longer expect $\boldsymbol{v}$ to be the closest lattice vector to $\boldsymbol{t}$. By the problem statement any lattice vector at distance $r \cdot g_d \gg g_d$ from $\boldsymbol{t}$ suffices as a solution, and by the Gaussian heuristic we expect there to be many of them. This problem is again well-known to become easier and easier as $r$ increases. For sufficiently large $r$, one can first run a form of basis reduction on a basis of $\mathcal{L}$ and then run Babai's nearest plane algorithm [Bab86] to recover a solution in polynomial time.

We expect the hardness of the problem, as a function of $r$, to reach its peak at $r = 1 \pm o(1)$, and we expect the problem to become easier both as $r$ becomes much bigger or becomes much smaller.

### 3.3 The costs of preprocessing

Besides the direct computational problem described above, where we are given a single lattice $\mathcal{L}$ and a single target vector $\boldsymbol{t}$ for which we wish to recover a nearby lattice vector, we will also consider preprocessing variants of these problems. In these variants any amount of preprocessing on $\mathcal{L}$ can essentially be done free of charge, before being given $\boldsymbol{t}$ and being tasked to find a nearby lattice point to $\boldsymbol{t}$. The motivation for this is that both in cryptographic applications and in algorithms appearing in cryptanalysis, one may face the problem of having to decode many targets for the same lattice $\mathcal{L}$. Then, rather than processing each of these separately with a direct closest vector algorithm, it may be beneficial to do a one-time preprocessing step on the lattice, producing data that may be useful for each of these instances, and then solving each of these problem instances faster with this preprocessed data. In reality the costs of the preprocessing phase are of course not negligible, but for a sufficiently large number of problem instances these preprocessing costs can potentially be amortized and may therefore be less important than the costs of the query phase.

Note that a naive solution for any such preprocessing problem would be to run no preprocessing, and running a standard closest vector algorithm to find a solution in the query phase. Non-trivial and potentially useful algorithms are those which take more time in the preprocessing phase (and perhaps even more time than any direct algorithm would require for solving a single closest vector problem instance), but which then require less time in the query phase. Such approaches would take more time for solving one problem instance from scratch, but perform better when solving a sufficiently large batch of closest vector problem instances on the same lattice. Depending on the application, one can then choose between algorithms with and without preprocessing.

### 3.4 Distinguishing and searching

Although most primal algorithms for solving closest vector problems come in only one flavor (directly finding a nearby lattice vector), algorithms based on the dual lattice come in two flavors: algorithms which directly solve the problem of finding a nearest lattice point, and algorithms which only return a yes/no answer whether the given vector lies *close* to the lattice or not. Informally, this corresponds to solving the following alternate distinguishing problem.

**Definition 4 (Planted closest vector problem, distinguishing).** *Let $\mathcal{L} \subset \mathbb{R}^d$ and $r > 0$ be given. Given a target $\boldsymbol{t} \in \mathbb{R}^d$, decide whether it is from the planted target distribution with radius $r$ or from the random target distribution.*

Algorithms for distinguishing can generally be turned into search algorithms with polynomial overhead. However, as this polynomial overhead may be significant, and as for certain applications within cryptanalysis or for undermining a security proof a distinguisher suffices, we will treat these problems separately. Moreover, depending on the bit security model (which again depends on the application), distinguishing may *not* be equivalent to searching, and may in fact be significantly easier.

### 3.5 Single-target and multi-target settings

When evaluating the complexity of an algorithm it is common to analyze its resource requirements to solve the problem with high probability. In the cryptographic setting, there is a generic way of obtaining a trade-off between complexity and success probability, given an algorithm for a search problem like BDD with low success probability: assume that one is given many independent instances of the problem and it is sufficient to solve any one of them. This allows for comparability across different algorithms in the cryptographic setting: if, e.g. in some system, public keys are represented by random targets, we require that out of many public keys, none of the secret keys can be recovered.

In the distinguishing case, the analog is to apply a low advantage distinguisher to many independent samples and take the majority. Again, this makes sense, if we consider targets representing ciphertexts: even given many encryptions of the same plaintext, they should be indistinguishable from many encryptions of another plaintext. We call this setting the *multi-target* setting.

In other settings, we may really want to solve BDD (either the distinguishing or the search version) on a single target with high probability. We call this the *single-target* setting. The algorithms we analyze in this work give a natural trade-off between the time complexity and success probability, determined by the number of dual vectors used and their lengths. Since usually there are only a limited number of very short vectors in a lattice, when using a large number of vectors to increase the success probability we need to make use of longer vectors. This has a negative impact on the power of the dual attack and thus this trade-off is worse than the generic approach in the multi-target setting. So in the context of dual attacks, making this distinction is important.

There is another reason to make this distinction: it seems that the dual distinguisher and the dual decoder are not necessarily equivalent in the multi-target setting. In the distinguishing case, clearly the optimal choice to solve the problem is to use a single shortest dual vector. Peeking ahead, applying this to a long list of $N = \exp((r/e)^2 d)$ independent targets that are either all drawn from the random target distribution or all from the planted target distribution with radius $rg_d$ will yield a distinguisher with high advantage (see Lemma 8). In contrast, it is unlikely that a single dual vector is sufficient to solve the search version of BDD, even once for $N$ independent targets. To see this, observe that one short dual vector only yields information about which hyperplane (defined by the dual vector) the closest point to a target appears to be in. At best, this may allow us to reduce the problem by one dimension. The probability that this reduction in dimension solves the problem is equivalent to the probability that after projecting onto the respective hyperplane, the target is decodable in polynomial time. Even given a dual HKZ-reduced basis for the projected lattice, the radius of the polynomial-time decodable targets is at most $O(g_{d-1}/\sqrt{d-1})$. The probability that the orthogonal projection of a random target at distance $rg_d$ in dimension $d$ lies in a $(d-1)$-dimensional ball with such a radius is proportional to $(r\sqrt{d})^{-d}$. This is super-exponentially small for all $r = \omega(1/\sqrt{d})$, so this will not yield a decoder with high success probability, even given $N = 2^{O(d)}$ targets.

## 4 Algorithms

In this section we will describe algorithms for solving closest vector problems using vectors from the dual lattice, both for distinguishing between a planted distribution and the random distribution, and for actually finding a nearby lattice vector to a (planted) target vector. The heuristic analysis of these algorithms will follow in Section 5, and results of experiments are reported in Section 6.

### 4.1 The Aharonov–Regev decoder

We will start with the Fourier-based derivation of the dual attack, which naturally results in a decoder similar to the one described by Aharonov–Regev [AR04] (and later used in [LLM06,DRS14]). First, suppose we have a target vector $\boldsymbol{t}$, and let $\boldsymbol{c}$ denote the closest lattice point to $\boldsymbol{t}$. Then by using (asymptotic) properties of Gaussians, we can derive the following approximations for the mass of a Gaussian at $\boldsymbol{c}-\boldsymbol{t}$:

$$\rho_s(\boldsymbol{c}-\boldsymbol{t}) \approx \sum_{\boldsymbol{v}\in\mathcal{L}} \rho_s(\boldsymbol{v}-\boldsymbol{t}) \approx \sum_{\boldsymbol{v}\in\mathcal{L}} \rho_s(\boldsymbol{v}+\boldsymbol{t}) \Big/ \sum_{\boldsymbol{v}\in\mathcal{L}} \rho_s(\boldsymbol{v}). \tag{2}$$

The first approximation relies on $\boldsymbol{c}$ being significantly closer to $\boldsymbol{t}$ than all other vectors, so that $\rho_s(\boldsymbol{c}-\boldsymbol{t}) \gg \rho_s(\boldsymbol{v}-\boldsymbol{t})$ for all other lattice vectors $\boldsymbol{v} \in \mathcal{L} \setminus \{\boldsymbol{c}\}$. In that case the sum essentially collapses to the biggest term $\rho_s(\boldsymbol{c}-\boldsymbol{t})$. The second approximation relies on $s \leq 1$ being sufficiently small, so that the sum in the denominator is approximately equal to 1.[4] Next, applying the Fourier transform [AR04], we translate the latter term to a function on the dual lattice:

$$\rho_s(\boldsymbol{c}-\boldsymbol{t}) \approx \sum_{\boldsymbol{w}\in\mathcal{L}^*} \rho_{1/s}(\boldsymbol{w}) \cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle) \Big/ \sum_{\boldsymbol{w}\in\mathcal{L}^*} \rho_{1/s}(\boldsymbol{w}). \tag{3}$$

Finally, we can approximate this infinite sum over dual vectors by only taking the sum over a set $\mathcal{W} \subset \mathcal{L}^*$ of short dual vectors (e.g. those of norm at most some radius $R$), which contribute the most to the numerator above. For this approximation to be accurate we require that the infinite sum over $\boldsymbol{w} \in \mathcal{L}^*$ is well approximated by the finite sum over only vectors $\boldsymbol{w} \in \mathcal{W}$:

$$\rho_s(\boldsymbol{c}-\boldsymbol{t}) \approx \frac{1}{M} \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle), \qquad M = \sum_{\boldsymbol{w}\in\mathcal{L}^*} \rho_{1/s}(\boldsymbol{w}). \tag{4}$$

Note that while the initial function $\rho_s(\boldsymbol{c}-\boldsymbol{t})$ cannot be evaluated without knowledge of $\boldsymbol{c}$, this last term is a finite sum over dot products of dual lattice vectors with the target vector, scaled with a constant $M$ which does not depend on the target $\boldsymbol{t}$. In particular, disregarding the scaling factor $1/M$, the finite sum can be seen as an indicator to the magnitude of $\rho_s(\boldsymbol{c}-\boldsymbol{t})$, which itself is large if $\boldsymbol{t}$ lies close to the lattice (to $\boldsymbol{c}$) and small if $\boldsymbol{t}$ lies far from the lattice. This finite sum can therefore be used to assess if $\boldsymbol{t}$ lies close to the lattice:

$$f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}) := \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle). \tag{5}$$

This quantity can be seen as an instantiation of [AR04, Lemma 1.3] on $\mathcal{W} \subset \mathcal{L}^*$: sampling from a discrete Gaussian restricted to $\mathcal{W}$ and taking the sum of cosines is equivalent to sampling from a uniform distribution over $\mathcal{W}$ and weighing the terms with appropriate Gaussian weights. The above decoder computes the expected value of this sum exactly, as $f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}) = \mathbb{E}_{\boldsymbol{w}\sim\mathcal{W}}[\rho_{1/s}(\boldsymbol{w}) \cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle)]$.

Note that there is one unspecified parameter $s > 0$ above, which corresponds to both the width of the Gaussian over the primal lattice and the reciprocal of the width of the Gaussian over the dual lattice. For the series of approximations to hold, we need both $s$ and $1/s$ to be small enough, so that in the primal lattice the mass of the entire sum over $\boldsymbol{v} \in \mathcal{L}$ is concentrated around $\boldsymbol{c}$, and the sum over all dual vectors is well approximated by the sum over only a finite subset $\mathcal{W} \subset \mathcal{L}^*$.

[4] For the method to work we only need the denominator to be constant in $\boldsymbol{t}$.

### 4.2 The Neyman–Pearson decoder

While the above derivation of the Aharonov–Regev decoder is quite straightforward, it is unclear whether this decoder is actually optimal for trying to distinguish (or search) using dual lattice vectors. Furthermore, the role of the parameter $s$ remains unclear; $s$ should neither be too small nor too large, and finding the optimal value is not obvious. Both Aharonov–Regev [AR04] and Dadush–Regev–Stephens-Davidowitz [DRS14] fixed $s = 1$, but perhaps one can do better (in theory and in practice) with a better choice of $s$.

In the following subsection we prove the following lemma, showing that from a probabilistic point of view, the following Neyman–Pearson decoder is optimal.

