Lower bounds on lattice sieving and information set decoding

Elena Kirshanova[1,2], Thijs Laarhoven[3]

[1] Immanuel Kant Baltic Federal University, Kaliningrad, Russia
[2] Horst Görtz Institute for IT-Security, Ruhr University Bochum
elenakirshanova@gmail.com
[3] Eindhoven University of Technology, Eindhoven, The Netherlands
mail@thijs.com

Abstract. In two of the main areas of post-quantum cryptography, based on lattices and codes, nearest neighbor techniques have been used to speed up state-of-the-art cryptanalytic algorithms, and to obtain the lowest asymptotic cost estimates to date [May–Ozerov, Eurocrypt'15; Becker–Ducas–Gama–Laarhoven, SODA'16]. These upper bounds are useful for assessing the security of cryptosystems against known attacks, but to guarantee long-term security one would like to have closely matching lower bounds, showing that improvements on the algorithmic side will not drastically reduce the security in the future. As existing lower bounds from the nearest neighbor literature do not apply to the nearest neighbor problems appearing in this context, one might wonder whether further speedups to these cryptanalytic algorithms can still be found by only improving the nearest neighbor subroutines.

We derive new lower bounds on the costs of solving the nearest neighbor search problems appearing in these cryptanalytic settings. For the Euclidean metric we show that for random data sets on the sphere, the locality-sensitive filtering approach of [Becker–Ducas–Gama–Laarhoven, SODA 2016] using spherical caps is optimal, and hence within a broad class of lattice sieving algorithms covering almost all approaches to date, their asymptotic time complexity of $2^{0.292d+o(d)}$ is optimal. Similar conditional optimality results apply to lattice sieving variants, such as the $2^{0.265d+o(d)}$ complexity for quantum sieving [Laarhoven, PhD thesis 2016] and previously derived complexity estimates for tuple sieving [Herold–Kirshanova–Laarhoven, PKC 2018]. For the Hamming metric we derive new lower bounds for nearest neighbor searching which almost match the best upper bounds from the literature [May–Ozerov, Eurocrypt 2015]. As a consequence we derive conditional lower bounds on decoding attacks, showing that also here one should search for improvements elsewhere to significantly undermine security estimates from the literature.

Keywords: nearest neighbor searching · locality-sensitive hashing · lattice sieving · information set decoding · lower bounds

1 Introduction

Post-quantum cryptography. After Shor's breakthrough work in the 90s [Sho94], showing that current solutions in public-key cryptography are vulnerable to quantum attacks, many researchers have shifted their attention towards developing new, quantum-safe alternatives. Within the field of post-quantum cryptography, arguably two subfields stand out: *lattice-based cryptography*, offering efficient, small, and versatile solutions [Reg05,Reg10,Gen09,GGH13] and relatively strong security guarantees [AD97,MR07,SSTX09,LPR10]; and *code-based cryptography*, relying on long-studied problems from coding theory, dating back as far as RSA [McE78,RSA78], and having remained unbroken ever since [Lan20]. In both these fields, it is crucial to obtain a good understanding of the true hardness of the underlying hard problems; both by trying to find new techniques that may lead to faster algorithms, and by studying what are the limits of known algorithms, when using algorithmic techniques we are currently aware of.

Hardness estimates for lattices. In the field of lattice-based cryptography, currently the fastest known approach for solving hard lattice problems is commonly referred to as *lattice sieving*. Theoretically, the fastest sieving algorithms for solving e.g. the shortest vector problem (SVP) on random d-dimensional lattices run in time $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$ [BDGL16] under plausible heuristic assumptions about random lattices[4]In practice all recent record-breaking computations on random lattices were done with sieving as well [svp20,ADH+19]. Accurately estimating the true cost of lattice sieving is therefore essential for choosing parameters for lattice-based cryptographic primitives. As the constant $\frac{1}{2}\log_2(\frac{3}{2}) \approx 0.292$ in the exponent has not been improved for several years now (with many improvements happening between 2008 and 2016), one might wonder whether this constant is optimal, and if one can confidently use it as an asymptotic lower bound on the cost of any algorithm trying to break the underlying lattice problem.

Hardness estimates for decoding. In the context of code-based cryptography, the most important algorithms to solve the problem of decoding random binary codes are information set decoding (ISD) algorithms. A random binary code of length d asymptotically has a minimum distance λ of the order $\lambda = \Theta(d)$.[5] In this regime all known ISD algorithms have a single-exponential running time $2^{\mathsf{c}d+o(d)}$, where the constant c has been improved over the last 60 years from $\mathsf{c} = 0.121$ [Pra62] through a series of works [Ste89,MMT11,BJMM12,MO15] to the current best leading constant $\mathsf{c} = 0.0885$ [BM18]. These runtimes hold for

[4] The literature on lattice algorithms is divided into two classes: algorithms with provable guarantees on the worst-case complexity for any input lattice [PS09,MV10a,ADRS15]; and algorithms making some heuristic assumptions about the "behavior" of random lattices, to obtain tighter average-case complexity estimates [NV08,GNR10,MV10b,Laa15a,ANSS18].

[5] We choose d to denote the *length* of the code rather than its minimum distance here, to be consistent with lattice and near neighbor literature.

average-case instances and are provable. The recent improvements in ISD come from a combination of various techniques, so it is important to pin down which techniques are already optimal and which should be further explored to see if the current best result from [BM18] can be improved upon.

Note that in this paper, we do not consider the so-called *sparse* error regime in decoding, i.e., when the error weight is promised to be $o(d)$. The aforementioned improvements for ISD do not hold in this regime, and the asymptotically fastest known algorithm for the sparse case is due to Prange's [Pra62].

Lower bounds for cryptanalytic algorithms. Both in the context of lattice algorithms and decoding random binary codes, most work has focused on upper bounds, i.e. constructing algorithms solving these problems as efficiently as possible. However, for applications in cryptography we are equally interested in (tight) lower bounds, stating that any attacker that tries to break the scheme by solving these underlying hard problems needs to spend at least this amount of time to find a solution. Any such lower bound would clearly be conditional on the approach used to solve the problem, but even such conditional lower bounds may already be valuable for choosing parameters in a more conservative manner than optimistically assuming that the current best algorithms are still the best algorithms an attacker can use in 20 years. Unfortunately not much is known about lower bounds in either area, with e.g. [ANSS18] obtaining lower bounds on lattice enumeration.

Nearest neighbor subroutines. Both in lattice sieving and in decoding, an important subroutine in the state-of-the-art algorithms for solving these problems is to solve a nearest neighbor problem in the ℓ_1 and ℓ_2-norms: given a large database of uniformly random vectors, store it in a convenient data structure such that, when given a random query vector, we can efficiently extract nearby vectors (under the corresponding metric) from the database. These relations were explicitly established in [Laa15a,MO15], and especially in lattice sieving many subsequent improvements were directly related to only improving the nearest neighbor subroutine [BGJ15,LdW15,BL16,BDGL16]. As a first step towards finding tight lower bounds on the overall decoding algorithms, we aim at obtaining lower bounds on the nearest neighbor subroutines, so that we can rule out further improvements which only target the nearest neighbor routine.

Nearest neighbor lower bounds. For the applications of interest in this paper (lattice sieving and decoding algorithms), the nearest neighbor methods that have worked best to date are hashing–based solutions, for which lower bounds have previously been studied in e.g. [MNP07,OWZ14,Chr17]. These lower bounds were mostly in a slightly different model than the models which naturally appear in cryptanalysis, and it is therefore unclear whether similar lower bounds apply in the context of cryptography, and whether the best nearest neighbor methods in these other models must also translate to the best methods for the problems of interest in cryptography.

On the strict inequivalence between different models. For the last question, we can explicitly derive a counterexample, showing that a method which is asymptotically optimal in one setting is not necessarily optimal in the other. Namely, for the often-considered *sparse regime*, cross-polytope hashing is known to be asymptotically optimal [TT07,AIL+15], but when applied to lattice sieving it leads to a suboptimal time complexity of $2^{0.298d+o(d)}$, compared to the $2^{0.292d+o(d)}$ obtained via the spherical filters of [BDGL16]. In other words: optimal solutions in other models may be suboptimal in our model, and lower bounds may not carry over to our setting either.

1.1 Contributions

After covering the preliminaries (Section 2), and explicitly describing the nearest neighbor search model considered in this paper and how it differs from other models commonly considered in the nearest neighbor literature (Section 3), our main contributions are covered in Sections 4–7:

Nearest neighbor searching on the Euclidean sphere (Section 4). For the problem of finding nearest neighbors in data sets uniformly distributed on the sphere, we prove that the best partitioning and filtering approaches – main subroutines in the hash-based Near neighbor searching – must necessarily be based on spherical caps. This shows that the spherical filters introduced in [BDGL16] and further analyzed in [ALRW17,Chr17] are optimal not only in the sparse regime, but also in the dense regime. Note that this result is even stronger than previous optimality results [AINR14,AIL+15,ALRW17], as there are no hidden order terms in the statement that spherical caps are optimal for shaping hash regions.

Application to lattice sieving and lattice-based cryptography (Section 5). As a direct application of the above result, we prove that within the framework of running a "pairwise" lattice sieve with some form of hash-based nearest neighbor search (a technique inside the sieves is described in e.g., [NV08,Laa15a,BDGL16]), the lattice sieve of Becker–Ducas–Gama–Laarhoven [BDGL16] is optimal, and the associated asymptotic time complexity $2^{0.292d+o(d)}$ is the best possible. Similar optimality results extend to the tuple sieving results of Herold–Kirshanova–Laarhoven [HKL18], the pairwise sieve with quantum speedups [Laa16], and applications to closest vector problems [DLvW20].

Nearest neighbor searching for the Hamming distance (Section 6). Moving from ℓ_2 to ℓ_1 norm, we show that spherical caps in Hamming space are optimal in the sparse regime and almost match the lower bound in the dense regime. We point to the source of the small discrepancy between our lower bound and what is achievable by spherical caps.

Application to decoding and code-based cryptography (Section 7). Similar to lattices, our lower bound for nearest neighbor searching on the Hamming cube

suggests that trying to improve *only* the nearest neighbor subroutine in information set decoding algorithms will not result in a noticeable asymptotic gain. For example, trying to replace a random code, which is used to construct spherical caps, with another code will not improve the overall algorithm.

However, the situation differs from lattices in the fact that near neighbor search is not necessarily the dominant subroutine and its complexity can be rebalanced with other combinatorial steps. This way, Both-May [BM18] were able to improve over [MO15] using the near neighbor routine differently. Thus one should interpret our lower bound as an indication that any faster algorithm for decoding will necessarily require a novel ideal of how (if at all) use near neighbor search.

2 Preliminaries

2.1 Notation

We write (M, d) for a metric space, where M is the underlying set and $d : M \times M \to \mathbb{R}$ is the distance function (metric) associated to this set. We write $\mathbb{1}\{E\}$ for the indicator function, which is 1 if event E holds and 0 otherwise. For random variables X sampled uniformly from a set S, we may write $X \sim S$. We denote vectors (lowercase) and matrices (uppercase) in boldface. We write $\|\cdot\|_p$ for the ℓ_p-norm, and in this work we will be using both the ℓ_1 and ℓ_2-norms. Throughout, d will always refer to the dimension of the space.

We denote the Euclidean sphere in d dimensions by $\mathcal{S}^{d-1} = \{\mathbf{z} \in \mathbb{R}^d : \|\mathbf{z}\|_2 = 1\} \subset \mathbb{R}^d$. On this sphere we will make use of the uniform surface measure σ which is normalized such that $\sigma(\mathcal{S}^{d-1}) = 1$. We write $\langle \cdot, \cdot \rangle$ for standard dot products.

We denote the Hamming cube in d dimensions by $\{0,1\}^d$. We define the binary entropy function for $x \in [0,1]$ as $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$. For asymptotic results on the Hamming cube, we shall be using the approximation for the binomial coefficient $\binom{d}{\alpha d} \approx 2^{H(\alpha)d}$ which holds for constant $\alpha \in (0,1)$ and large d.

2.2 Lattices

A full-rank lattice $\mathcal{L}(\mathbf{B})$ is a discrete additive subgroup of $\mathbb{R}^d$ generated by the columns of a matrix $\mathbf{B} \in \mathbb{R}^{d \times m}$ (with polynomially-sized entries). Various hard lattice problems have been studied over time, with the shortest and closest vector problems being the classical hard problems. We state the shortest vector problem below, as efficient algorithms for this (exact) problem are often a key ingredient for the best cryptanalytic attacks for lattice-based cryptosystems. For simplicity, one may assume that the rank m below is equal to d.

Definition 1 (The shortest lattice vector problem). *Let d, m be positive integers, and suppose we are given a basis $\mathbf{B} \in \mathbb{R}^{d \times m}$ generating a lattice $\mathcal{L} = \{\mathbf{Bz} : \mathbf{z} \in \mathbb{Z}^m\} \subset \mathbb{R}^d$. Find a vector $\mathbf{s} \in \mathcal{L}$ satisfying $\|\mathbf{s}\|_2 = \min_{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{v}\|_2$.*

We express complexities for algorithms for solving lattice problems in terms of their main security parameter d, i.e. in the form $2^{\mathsf{c}d + o(d)}$ for a constant c.

2.3 Codes

We refer to a binary linear code $\mathcal{C}$ as a $[d, k, \lambda]$-code, with d being the dimension, k the rank of the code, and λ the minimum distance. While the shortest lattice vector problem is one of the central hard problems on lattices, upon which the security of lattice-based cryptography relies, the following problem is crucial in understanding the security of code-based cryptosystems.

Definition 2 (The information set decoding problem). *Let d, k, λ be positive integers, and suppose we are given a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(d-k)\times d}$ and a syndrome vector $\mathbf{s} \in \mathbb{F}_2^{d-k}$ satisfying $\mathbf{s} = \mathbf{He}$ for some $\mathbf{e} \in \mathbb{F}_2^d$ with Hamming weight $w := \|\mathbf{e}\|_1 \leq \lambda$. Find the error vector $\mathbf{e}$.*

In the analysis of information set decoding algorithms, it is common to relate the parameter w (the error weight) to the rank of the code k and to the dimension d. To do so, we make use of the Gilbert–Varshamov bound which states that $\frac{k}{d} = 1 - H\left(\frac{w}{d}\right)$ as $d \to \infty$. This gives us a way to express w as a function of d and k. Then for any chosen $k \in (0, 1)$, the runtime of an information set decoding algorithm simplifies to the form $2^{\mathsf{c}d+o(d)}$ for some constant c. We are interested in the setting when $w = \Theta(d)$, the so-called *dense* regime.

3 Nearest neighbor model

3.1 Closest pairs problem

For the applications in post-quantum cryptanalysis, which are ultimately the main objective of this study, we are commonly interested in solving the following general closest pairs problem: finding nearby pairs of vectors in a given list of vectors living in some bounded metric space.

Definition 3 (Closest pairs problem). *Let (M, d) be a bounded metric space, and let $r \geq 0$ be a given target distance. Let $L \subset M$ be a finite subset of M, with elements drawn uniformly at random from M. Find almost all*[6] *pairs $\mathbf{x}, \mathbf{y} \in L$ satisfying $d(\mathbf{x}, \mathbf{y}) \leq r$.*

In the above definition, we assume the list L follows a uniform distribution over the underlying metric space M; in the applications for the Euclidean sphere and Hamming cube it will be clear what this uniform distribution looks like. This is different from various other models in the nearest neighbor literature, where

[6] The term "almost all" can intuitively be interpreted as finding at least 90% of all such pairs (or, if only one such pair exists, making sure it is found with probability at least 0.90). Although this minimum success rate is not a hard limit, and the high-level ideas would still work if only e.g. 50% or 10% of all pairs are found, the complexities of these underlying algorithms are usually inversely proportional to the ratio of good pairs that are found in the closest pairs subroutine: finding a smaller ratio of good pairs commonly means having to use bigger lists, which in turn translates to a higher space complexity and a higher overall runtime due to having to search bigger lists.

one might aim to find a solution to the closest pairs problem which works even for worst-case data sets, albeit with a certain approximation factor. In cryptanalytic applications, these uniform distributions appear naturally, and average-case analyses give a better idea of the overall performance than worst-case analyses.

A common approach for solving variants of the closest pairs problem is by first building, and then repeatedly querying a well-chosen nearest neighbor data structure:

1. Initialize a nearest neighbor data structure $\mathcal{D}$;
2. Populate this data structure $\mathcal{D}$ with all elements $\mathbf{x} \in L$;
3. For each $\mathbf{x} \in L$, query the data structure $\mathcal{D}$ to find nearby $\mathbf{y} \in L$, $\mathbf{x} \neq \mathbf{y}$, with $d(\mathbf{x}, \mathbf{y}) \leq r$.

Note that within this framework, we need to index the list L in the data structure $\mathcal{D}$ (corresponding to $|L|$ *insertions*), and we need to run $|L|$ queries on the list L to find almost all closest pairs (corresponding to $|L|$ *queries*). While there is often a trade-off between the insertion and query complexities for such nearest neighbor data structures, this outline naturally tells us that to optimize the overall time complexity for solving the closest pairs problem, we should balance the insertion and query complexities. If insertions and queries can both be done in time $|L|^{\rho+o(1)}$ for some $\rho \in (0, 1)$, then the above algorithm would solve the closest pairs problem in time and memory $|L|^{1+\rho+o(1)}$. There exists memory-efficient version of the above approach that uses only $|L|^{1+o(1)}$ memory [BGJ15,BDGL16] that consists in building $\mathcal{D}$ "on-the-fly". Each bucket is processed once constructed and is never stored. Such a modification improves memory but not the runtime of the algorithm, hence, the lower bound we obtain applies.

3.2 Nearest neighbor problem

As outlined above, the problem of finding all close pairs in a long list can be solved via the nearest neighbor problem.

Definition 4 (Nearest neighbor problem). *Let (M, d) be a bounded metric space, and let $r \geq 0$ be a given target distance. Let $L \subset M$ be a finite subset of M, with elements drawn uniformly at random from M. Preprocess L in a data structure such that, when later given a uniformly random query $\mathbf{x} \in M$, we can efficiently find almost all vectors $\mathbf{y} \in L$ satisfying $d(\mathbf{x}, \mathbf{y}) \leq r$.*

Similar to the closest pairs problem, we assume that the data set is drawn uniformly at random from the space M, which we therefore assume is bounded. We also assume that the query vector $\mathbf{x} \in M$ is drawn uniformly at random from M, which closely matches the nearest neighbor subroutine that needs to be solved to solve the closest pairs problem defined earlier.

3.3 Hash-based nearest neighbor searching

While many solutions have been proposed for solving such nearest neighbor problems, the most promising approaches for high-dimensional problem instances all

Algorithm 3.1 Hash-based nearest neighbor searching

SCHEME PARAMETERS:
- $t \in \mathbb{N}$ — the number of hash regions
- $r \in \mathbb{R}$ — target distance
- $U_1, \ldots, U_t \subset M$ — hash regions for insertions
- $Q_1, \ldots, Q_t \subset M$ — hash regions for queries

function INSERT($\mathbf{y}$) ▷ Add $\mathbf{y}$ to all relevant buckets
  for all $i \in [t]$ **with** $\mathbf{y} \in U_i$ **do**
    $B_i \leftarrow B_i \cup \{\mathbf{y}\}$

function QUERY($\mathbf{x}$) ▷ Find near neighbors $\mathbf{y} \in L$ with $d(\mathbf{x}, \mathbf{y}) \leq r$
  $C \leftarrow \varnothing$
  for all $i \in [t]$ **with** $\mathbf{x} \in Q_i$ **do**
    for all $\mathbf{y} \in B_i$ **with** $d(\mathbf{x}, \mathbf{y}) \leq r$ **do**
      $C \leftarrow C \cup \{\mathbf{y}\}$
  return C

function PREPROCESS(L) ▷ Store all $\mathbf{y} \in L$ in the data structure
  $B_1, \ldots, B_t \leftarrow \varnothing$
  for all $\mathbf{y} \in L$ **do**
    INSERT($\mathbf{y}$)

function CLOSESTPAIRS(L) ▷ Find close pairs $\{\mathbf{x}, \mathbf{y}\} \in L$ with $d(\mathbf{x}, \mathbf{y}) \leq r$
  PREPROCESS(L)
  $P \leftarrow \varnothing$
  for all $\mathbf{x} \in L$ **do**
    $P \leftarrow P \cup (\{\mathbf{x}\} \times \text{QUERY}(\mathbf{x}))$
  return P

seem to be based around the idea of (randomized) *divide and conquer*: divide the space in regions, and solve the closest pairs problem (nearest neighbor problem) in each region separately. By using well-chosen *hash regions*, and by using many rerandomizations to account for unfortunate separations of nearby vectors, we hope that each pair of nearby vectors will eventually end up in the same hash region at least once.

Formally, with the added generalization that combinations of these hash regions do not necessarily have to form a partition of the space [BDGL16,ALRW17], this leads to the following definition of hash-based nearest neighbor searching.