**Lemma 2 (Optimal decoder).** *Let $r > 0$ be given. Suppose we wish to distinguish between samples from the random target distribution and samples from the planted target distribution with radius $r \cdot g_d$, given a set of dual vectors $\mathcal{W} \subset \mathcal{L}^*$, and based only on dot products $\langle \boldsymbol{w}, \boldsymbol{t} \rangle \bmod 1$. Let $s > 0$ be defined as:*

$$s = r \cdot g_d \cdot \sqrt{\frac{2\pi}{d}} \approx \frac{r}{\sqrt{e}}. \tag{6}$$

*Then an optimal distinguisher consists of computing the following quantity and making a decision based on whether this quantity exceeds a threshold $\eta$:*

$$f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}) := \sum_{\boldsymbol{w} \in \mathcal{W}} \ln \left( 1 + 2 \sum_{k=1}^{\infty} \rho_{1/s}(k\boldsymbol{w}) \cos(2\pi k \langle \boldsymbol{w}, \boldsymbol{t} \rangle) \right). \tag{7}$$

At first sight this decoder shares many similarities with the Aharonov–Regev decoder, in the form of the sum over Gaussian weights and cosines with dot products. The relation with the Aharonov-Regev decoder can be formalized (see the next subsection) to show that, up to order terms, they are scalar multiples:

$$f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}) \sim 2 \cdot f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}). \tag{8}$$

In contrast to the Aharonov–Regev decoder however, the Neyman–Pearson decoder comes with a guarantee of optimality within the model described in Section 3. Moreover, this decoder comes with an explicit description of the optimal parameter $s$ to use to achieve the best performance. We will later use this as a guideline for choosing parameters $s$ for the Aharonov–Regev decoder in practice.

Using the asymptotic (heuristic) relation $g_d \approx \sqrt{d/(2\pi e)}$, the above optimal parameter $s$ is approximately $s \approx r/\sqrt{e}$. For BDD problems with radius close to the average-case CVP radius, we have $r \approx 1$ and $s \approx 1/\sqrt{e}$. As the BDD radius decreases, the optimal value $s$ decreases linearly with $r$ as well, and thus the width $1/s$ of the Gaussian in the dual increases and scales as $1/r$. As $r$ decreases, the Gaussian over the dual becomes flatter and flatter, and most terms in the summations will roughly have equal weights. This suggests that at least for the easier BDD instances of small radius, using equal weights in the summation may be almost as good as using Gaussian weights.

### 4.3 Derivation of the Neyman–Pearson decoder

For the problem of constructing an efficient decoder we will approach the problem from a hypothesis testing point of view, and use known results from the relevant literature to derive an optimal decoder which describes what is the optimal distinguishing method based on the available evidence. This derivation relies on the classical Neyman–Pearson lemma, stated below.

**Lemma 3 (Neyman–Pearson lemma [NP33]).** *Suppose we wish to distinguish between two hypotheses $H_0$ and $H_1$, given some evidence $X$. Then out of all statistical tests achieving a certain false positive probability $\varepsilon_0$ (i.e. deciding that $H_1$ holds while $H_0$ is actually true), the test minimizing the false negative error $\varepsilon_1$ (i.e. deciding that $H_0$ holds, when $H_1$ is true) is of the form:*

$$\frac{\Pr(X \mid H_0)}{\Pr(X \mid H_1)} > \eta : \textit{Accept } H_0, \qquad \frac{\Pr(X \mid H_0)}{\Pr(X \mid H_1)} < \eta : \textit{Reject } H_0. \tag{9}$$

*Here $\eta = \eta(\varepsilon_0) \in \mathbb{R}$ is some threshold determined by the false positive probability.*

This lemma can be interpreted as follows: to decide between two hypotheses, based on a limited amount of evidence, the best approach is to consider the ratio of probabilities of the evidence under either hypothesis, and see if this quantity is smaller or larger than some threshold $\eta$. This ratio of probabilities under the two different hypotheses is often called the likelihood ratio:

$$\Lambda(x) := \frac{\Pr(X = x \mid H_0)}{\Pr(X = x \mid H_1)}. \tag{10}$$

As deciding whether $\Lambda(x) > \eta$ is equivalent to deciding whether $\ln \Lambda(x) > \eta' := \ln \eta$, sometimes log-likelihood ratios are studied for convenience, as this turns products of (independent) probabilities into sums. Note that for continuous distributions, the probabilities can be replaced by the corresponding probability density functions under $H_0$ and $H_1$, evaluated at $x$.

In our application, for a given $r > 0$ (the radius of the planted target distribution is $r \cdot g_d$), we wish to distinguish between the following two hypotheses:

- $\mathbf{H_0}$: The target $\boldsymbol{t}$ was sampled from the random target distribution.
- $\mathbf{H_1}$: The target $\boldsymbol{t}$ was sampled from the planted distribution with radius $r$.

The evidence $X$ here consists of having knowledge of a set of dual vectors $\mathcal{W} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}$, and dual attacks commonly consist of using the values of the dot products $x_i = (\langle \boldsymbol{w}_i, \boldsymbol{t} \rangle \bmod 1)$ as evidence. The distribution of $x$ is different under these two hypotheses, but in both cases we have a heuristic model describing what we expect their densities to be. Therefore we can explicitly write out these (log-)likelihood ratios, and obtain an explicit expression for the optimal decoder.

Under the hypothesis $H_0$ it is clear that the random variable $X = (\langle \boldsymbol{w}, \boldsymbol{t} \rangle \bmod 1)$ follows a uniform distribution on $[-1/2, 1/2)$, whose density at any point in this interval equals 1. The numerator of the likelihood ratio therefore always equals 1. For the denominator, as explained in Section 5.1 the distribution of $X$ is that of a Gaussian (with mean 0 and variance $\|\boldsymbol{w}\|^2 r^2/d$) modulo 1, whose density we will denote by $f_X(x)$. The likelihood ratio for samples $X_1, \ldots, X_N$, evaluated at $x_1, \ldots, x_N$, thus satisfies:

$$\Lambda(x_1, \ldots, x_N) = \prod_{i=1}^{N} \frac{1}{f_{X_i}(x_i)}\,. \tag{11}$$

Note that as the norms of the dual vectors $\boldsymbol{w}_i$ may be different, the densities $f_{X_i}$ in the denominator are not the same. For convenience we will switch to the log-likelihood score $\ln \Lambda(x_1, \ldots, x_N)$, so that the optimal score function turns into a sum of individual scores. Substituting the expression for $f_X(x)$ from (27), we obtain:

$$\ln \Lambda(x_1, \ldots, x_N) = -\sum_{\boldsymbol{w} \in \mathcal{W}} \ln \left( 1 + 2 \sum_{k=1}^{\infty} \exp\left(-2\pi^2 k^2 \sigma_i^2\right) \cos(2\pi k \langle \boldsymbol{w}, \boldsymbol{t} \rangle) \right). \tag{12}$$

Here $\sigma_i^2 = \|\boldsymbol{e}\|^2 \|\boldsymbol{w}_i\|^2/d = r^2 g_d^2 \|\boldsymbol{w}_i\|^2/d$ is the width of the Gaussian distribution obtained via $\langle \boldsymbol{w}_i, \boldsymbol{t} \rangle$, and $\boldsymbol{e}$ is the error vector in the planted target parameterization $\boldsymbol{t} = \boldsymbol{c} + \boldsymbol{e}$ with norm $r \cdot g_d$. Note that by flipping the sign of the summation, we obtain an equivalent decoder where we accept $H_1$ rather than $H_0$ if some threshold is exceeded. Expanding $\sigma_i$, and substituting $s = r g_d \sqrt{2\pi/d}$, so that we obtain a nice and simple Gaussian weight function for the exponential, this leads to the following result.

**Lemma 4 (Optimal Neyman–Pearson decoder).** *Let a set of dual vectors $\mathcal{W} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}$ and a radius $r > 0$ be given. Suppose we wish to decide if a target vector $\boldsymbol{t} \in \mathbb{R}^d$ comes from the random target distribution or from the planted target distribution with radius $r \cdot g_d$, based only on the evidence $\{x_1, \ldots, x_N\}$, with $x_i = (\langle \boldsymbol{w}_i, \boldsymbol{t} \rangle \bmod 1)$. Let the Gaussian width $s$ be defined as:*

$$s = r \cdot g_d \cdot \sqrt{\frac{2\pi}{d}}\,. \tag{13}$$

*Then the optimal strategy, minimizing the false negative rate for any fixed false positive rate, is to test whether* $f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}) > \eta$ *or not, with* $f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t})$ *given below:*

$$f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}) := \sum_{\boldsymbol{w}\in\mathcal{W}} \ln\left(1 + 2\sum_{k=1}^{\infty} \rho_{1/s}(k\boldsymbol{w})\cos(2\pi k\langle \boldsymbol{w}, \boldsymbol{t}\rangle)\right), \tag{14}$$

$$= \sum_{\boldsymbol{w}\in\mathcal{W}} \ln\left(1 + 2\exp\left(-\frac{2\pi^2 r^2 g_d^2 \|\boldsymbol{w}\|^2}{d}\right)\cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle) + \dots\right) \tag{15}$$

*The threshold* $\eta$ *controls the trade-off between the false positive/negative errors.*

Note that of course, if one takes into account the underlying problem structure, one can observe that the vectors $\boldsymbol{w}_i$ likely form a basis of the dual lattice, from which one can derive a basis for the primal lattice, and from which one can then run a costly CVP algorithm to recover the nearest lattice point to $\boldsymbol{t}$. If however we only use the dot products with the dual vectors (modulo 1), the above lemma says we cannot do better than computing the above quantity and seeing if it exceeds a properly chosen threshold.

Observe that as the Aharonov–Regev decoder fits in the same model, the previous lemma shows that the Aharonov–Regev decoder is not optimal and using the Neyman–Pearson decoder should work strictly better, assuming that the slightly simplified model in this paper is accurate[5]. The following lemma however shows that the Aharonov–Regev decoder is asymptotically almost as good as the optimal Neyman–Pearson decoder.

**Corollary 2 (Equivalence with the Aharonov–Regev decoder).** *Suppose that* $r\cdot\min_{\boldsymbol{w}\in\mathcal{W}} \|\boldsymbol{w}\| = \omega(1)$. *Then, ignoring lower order terms:*

$$f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}) \sim 2 \cdot f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}). \tag{16}$$

*Proof.* For $r\cdot\min_{\boldsymbol{w}\in\mathcal{W}} \|\boldsymbol{w}\| = \omega(1)$, using that $g_d = \Theta(\sqrt{d})$ we have $\rho_{1/s}(\boldsymbol{w}) = o(1)$, so that the dual Gaussian weights decay quickly as either the norm of the argument $\boldsymbol{w}$ increases, or the parameter $k$ in the infinite summation in (14) increases. In particular, the summation over $k$ is dominated by the term with $k = 1$. Ignoring lower order terms, the argument of the logarithm is therefore of the form $1+2\rho_{1/s}(\boldsymbol{w})\cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle) = 1 + o(1)$, and we can use a Taylor expansion of $\ln(1 + x) = x - x^2/2 + O(x^3)$ for small $x = o(1)$ to eliminate the logarithm. What remains is twice the sum appearing in the Aharonov–Regev decoder, plus lower order terms.

As described in Section 5, the condition on $r$ can be read as "suppose that the distinguishing advantage is $o(1)$". For smaller $r$ the condition may not hold, but in that regime the problem can be solved in polynomial time. The main regimes of interest are those where the condition on $r$ is indeed satisfied.

### 4.4 The simple decoder

While the Neyman–Pearson decoder is in a sense a more complicated decoder than the Aharonov–Regev decoder, attempting to achieve a superior theoretical performance, the third decoder we consider favors simplicity over optimality.

In most applications, the set $\mathcal{W}$ will consist of short dual vectors, which will mostly have similar Euclidean norms. For instance, if $\mathcal{W}$ is the output of a sieve, we expect the ratio between the norm of the shortest and longest vectors in $\mathcal{W}$ to be approximately $\sqrt{4/3}$, i.e. at most a 16% difference in norms between the shortest and longest vectors in $\mathcal{W}$. As all norms are very similar anyway, one may attempt to simplify the decoder even further by removing this weighing factor, and using the following "simple" decoder instead:

$$f_{\mathrm{simple}}^{(\mathcal{W})}(\boldsymbol{t}) := \sum_{\boldsymbol{w}\in\mathcal{W}} \cos(2\pi\langle \boldsymbol{w}, \boldsymbol{t}\rangle). \tag{17}$$

[5] Concretely, in our analysis we assumed that the dot product between uniformly random unit vectors is exactly Gaussian, when in reality it follows a beta distribution which converges to a Gaussian distribution as $d \to \infty$.

**Algorithm 1** A dual distinguisher

**Require:** A target $\boldsymbol{t} \in \mathbb{R}^d$, a set of dual vectors $\mathcal{W} \subset \mathcal{L}^*$, a decoder $f$
**Ensure:** The boolean output estimates whether $\boldsymbol{t}$ lies close to the lattice or not
1: Choose a threshold $\eta > 0$
2: **return** $f^{(\mathcal{W})}(\boldsymbol{t}) > \eta$

**Algorithm 2** A dual search algorithm

**Require:** A target $\boldsymbol{t} \in \mathbb{R}^d$, a set of dual vectors $\mathcal{W} \subset \mathcal{L}^*$, a decoder $f$
**Ensure:** The output vector is an estimate for the closest lattice vector to $\boldsymbol{t}$
1: Choose a step size parameter $\delta > 0$
2: **while** $\|\text{Babai}(\boldsymbol{t}) - \boldsymbol{t}\| > \frac{1}{2}\lambda_1(\mathcal{L})$ **do** ▷ any bound $0 < R \leq \frac{1}{2}\lambda_1(\mathcal{L})$ works
3: $\quad \boldsymbol{t} \leftarrow \boldsymbol{t} + \delta \cdot \nabla f^{(\mathcal{W})}(\boldsymbol{t})$
4: **return** Babai($\boldsymbol{t}$)

From an alternative point of view, this simple decoder can be seen as the limiting case of the Aharonov–Regev decoder where we let $s \to 0^+$. For small $s$, the width of the dual Gaussian $1/s$ increases, and all vectors will essentially have the same probability mass after normalization. As the optimal parameter $s$ decreases with the BDD radius $r$ for the Neyman–Pearson decoder, this limiting case may be most relevant for BDD instances with small radius.

Although this decoder may not come with the same optimality guarantees as the Neyman–Pearson decoder, and only approximates it, this decoder does have one particularly convenient property: it can be computed without knowledge of $r$! Whereas the other decoders require knowledge of $r$ to compute the optimal Gaussian width $s$, the simple decoder is a function only of $\mathcal{W}$ and $\boldsymbol{t}$. This might make this decoder more useful in practice, as $r$ may not be known.

### 4.5 Distinguishing algorithms

As already described above, given any decoder $f$ we can easily use this decoder as a distinguisher to decide if we are close to the lattice or not, by evaluating it at $\boldsymbol{t}$ and seeing whether the resulting value is small or large. Formally, this approach is depicted in Algorithm 1. Besides the decoder $f$ and the set of dual vectors $\mathcal{W}$, the only parameter that needs to be chosen is the threshold $\eta$, which controls the trade-off between the false positive (deciding $\boldsymbol{t}$ lies close to the lattice when it does not) and false negative error probabilities. Increasing $\eta$ decreases the false positive rate and increases the number of false negatives.

### 4.6 Search algorithms

To actually find the closest vector to a target vector, we will use the classical gradient ascent approach, which was previously used in [LLM06,DRS14]. Recall that up to scaling, all decoders approximate a Gaussian mass at $\boldsymbol{c} - \boldsymbol{t}$:

$$\exp\left(-\frac{\pi}{s^2}\|\boldsymbol{c} - \boldsymbol{t}\|^2\right) = \rho_s(\boldsymbol{c} - \boldsymbol{t}) \approx C \cdot f^{(\mathcal{W})}(\boldsymbol{t}). \tag{18}$$

Computing the gradient of the Gaussian as a function of $\boldsymbol{t}$, we obtain the relation:

$$\frac{2\pi}{s^2}(\boldsymbol{c} - \boldsymbol{t}) \exp\left(-\frac{\pi}{s^2}\|\boldsymbol{c} - \boldsymbol{t}\|^2\right) = \nabla\rho_s(\boldsymbol{c} - \boldsymbol{t}) \approx C \cdot \nabla f^{(\mathcal{W})}(\boldsymbol{t}). \tag{19}$$

For the exact Gaussian mass function we can isolate $\boldsymbol{c}$, which then gives us an approximation for $\boldsymbol{c}$ in terms of $f$ and its gradient:

$$\boldsymbol{c} = \boldsymbol{t} + \frac{s^2}{2\pi} \cdot \frac{\nabla\rho_s(\boldsymbol{c} - \boldsymbol{t})}{\rho_s(\boldsymbol{c} - \boldsymbol{t})} \approx \boldsymbol{t} + \frac{s^2}{2\pi} \cdot \frac{\nabla f^{(\mathcal{W})}(\boldsymbol{t})}{f^{(\mathcal{W})}(\boldsymbol{t})}. \tag{20}$$

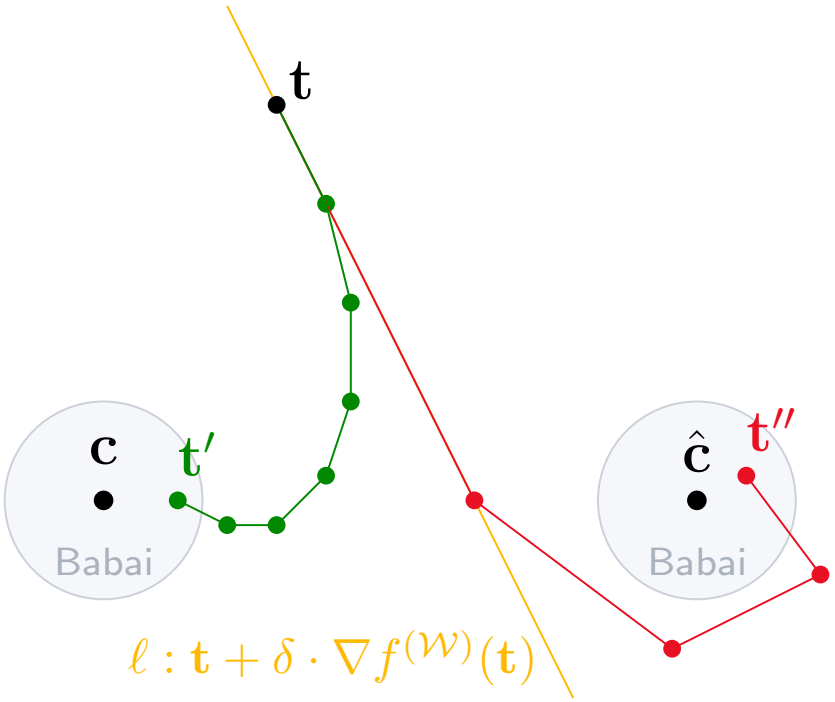



**Fig. 2.** A sketch of the dual search algorithm based on the gradient ascent approach. Starting from a target vector $\boldsymbol{t}$ we move in the direction of the approximate gradient, $\nabla f^{(\mathcal{W})}(\boldsymbol{t})$, which ideally points directly to the true closest vector $\boldsymbol{c}$. We repeatedly update the target vector $\boldsymbol{t}$ to new targets $\boldsymbol{t}'$ and recompute the gradient at this new point at each iteration. Using a small step size $\delta$ (the green path) we will likely require more steps until convergence, but we run a lower risk of *overshooting* and converging towards a different lattice vector $\hat{\boldsymbol{c}}$ (the red path). The gray balls around the lattice points $\boldsymbol{c}$ and $\hat{\boldsymbol{c}}$ denote the radius where Babai's polynomial-time algorithms are guaranteed to find the nearest vector $\boldsymbol{c}$, given as input a target vector within one of these balls. Note that we do not necessarily have to continue until $\boldsymbol{t}'$ lies within one of these balls; as soon as $\|\text{Babai}(\boldsymbol{t}') - \boldsymbol{t}'\| \leq \frac{1}{2}\lambda_1(\mathcal{L})$, we know that $\text{Babai}(\boldsymbol{t}')$ must be the closest vector to $\boldsymbol{t}'$ and we can terminate the gradient ascent.

Here the direction $\nabla f^{(\mathcal{W})}(\boldsymbol{t})$ can be seen as an approximation to the direction we need to move in, starting from $\boldsymbol{t}$, to move towards the closest vector $\boldsymbol{c}$. The scalar $s^2/(2\pi f^{(\mathcal{W})}(\boldsymbol{t}))$ controls the step size in this direction; this would be our best guess to get as close to $\boldsymbol{c}$ in one step as possible. In practice one may choose to use a smaller step size $\delta$, and repeatedly iterate replacing $\boldsymbol{t}$ by $\boldsymbol{t}' = \boldsymbol{t} + \delta \cdot \nabla f^{(\mathcal{W})}(\boldsymbol{t})$ to slowly move towards $\boldsymbol{c}$. Ideally, iterating this procedure a few times, our estimates will get more accurate, and $\boldsymbol{t}'$ will eventually lie close enough to $\boldsymbol{c}$ for polynomial-time closest vector algorithms (e.g. Babai's algorithms [Bab86]) to recover $\boldsymbol{c}$.

The above gradient ascent approach is formalized in Algorithm 2, and a sketch of the procedure is given in Figure 2. This figure gives some rough intuition to the order of magnitude of the ideal step size, the intuitive dependence of the rate of convergence on the step size, and why we want the gradient to point towards $\boldsymbol{c}$, starting from $\boldsymbol{t}$. As long as $\langle \nabla f^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{t} - \boldsymbol{c} \rangle < 0$, we can always take a sufficiently small step in the direction of the gradient and end up with a shifted target $\boldsymbol{t}'$ which lies (slightly) closer to $\boldsymbol{c}$ than the original target $\boldsymbol{t}$.

For illustration, note that we can compute gradients for the decoders explicitly. For instance, for the Aharonov–Regev decoder we have:

$$\nabla f_{\text{AR}}^{(\mathcal{W})}(\boldsymbol{t}) = -2\pi \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \sin(2\pi \langle \boldsymbol{w}, \boldsymbol{t} \rangle) \cdot \boldsymbol{w}. \tag{21}$$

Our best guess for the nearest lattice point $\boldsymbol{c}$, given $\boldsymbol{t}$ and $\mathcal{W}$, would be:

$$\boldsymbol{c}_{\text{AR}}^{(\mathcal{W})} = \boldsymbol{t} - s^2 \cdot \frac{\sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \sin(2\pi \langle \boldsymbol{w}, \boldsymbol{t} \rangle) \cdot \boldsymbol{w}}{\sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \cos(2\pi \langle \boldsymbol{w}, \boldsymbol{t} \rangle)}. \tag{22}$$

For the Neyman–Pearson decoder and simple decoder we can similarly derive explicit expressions for the gradient, and obtain preliminary estimates for $\boldsymbol{c}$ based on the provided evidence.

### 4.7 Choosing the set of dual vectors $\mathcal{W}$

Although some long dual vectors may contribute more to the output of the decoders if they happen to be almost parallel to $\boldsymbol{t} - \boldsymbol{c}$, overall the largest contribution to the sums appearing in all three decoders comes from the shortest dual vectors. Some logical choices for $\mathcal{W}$ would therefore be:

1. $\mathcal{W} = \{\boldsymbol{s}^*\}$, where $\boldsymbol{s}^*$ is a shortest non-zero vector of $\mathcal{L}^*$;
2. $\mathcal{W} = \mathcal{L}^* \cap B(R \cdot g_d)$, i.e. all dual lattice vectors of norm at most $R \cdot g_d$;
3. $\mathcal{W} = \text{Sieve}(\mathcal{L}^*)$, i.e. the output of a lattice sieve applied to (a basis of) $\mathcal{L}^*$;
4. $\mathcal{W} = \text{BKZSieve}(\mathcal{L}^*, k)$, i.e. running a sieving-based BKZ algorithm with block size $k$ on a basis of $\mathcal{L}^*$, and outputting the database of short vectors found in the top block of the basis [ADH+19].

The first choice commonly does not suffice to actually distinguish for one target vector, but the distinguishing advantage per vector is maximized for this option (see Section 5). This choice of $\mathcal{W}$ is often considered in cryptanalytic contexts when studying the performance of dual attacks on LWE. The second choice is a more realistic choice when we actually wish to distinguish or find a closest lattice point to a single target vector, rather than just maximizing the distinguishing advantage. Computing this set $\mathcal{W}$ may be costly (see [DLdW19] for a preprocessing method attempting to achieve this), but in preprocessing problems this is likely the best set $\mathcal{W}$ one can possibly use. The third option is an approximation to the second approach, and may make more sense in practical contexts. The fourth option is a more sensible option when the costs of the preprocessing phase are taken into account as well, as then we commonly wish to balance the preprocessing costs and the costs of the dual attack via a suitable choice of the block size $k$.