Definition 5 (Hash-based nearest neighbor searching). *Let the data set $L \subset M$ and target radius $r > 0$ be given. To solve the nearest neighbor problem, hash-based nearest neighbor searching preprocesses the data set L and processes queries $\mathbf{x}$ as outlined in Algorithm 3.1.*

Observe that the pseudocode in Algorithm 3.1 is not quite precise on how we recover the indices $i \in [t]$ with either $\mathbf{y} \in U_i$ (for insertions) or $\mathbf{x} \in Q_i$ (for queries). A naive linear search would take time t, by checking for each i if the condition is satisfied. If there is some additional structure in these hash regions

U_i and Q_i, then ideally we may hope for an algorithm finding the set $Y = \{i \in [t] : \mathbf{y} \in U_i\}$ in time $O(|Y|)$, and the set $X = \{i \in [t] : \mathbf{x} \in Q_i\}$ in time $O(|X|)$. Throughout we will often assume the existence of an oracle $\mathcal{O}$ which achieves these optimal time complexities, as the technicalities for implementing this (as in e.g. [BDGL16,ALRW17]) are not necessary for understanding our results, and may distract the reader from the essence of our contributions.

At the end of the query phase, we search the set of candidates $C = \cup_{i:\mathbf{x}\in Q_i} B_i$ for potential nearest neighbors to $\mathbf{x}$. Ideally we would like this set C to only contain nearby vectors in the data set, and not any other vectors. In other words, ideally we would like to guarantee that for random vectors $\mathbf{y} \in L$ the event $\{\mathbf{x} \in Q_i, \mathbf{y} \in U_i\}$ is rare, while for nearby vectors $\mathbf{y} \in L$ the probability of $\{\mathbf{x} \in Q_i, \mathbf{y} \in U_i\}$ happening is large. Therefore, the following quantities are of interest, which capture the probabilities of hash collisions for nearby and random vectors.

Definition 6 (Collision probabilities). *Given a hash-based nearest neighbor scheme, with hash regions $U_1, \ldots, U_t$ and $Q_1, \ldots, Q_t$, and a target distance $r > 0$, we define the following quantities:*

$$p_1 := \sum_{i=1}^{t} p_{1,i}, \qquad p_{1,i} := \Pr_{\mathbf{x},\mathbf{y}\sim M}(\mathbf{x} \in Q_i, \mathbf{y} \in U_i \mid d(\mathbf{x},\mathbf{y}) \leq r), \tag{1}$$

$$p_2 := \sum_{i=1}^{t} p_{2,i}, \qquad p_{2,i} := \Pr_{\mathbf{x},\mathbf{y}\sim M}(\mathbf{x} \in Q_i, \mathbf{y} \in U_i). \tag{2}$$

To obtain the best performance for a hash-based scheme, we wish to maximize p_1 and minimize p_2. An often considered quantity capturing both these goals is $\rho := \ln p_1 / \ln p_2$. Maximizing p_1 and minimizing p_2 means making the exponent ρ as small as possible, and when the parameters of the scheme are chosen to balance insertion and query costs (and one assumes the existence of an efficient oracle for finding relevant buckets), both these costs can be made equal to $\tilde{O}(n^\rho)$. In general however one can obtain arbitrary trade-offs between the costs of this approach, as described in e.g. [Laa15b,BDGL16,ALRW17]. The shapes of the hash buckets may vary, but intuitively the relative sizes of Q_i and U_i control the trade-off between the query time on the one hand, and the insertion time, preprocessing time, and memory complexity on the other hand as follows:

- For $Q_i \subset U_i$, we are more selective with buckets in the query phase, often leading to better query times but worse insertion and preprocessing complexities, as we will need more buckets to guarantee we still find the nearest neighbors in the few buckets we query for near neighbors.
- For $Q_i \supset U_i$, we are less selective in the query phase, and overall we need a smaller number of buckets t (less memory, better preprocessing time) to make sure we find the nearest neighbor in one of the queried buckets. However, as we also consider "bad quality" hash buckets, we will commonly spend more time in the query phase. (Choosing $Q_i \supset U_i$ is intuitively similar to *probing* in locality-sensitive hashing literature [Pan06,AIL+15].)

– For $Q_i = U_i$, we balance the query and insertion complexities. This is sometimes called the balanced regime, and most lower bounds from the literature on ρ apply to this regime.

Usually it does not make sense to choose regions U_i and Q_i for which neither $U_i \subseteq Q_i$ nor $Q_i \subseteq U_i$; we want $\mathbf{x}$ and $\mathbf{y}$ to be as similar as possible, so if we know $\mathbf{y} \in U_i$ we will want to compare $\mathbf{x}$ to $\mathbf{y}$ only if $\mathbf{x}$ lies in a similar region in space.

3.4 Assumptions about the data set

While most of the above model is still very much in line with most of the existing (hash-based) nearest neighbor literature, and lower bounds that have previously appeared, there are some subtle differences we make about the data set, which warrant the new search for lower bounds in this paper. We will describe the two key properties below, which have to do with two assumptions about the data set: the distribution of points, and the size of the data set n relative to d.

The distribution of the data set. As described in the nearest neighbor definitions above, in this paper we specifically assume that the data set follows a uniform distribution over the underlying metric space. (Concretely we will consider the Euclidean sphere and the Hamming cube, for which this uniform distribution is well-defined.) Most literature on the nearest neighbor problem however makes no such assumptions, and aims to provide solutions for worst-case data sets. In practice however it often turns out that these "random data sets" are, in fact, worst-case data sets for most hash-based solutions [AINR14,AR15,ALRW17]. One may argue that here we are making stronger assumptions about the problem than in most of the past literature. On the other hand, in most applications the most natural distribution of points for the data set is uniform, and uniform data sets are often considered the hardest to deal with anyway. One could therefore consider this as only a minor additional assumption. Note that without this additional assumption, we would not be able to strengthen our model compared to previous work as described in the next paragraph.

The sparsity of data set. Most past work on nearest neighbor searching focused specifically on the so-called *sparse regime*, where the number of points n in the data set scales as $n = 2^{o(d)}$, or equivalently $\log n = o(d)$. For $\log n \ll d$, i.e. for extremely sparse data sets, one can always use a dimension reduction step [JL84] to obtain $\log n \propto d/\log d$; one can always go from an extremely sparse data set to a less sparse data set. This is however the limit, and one cannot reduce the dimensionality to $\log n \propto d$ without losing guarantees on the preservation of distances between points in the data set. The entire sparse regime can therefore be reduced by only solving the regime where $\log n \propto d/\log d$, but this leaves open the regime where $\log n = \Omega(d)$. The latter is exactly the regime of interest for the cryptanalytic applications in this paper, and unfortunately lower bounds are specifically tailored to the sparse regime.

To summarize: whereas most past work made no assumptions about the distribution of the data set, it did make assumptions about the sparsity of the data set. In this paper we make no assumptions about the sparsity of the data set, but we do specifically assume that the data set follows a uniform distribution.

3.5 Inapplicability of existing lower bounds

Various lower bounds have previously been derived for (hash-based) nearest neighbor searching in a long series of works [MNP07,PTW10,OWZ14,Chr17], but all of these have focused on the sparse regime, discussed above. As we are interested in the dense regime of $\log n = O(d)$, one might wonder whether applying the same lower bounds to the dense regime is just a "technicality", and if schemes which are known to be asymptotically optimal in the sparse regime are also optimal in the dense regime.

We can counter this reasoning with an explicit counterexample, showing that indeed the study in this paper is needed. For the sparse regime and for the angular distance (or nearest neighbor searching on the sphere; see Section 4), different schemes are known to be optimal:

- The spherical hashing from [AINR14] and the cross-polytope hashing from e.g. [TT09,AIL$^+$15] are both known to be optimal for the sparse regime. They both achieve the optimal scaling of the query exponent ρ for the balanced regime as $\rho \sim 1/(2c^2-1)$ for random data sets, when the target distance r is a factor c less than the average distance on the unit sphere ($\sqrt{2}$). Matching lower bounds are known [AINR14,AR15,AR16,ALRW17] showing their optimality for the sparse regime. When applying these schemes in the context of lattice sieving, where we substitute the nearest neighbor step by these optimized hashing schemes, the best possible time complexity for solving lattice problems in dimension d with both these hash-based approaches becomes $2^{0.297\ldots d+o(d)}$ [LdW15,BL16].
- Later on, spherical filtering was presented in [BDGL16], and further studied in [Laa15b,ALRW17]. Spherical filtering is also known to be optimal in the sparse regime, again obtaining the optimal scaling of $\rho \sim 1/(2c^2 - 1)$, up to lower order terms. When applying these results to lattice sieving however, again substituting this scheme for the nearest neighbor step that needs to be done, the time complexity for solving lattice problems becomes $2^{0.292\ldots d+o(d)}$ [BDGL16]. In other words, using this nearest neighbor scheme leads to a strict asymptotic improvement over the previous results from [LdW15,BL16], even though these other results were also relying on a hash-based scheme which was known to be optimal *in the sparse regime*.

The essence lies exactly in the fact that all existing lower bounds were derived specifically for the sparse regime, and do not necessarily carry over to the dense regime. And as the above situation in lattice sieving shows, indeed asymptotically optimal schemes in the sparse regime may be strictly suboptimal in the dense regime. This motivates the study of this work: to derive lower bounds for the dense regime, which do apply to regimes of interest in cryptanalysis (and potentially in other applications with dense data sets as well).

4 Nearest neighbor searching on the Euclidean sphere

For the Euclidean sphere, we instantiate the metric space (M, d) from Section 3 by the Euclidean metric $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ and the unit sphere $M = \mathcal{S}^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2 = 1\}$. Throughout Sections 4–5, we will write $\|\cdot\| = \|\cdot\|_2$ for the Euclidean norm.

4.1 The Baernstein–Taylor rearrangement inequality

A key ingredient for deriving the optimal hash-based approaches for the Euclidean sphere is the following result of Baernstein–Taylor from the 1970s [BT76]. This inequality is closely related to the Riesz–Sobolev rearrangement inequality [Rie30], but instantiated on the unit sphere rather than the entire real space. The original statement and its proof can be found in [BT76, Theorem 2]. Below σ denotes the normalized surface measure on $\mathcal{S}^{d-1}$, such that $\sigma(\mathcal{S}^{d-1}) = 1$.