## 5 Asymptotics

To study the performance of the algorithms from the previous section, we will first analyze the distributions of the output of the decoder, when the target is either planted or random. Then we will analyze how this can be used to assess the costs of distinguishing and searching for nearest vectors.

### 5.1 Gaussians modulo 1

From the definition of the planted target distribution with radius $r \cdot g_d$, the target is of the form $\boldsymbol{t} = \boldsymbol{c} + \boldsymbol{e}$ where $\boldsymbol{c} \in \mathcal{L}$ and $\boldsymbol{e}$ is sampled uniformly at random from the sphere of radius $r \cdot g_d$. All decoders consider dot products between the target and dual lattice vectors, for which we have:

$$\langle \boldsymbol{t}, \boldsymbol{w} \rangle = \langle \boldsymbol{c} + \boldsymbol{e}, \boldsymbol{w} \rangle = \langle \boldsymbol{c}, \boldsymbol{w} \rangle + \langle \boldsymbol{e}, \boldsymbol{w} \rangle \in \langle \boldsymbol{e}, \boldsymbol{w} \rangle + \mathbb{Z}. \tag{23}$$

Here $\langle \boldsymbol{c}, \boldsymbol{w} \rangle \in \mathbb{Z}$ since $\boldsymbol{c} \in \mathcal{L}$ and $\boldsymbol{w} \in \mathcal{L}^*$. As in all decoders the quantity $\langle \boldsymbol{t}, \boldsymbol{w} \rangle$ is the argument of a cosine, with multiplicative factor $2\pi$, we are interested in the distribution of $\langle \boldsymbol{t}, \boldsymbol{w} \rangle \bmod 1$, which equals the distribution of $\langle \boldsymbol{e}, \boldsymbol{w} \rangle \bmod 1$.

Now, by assumption the distribution of $\boldsymbol{e}$ is independent from $\boldsymbol{w}$. By either assuming the distribution of $\boldsymbol{e}$ is spherically symmetric or the distribution of $\boldsymbol{w}$ is spherically symmetric[6], we can rewrite the above as follows:

$$\langle \boldsymbol{e}, \boldsymbol{w} \rangle = \|\boldsymbol{e}\| \cdot \|\boldsymbol{w}\| \cdot \left\langle \tfrac{\boldsymbol{e}}{\|\boldsymbol{e}\|}, \tfrac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right\rangle \sim \|\boldsymbol{e}\| \cdot \|\boldsymbol{w}\| \cdot \langle \boldsymbol{r}_1, \boldsymbol{r}_2 \rangle, \ \boldsymbol{r}_1, \boldsymbol{r}_2 \sim \mathcal{S}^{d-1}. \tag{24}$$

Here $\boldsymbol{r}_1, \boldsymbol{r}_2$ are drawn uniformly at random from $\mathcal{S}^{d-1}$. Without loss of generality we may fix $\boldsymbol{r}_1 = \boldsymbol{e}_1$ as the first unit vector, so that the dot product is the first coordinate of the random unit vector $\boldsymbol{r}_2$. In high dimensions the uniform distribution on the unit sphere is essentially equivalent to sampling each coordinate independently from a Gaussian with mean 0 and variance $1/d$ (and normalizing[7]), so up to order terms the distribution of the first coordinate of $\boldsymbol{r}_2$ is equal to the Gaussian distribution with mean 0 and variance $1/d$. So focusing on the regime of large $d$, it follows that:

$$\langle \boldsymbol{e}, \boldsymbol{w} \rangle \sim \mathcal{N}\left(0, \ \tfrac{1}{d} \cdot \|\boldsymbol{w}\|^2 \cdot \|\boldsymbol{e}\|^2\right). \tag{25}$$

For the original dot product, taken modulo 1, we thus obtain:

$$(\langle \boldsymbol{t}, \boldsymbol{w} \rangle \bmod 1) \sim \mathcal{N}\left(0, \ \tfrac{1}{d} \cdot \|\boldsymbol{w}\|^2 \cdot \|\boldsymbol{e}\|^2\right) \bmod 1. \tag{26}$$

This distribution is sketched in Figure 3a.

To study the asymptotic performance of various decoders, we first derive more concrete asymptotics on the shape of this distribution below. More precisely, we study the shape of the distribution when starting with a Gaussian distribution on $\mathbb{R}$, and only taking the fractional part, i.e. evaluating a Gaussian modulo 1. For completeness we provide a proof as well, although we do not claim novelty for this result; various works on the dual attack have previously derived similar expressions for the shape of Gaussians modulo 1.

**Lemma 5 (Properties of Gaussians modulo 1).** *Let $Z \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = \omega(1)$, and let $X = [Z \bmod 1] \in [-\frac{1}{2}, \frac{1}{2})$ denote the Gaussian distribution with variance $\sigma^2$, evaluated modulo 1. Let $\rho = \exp(-2\pi^2\sigma^2)$. Then the density $f_X$ of $X$ satisfies:*

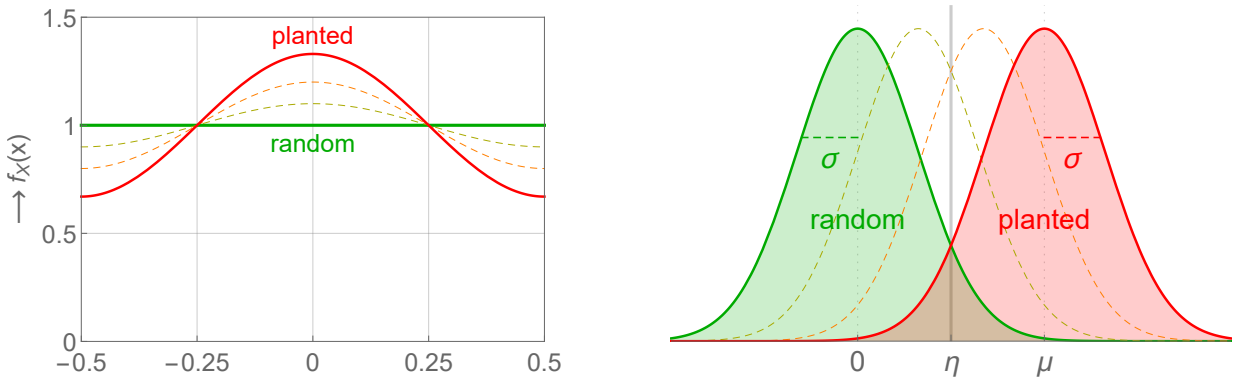
$$f_X(x) = 1 + 2\sum_{k=1}^{\infty} \rho^{k^2} \cos(2\pi k x) \tag{27}$$

$$= 1 + 2\rho\cos(2\pi x) + O(\rho^4), \tag{28}$$

$$\ln f_X(x) = 2\rho\cos(2\pi x) - 2\rho^2\cos^2(2\pi x) + \tfrac{8}{3}\rho^3\cos^3(2\pi x) + O(\rho^4). \tag{29}$$

[6] The following argument therefore holds not only if $\boldsymbol{t}$ is from the planted target distribution, but also if $\boldsymbol{t}$ is fixed and the distribution of the dual vectors is modeled via the Gaussian heuristic.

[7] For large $d$ the squared norm of such a vector follows a chi-squared distribution, which is closely concentrated around 1.




(a) Density of Gaussians modulo 1.

(b) Output distributions of decoders.

**Fig. 3.** A graphical sketch of Gaussians modulo 1, with varying planted target radii, and the output distributions of the three decoders. We wish to distinguish between Gaussians with similar standard deviations but different means.

*Furthermore, for $x \sim X$ and $z \sim Z$, the following equations hold:*

$$\mathbb{E}[\cos(2\pi x)] = \rho, \qquad \mathrm{Var}[\cos(2\pi x)] = \tfrac{1}{2} - \rho^2 + O(\rho^4), \tag{30}$$

$$\mathbb{E}[\ln f_X(x)] = \rho^2 + O(\rho^4), \qquad \mathrm{Var}[\ln f_X(x)] = 2\rho^2 + O(\rho^4), \tag{31}$$

$$\mathbb{E}[z\sin(2\pi z)] = 2\pi\sigma^2\rho, \qquad \mathrm{Var}[z\sin(2\pi z)] = \tfrac{1}{2}\sigma^2 - \tilde{O}(\rho^2), \tag{32}$$

*Proof.* The lemma contains many claims, which we will prove one by one below.

**The density.** First, the density $f_X(x)$ at any point $x \in (-1/2, 1/2)$ can be expressed as a sum of standard Gaussian densities at $x + m$ for $m \in \mathbb{Z}$:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{m\in\mathbb{Z}} \exp\left(-\frac{(x+m)^2}{2\sigma^2}\right). \tag{33}$$

Let us now express $f_X(x)$ in terms of its Fourier series, to obtain a simplified expression. For the Fourier coefficients we obtain:

$$c_k = \int_{-1/2}^{1/2} f_X(x)\exp(2\pi ikx)dx \tag{34}$$

$$= \frac{1}{\sigma\sqrt{2\pi}} \sum_{m\in\mathbb{Z}} \int_{-1/2}^{1/2} \exp\left(-\frac{(x+m)^2}{2\sigma^2}\right)\exp(2\pi ikx)dx \tag{35}$$

$$= \frac{1}{\sigma\sqrt{2\pi}} \sum_{m\in\mathbb{Z}} \int_{m-1/2}^{m+1/2} \exp\left(-\frac{x^2}{2\sigma^2}\right)\exp(2\pi ikx)dx \tag{36}$$

$$= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right)\exp(2\pi ikx)dx. \tag{37}$$

As the argument is even, the imaginary part will evaluate to 0, and we are only left with an integration over the real part, $\exp(-x^2/(2\sigma^2))\cos(2\pi kx)$, which can be evaluated easily as [AS72, Equation 7.4.6]:

$$c_k = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right)\cos(2\pi kx)dx = \exp(-2\pi^2k^2\sigma^2) = \rho^{k^2}. \tag{38}$$

Note that $c_0 = 1$ and $c_{-k} = c_k$. We can now rewrite $f_X(x)$ in terms of its Fourier series as follows:

$$f_X(x) = \sum_{k\in\mathbb{Z}} c_k\cos(2\pi kx) = 1 + 2\sum_{k=1}^{\infty} \rho^{k^2}\cos(2\pi kx). \tag{39}$$

For $\sigma = \omega(1)$ we have $\rho = o(1)$ and the magnitude of the terms quickly decreases. Using the bounds $|\cos(2\pi kx)| \leq 1$ for all $x$ and $k$, and $k^2 \geq 2 + k$ for all $k \geq 2$, we get $|\sum_{k=2}^{\infty} \rho^{k^2} \cos(2\pi kx)| \leq \sum_{k=2}^{\infty} \rho^{k+2} = \rho^4/(1-\rho) = O(\rho^4)$. By only separating the term $k = 1$ from the sum, we therefore get the final expression as claimed:

$$f_X(x) = 1 + 2\rho\cos(2\pi x) + O(\rho^4). \tag{40}$$

**The log-density.** Starting from the above density, using the Taylor expansion $\ln(1+\delta) = \delta - \frac{1}{2}\delta^2 + \frac{1}{3}\delta^3 + O(\delta^4)$ for small $\delta$, and observing that $\delta = 2\rho\cos(2\pi x) = o(1)$ for $\sigma = \omega(1)$, the result follows.

**Expectations and variances.** All the stated equations follow from writing out the expectations by their definitions, using the aforementioned expressions for $f_X(x)$ and $\ln f_X(x)$:

$$\mathbb{E}[g(x)] = \int_{-1/2}^{1/2} f_X(x) g(x) dx, \qquad \mathrm{Var}[g(x)] = \mathbb{E}[g(x)^2] - \mathbb{E}[g(x)]^2. \tag{41}$$

In the derivation of the first four equations we further used the following two results:

$$\int_{-1/2}^{1/2} \cos^2(2\pi x) dx = \frac{1}{2}, \qquad \int_{-1/2}^{1/2} \cos^{2k+1}(2\pi x) dx = 0 \quad (k \in \mathbb{Z}). \tag{42}$$

For these first four results, what then remains is a technical exercise using Taylor series. For the last two results, involving sines of Gaussians, we use complex expansions of which we take the imaginary part to obtain the corresponding integrals over sines,[8] after which the results again follow from standard properties of expectations and variances. □

### 5.2 Output distributions of the decoders

Using Lemma 5, we can easily derive expressions for the means $\mu$ and variances $\sigma^2$ for each of the decoders from the previous section, assuming that $\boldsymbol{t}$ was sampled from the planted target distribution for radius $r$. These can then be used to assess the performance of these decoders. We remark that some of the analysis for $\mu_{\text{simple}}$ and $\sigma_{\text{simple}}$ was also recently carried out in [EJK20] in the context of analyzing small secret LWE.

**Lemma 6 (Output distribution of the decoders).** *Let $\mathcal{W} \subset \mathcal{L}^*$. Let $s = r g_d \sqrt{2\pi/d}$. Let $\boldsymbol{t}$ be sampled from the planted target distribution with radius $r \cdot g_d$. Then over the randomness of $\boldsymbol{t}$:*

$$\mathbb{E}\left[f_{\text{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2, \qquad \mathrm{Var}\left[f_{\text{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx \tfrac{1}{2} \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2 \left[1 + O(\rho_{1/s}(\boldsymbol{w})^2)\right],$$

$$\mathbb{E}\left[f_{\text{NP}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx 2 \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2 \left[1 + O(\rho_{1/s}(\boldsymbol{w})^2)\right], \qquad \mathrm{Var}\left[f_{\text{NP}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx 2 \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2 \left[1 + O(\rho_{1/s}(\boldsymbol{w})^2)\right],$$

$$\mathbb{E}\left[f_{\text{simple}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w}), \qquad \mathrm{Var}\left[f_{\text{simple}}^{(\mathcal{W})}(\boldsymbol{t})\right] \approx \tfrac{1}{2} \sum_{\boldsymbol{w}\in\mathcal{W}} 1 \left[1 + O(\rho_{1/s}(\boldsymbol{w})^2)\right].$$

*Here $\approx$ denotes approximating the beta distributions, arising when taking dot products between vectors drawn uniformly at random from a sphere, by normal distributions. For all three decoders, for targets from the random target distribution we have $\hat{\mu} = 0$ and $\hat{\sigma}^2 = \sigma^2$, i.e. for random targets the mean of the output of the decoder will be 0 and the variance will be the same as for the planted target distribution the decoder was designed for.*

*Proof.* All these results are based on straightforward expansions of the associated expectations and variances, using e.g. that the expectation of a sum equals the sum of expectations. We illustrate two of these expansions

[8] https://stats.stackexchange.com/q/294007/36155

below. For the expected value of the output of the Aharonov–Regev decoder we have:

$$\mathbb{E}_{\boldsymbol{t}}\left[f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right] = \mathbb{E}_{\boldsymbol{t}}\left[\sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})\cos(2\pi\langle \boldsymbol{w},\boldsymbol{t}\rangle)\right] \tag{43}$$

$$= \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})\cdot \mathbb{E}_{\boldsymbol{t}}[\cos(2\pi\langle \boldsymbol{w},\boldsymbol{t}\rangle)] \tag{44}$$

$$\approx \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})\cdot \mathbb{E}_{z_w}[\cos(2\pi z_w)] \tag{45}$$

$$= \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}^2(\boldsymbol{w}). \tag{46}$$

Here $z_w \sim \mathcal{N}(0, \|\boldsymbol{e}\|^2\cdot\|\boldsymbol{w}\|^2/d)$ is used to approximate the beta distribution that arises when taking dot products between vectors sampled uniformly and independently from a sphere. The last equality then follows from Lemma 5 to evaluate this expected value. For the variance we have:

$$\mathrm{Var}_{\boldsymbol{t}}\left[f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right] = \mathrm{Var}_{\boldsymbol{t}}\left[\sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})\cos(2\pi\langle \boldsymbol{w},\boldsymbol{t}\rangle)\right] \tag{47}$$

$$= \sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}^2(\boldsymbol{w})\cdot \mathrm{Var}_{\boldsymbol{t}}[\cos(2\pi\langle \boldsymbol{w},\boldsymbol{t}\rangle)] \tag{48}$$

$$\approx \frac{1}{2}\sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}^2(\boldsymbol{w}). \tag{49}$$

We leave the other four derivations to the reader. □

Figure 3b thus illustrates the situation well, in that in both cases the output distribution follows a Gaussian with the same variance, but the mean depends on whether $\boldsymbol{t}$ was planted or random.

In all applications we try to distinguish/search for a closest vector using a large number of samples $\langle \boldsymbol{w},\boldsymbol{t}\rangle$, either using many dual vectors $\boldsymbol{w}$ or many targets $\boldsymbol{t}$, and taking appropriate sums as defined in the decoders. As the terms of the summation are approximately identically distributed[9], by the central limit theorem we expect the output of the decoder to be approximately Gaussian, with the above means and variances.

Assuming that the output distributions are perfectly Gaussian, as sketched in Figure 3b, we can distinguish with constant errors iff $\mu/\sigma$ is at least constant, with the ratio implying the distinguishing advantage. Considering the squares of this quantity for convenience, for the AR and NP decoders we have:

$$\frac{\mathbb{E}\left[f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right]^2}{\mathrm{Var}\left[f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t})\right]} \approx \frac{\mathbb{E}\left[f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t})\right]^2}{\mathrm{Var}\left[f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t})\right]} \approx 2\sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2. \tag{50}$$

As long as this quantity is at least constant, we can confidently distinguish between planted and random targets. For the simple decoder we get a slightly different expression for this ratio:

$$\frac{\mathbb{E}\left[f_{\mathrm{simple}}^{(\mathcal{W})}(\boldsymbol{t})\right]^2}{\mathrm{Var}\left[f_{\mathrm{simple}}^{(\mathcal{W})}(\boldsymbol{t})\right]} \approx \frac{2}{|\mathcal{W}|}\left(\sum_{\boldsymbol{w}\in\mathcal{W}} \rho_{1/s}(\boldsymbol{w})\right)^2. \tag{51}$$

Observe that the Cauchy–Schwarz inequality applied to the $|\mathcal{W}|$-dimensional vectors $\boldsymbol{x} = (1)_{\boldsymbol{w}\in\mathcal{W}}$ and $\boldsymbol{y} = (\rho_{1/s}(\boldsymbol{w}))_{\boldsymbol{w}\in\mathcal{W}}$ shows that this quantity for the simple decoder is never larger than for the other two decoders, with equality only iff the weights $\rho_{1/s}(\boldsymbol{w})$ are all the same. As we wish to maximize this ratio to maximize our distinguishing advantage, the simple decoder is not better than the other decoders, and is not much worse if vectors in $\mathcal{W}$ have similar norms.

[9] For the simple decoder indeed the distribution of each term is identical. Due to the weighing factors $\rho_{1/s}(\boldsymbol{w})$ in the other two decoders, the terms are not quite identically distributed. However, in most cases of interest, the important contribution to the output distribution comes from a subset of vectors of $\mathcal{W}$ with almost equal norms.

### 5.3 The direction of the gradient

A similar argument can be used to analyze the costs of the gradient ascent approach. To make progress, we need the gradient to point in the direction of $\boldsymbol{c}$, starting from $\boldsymbol{t}$: as long as the gradient is pointing in this direction, a small step in the direction of the gradient will bring us closer to $\boldsymbol{c}$. Studying the associated random variable $\langle \nabla f^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{t} - \boldsymbol{c} \rangle$, we again see that by the central limit theorem this will be approximately Gaussian. The following lemma describes the means and variances for different decoders.

**Lemma 7 (Direction of the gradient).** *Let $\mathcal{W} \subset \mathcal{L}^*$. Let $\rho_{\boldsymbol{w}} = \rho_{1/s}(\boldsymbol{w})$ for $s = r g_d \sqrt{2\pi/d}$. Let $\boldsymbol{t}$ be sampled from the planted target distribution as $\boldsymbol{t} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{c} \in \mathcal{L}$ and $\boldsymbol{e}$ uniform from $r \cdot g_d \cdot \mathcal{S}^{d-1}$. Then:*

$$\mathbb{E}\left[\langle \nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx -2\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{\boldsymbol{w}}^2, \qquad \mathrm{Var}\left[\langle \nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx \pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{\boldsymbol{w}}^2,$$

$$\mathbb{E}\left[\langle \nabla f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx -4\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{\boldsymbol{w}}^2, \qquad \mathrm{Var}\left[\langle \nabla f_{\mathrm{NP}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx 4\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{\boldsymbol{w}}^2,$$

$$\mathbb{E}\left[\langle \nabla f_{\mathrm{simple}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx -2\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{\boldsymbol{w}}, \qquad \mathrm{Var}\left[\langle \nabla f_{\mathrm{simple}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] \approx \pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2.$$

*Proof.* Similar to the proof of Lemma 6 above, this is mostly a matter of applying properties from Lemma 5, and writing out the resulting expectations and variances. We again sketch the procedure by writing out the expressions for the Aharonov–Regev decoder. First, the gradient can be explicitly written as:

$$\nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}) = -2\pi \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \sin(2\pi \langle \boldsymbol{w}, \boldsymbol{t} \rangle) \cdot \boldsymbol{w}. \tag{52}$$

Note that the term $\langle \boldsymbol{w}, \boldsymbol{t} \rangle$ inside the sine can equivalently be replaced by $\langle \boldsymbol{w}, \boldsymbol{t} \rangle$, as $\langle \boldsymbol{w}, \boldsymbol{c} \rangle \in \mathbb{Z}$ and the sine is periodic modulo $2\pi$. Taking the dot product with $\boldsymbol{e}$, we obtain:

$$\begin{aligned}
\mathbb{E}\left[\langle \nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle\right] &= \mathbb{E}_{\boldsymbol{e}}\left[-2\pi \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \sin(2\pi \langle \boldsymbol{w}, \boldsymbol{e} \rangle) \cdot \langle \boldsymbol{w}, \boldsymbol{e} \rangle\right] && (53)\\
&= -2\pi \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \, \mathbb{E}_{\boldsymbol{e}}\left[\langle \boldsymbol{w}, \boldsymbol{e} \rangle \cdot \sin(2\pi \langle \boldsymbol{w}, \boldsymbol{e} \rangle)\right] && (54)\\
&\approx -2\pi \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w}) \, \mathbb{E}_{z_w}\left[z_w \sin(2\pi z_w)\right], && (55)\\
&= -2\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{1/s}^2(\boldsymbol{w}). && (56)
\end{aligned}$$

In the last expression, the Gaussian random variable $z_w \sim \mathcal{N}(0, \|\boldsymbol{e}\|^2 \|\boldsymbol{w}\|^2/d)$ approximates the beta distribution one obtains when taking dot products between random vectors from a sphere. The expectations over $z_w$ then follow from Lemma 5, where we then use the definition of $s = r \cdot g_d \cdot \sqrt{2\pi/d}$ to obtain a term $\rho_{1/s}(\boldsymbol{w}) = \exp(-2\pi^2\sigma^2)$ with $\sigma^2 = \|\boldsymbol{e}\|^2\|\boldsymbol{w}\|^2/d)$, and multiplicative factors $s^2$ and $\|\boldsymbol{w}\|^2$ from the contribution $2\pi\sigma^2$ to the expectation of $z \sin(2\pi z)$. The other five asymptotic relations can be derived in a similar fashion. □

Note that the above lemma describes that, as long as the mean of $\langle \nabla f^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{e} \rangle$ is a constant multiple of the standard deviation, with high probability the gradient points a "significant amount" in the direction of $\boldsymbol{t}$. To satisfy this, we again get a similar condition on $\sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}^2(\boldsymbol{w})$ being non-negligible, i.e. we get the same heuristic complexities for the gradient ascent to make progress as for the distinguisher to successfully distinguish, which motivates Lemma 10.