Lemma 1 (Baernstein–Taylor inequality for $\mathcal{S}^{d-1}$ [BT76, Theorem 2]). *Let $f, g : \mathcal{S}^{d-1} \to \mathbb{R}$ be arbitrary Lebesgue-integrable functions. Let $h : [-1, 1] \to \mathbb{R}$ be a non-decreasing, bounded, and measurable function. Let $f^*, g^* : \mathcal{S}^{d-1} \to \mathbb{R}$ be functions satisfying the following conditions:*

- *$f^*(\mathbf{z})$ only depends on the first coordinate z_1 of $\mathbf{z}$ and is a non-decreasing function of z_1;*
- *$g^*(\mathbf{z})$ only depends on the first coordinate z_1 of $\mathbf{z}$ and is a non-decreasing function of z_1;*
- *For all $\lambda \in \mathbb{R}$: $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : f^*(\mathbf{z}) > \lambda\}) = \sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : f(\mathbf{z}) > \lambda\})$;*
- *For all $\lambda \in \mathbb{R}$: $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : g^*(\mathbf{z}) > \lambda\}) = \sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : g(\mathbf{z}) > \lambda\})$.*

Then:

$$\iint\limits_{\mathcal{S}^{d-1} \times \mathcal{S}^{d-1}} f(\mathbf{x}) g(\mathbf{y}) h(\langle \mathbf{x}, \mathbf{y} \rangle) \, \mathrm{d}\sigma(\mathbf{x}) \, \mathrm{d}\sigma(\mathbf{y}) \leq \iint\limits_{\mathcal{S}^{d-1} \times \mathcal{S}^{d-1}} f^*(\mathbf{x}) g^*(\mathbf{y}) h(\langle \mathbf{x}, \mathbf{y} \rangle) \, \mathrm{d}\sigma(\mathbf{x}) \, \mathrm{d}\sigma(\mathbf{y}).$$

4.2 Optimal hash collision probabilities

At first sight it may not be obvious how the above inequality is useful for us. The following corollary shows that with a proper instantiation of the functions f, g, h this naturally leads to an upper bound on collision probabilities for regions on the sphere in the hash-based nearest neighbor framework.

Theorem 1 (Collision probabilities for $\mathcal{S}^{d-1}$). *Let $Q, U \subseteq \mathcal{S}^{d-1}$ be arbitrary subsets of the sphere, and let $\mathcal{C}_Q, \mathcal{C}_U \subseteq \mathcal{S}^{d-1}$ be spherical caps of the following form:*

$$C_Q := \{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \alpha\}, \quad \text{with } \alpha \in [-1, 1] \text{ such that } \sigma(C_Q) = \sigma(Q),$$
$$C_U := \{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}, \quad \text{with } \beta \in [-1, 1] \text{ such that } \sigma(C_U) = \sigma(U).$$

Then, for any $\gamma \in [-1, 1]$ we have:

$$\Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in Q, \mathbf{y} \in U \mid \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma] \leq \Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U \mid \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma],$$

$$\Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in Q, \mathbf{y} \in U] = \Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U].$$

Proof. The second equality follows trivially by factoring the joint probability into two individual probabilities, and noting that the spherical caps $\mathcal{C}_Q, \mathcal{C}_U$ have the same volume as the sets Q, U:

$$\Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in Q, \mathbf{y} \in U] = \sigma(Q) \cdot \sigma(U) = \sigma(\mathcal{C}_Q) \cdot \sigma(\mathcal{C}_U) = \Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U].$$

The first inequality follows almost directly from the Baernstein–Taylor inequality with the proper choice of functions. We define the functions f, g, h as:

$$f(\mathbf{x}) := \mathbb{1}\{\mathbf{x} \in Q\}, \qquad g(\mathbf{y}) := \mathbb{1}\{\mathbf{y} \in U\}, \qquad h(s) := \mathbb{1}\{s \geq \gamma\}.$$

Note that, for $\lambda \in \mathbb{R}$, the functions f and g satisfy:

$$\sigma(\{f > \lambda\}) = \begin{cases} 1, & \lambda < 0; \\ \sigma(Q), & 0 \leq \lambda < 1; \\ 0, & 1 \leq \lambda; \end{cases} \qquad \sigma(\{g > \lambda\}) = \begin{cases} 1, & \lambda < 0; \\ \sigma(U), & 0 \leq \lambda < 1; \\ 0, & 1 \leq \lambda. \end{cases}$$

For the function f^* from Lemma 1 we need $\sigma(\{f^* > \lambda\}) = \sigma(\{f > \lambda\})$ to hold for all $\lambda \in \mathbb{R}$, with f^* only depending on x_1 and being non-decreasing in x_1. To satisfy $f^*(x_1) > 0$ with measure $\sigma(Q)$ and $f^*(x_1) \geq 0$ with measure 1, it follows that $f^*(x_1) = 0$ with measure $1 - \sigma(Q)$. Similarly $f^*(x_1) = 1$ with measure $\sigma(Q)$. This means that $f^*(x_1)$ must be a heaviside step function in one variable $x_1 \in [-1, 1]$, with an increase from 0 to 1 at the value $x_1 = \alpha$ satisfying $\sigma(Q) = \sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \alpha\})$. Defining $\mathcal{C}_Q := \{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \alpha\}$ for the above α, this translates to $\sigma(Q) = \sigma(\mathcal{C}_Q)$, and together with a similar derivation for g^* we obtain the expressions:

$$f^*(\mathbf{x}) := \mathbb{1}\{\mathbf{x} \in \mathcal{C}_Q\}, \quad \text{with } \mathcal{C}_Q = \{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \alpha\} \text{ such that } \sigma(Q) = \sigma(\mathcal{C}_Q);$$

$$g^*(\mathbf{y}) := \mathbb{1}\{\mathbf{y} \in \mathcal{C}_U\}, \quad \text{with } \mathcal{C}_U = \{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\} \text{ such that } \sigma(U) = \sigma(\mathcal{C}_U).$$

Now, with all conditions for Lemma 1 satisfied, we can instantiate the Baernstein–Taylor inequality for these functions f, f^*, g, g^*, h. Observing that the integrals can be interpreted as probabilities, and combining the indicator functions, we obtain:

$$\begin{aligned} &\Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in Q, \mathbf{y} \in U, \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma] = \\ &\iint_{\mathcal{S}^{d-1}\times\mathcal{S}^{d-1}} \mathbb{1}\{\mathbf{x} \in Q, \mathbf{y} \in U, \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma\}\, d\sigma(\mathbf{x})\, d\sigma(\mathbf{y}) \\ &\leq \iint_{\mathcal{S}^{d-1}\times\mathcal{S}^{d-1}} \mathbb{1}\{\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U, \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma\}\, d\sigma(\mathbf{x})\, d\sigma(\mathbf{y}) = \\ &\Pr_{\mathbf{x},\mathbf{y}\sim\mathcal{S}^{d-1}}[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U, \langle \mathbf{x}, \mathbf{y}\rangle \geq \gamma]. \end{aligned}$$

Note that the above derivation applies for all $\gamma \in [-1, 1]$. Now finally, we can easily obtain a similar inequality for the conditional probabilities as follows, where all probabilities are over $\mathbf{x}, \mathbf{y} \sim \mathcal{S}^{d-1}$:

$$\begin{aligned}\Pr\left[\mathbf{x} \in Q, \mathbf{y} \in U \mid \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right] &= \frac{\Pr\left[\mathbf{x} \in Q, \mathbf{y} \in U, \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right]}{\Pr\left[\langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right]} \\ &\leq \frac{\Pr\left[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U, \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right]}{\Pr\left[\langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right]} = \Pr\left[\mathbf{x} \in \mathcal{C}_Q, \mathbf{y} \in \mathcal{C}_U \mid \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\right].\end{aligned}$$

This completes the proof of the first inequality.

The above theorem states that, if we replace the hash regions Q and U by spherical caps of equal volume as Q and U, then (i) uncorrelated pairs of vectors are still equally likely to be found as candidate near neighbors, while (ii) nearby pairs of vectors are at least as likely (and perhaps more likely) to be considered as potential near neighbors. So ignoring e.g. the potential decoding overhead or the cost of membership queries for these different hash regions, this shows that the optimal choice for the hash regions is to use spherical caps. Note that for this optimality to hold, it is crucial that $\mathcal{C}_Q, \mathcal{C}_U$ are spherical caps centered at the same point on the sphere, although the same inequalities hold if both are centered at a different point $\mathbf{v} \in \mathcal{S}^{d-1}$ with $\mathbf{v} \neq \mathbf{e}_1$.

4.3 Optimal hash-based nearest neighbor searching

The previous result suggests that using spherical caps is optimal, and the following result formalizes this statement in the *asymptotic* setting. Here by "optimal" we mean that choosing the hash regions U_i or Q_i of shape different from spherical caps will not asymptotically improve the performance of Algorithm 3.1.

Theorem 2 (Spherical caps are optimal for $\mathcal{S}^{d-1}$). *Suppose we have access to an efficient decoding oracle for retrieving relevant hash regions. Then to get the best asymptotic performance for hash-based nearest neighbor searching, the following choice of hash regions is asymptotically optimal:*

- *Choose $t \in \mathbb{N}$, and for each $i \in [t]$ choose thresholds $\alpha_i, \beta_i \in [-1, 1]$ and draw $\mathbf{v}_i \sim \mathcal{S}^{d-1}$;*
- *Define $Q_i = \{\mathbf{z} \in \mathcal{S}^{d-1} : \langle \mathbf{z}, \mathbf{v}_i \rangle \geq \alpha_i\}$ and $U_i = \{\mathbf{z} \in \mathcal{S}^{d-1} : \langle \mathbf{z}, \mathbf{v}_i \rangle \geq \beta_i\}$.*

Proof. First, observe that with access to an efficient decoding algorithm, the costs of the hash-based nearest neighbor search are equal for two schemes which use regions of equal size; the data set and queries are assumed to be uniform, and therefore the number of hash collisions within each bucket and the number of buckets to check only depend on their volumes, and not on their shapes. Given the volumes of the regions, and the number of regions, the costs in terms of having to compare a query $\mathbf{x}$ with random vectors $\mathbf{y} \in L$ which are not near neighbors, does not depend on the shapes of the regions. The only thing that is influenced by the (relative) shapes of the regions is the probability of finding

nearby vectors in the list: given a query $\mathbf{x} \sim \mathcal{S}^{d-1}$, the probability of finding a nearby vector $\mathbf{y} \in L$ with $\langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma$ in at least one of the t potential buckets.

Recall that the hash collision probabilities for nearby vectors can be expressed in terms of probabilities of inserting and querying the same bucket, for at least one of the indices $i = 1, \ldots, t$. Letting $E_i = \{\mathbf{x} \in Q_i, \mathbf{y} \in U_i \mid \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\}$ denote the event that for a nearby vector $\mathbf{y}$ to the query $\mathbf{x}$, we insert $\mathbf{y}$ into bucket U_i and we later query Q_i for $\mathbf{x}$ in the query phase. Then we have:

$$p_1 = \Pr\left[\bigcup_{i=1}^{t} E_i\right] \leq \sum_{i=1}^{t} \Pr[E_i] = \sum_{i=1}^{t} \Pr_{\mathbf{x},\mathbf{y} \sim \mathcal{S}^{d-1}} [\mathbf{x} \in Q_i, \mathbf{y} \in U_i \mid \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma]\,. \quad (3)$$

The first inequality becomes more of an equality when the events are more disjoint; this tells us that ideally we should minimize the probabilities that two events E_i and E_j happen at the same time, e.g. by carefully spreading out these hash regions over the unit sphere[7]. Note that asymptotically, as analyzed in e.g. [BDGL16,Laa15b], we do indeed have $\Pr\left[\bigcup_{i=1}^{t} E_i\right] = \sum_{i=1}^{t} \Pr[E_i] \cdot (1+o(1))$ for all common parameter choices, as it is extremely unlikely that multiple events E_i happen at the same time for random $\mathbf{v}_i$. So the right hand side of (3) is asymptotically equal to p_1.

Finally, by Theorem 1 the right hand side of (3) is maximized when the shapes of the regions are spherical caps. So the probability of finding nearby vectors is maximized when the Q_i and U_i are spherical caps centered around the same vector $\mathbf{v}_i$ on the sphere. With the other collision probability p_2 being invariant under these replacements of arbitrary regions by equal-volume spherical caps, and with the decoding costs assumed to be not an issue, this shows that up to lower order terms, this hash-based scheme is optimal.

All that now remains is choosing the thresholds α_i and β_i. The following result shows that all the β_i's should be equal to get the best asymptotic performance, and that their optimal value is determined purely by the list size n. For the α_i we also derive that they should all be equal to the same value α, but together with t this parameter allows us to obtain trade-offs between the query and update complexities of the underlying hash-based scheme.

In the following theorem by "optimal" we mean that choosing the spherical caps U_i's (or Q_i's) of different sizes for different i will not improve the performance of Algorithm 3.1.

Theorem 3 (Equal spherical caps are optimal for $\mathcal{S}^{d-1}$). *Suppose we have access to an efficient decoding oracle for retrieving relevant hash regions. Then to get the best asymptotic performance for hash-based nearest neighbor searching, the following choice is asymptotically optimal:*

- *Choose $t \in \mathbb{N}$, choose $\alpha \in [-1, 1]$ and compute β such that $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}) \approx 1/n$;*

[7] This further illustrates the need for good spherical codes for determining where to place these vectors $\mathbf{v}_i$ to obtain the best performance in practice [AI06,TT07,AIL+15,Laa20].

– *For each* $i \in [t]$ *draw* $\mathbf{v}_i \sim \mathcal{S}^{d-1}$*;*
– *Define* $Q_i = \{\mathbf{z} \in \mathcal{S}^{d-1} : \langle \mathbf{z}, \mathbf{v}_i \rangle \geq \alpha\}$ *and* $U_i = \{\mathbf{z} \in \mathcal{S}^{d-1} : \langle \mathbf{z}, \mathbf{v}_i \rangle \geq \beta\}$.

Proof. Compared to the optimality result from Theorem 2 we need to prove that (1) fixing one parameter β, rather than choosing each separately, cannot decrease the asymptotic performance; and (2) with β fixed, it does not make sense to use different values α_i for the different buckets.

Fixing $\beta_i \equiv \beta$. For populating the buckets B_i, observe that we do not want most buckets to be empty (which happens when β_i is too large). In that case the overhead of retrieving these hash buckets will be much larger than the actual comparisons with potential near neighbors, as the number of buckets is larger than the number of vectors in these buckets. If many buckets are empty, we would be better off creating larger buckets, corresponding to larger spherical caps, until these buckets contain at least a few vectors each, decreasing the decoding cost and not affecting other costs more than $n^{o(1)}$. So we never want to choose β_i such that $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}) \ll 1/n$.

On the other hand, if we use spherical caps with too small parameters β_i, then these buckets will contain $n^{\Theta(1)}$ vectors each. Note that such a bucket corresponds to a spherical cap, which can essentially be seen as a sphere of one dimension less, with a smaller radius, and where again the vectors in this bucket are uniformly distributed over this lower-dimensional sphere. This is again a NNS instance on a smaller sphere, and we can do better than to put all $n^{\Theta(1)}$ vectors in one big list and having to query the whole list when we want to search this region for near neighbors. It cannot be worse to partition this bucket into smaller buckets, so that we can either choose α so large that the entire list is queried (if necessary), or we can choose α larger to only query some of these smaller buckets. So we also do not want to choose β_i too small, such that $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}) \gg 1/n$.

In other words, we want each β_i to satisfy $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}) \propto 1/n$. Small deviations in individual bucket sizes may not be worse in practice, but asymptotically we need all β_i to be approximately equal to the β satisfying $\sigma(\{\mathbf{z} \in \mathcal{S}^{d-1} : z_1 \geq \beta\}) = 1/n$ (see [HKL18][Appendix A]).

Fixing $\alpha_i \equiv \alpha$. With all β_i fixed to the same value β, and with all buckets containing (in expectation) a small number of vectors, the parameters α_i now control when buckets are queried. Note that for a fixed β, all buckets are identically shaped as a spherical cap of a fixed size, and with the data set being uniform on the sphere, all buckets are essentially equivalent. For a given query $\mathbf{x}$ however, the distribution of dot products $\langle \mathbf{x}, \mathbf{y} \rangle$ for vectors $\mathbf{y} \in B_i$ depends on $\langle \mathbf{x}, \mathbf{v}_i \rangle$: if $\mathbf{x}$ is almost equal to $\mathbf{v}_i$, we have a stronger guarantee that the vectors in this bucket (which are uniform in a spherical cap centered at $\mathbf{y}_i$) are close to $\mathbf{x}$ as well. On the other hand, if $\langle \mathbf{x}, \mathbf{v}_i \rangle$ is relatively small, then the vectors centered around $\mathbf{v}_i$ will on average be further away from $\mathbf{x}$. As each bucket contains equally many vectors, we therefore want to select only the buckets with the best potential for near neighbors, i.e. those buckets for which $\langle \mathbf{x}, \mathbf{v}_i \rangle$ is largest. Sorting the buckets by $\langle \mathbf{x}, \mathbf{v}_i \rangle$ and only going through the highest-quality buckets is

equivalent to selecting a single appropriate parameter α and only checking those buckets for which $\langle \mathbf{x}, \mathbf{v}_i \rangle \geq \alpha$.

So ultimately, we may set $\beta_i \equiv \beta$ to one fixed value, determined immediately by n and d, and fix $\alpha_i \equiv \alpha$ to one value which together with t then trades off the space and query complexities.

Note that the optimal choice of α is not obvious. The free parameters α and t together control the trade-off between the query time complexity and the update complexity. Concretely we can minimize for the query time by choosing both α and t to be large (generate a large number of buckets, and only query the buckets for which $\mathbf{v}_i$ is almost identical to $\mathbf{x}$), or we can minimize for the update and space complexities by choosing α and t to be small (using fewer hash buckets, but being less selective in the query phase and visiting most of these buckets).

Summarizing, the asymptotically optimal scheme (up to order terms) is now written all the way down up to selecting the best parameters t, α, and implementing such an efficient decoding oracle. This problem has previously been studied in [BDGL16,Laa15b,ALRW17], and here we will merely state that the schemes analyzed in these works are therefore optimal.

Theorem 4 (Spherical filtering is optimal for $\mathcal{S}^{d-1}$). *The hash-based near neighbor schemes studied in [Laa15b,BDGL16,ALRW17] are optimal within the hash-based framework for uniformly random data sets on the sphere.*

4.4 Results for dense data sets

Note that [ALRW17] already claimed optimality of the filtering approach described in [BDGL16,ALRW17], by proving matching lower bounds *in the sparse regime*. For the dense regime, no lower bounds were previously known, and as explained in Section 3.4 this was not just a matter of applying optimal algorithms from the sparse regime to the dense regime and claiming optimality in the dense regime as well. Our results settle the issue for uniformly random data sets, showing that spherical caps of specific sizes are indeed optimal.

The resulting optimal complexities for the dense regime can be found in e.g. [Laa15b, Theorem 2], where the parameters α and t were optimized to obtain the best performance. We restate these upper bounds below, where based on our lower bounds we now add that these trade-offs are optimal for the dense regime.

Theorem 5 (Trade-offs for the dense regime). *Let $\theta \in (0, \frac{1}{2}\pi)$, let the target dot product be $\langle \mathbf{x}, \mathbf{y} \rangle \geq \cos\theta$, and let the data set consist of $n = 2^{\Theta(d)}$ random points on the unit sphere. Then to obtain asymptotically optimal trade-offs for the query and update complexities, we should choose $u \in [\cos\theta, 1/\cos\theta]$ and set the parameters as:*

$$\alpha = u \cdot \sqrt{1 - n^{-2/d}}, \qquad \beta = \sqrt{1 - n^{-2/d}}.$$

We can then find nearest neighbors on the Euclidean sphere with query and update exponents:

$$\rho_q = \frac{-d}{2\log n}\log\left[1-\left(1-n^{-\frac{2}{d}}\right)\frac{1+u^2-2u\cos\theta}{\sin^2\theta}\right] + \frac{d}{2\log n}\log\left[1-\left(1-n^{-\frac{2}{d}}\right)u^2\right],$$
$$\rho_u = \frac{-d}{2\log n}\log\left[1-\left(1-n^{-2/d}\right)\frac{1+u^2-2u\cos\theta}{\sin^2\theta}\right] - 1.$$

The resulting algorithm has a query time complexity $\tilde{O}(n^{\rho_q})$, an update time complexity $\tilde{O}(n^{\rho_u})$, a preprocessing time complexity $\tilde{O}(n^{1+\rho_q})$, and a total space complexity of $\tilde{O}(n^{1+\rho_q})$. The total number of filters scales as $t = \tilde{O}(n^{1+\rho_q})$.

While the above formulas are a bit more technical, note that the query and update exponents only involve the input parameters d, n, θ and the trade-off parameter u. Choosing $u = 1$ leads to a "balanced" trade-off with $\rho_q = \rho_u$, and e.g. for the lattice sieving regime of the next section, where $\theta = \frac{\pi}{2}$ and $n = (4/3)^{d/2+o(d)}$, for $u = 1$ we obtain $\rho_q = \rho_u = \log(9/8)/\log(4/3)$ with query complexity $n^{\rho} = (9/8)^{d/2+o(d)}$ and closest pairs complexity $n^{1+\rho} = (3/2)^{d/2+o(d)}$.

5 Application to lattice sieving and lattice-based cryptography

With the results from Section 4 in mind, showing that the best hash-based nearest neighbor search technique is what has already been studied in the context of lattice cryptanalysis, we immediately get conditional optimality results for various current lattice sieving approaches. These optimality results are all under the assumption that we are only allowed to make tweaks to the nearest neighbor subroutine within these algorithms.