As before, to get an idea of the costs of this method, we can look at the ratio between the squared means and the standard deviations. For the AR decoder for instance, this quantity scales as:

$$\frac{\mathbb{E}\left[\langle \nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{t} - \boldsymbol{c} \rangle\right]^2}{\mathrm{Var}\left[\langle \nabla f_{\mathrm{AR}}^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{t} - \boldsymbol{c} \rangle\right]} = 4\pi s^2 \sum_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|^2 \rho_{1/s}^2(\boldsymbol{w}). \tag{57}$$

Up to polynomial factors in $d$ (e.g. $s^2$ and $\|\boldsymbol{w}\|^2$), this rough cost estimate is equivalent to the condition for the distinguisher to succeed. This is unsurprising, as in many cases there is a polynomial-time reduction between searching and distinguishing.

### 5.4 Cost estimates

Based on the sketch of Figure 3b, as described above we can get concrete cost estimates for these algorithms by inspecting the means and variances, and computing the right ratio. These can be estimated both with and without using specific information about the data set or the lattice.

**Data-dependent estimates, example.** From Figure 3b, we may estimate that the output distributions of both decoders are roughly Gaussian. Suppose we wish to balance the false positive and false negative error rates, and we want to guarantee that each error rate is at most 10%. If $X$ is from a standard normal distribution, then the tail $\Pr(X \geq x)$ is 10% iff $x \approx 1.281551\ldots$. Alternatively, if $X \sim \mathcal{N}(0, \sigma^2)$, we have $\Pr(X \geq 1.2816\sigma) \approx 0.10$. As we estimate that the standard deviations of both output distributions are roughly the same, and the mean of the random target distribution is 0, we will put the distinguishing threshold $\eta$ halfway, around $\mu/2$, with $\mu$ denoting the mean of the planted output distribution. To distinguish 90th percentiles we therefore need $\mu/2 \approx 1.2816\sigma$, or squaring and rewriting, we need $\mu^2/\sigma^2 \approx (2 \cdot 1.2816)^2 \approx 6.5695$.

Suppose we have a list $\mathcal{W}$ of many short dual vectors, sorted from smallest to largest. For e.g. the Neyman–Pearson decoder we can then start computing the summation from (50), starting from the shortest dual vectors, and adding more and more vectors until the sum exceeds 6.5695. Around the point where the partial sum exceeds 6.5695, we expect to have enough vectors to distinguish with false positive and false negative error rates of at most 10%.

**Data-independent estimates, example.** Continuing from the previous example, we can obtain even more direct estimates of the required number of vectors to distinguish with a certain confidence based on heuristics of vector norms in the dual lattice. Using the Gaussian heuristic, we estimate that the norm of the $i$th shortest dual vector satisfies $\|\boldsymbol{w}_i\| = i^{1/d} \cdot g_d$. Similar to the data-dependent setting, we can then compute the partial sum:

$$\frac{\mu^2}{\sigma^2} \approx 2 \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2 \approx 2 \sum_{i=1}^{n} \exp(-2\pi s^2 i^{2/d} g_d^2). \tag{58}$$

For given $s$ (e.g. $s = r \cdot g_d \cdot \sqrt{2\pi/d}$), the above expression is only a function of the number of points $n$ used for distinguishing, the lattice dimension $d$, the target radius $r$, and the Gaussian heuristic $g_d = \Gamma(d/2+1)^{1/d}/\sqrt{\pi}$. Given $d$ and $r$ we can compute this sum and find the value $n$ for which we first expect the sum to exceed 6.5695. This could then be used as an estimate for the number of dual lattice vectors needed to distinguish with false positive and false negative error rates of at most 10%.

### 5.5 Closest vector problems with preprocessing

We study two separate applications: (i) we are given a large number of target vectors, and we wish to decide whether they come from the planted target distribution or from the random distribution; and (ii) we have only one target vector and we either want to distinguish correctly or find a closest lattice point to it. The former problem has a lower asymptotic complexity, as it allows us to reuse the shortest dual vector for different targets.

**Lemma 8 (Multi-target distinguishing, with preprocessing).** *Suppose an arbitrary number of targets $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots$ is given, which are either all from the random target distribution or all from the planted target distribution with given radius $r \cdot g_d$, for arbitrary $r > 0$. Let $\boldsymbol{w} \in \mathcal{L}^*$ be the shortest dual vector of norm $\|\boldsymbol{w}\| = g_d(1 + o(1))$, and suppose $\boldsymbol{w}$ is known. Then heuristically we can distinguish between these two cases using $\boldsymbol{w}$ in time:*

$$T = 2^{r^2(d+o(d))/(e^2 \ln 2)} \approx 2^{0.195 r^2 (d+o(d))}. \tag{59}$$

*Proof.* As described before, we can confidently distinguish if the sum of the squared weights $\rho_{1/s}(\boldsymbol{w})^2$ is constant, where $s = r \cdot g_d \cdot \sqrt{2\pi/d}$ as before. In this case we sum over different targets $\boldsymbol{t}$, rather than vectors $\boldsymbol{w}$, so using $T$ target vectors, the sum equals $T \cdot \rho_{1/s}(\boldsymbol{w})^2 = T \cdot \exp(-2\pi s^2 \|\boldsymbol{w}\|^2) = T \cdot \exp(-4\pi^2 r^2 g_d^4/d) = T \cdot \exp(-r^2(d + o(d))/e^2)$, where we used the relation $g_d = \sqrt{d/(2\pi e)}(1 + o(1))$ for large $d$. This sum is then constant for the given expression for $T$. □

For $r = 1$ we need $2^{0.195d+o(d)}$ target vectors to distinguish planted targets from random. This only works when using many targets, as otherwise the dual vectors increase in length and increase the asymptotic complexity. Note that we do not claim to find a closest vector to any of the targets in this amount of time.

We stress that even for $r \gg 1$ we obtain a non-zero distinguishing advantage. This is because even when planting a target at distance $10 \cdot \lambda_1(\mathcal{L})$, the resulting distribution is slightly different from the uniform distribution (modulo the fundamental domain). As expected, we do observe that for large $r$ the distribution becomes almost indistinguishable from random.

**Lemma 9 (Single-target distinguishing, with preprocessing).** *Suppose we are given a single target $\boldsymbol{t}$ from either the random target distribution or the planted target distribution with radius $r \cdot g_d$, with $r > 0$ known. Let $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots \in \mathcal{L}^*$ be the (given) shortest vectors of the dual lattice, and suppose their norms follow the Gaussian heuristic prediction, $\|\boldsymbol{w}_n\| = n^{1/d} g_d(1 + o(1))$. Then heuristically we can distinguish whether $\boldsymbol{t}$ is random or planted in time:*

$$T = \alpha^{d+o(d)}, \qquad (\textit{with } \alpha := \min\{\beta : e^2 \ln \beta = \beta^2 r^2\}.) \tag{60}$$

*Proof.* As before, we wish to use a number of vectors $T$ such that the quantity $T \cdot \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2$ is constant. Using the shortest $T = \alpha^{d+o(d)}$ dual vectors as $\mathcal{W}$, we notice that the above sum can asymptotically be written as:

$$T \cdot \sum_{\boldsymbol{w} \in \mathcal{W}} \rho_{1/s}(\boldsymbol{w})^2 \approx \max_{x \in [1,\alpha]} x^d \cdot \rho_{1/s}(x g_d)^2 = \alpha^d \cdot \rho_{1/s}(\alpha g_d)^2. \tag{61}$$

Since we wish to make $T$ as small as possible, we want to minimize over $\alpha$ such that this quantity is at least constant. Substituting the value $s$ as before, this leads to the given definition of $\alpha$. □

For what approximately corresponds to average-case CVPP, substituting $r = 1$ leads to $\alpha \approx 1.2253$ and $T \approx 2^{0.293d+o(d)}$, where we stress that the constant $0.2931\ldots$ is *not* the same as the $\log_2(3/2)/2 \approx 0.2924\ldots$ exponent of sieving for SVP. For smaller radii the complexities decrease quickly, and similar to the multi-target distinguishing case the complexity scales as $\exp O(r^2 d)$, with the complexity for BDD with preprocessing becoming polynomial when $r = O(\sqrt{(\ln d)/d})$. This is depicted in Figures 1a and 1b.

For finding closest vectors, Algorithm 2 gives a natural heuristic approach based on a gradient ascent. As argued above, the quantity $\langle \nabla f^{(\mathcal{W})}(\boldsymbol{t}), \boldsymbol{t} - \boldsymbol{c} \rangle$ may be of particular interest. This quantity is sufficiently bounded away from 0 for similar parameters as for distinguishing to work, thereby suggesting that similar asymptotics likely hold for searching and distinguishing. After all, if we can use a distinguisher to correctly determine when we are close to the lattice or not, using it as a black box we can take small steps in well-chosen directions starting from $\boldsymbol{t}$ and determine if we are making progress or not. In the next section we will compare experimental results for searching and distinguishing to further strengthen the following claim.

**Lemma 10 (Single-target searching, with preprocessing).** *Heuristically we expect to be able to solve the planted closest vector problem with preprocessing with radii $r \cdot g_d$ for $r < 1$ with similar complexities as Lemma 9.*

Finally, while the above results cover distinguishing planted cases at arbitrary radii, and searching close vectors for radii $r < 1$, this does not give any insights how to solve the search version of approximate CVPP. Experimentally, as well as intuitively, it is unclear if any guarantees can be given on the quality of the output of such a gradient ascent method if the algorithm fails to find the nearest lattice point. This may be inherent to the dual approach, similar to how the poor performance for BDD with preprocessing may be inherent to the primal approximate Voronoi cell approach. Note however that we can always achieve the same complexities for approximate CVPP as for "average-case CVPP" (BDDP with radius approaching $g_d$) as indicated in Figures 1a and 1b). We leave it as an open problem to study if the dual attack can be improved for approximate CVPP, compared to exact CVPP.

### 5.6 Closest vector problems without preprocessing

While the previous results focused on being given the short dual vectors, here we will take into account the costs for generating these dual vectors as well. We will focus on using some form of sieving based block reduction, which covers the most sensible, efficient choices of preprocessing before running the dual attack.

**Lemma 11 (Single-target searching, without preprocessing).** *Heuristically we expect to be able to solve the planted closest vector problem (without preprocessing) with target radius $r \cdot g_d$ for $r = d^\alpha$, $\alpha < 0$, in time $T$ as follows:*

$$T = (3/2)^{(d+o(d))/(2-4\alpha)} \approx 2^{0.292(d+o(d))/(1-2\alpha)}. \tag{62}$$

*Proof.* Let us parameterize the block size $k$ for the initial BKZ preprocessing as $k = \beta d$. For $\beta = 1 - o(1)$, at the same asymptotic cost as BKZ with block size $k$ we can run CVP-style sieving on the entire lattice as described in [Laa16b] to solve average-case CVP and BDD with radius approaching $g_d$. As BDD with radius a constant multiple of $g_d$ requires a block size $d - o(d)$, we will focus on the regime where $r$ scales with $d$, i.e. $r = d^\alpha$ for $\alpha < 0$, so that $\beta \in (0, 1)$.