5.1 Lattice sieving

The lattice sieving approach introduced by Ajtai–Kumar–Sivakumar [AKS01] is currently the best known method for solving the shortest vector problem in practice on random high-dimensional lattices. For a d-dimensional lattice, the time and memory complexity are both of the order $2^{\Theta(d)}$, compared to a time complexity of $2^{\Omega(d\log d)}$ for enumeration-based approaches [Kan83,FP85,GNR10].

Given as input an arbitrary basis $\mathbf{B}$ of a lattice, sieving algorithms start by sampling an exponentially long list L of lattice vectors using efficient discrete Gaussian sampling procedures like [Kle00,GPV08]. Note that sampling exactly from a discrete Gaussian is not important; all that matters is that the sampled points are distinct, and are as short as possible. The points from the list are then combined to produce new shorter vectors $\mathbf{z} = \mathbf{x} - \mathbf{y}$ where $\mathbf{x}, \mathbf{y} \in L$. Note that $\mathbf{z}$ is short if and only if $\mathbf{x}$ and $\mathbf{y}$ are "near neighbors" in space, and this naturally leads us to using closest pairs algorithms for performing these sieving steps. The process of sieving is then executed iteratively with the new and shorter vectors

added to the list (and longer vectors getting removed from the list), until we ultimately find a shortest vector in our list.

The complexity of sieving algorithms is determined by the size of the starting list required for the iterative process to succeed, and by the complexity of finding short pairwise combinations of vectors in the list to form new short vectors. Note that by volume arguments over the sphere, if all lattice vectors in the list L have roughly the same norm, then (i) for a list of size $n = |L| \ll (4/3)^{d/2+o(d)}$ we expect the number of nearby pairs $\mathbf{x}, \mathbf{y} \in L$ with $\|\mathbf{x}-\mathbf{y}\| < \|\mathbf{x}\|$ to be significantly less than n, while (ii) for a list of size $n = (4/3)^{d/2+o(d)}$ we do expect the number of such pairs to be proportional to n. So if we wish to repeat this sieving step a polynomial number of times and end up with sufficiently many new vectors each time, we need the input list to be of size $n = (4/3)^{d/2+o(d)}$. The closest pairs subroutine then consists of: given a list of n vectors of roughly equal norms as input, find all pairs of vectors whose mutual distance is shorter than their individual norms. This translates to a target angle of $\pi/3$.

The above requirements on the algorithm lead to the following results, where we know that within the hash-based nearest neighbor framework, the results from Theorem 5 are optimal. So unless we modify other parts of the algorithm, or solve the closest pairs problem differently, these complexities are optimal for the standard pairwise sieving framework.

Theorem 6 (Classical sieve, heuristic). *Suppose we use a pairwise sieve with a hash-based nearest neighbor search subroutine to solve the closest pairs problem. Then the following time and space complexities of Becker–Ducas–Gama–Laarhoven [BDGL16] are asymptotically optimal:*

$$T = \left(\frac{3}{2}\right)^{d/2+o(d)} \approx 2^{0.292d+o(d)}, \qquad S = \left(\frac{4}{3}\right)^{d/2+o(d)} \approx 2^{0.208d+o(d)}.$$

Note that the space complexity S is determined by the list sizes required by the sieving, but not by the number of the buckets from the hashing as in Theorem 5. This is due to the fact that the number of query points in sieving is S, hence, a one can use a memory-efficient bucket processing (see end of Section 3.3).

Lattice sieving variants. Various variants of lattice sieving have been studied, aiming to solve slightly different problems or optimizing other parts of the underlying algorithm. We will briefly cover three of these variants: (i) quantum sieving [LMvdP15,Laa16,KMPM19], (ii) tuple sieving [BLS16,HK17,Laa17,HKL18], and (iii) sieving for the closest vector problem with preprocessing [Laa21,DLvW20]. Almost these algorithms (exception is the tuple sieve from [BLS16]) use near neighbor routines. Therefore, our lower bounds apply: if we are only allowed to replace the nearest neighbor subroutine by some other hash-based nearest neighbor subroutine, then we cannot do better than the above results. Of course, this does not rule out potential improvements coming from another modifications.

Relevance for lattice-based cryptography. As a take-away for cryptographic applications, one can view our lower bounds on sieving with nearest neighbor search-

ing as a further motivation for most concrete parameter selection methods currently used in practice, which assume that the leading time complexity exponents 0.292 and 0.265 are the best an attacker can do [BDK+18,BGML+18,BCD+16]. There is always the possibility that faster algorithms will be found, but if an attacker uses sieving with some form of nearest neighbor searching, they will not be able to improve upon these exponents.

The question remains how to estimate concrete costs in e.g. dimension 768 or 1024, as our lower bounds and most asymptotic analyses of upper bounds are asymptotic: the exponent scales as $0.292d + o(d)$ for large d (or $0.265d + o(d)$ quantumly), but the $o(d)$ may be arbitrarily small or large when d is fixed. Some past work has looked at trying to estimate the $o(d)$-term of the best upper bounds [Sch19,AGPS19].

Observe that when studying concrete attack costs in fixed dimensions d, it is also necessary to take into account further potential subexponential speedups, proposed in e.g. [Duc18,ADH+19,DLdW20]. Furthermore it may not be sufficient to only look at the asymptotically fastest approaches: in a fixed dimension d, another nearest neighbor method may have less overhead in practice and lead to better time and space complexities than the spherical filters, which match our asymptotic lower bounds. Especially here, where the gap between the time complexities for sieving with spherical filtering ($0.292d+o(d)$) and cross-polytope hashing ($0.298d + o(d)$) is so small, there is no guarantee that spherical filtering will be faster than cross-polytope hashing.

6 Nearest neighbor searching on the Hamming cube

We instantiate the nearest neighbor problem from Definition 4 with the the Hamming cube $M = \{0,1\}^d$ and the Hamming metric $d(\mathbf{x},\mathbf{y}) = |\{i \in [d] : x_i \neq y_i\}| = \|\mathbf{x}-\mathbf{y}\|_1$. Throughout Sections 6–7, we will write $\|\cdot\| = \|\cdot\|_1$ for the Hamming distance, and for the Hamming weight of vectors on the Hamming cube. It will further be easier to work with dimensionless versions of Hamming distances. In particular, we will denote the dimensionless target distance of the nearest neighbor problem by γ, i.e., $\gamma := r/d$. This applies to other distances we introduce below.

We start this section by obtaining a lower bound on nearest neighbor search using the result of Andoni–Razenstein [AR16]. Next we show that the algorithm of May–Ozerov [MO15] matches this lower bound in the sparse regime and comes extremely close to it in the dense regime.

6.1 The Andoni–Razenshteyn lower bound

Following [AR16], for $\mathbf{x} \in \mathbb{F}_2^d$ and $0 \leq \gamma < 1/2$, let us denote by $N_\gamma(\mathbf{x})$ a vector from $\mathbb{F}_2^d$ such that $(N_\gamma(\mathbf{x}))_i = x_i$ with probability $1-\gamma$ and $(N_\gamma(\mathbf{x}))_i = x_i \oplus 1$ with probability γ. So for any $\mathbf{x}$ and $N_\gamma(\mathbf{x})$, the Hamming distance between

them is on expectation $\gamma \cdot d$. For any hash function h, define

$$p_1 = \Pr_{\substack{\mathbf{x}\sim\mathbb{F}_2^d \\ \mathbf{y}\sim N_\gamma(\mathbf{x})}} [h(\mathbf{x}) = h(\mathbf{y})], \qquad p_2 = \Pr_{\mathbf{x},\mathbf{y}\sim\mathbb{F}_2^d} [h(\mathbf{x}) = h(\mathbf{y})].$$

We are interested in the quantity $\rho = \ln(1/p_1)/\ln(1/p_2)$, which defines the complexity of the nearest neighbor search when applied to the closest pairs problem. In particular, we are interested in a lower bound on ρ given in the following lemma.

Lemma 2 (Collision probabilities for $\{0,1\}^d$ [AR16, Lemma 5]). *For every hash function $h : \{0,1\}^d \to \mathbb{Z}$ and every $0 \le \gamma \le 1/2$:*

$$\Pr_{\substack{\mathbf{x}\sim\mathbb{F}_2^d \\ \mathbf{y}\sim N_\gamma(\mathbf{u})}} [h(\mathbf{x}) = h(\mathbf{y})] \le \Pr_{\mathbf{x},\mathbf{y}\sim\mathbb{F}_2^d} [h(\mathbf{x}) = h(\mathbf{y})]^{\frac{\gamma}{1-\gamma}}. \tag{4}$$

This lemma gives the relation between the probabilities p_1, p_2 and thus, tells what is the best sensitivity parameter ρ we can hope for. Namely, for the target distance $r = \gamma d$, using the above lemma we obtain the lower bound $\rho \ge \gamma/(1-\gamma)$. So the best we could achieve is the query time $T^{\texttt{Query}} = |L|^\rho$ and the total runtime of the nearest neighbor problem is $T = |L|^{1+\rho}$, which is the runtime of both the preprocessing step and the query step, when the number of queries is $|L|$. Taking the logarithm, we obtain the following lower bound:

$$\log_2 T \ge \frac{1}{1-\gamma} \log_2 |L|. \tag{5}$$

Next we compare the obtained lower bound with what is achieved in [MO15].

6.2 Spherical caps on the Hamming cube

For the dense case, the best known algorithm for the nearest neighbor problem is due to May–Ozerov [MO15] (see a recent result of Esser et. al [EKZ21] for a different analysis of this algorithm). At the heart of May–Ozerov is a hashing technique analogous to the one defined in Theorem 3, which is based on spherical caps in the Hamming space. As the main application of this hashing technique is to solve the closest pairs problem, we shall describe it the setting when the insert regions U_i and the query regions Q_i are the same.

The set up for the nearest neighbor data structure is as follows. An insertion region is defined by a center $\mathbf{v}_i \subseteq \mathbb{F}_2^d$ of the spherical cap $U_i = \{\mathbf{z} \in \mathbb{F}_2^d : \|\mathbf{z} - \mathbf{v}_i\| \le \beta\} \subseteq \mathbb{F}_2^d$, where β is the insertion parameter subject to optimization. The purpose of these regions is similar to the Euclidean metric case: when two vectors end up in the same region, i.e., both are close to some $\mathbf{v}_i$, then these vectors are also likely to be nearby to one another on the cube.

Given on input a list $L \subset \mathbb{F}_2^d$, the nearest neighbor search assigns each $\mathbf{y} \in L$ to its regions thus defining the buckets as $B_i = U_i \cap L$. The nearest neighbor data structure $\mathcal{D}$ consists of the union of all these buckets. Given a query $\mathbf{x}$ we

then look at all buckets B_i that are α-close to $\mathbf{x}$ (i.e., all $\mathbf{v}_i$ with $\|\mathbf{x} - \mathbf{v}_i\| \leq \alpha$), and we check if any of the vectors $\mathbf{y}$ stored in these buckets gives a solution to the nearest neighbor problem with parameter γ.

Similar to Theorem 3, we assume that we can efficiently find all relevant centers to a given point. An efficient procedure for that is called the 'stripes technique' and is described in [MO15]. The idea is to make the filter vectors structured (i.e., a concatenation of several codewords from some lower-dimensional codes). We will not describe this technique here in detail (for that, see [BDGL16,MO15]), but remark the main advantage of such a construction: it allows us find all close buckets in time (up to lower-order terms) equal to the output size.

When nearest neighbor searching is applied to the closest pairs problem, the number of queries is equal to $|L|$. In this case, the optimal choice of parameters is $\alpha = \beta = H^{-1}(1 - \log_2 |L|/d)$ so that the runtime T of the nearest neighbor search step are determined by the total number of buckets $|U_i|$ which we denote as t. This number is computed in [MO15, Theorem 1].

Theorem 7 (Hash-based complexities for $\{0,1\}^d$ [MO15, Thm. 1]). *To solve the nearest neighbor problem in the Hamming metric with some fixed target $0 \leq \gamma \leq 1/2$, with $\gamma = \Theta(d)$, the May–Ozerov algorithm uses a number t of hash regions satisfying:*

$$\log_2 t = (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1 - \log_2 |L|/d) - \gamma/2}{1-\gamma}\right)\right). \tag{6}$$

The following observation is important for our result: when the list size $|L|$ becomes subexponential in the dimension d, then the number of hash regions given above converges to $|L|^{\frac{1}{1-\gamma}}$. More precisely, [MO15, Corollary 1] shows that:

$$\lim_{\frac{1}{d}\log_2|L|\to 0} \frac{\log_2 t}{\log_2 |L|} = \frac{1}{1-\gamma}. \tag{7}$$

We shall next compare the lower and upper bounds for nearest neighbor searching on the Hamming cube.

6.3 Comparison between upper and lower bounds

Notice first that the lower bound given in Equation 5 matches exactly the performance of the May–Ozerov upper bound in the setting when the input list size is subexponential in the dimension, i.e., in the *sparse* regime.

Decoding algorithms we discuss in the next section work in the *dense* regime, i.e., when $|L| = 2^{cd}$ for a constant c. In this regime the above lower bound does not exactly match the complexity of May–Ozerov given in Equation (6) as one can see from the plot given in Figure 1, where we compare the two nearest neighbor search runtimes given in Equation (5). For a given target distance γ we set $|L| = 2^{\frac{1}{2} - \frac{1}{2}H(\gamma)}$, so we expect only sub-exponentially many pairs from L to satisfy the target distance condition, assuming L consists of uniformly randomly

vectors. Notice that the larger γ is, the smaller the list sizes we choose and the closer both bounds are to each other. This is consistent with the fact that May–Ozerov is optimal in the sparse regime.

One source of the discrepancy between the upper bound of May–Ozerov and the lower bound based on the Andoni–Razensteyn result is that the latter uses the probabilistic distance between the two close vectors $\mathbf{x}, \mathbf{y}$, namely the distance follows a binomial distribution with expected value $\gamma \cdot d$, while the algorithm of [MO15] targets to find $\mathbf{x}, \mathbf{y}$ whose distance is *at most* $\gamma \cdot d$ (with high concentration at the boundary). The tails of the distributions of the distances differ in these two cases leading to a gap between the bounds.

Another source of the gap lies in an inequality which Andoni–Razensteyn used in the proof of Lemma 2. In particular, they use the fact (see [KV15] for a proof) that for an arbitrary set $A \subseteq \mathbb{F}_2^d$, $\Pr_{\mathbf{x}\sim\mathbb{F}_2^d, \mathbf{y}\sim N_\gamma(\mathbf{u})}[\mathbf{x} \in A \mid \mathbf{y} \in A] \leq \left(|A|/2^d\right)^{\gamma/(1-\gamma)}$. This inequality is not tight when A is chosen to be a spherical cap in the Hamming space. This leaves the question of whether one can construct a set A, which would be useful for nearest neighbor searching (that is, it would have an efficient membership oracle), and for which the inequality holds with equality. That would give an improvement to nearest neighbor searching in the dense regime, albeit a very small one, as we shall see in the next section.

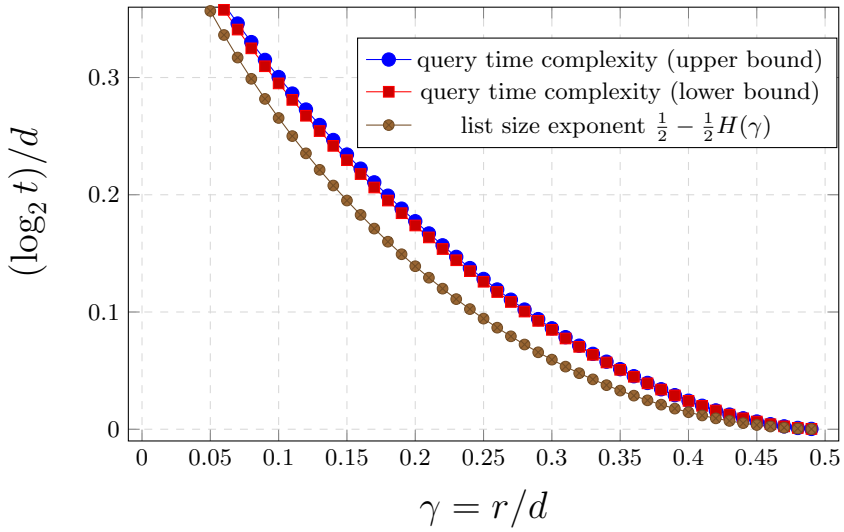

Fig. 1: Nearest neighbor search runtime exponents (dimensionless) for the target distance γ for lists of sizes $2^{\left(\frac{1}{2}-\frac{1}{2}H(\gamma)\right)d}$, i.e., we expect sub-exponentially many solutions. Upper bounds are determined by the number of hash regions t and follow from Equation (6), while lower bounds are based on Equation (5).

7 Application to decoding and code-based cryptography

All currently known fastest information set decoding algorithms for the *dense* setting make use of nearest neighbor searching. The goal of this chapter is to see how far down we could push the complexity of these decoding algorithms if we had a nearest neighbor search technique that matches the lower bound derived in the previous section.

In this section we will consider two algorithms: Stern's algorithm [Ste89], and the most recent algorithm of Both–May [BM18]. The first is the simplest information set decoding algorithm where nearest neighbor searching can be applied, while the second is the one that achieves the best currently known asymptotic time complexities.

7.1 Stern's algorithm

Recall from Definition 2, that as input the information set decoding problem receives a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{d-k \times d}$ and a syndrome $\mathbf{s} \in \mathbb{F}_2^{d-k}$. Stern's algorithm transforms the parity check matrix $\mathbf{H}$ into systematic form $[\mathbf{Q} \mid \mathbb{I}_{d-k}]$ (provided the last $d-k$ columns of $\mathbf{H}$ form an invertible matrix, which happens with constant success probability). The same transformation is applied to the syndrome $\mathbf{s}$ giving a new syndrome $\bar{\mathbf{s}}$. So the task is to find $\mathbf{e}$ that satisfies the equation:

$$[\mathbf{Q} \mid \mathbb{I}_{d-k}] \cdot \mathbf{e} = \bar{\mathbf{s}} \quad \text{for} \quad \mathbf{Q} \in \mathbb{F}_2^{d-k \times k}. \tag{8}$$

Stern's algorithm searches for a vector $\mathbf{e}$ whose weight is $p > 0$ on the last $d-k$ coordinates (hence, weight $w-p$ on the first k coordinates). The probability that this happens is $P = \binom{k}{p}\binom{d-k}{w-p}/\binom{d}{w}$. The inverse of this quantity is the expected number of permutations we need to apply on $\mathbf{H}$ to obtain the desired weight distribution on $\mathbf{e}$. Once a good permutation is found, Equation (8) rewrites as:

$$\mathbf{Q}\mathbf{e}_1 + \mathbf{Q}\mathbf{e}_2 + \mathbf{e}_3 = \bar{\mathbf{s}} \implies \mathbf{Q}\mathbf{e}_1 \approx \mathbf{Q}\mathbf{e}_2 + \bar{\mathbf{s}}. \tag{9}$$

Here $\mathbf{e}_1$ has weight $p/2$ on the first $k/2$ coordinates and is 0 on the last $d-k/2$ coordinates, $\mathbf{e}_2$ has weight $p/2$ on the coordinates $\{k/2+1, \ldots, k\}$ and is 0 elsewhere, and $\|\mathbf{e}_3\| = w-p$. Enumerating over all $\mathbf{e}_1$ and $\mathbf{e}_2$ into the lists $L_1 = \{(\mathbf{Q}\mathbf{e}_1, \mathbf{e}_1)\}$ and $L_2 = \{(\mathbf{Q}\mathbf{e}_2 + \mathbf{s}, \mathbf{e}_2)\}$, we receive an instance of the nearest neighbor problem with target distance $w-p$ in the Hamming metric.