Running BKZ on the dual lattice with block size $k$ requires time $(3/2)^{d/2+o(d)}$ as outlined in the preliminaries. The last step of a sieving-based BKZ run considers the top block of size $k$, and runs sieving on it to essentially find the $(4/3)^{d/2+o(d)}$ shortest vectors in this block. The Gaussian heuristic in this block equals $g_k^{(d-1)/(k-1)} = g_{\beta d}^{1/\beta+o(1)}$ for constant $\beta$, and the sieving list returns $(4/3)^{d/2+o(d)}$ vectors, most of which have norm $\sqrt{4/3} \cdot g_{\beta d}^{1/\beta+o(1)}$. With these vectors we are therefore able to successfully distinguish in the dual attack, if the following condition is satisfied:

$$(4/3)^{d/2+o(d)} \cdot \rho_{1/s}\left(\sqrt{4/3} \cdot g_{\beta d}^{1/\beta+o(1)}\right) = \Theta(1). \tag{63}$$

As the left hand term is exponential in $d$, for large $d$ this inequality can be satisfied if and only if the leading term in $d$ is constant or increasing with $d$. Gathering exponents, and using crude but sufficiently precise estimates such as $(4/3)^{d/2+o(d)} = \exp O(d)$ and $g_{\beta d}^{1/\beta} = O(d^{1/(2\beta)+o(1)})$, we get:

$$\exp\left[O(d) - O(r^2 d^{1/\beta+o(1)})\right] \geq \Theta(1). \tag{64}$$

If $r^2 d^{2/\beta+o(1)} = o(d)$ the exponent is increasing in $d$ and the inequality is satisfied. If $r^2 d^{2/\beta+o(1)} = \omega(d)$ the exponent is negative for large $d$, and we fail to distinguish. Writing $r^2 = d^{2\ln r/\ln d}$, and taking logarithms, this leads to the condition $\beta \geq \ln d/(\ln d - 2\ln r)$, which for $r = d^\alpha$ translates to $\beta \geq 1/(1-2\alpha)$. Finally, observe that the preprocessing step takes time $(3/2)^{d/2+o(d)}$, and the distinguishing/search phase only takes time $(4/3)^{d/2+o(d)}$. The overall time complexity is therefore equal to the time of the basis reduction step. □

The resulting trade-offs between the radius $r$ and the time complexity $T$ are depicted in Figures 1c and 1d. These figures are similar to the plots for the preprocessing case, except that the x-axes have been stretched out significantly; to reduce the constant in the exponent one needs $r$ to scale polynomially with $d$ (as opposed to $r = O(1)$ for the preprocessing case), and to achieve polynomial time complexity one needs $r = 2^{-O(d)}$ (cf. $r = O(\sqrt{\log d/d})$ for the preprocessing case). Note that the latter regime matches what we intuitively expect; a BDD instance with exponentially small radius can also be solved in polynomial time using LLL basis reduction followed by one of Babai's algorithms.

Finally, similar to the preprocessing case we do not expect the dual approach to work well for solving approximate CVP. Finding a way to give better guarantees for approximate CVP(P), or showing that the dual attack is unable to solve this problem efficiently, is left as an open problem.

## 6 Experiments

### 6.1 Setup

To verify our assumptions and the heuristic estimates, we conducted the following experiments. We used G6K [ADH+19] to compute a large set of short vectors $\mathcal{W}_d$ for the SVP challenge lattice (seed 0) in dimension $d \in \{55, 60, 65, 70, 75, 80\}$ by extracting the vectors in the sieving database after a full sieve on these lattices. To give a sense of the size and the shape of the $\mathcal{W}_d$, we list these parameters in Table 1. Note that these are not the $|\mathcal{W}_d|$ shortest vectors, so our results are likely to be slightly sub-optimal, but, as outlined below, these databases are sufficient to verify our heuristic estimates. We treated the challenge lattices as the dual of the lattices we target to solve BDD for and created samples from the planted target distribution with parameter $r \cdot g_d$ , where we arbitrarily fixed the closest lattice point to be $\mathbf{0}$.

### 6.2 Evaluating the distinguishers

Our first experiment was targeted at verifying the heuristic complexity of the distinguishing algorithms and comparing the three different score functions. For this, we chose 1000 different targets of length $rg_d$ for various $r \in [0.5, 0.9]$. For each of them, we computed the score functions $f_j$ for $j \in \{\text{simple}, \text{AR}, \text{NP}\}$ using the $i$ shortest vectors of $\mathcal{W}_d$. We denote the resulting set of scores as $f_j^i(\mathcal{W}_d, r)$. (For practicality reasons we only computed $f_j^i(\mathcal{W}_d, r)$ for $i$ being multiples of 100.) Additionally, we computed the score functions $\hat{f}_j^i(\mathcal{W}_d, r)$ for targets from the random target distribution, i.e. we chose $\langle \boldsymbol{t}, \boldsymbol{w} \rangle$ uniformly at random from $[-0.5, 0.5]$ for all dual vectors $\boldsymbol{w}$. We now evaluated how well the score functions can distinguish between targets at distance $rg_d$ from the random target distribution. Let $p \in [0, 0.5)$ be a parameter. For each $r$ we computed the minimal $i$ such that $f_j^i(\mathcal{W}_d, r)$ and $\hat{f}_j^i(\mathcal{W}_d, r)$ overlap by at most a $p$-fraction. In other words, we computed the minimal $i$ such that the $p$-percentile of $f_j^i(\mathcal{W}_d, r)$ is larger than the $(1-p)$-percentile of $\hat{f}_j^i(\mathcal{W}_d, r)$. Additionally, we computed two types of predictions for the distinguishing complexity. First, we used the vectors in $\mathcal{W}_d$ to predict the complexity of the $f_{\text{AR}}$ distinguisher by assuming that the result of the score function under the planted and random target distributions was a perfect Gaussian. We call this the data dependent (d.d.) estimate. Second, we also estimated the complexity in a similar way, but by assuming that $\mathcal{W}_d$ contained all the shortest vectors in the dual lattice and that the lengths of these followed the distribution given by the Gaussian heuristic. We denote this by the data independent (d.i.) estimate. The results show that the estimates are very close to the experimental data, which in turn verifies the assumption that the result of the score function can indeed be well approximated by a Gaussian. For $f_{\text{simple}}$ this was independently verified in [EJK20] in a similar setting.

Exemplary results are plotted in Figures 4a and 4b, for varying radius and dimension, respectively, and $p = 0.1$. They represent typical examples from our data sets. We fitted models to the curves to evaluate the dependency of the practical complexity on $r$ ($d$, resp.). Figure 1a suggests that at least for $r$ not too close to 1, the dependency on $r$ should be roughly quadratic. If all dual vectors in the database had length $\|\boldsymbol{w}\| = g_d$, we would expect a curve of the form $\exp((d/e^2)r^2) + O(1)$. However, there are longer vectors in our sets $\mathcal{W}_d$, and the vectors are used in increasing length. This has a progressively negative effect on the complexity, which explains the worse constants computed for the curves in Figure 4a. On the other hand, the curves in Figure 4b adhere very closely to the heuristic value computed in Lemmas 9. The slight variances in the leading constant in the exponent can be easily explained with the small set of data points used to fit the curves. Finally, we see that the score functions all perform similarly well. Since $f_{\text{simple}}$ is the fastest to compute, this will likely give the best results in practice for distinguishing.

### 6.3 Evaluating the search algorithms

In the second part of our experiments, we evaluated the decoding algorithm based on gradient ascent. During our experiments we noticed that $f_{\text{AR}}$ and $f_{\text{simple}}$ perform similarly well, so we arbitrarily choose to report the results for $f_{\text{AR}}$. We chose the step size such that the update corresponds to (22) as a sensible first guess. Otherwise, the methodology was similar to the previous set of experiments. We only performed one step of the gradient ascent algorithm and declared success if it resulted in a target closer to the lattice than the initial target. Intuitively, since after a successful first step the problem becomes easier, this should be a good

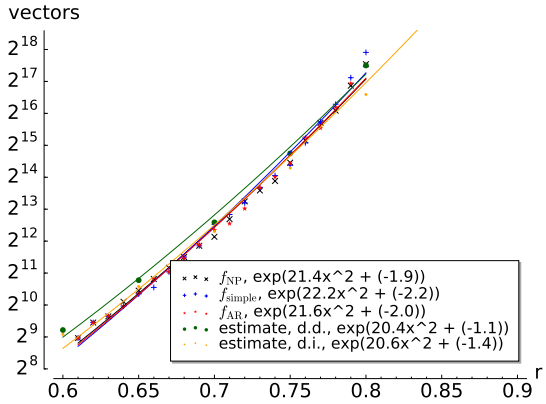
| Dimension | 55 | 60 | 65 | 70 | 75 | 80 |
|---|---|---|---|---|---|---|
| $\lvert\mathcal{W}_d\rvert$ | 8729 | 17918 | 36783 | 75510 | 155007 | 318199 |
| $\min \lVert\boldsymbol{w}\rVert/g_d$ | 1.0815 | 1.0147 | 1.0575 | 1.0377 | 1.0527 | 1.0306 |
| $\max \lVert\boldsymbol{w}\rVert/g_d$ | 1.3161 | 1.2945 | 1.2888 | 1.2812 | 1.2748 | 1.2632 |

**Table 1.** Parameters of the set of short vectors used in experiments.

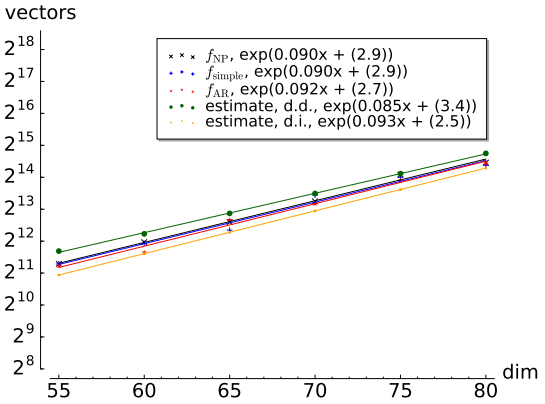
proxy for the success probability of the full algorithm. Exemplary results are given in Figures 4c and 4d again depending on $r$ and the dimension $d$, respectively. The results are similar to the distinguishing case, with close adherence to the heuristic estimate of Lemma 9 w.r.t. the dependence on $d$. As expected, the success probability only affects constant factors in the complexity.

In the previous experiments, we only considered the gradient ascent method with a fixed step size given by the analysis in Section 4.6. Since we only compute an approximation of the true step, it stands to reason that in some cases a poor approximation may lead to failure of the algorithm, even though the computed update is roughly pointing in the right direction. One may hope to reduce the number of such cases by reducing the step size. This should give a trade-off between the success probability of the algorithm and the running time. In order to evaluate the potential of this approach, we repeated the above experiments, but we now deemed any trial successful where the first update $\boldsymbol{u}$ satisfied $\langle \boldsymbol{u}, \boldsymbol{e} \rangle < 0$, where $\boldsymbol{e}$ is the error vector of the target. If this is satisfied, this implies that there is a step size such that the update succeeds in reducing the distance to the lattice. The results are shown in Figures 4e and 4f, analogously to the ones above. They show that the results only differ in small constant factors ($\lesssim e$). In practice, it might be worth exploring this trade-off, but we leave this for future work.

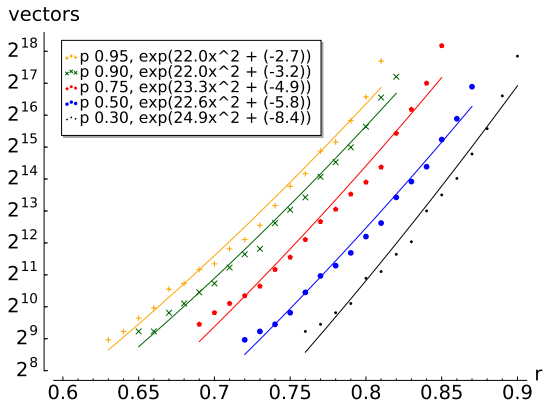
To give a sense of how many steps the search algorithm needs, we ran another set of experiments: for various radii and number of vectors from $\mathcal{W}_{80}$ we ran the full algorithm until we had successfully decoded 100 targets in the 80-dimensional lattice. For the successful invocations we recorded the number of required steps. The results are given in Figure 7. There is a small caveat regarding the implementation of our algorithm. At least after performing the first update, it is in general not clear how close the new target is to the lattice, which impacts the optimal choice for parameter $s$, which should be chosen for the next step (see Lemma 6). We exploited the fact that we know the closest lattice vector to recompute this $s$ for the subsequent steps. So the concrete number of steps we report might be somewhat optimistic. In practice, one could use the simple decoder to estimate the distance to the lattice, which we expect not to impact the running time significantly.



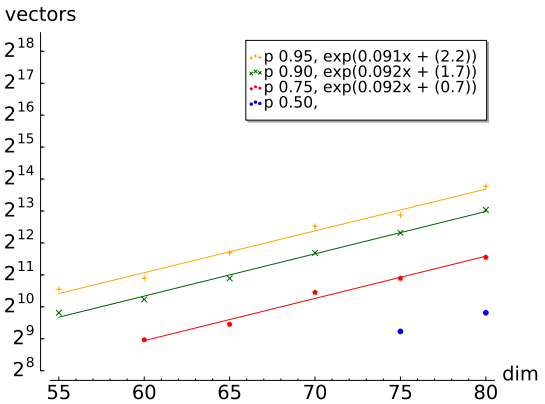

(a) Complexity of distinguishing the planted target distribution with radius $r \cdot g_d$ from the random target distribution with success probability 0.90 in dimension 80.



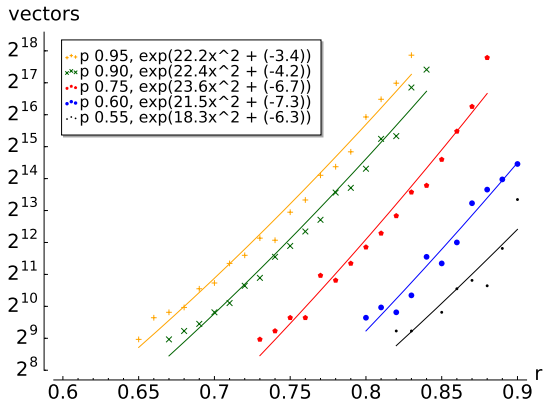

(b) Complexity of distinguishing a planted target with radius $0.75 \cdot g_d$ from a random target with success probability 0.90. The theoretical estimate from Lemma 9 is $\exp(0.0913d + o(d))$.



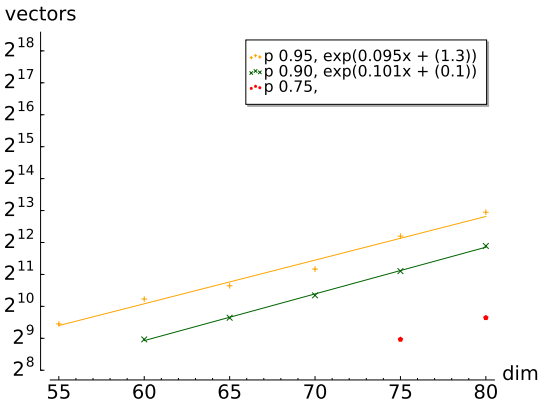

(c) Complexity of decoding a target at distance $r \cdot g_d$ with success probability $p$ in dimension 80.




(d) Complexity of decoding a target with radius $0.75 \cdot g_d$ with success probability $p$. Lemma 10 predicts $\exp(0.0913d + o(d))$.

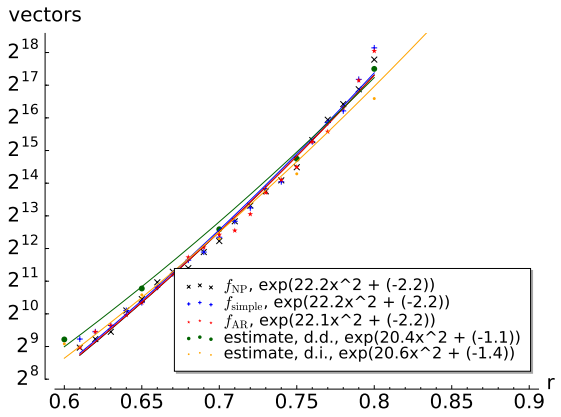



(e) "Lower bound" for complexity of decoding a target with radius $r \cdot g_d$ in dimension 80 with success probability $p$.

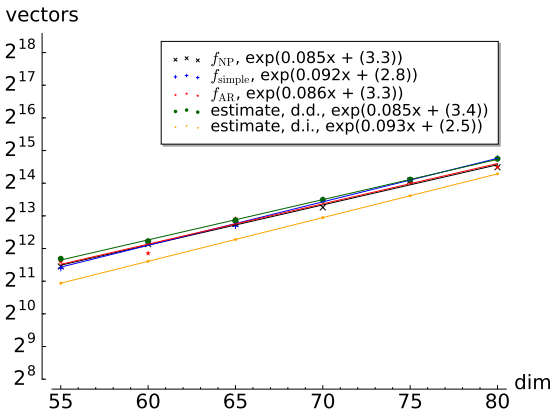



(f) "Lower bound" for complexity of decoding a target with radius $0.75 \cdot g_d$ with success probability $p$.

**Fig. 4.** Exemplary results, for distinguishing (a,b), decoding (c,d), and a "lower bound" (e,f). Left (a,c,e): $d = 80$ fixed, varying $r$. Right (b,d,f): $r = 0.75$ fixed, varying $d$.

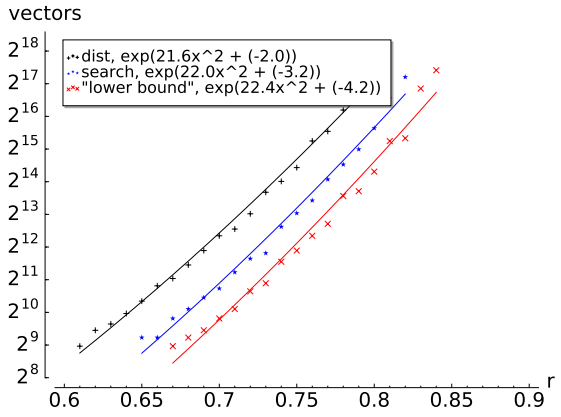



(a) Complexity of distinguishing the planted target distribution with radius $r \cdot g_d$ from the planted target distribution with radius $g_d$ with success probability 0.90 in dimension 80.

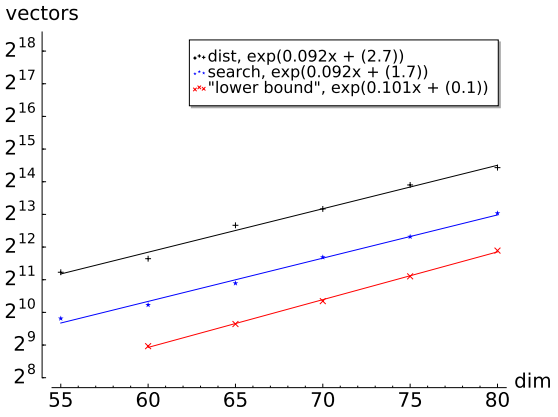



(b) Complexity of distinguishing the planted target distribution with radius $0.75 \cdot g_d$ from the planted target distribution with radius $g_d$ with success probability 0.90.

**Fig. 5.** Similar to Figures 4a and 4b, but distinguishing the planted target distribution of radius $r \cdot g_d$ from another planted target distribution with radius $g_d$, instead of from the random target distribution.

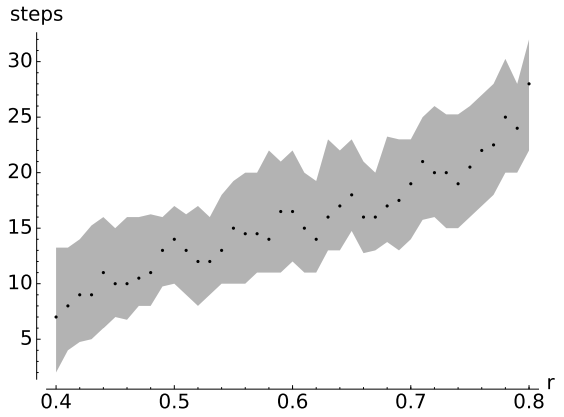



(a) Aggregation of the plots in Figure 4a, 4c and 4e, where we chose $f_{\mathrm{AR}}$ for the first and $p = 0.9$ for the latter two.

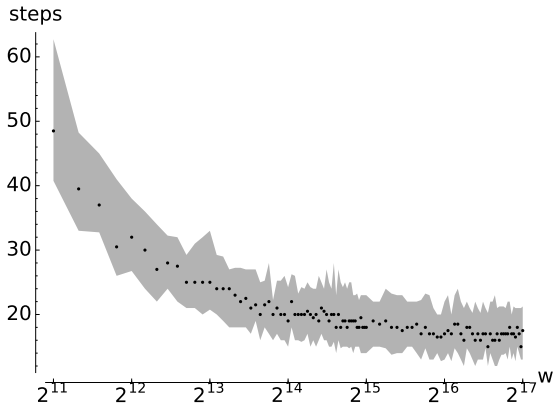



(b) Aggregation of the plots in Figure 4b, 4d and 4f, where we chose $f_{\mathrm{AR}}$ for the first and $p = 0.9$ for the latter two.

**Fig. 6.** Aggregation of the plots of Figure 4(vertically).




(a) Steps required to decode target at radius $r \cdot g_d$ using $2^{14}$ vectors.




(b) Steps required to decode target at radius $0.75 \cdot g_d$ using $w$ vectors.

**Fig. 7.** Median number of steps required to decode target in 80 dimensional lattice. Gray area marks the 25- and 75-percentile.

## References


ADH+19. Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *EUROCRYPT*, pages 717–746, 2019.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In *USENIX Security Symposium*, pages 327–343, 2016.

AGVW17. Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In *ASIACRYPT*, pages 297–322, 2017.

AKS01. Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

AN17. Yoshinori Aono and Phong Q. Nguyên. Random sampling revisited: lattice enumeration with discrete pruning. In *EUROCRYPT*, pages 65–102, 2017.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, pages 1–38, 2015.

AR04. Dorit Aharonov and Oded Regev. Lattice problems in $\mathsf{NP} \cap \mathsf{coNP}$. In *FOCS*, pages 362–371, 2004.

AS72. Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Formulas.* Dover Publications, 1972.

Bab86. László Babai. On lovasz lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

BBD09. Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-quantum cryptography.* Springer, 2009.

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24, 2016.

BKV19. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. *Cryptology ePrint Archive, Report 2019/498*, 2019.

DLdW19. Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate Voronoi cells. In *PQCrypto*, pages 3–22, 2019.

DLdW20. Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Sieve, enumerate, slice, and lift: Hybrid lattice algorithms for SVP via CVPP. In *Africacrypt*, pages 301–320, 2020.

DLvW20. Leo Ducas, Thijs Laarhoven, and Wessel van Woerden. The randomized slicer for CVPP: sharper, faster, smaller, batchier. In *PKC*, pages 3–36, 2020.

DRS14. Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *CCC*, pages 98–109, 2014.

DSvW21. Léo Ducas, Marc Stevens, and Wessel van Woerden. Advanced lattice sieving on gpus, with tensor cores. Cryptology ePrint Archive, Report 2021/141, 2021. `https://eprint.iacr.org/2021/141`.

Duc18. Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. In *EUROCRYPT*, pages 125–145, 2018.

EJK20. Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a hybrid approach to solve small secret lwe. Cryptology ePrint Archive, Report 2020/515, 2020. `https://eprint.iacr.org/2020/515`.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

GN08. Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *STOC*, pages 207–216. ACM, 2008.

HKM18. Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving LWE. *Designs, Codes and Cryptography*, 86:55–83, 2018.

Kan83. Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206, 1983.

Kan87. Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operation research*, 12(3):415–440, August 1987.

Laa16a. Thijs Laarhoven. *Search problems in cryptography.* PhD thesis, Eindhoven University of Technology, 2016.

Laa16b. Thijs Laarhoven. Sieving for closest lattice vectors (with preprocessing). In *SAC*, pages 523–542, 2016.

Laa20. Thijs Laarhoven. Approximate Voronoi cells for lattices, revisited. *Journal of Mathematical Cryptology*, 15(1):60–71, 2020.

LLL82. Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

LLM06. Yi-Kai Liu, Vadim Lyubashevsky, and Daniele Micciancio. On bounded distance decoding for general lattices. In *RANDOM*, pages 450–461, 2006.

LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, pages 319–339, 2011.

MV10. Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010.

MW15. Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *SODA*, pages 276–294, 2015.

MW16. Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *EUROCRYPT*, pages 820–849, 2016.

NP33. Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Phil. Trans. R. Soc. Lond. A*, 231(694-706):289–337, 1933.

oSN17. The National Institute of Standards and Technology (NIST). Post-quantum cryptography, 2017.

PMHS19. Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In *EUROCRYPT*, pages 685–716, 2019.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

Reg10. Oded Regev. The learning with errors problem (invited survey). In *CCC*, pages 191–204, 2010.

Sch87. Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, 1987.

SE94. Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(2–3):181–199, 1994.

Sho94. Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *FOCS*, pages 124–134, 1994.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, pages 617–635, 2009.

svp20. SVP challenge, 2020. `http://latticechallenge.org/svp-challenge/`.