May–Ozerov in [MO15] propose to solve this task with nearest neighbor searching and obtain the runtime of Stern's algorithm as illustrated in Figure 2. We compare this with the decoding complexities if instead of the upper bound of May–Ozerov, the lower bound runtimes from Equation (5) are substituted. Note that this is different from the comparison given in Figure 1 since the complexity of Stern's algorithm is not only determined by the complexity of the nearest neighbor subroutine, but also by the number of permutations. For various code rates k, Figure 2 compares the runtime exponents c for Stern's algorithm when (i) the nearest neighbor technique of [MO15] as in Equation (6) is used, or (ii)

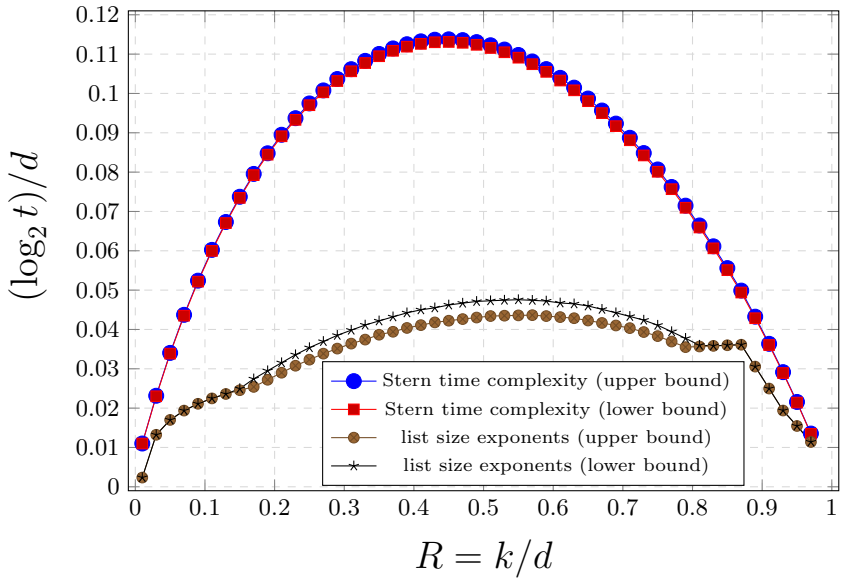

Fig. 2: Runtime exponents for Stern's information set decoding algorithm when either May–Ozerov's nearest neighbor search approach is used (blue circles), or when the lower bound is implemented (red squares). The code rate R is on the horizontal axis. The other two plots show the list size exponents that Stern's algorithm runs the nearest neighbor step on. The faster nearest neighbor searching (the lower bound) allows larger lists, hence there is a bigger gap between the list sizes in the most dense regime (around $R = 0.5$). The larger or the smaller the code rate is, the closer we are to the sparse setting, so the closer lower and upper bounds to each other.

the lower bound for nearest neighbor searching is used. It also gives corresponding list sizes $|L_1| = |L_2| = \binom{k}{p} \approx 2^{kH(p/k)}$, but note that the optimal value for p slightly differs between the two runtimes. As the nearest neighbor search step becomes cheaper, the p is allowed to increase leading to larger lists.

7.2 The Both–May algorithm

The recent information set decoding algorithm due to Both–May [BM18] significantly improves Stern's algorithm, and is currently the fastest algorithm for solving the information set decoding problem in the dense regime. We shall not describe the algorithm here but point out that the algorithm uses two-step nearest neighbor searching.

Similar to Stern's algorithm we compare the runtime of the Both–May algorithm when for the nearest neighbor steps, either (i) the best known nearest neighbor approach of May–Ozerov is used, or (ii) the lower bound given in Equation (5) is used. Optimal runtimes for each code rate k are given in Figure 3.

We notice that the Both–May algorithm, while being quite close to the lower bound, leaves more potential for improvement than Stern's algorithm. This can be explained by looking at the lists sizes: the Both–May algorithm allows for larger lists that Stern's algorithm, thus making the contribution of the nearest neighbor subroutine more substantial. Still, our conclusion is that a potential improvement in the nearest neighbor subroutine will not significantly improve the overall algorithm.

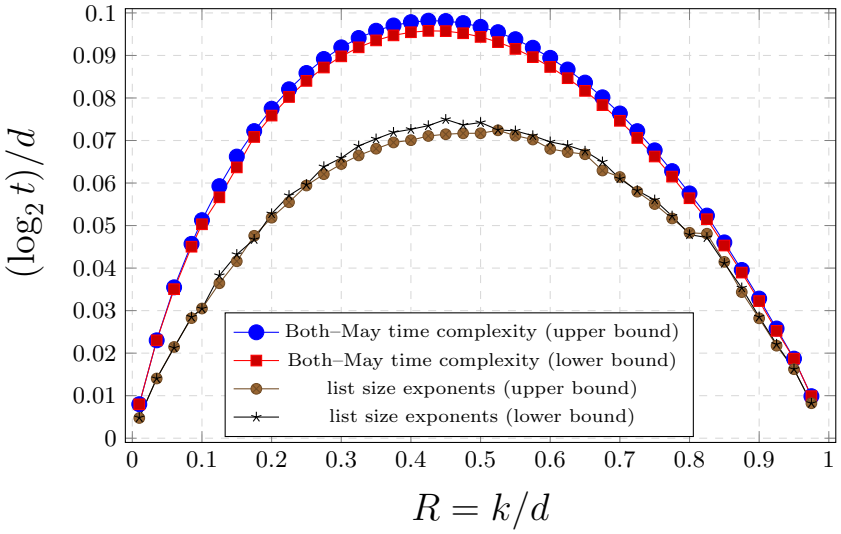

Fig. 3: Runtime exponents for the Both–May [BM18] algorithm when either the May–Ozerov [MO15] upper bound is used (blue circles), or our lower bound is substituted (red squares). The other two plots show the list size exponents that nearest neighbor searching receives on input. The code rate R is on the X-axis.

7.3 Relevance for code-based cryptography

The hardness of the information set decoding problem is essential for the security of prominent code-based cryptosystems, such as the McEliece cryptosystem [McE78]. All proposed constructions for the NIST standardization competition [BCL+19] work in the regime where the error is of weight sub-linear in d, thus making Stern's algorithm and other faster information set decoding algorithms like [BM18] *asymptotically* irrelevant [CTS16]. This does not imply, however, that these information set decoding algorithms are *practically* irrelevant for concrete parameters. To the best of our knowledge, the question of the exact complexity of the fastest information set decoding algorithms using recent improvements has not been investigated. This leaves the possibility that information set decoding algorithms which do use nearest neighbor techniques will

eventually be recognized as being actually applicable to cryptographic parameters as well, in which case our lower bounds may serve as conservative estimates for potential attack costs, and for choosing parameters.

Acknowledgments

Elena Kirshanova is supported by the "5-100" Russian academic excellence project and by the Young Russian Mathematics scholarship. Thijs Laarhoven is supported by a Veni grant from NWO under project number 016.Veni.192.005. Part of this work was done while both authors were visiting the Simons Institute for the Theory of Computing at UC Berkeley for the Spring 2020 program "Lattices: Algorithms, Complexity, and Cryptography."

References

AD97. Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997.

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.

ADRS15. Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In *STOC*, pages 733–742, 2015.

AGPS19. Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. *Cryptology ePrint Archive 2019/1161*, 2019.

AI06. Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006.

AIL+15. Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015.

AINR14. Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014.

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

ALRW17. Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, pages 47–66, 2017.

ANSS18. Yoshinori Aono, Phong Q. Nguyen, Takenobu Seito, and Junji Shikata. Lower bounds on lattice enumeration with extreme pruning. In *CRYPTO*, pages 608–637, 2018.

AR15. Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015.

AR16. Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. In *SOCG*, pages 1–15, 2016.

BCD+16. Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *CCS*, pages 1006–1018, 2016.

BCL+19. Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic McEliece: conservative code-based cryptography, 2019.

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.

BDK+18. Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. In *Euro S&P*, pages 353–367, 2018.

BGJ15. Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. *Cryptology ePrint Archive, Report 2015/522*, pages 1–14, 2015.

BGML+18. Sauvik Bhattacharya, Oscar Garcia-Morchon, Thijs Laarhoven, Ronald Rietman, Markku-Juhani O. Saarinen, Ludo Tolhuizen, and Zhenfei Zhang. Round5: Compact and fast post-quantum public-key encryption. *Cryptology ePrint Archive, Report 2018/725*, 2018.

BJMM12. Anja Becker, Antoine Joux, Alexandre May, and Alexandre Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, pages 520–536, 2012.

BL16. Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016.

BLS16. Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. In *ANTS*, pages 146–162, 2016.

BM18. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security. In *PQCrypto*, pages 25–46, 2018.

BT76. Albert Baernstein and B.A. Taylor. Spherical rearrangements, subharmonic functions, and $*$ -functions in n -space. *Duke Math. J.*, 43(2):245–268, 06 1976.

Chr17. Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017.

CTS16. Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Proceedings of the 7th International Workshop on Post-Quantum Cryptography - Volume 9606*, PQCrypto 2016, page 144–161, 2016.

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

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

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

EKZ21. Andre Esser, Robert Kübler, and Floyd Zweydinger. A faster algorithm for finding closest pairs in hamming metric, 2021.

FP85. Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice. *Mathematics of Computation*, 44(170):463–471, 1985.

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

GGH13. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

GNR10. Nicolas Gama, Phong Q. Nguyên, and Oded Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, pages 257–278, 2010.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

HK17. Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in Euclidean norm. In *PKC*, pages 16–40, 2017.

HKL18. Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In *PKC*, pages 407–436, 2018.

JL84. William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(1):189–206, 1984.

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

Kle00. Philip Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941, 2000.

KMPM19. Elena Kirshanova, Erik Martensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *ASIACRYPT*, pages 521–551, 2019.

KV15. Subhash A. Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1. *J. ACM*, 62(1), 2015.

Laa15a. Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015.

Laa15b. Thijs Laarhoven. Tradeoffs for nearest neighbors on the sphere. *arXiv:1511.07527 [cs.DS]*, pages 1–16, 2015.

Laa16. Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2016.

Laa17. Thijs Laarhoven. Faster tuple lattice sieving using spherical locality-sensitive filters. *arXiv:1705.02828 [cs.DS]*, pages 1–14, 2017.

Laa20. Thijs Laarhoven. Polytopes, lattices, and spherical codes for the nearest neighbor problem. In *ICALP*, 2020.

Laa21. Thijs Laarhoven. Approximate Voronoi cells for lattices, revisited. *Journal of Mathematical Cryptology*, 15:1–21, 2021.

Lan20. Tanja Lange. Overview of code-based crypto assumptions. Talk at Quantum Cryptanalysis of Post-Quantum Cryptography, 2020.

LdW15. Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *LATINCRYPT*, pages 101–118, 2015.

LMvdP15. Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.

McE78. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *The Deep Space Network Progress Report*, pages 114–116, 1978.

MMT11. Alexandre May, Alexandre Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.54n})$. In *ASIACRYPT*, volume 7073 of LNCS, pages 107–124, 2011.

MNP07. Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal of Discrete Mathematics*, 21(4):930–935, 2007.

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.

MR07. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

MV10a. 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.

MV10b. Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480, 2010.

NV08. Phong Q. Nguyên and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008.

OWZ14. Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory*, 6(1):5:1–5:13, 2014.

Pan06. Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8:5–9, 1962.

PS09. Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *Cryptology ePrint Archive, Report 2009/605*, pages 1–7, 2009.

PTW10. Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *FOCS*, pages 805–814, Oct 2010.

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.

Rie30. Frederic Riesz. Sur une inégalité intégrale. *Journal of the London Mathematical Society*, s1-5(3):162–168, 1930.

RSA78. Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

Sch19. John Schanck. Sieve tables, 2019.

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.

Ste89. Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, pages 106–113, 1989.

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

TT07. Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007.

TT09. Kengo Terasawa and Yuzuru Tanaka. Approximate nearest neighbor search for a dataset of normalized vectors. *IEICE Transactions on Information and Systems*, 92(9):1609–1619, 2009.