Tradeoffs for nearest neighbors on the sphere

Tradeoffs for nearest neighbors on the sphere

Thijs Laarhoven[*]

September 13, 2016

Abstract

We consider tradeoffs between the query and update complexities for the (approximate) nearest neighbor problem on the sphere, extending the spherical filters recently introduced by [Becker–Ducas–Gama–Laarhoven, SODA'16] to sparse regimes and generalizing the scheme and analysis to account for different tradeoffs. In a nutshell, for the sparse regime the tradeoff between the query complexity $n^{\rho_{\mathrm{q}}}$ and update complexity $n^{\rho_{\mathrm{u}}}$ for data sets of size n can be summarized by the following equation in terms of the approximation factor c and the exponents ρ_{q} and ρ_{u}:

$$c^2\sqrt{\rho_{\mathrm{q}}} + (c^2-1)\sqrt{\rho_{\mathrm{u}}} = \sqrt{2c^2-1}.$$

For small $c = 1+\varepsilon$, minimizing the time for updates leads to a linear space complexity at the cost of a query time complexity of approximately $n^{1-4\varepsilon^2}$. Balancing the query and update costs leads to optimal complexities of $n^{1/(2c^2-1)}$, matching lower bounds from [Andoni–Razenshteyn, 2015] and [Dubiner, IEEE Trans. Inf. Theory 2010] and matching the asymptotic complexities previously obtained by [Andoni–Razenshteyn, STOC'15] and [Andoni–Indyk–Laarhoven–Razenshteyn–Schmidt, NIPS'15]. A subpolynomial query time complexity $n^{o(1)}$ can be achieved at the cost of a space complexity of the order $n^{1/(4\varepsilon^2)}$, matching the lower bound $n^{\Omega(1/\varepsilon^2)}$ of [Andoni–Indyk–Pătraşcu, FOCS'06] and [Panigrahy–Talwar–Wieder, FOCS'10] and improving upon results of [Indyk–Motwani, STOC'98] and [Kushilevitz–Ostrovsky–Rabani, STOC'98] with a considerably smaller leading constant in the exponent.

For large c, minimizing the update complexity results in a query complexity of $n^{2/c^2+O(1/c^4)}$, improving upon the related asymptotic exponent for large c of [Kapralov, PODS'15] by a factor 2, and matching the lower bound $n^{\Omega(1/c^2)}$ of [Panigrahy–Talwar–Wieder, FOCS'08]. Balancing the costs leads to optimal complexities of the order $n^{1/(2c^2-1)}$, while a minimum query time complexity can be achieved with update and space complexities of approximately $n^{2/c^2+O(1/c^4)}$ and $n^{1+2/c^2+O(1/c^4)}$, also improving upon the previous best exponents of Kapralov by a factor 2 for large n and c.

For the regime where n is exponential in the dimension, we obtain further improvements compared to results obtained with locality-sensitive hashing. We provide explicit expressions for the query and update complexities in terms of the approximation factor c and the chosen tradeoff, and we derive asymptotic results for the case of the highest possible density for random data sets.

1 Introduction

Approximate nearest neighbors (ANN). A central computational problem in many areas of research, such as machine learning, coding theory, pattern recognition, data compression, and cryptanalysis [Bis06, Dub10, DHS00, Her15, Laa15, MO15, SDI05], is the *nearest neighbor problem*: given a d-dimensional data set $\mathcal{D} \subset \mathbb{R}^d$ of cardinality n, design a data structure and preprocess $\mathcal{D}$ in a way that, when later given a query vector $\boldsymbol{q} \in \mathbb{R}^d$, we can quickly find a nearest vector to $\boldsymbol{q}$ in $\mathcal{D}$. A common relaxation of this problem is the *approximate nearest neighbor problem (ANN)*: given that the nearest neighbor in $\mathcal{D}$ lies at distance at most r from $\boldsymbol{q}$, design an efficient algorithm that finds an element $\boldsymbol{p} \in \mathcal{D}$ at distance at most $c \cdot r$ from $\boldsymbol{q}$, for a given approximation factor $c > 1$. We will consider the case where d scales with n; for fixed d it is well-known that one can answer queries in timen^{ρ} with $\rho = o(1)$ with only a polynomial increase in the space complexity [AMN+94].

[*] Eindhoven University of Technology, Eindhoven, The Netherlands. E-mail: mail@thijs.com

ANN on the sphere. Depending on the notion of distance, different solutions have been proposed in the literature (e.g. [AHL01, DIIM04, GIM99, LRU14, WSSJ14]). In this work we will restrict out attention to the *angular distance*, where two vectors are considered nearby iff their common angle is small [AIL+15, Cha02, Laa15, SSLM14, STS+13]. This equivalently corresponds to spherical ANN under the ℓ_2-norm, where the entire data set is assumed to lie on the Euclidean unit sphere. Recent work of Andoni and Razenshteyn [AR15a] showed how to reduce ANN in the entire Euclidean space to ANN on the sphere, which further motivates why finding optimal solutions for the spherical case is relevant. For spherical, low-density settings ($n = 2^{o(d)}$), we will further focus on the *random* setting described in [AR15a], where *nearby* corresponds to a Euclidean distance of $r = \frac{1}{c}\sqrt{2}$ on the unit sphere (i.e. an angle $\theta = \arccos(1 - 1/c^2)$) and *far away* corresponds to a distance $c \cdot r = \sqrt{2}$ (angle $\psi = \frac{1}{2}\pi$).

Spherical locality-sensitive hashing. A well-known method for solving ANN in high dimensions is *locality-sensitive hashing (LSH)* [IM98]. Using locality-sensitive hash functions, with the property that nearby vectors are more likely to be mapped to the same output value than distant pairs of vectors, one builds several hash tables with buckets containing sets of vectors with the same hash value. To answer a query $\boldsymbol{q}$, one computes $\boldsymbol{q}$'s hash values and checks the corresponding buckets in each of the hash tables for potential near neighbors. For spherical ANN in the random setting, two recent works [AR15a, AIL+15] have shown how to solve ANN with query time $\tilde{O}(n^{\rho})$ and space $\tilde{O}(n^{1+\rho})$ with $\rho = \frac{1}{2c^2-1} + o(1)$ where $o(1) \to 0$ as $n \to \infty$. For large c and n, this improved upon e.g. hyperplane LSH [Cha02] and Euclidean LSH [AI06]. Within the class of LSH algorithms, these results are known to be essentially optimal [AR15b, Dub10].

Spherical locality-sensitive filters. Recently Becker–Ducas–Gama–Laarhoven [BDGL16] introduced spherical filters, which map the data set $\mathcal{D}$ to a subset $\mathcal{D}' \subseteq \mathcal{D}$ consisting of all points lying in a certain spherical cap. Filtering could be considered a relaxation of locality-sensitive hashing: for LSH a hash function is required to *partition* the space in regions, while for LSF this is not necessarily the case. Similar filtering constructions were previously proposed in [Dub10, MO15]. For dense data sets of size $n = 2^{\Theta(d)}$, the approach of [BDGL16] led to a query exponent $\rho < 1/(2c^2 - 1)$ for the random setting.

Asymmetric ANN. The exponents ρ described so far are all for *balanced* or *symmetric* ANN: both the time to answer a query and the time to insert/delete vectors from the data structure are then equal to $\tilde{O}(n^{\rho})$, and the time complexity for preprocessing the data and the total space complexity are both equal to $\tilde{O}(n^{1+\rho})$. Depending on the application however, it may be desirable to obtain a different tradeoff between these costs. In some cases it may be beneficial to use even more space and more time for the preprocessing, so that queries can be answered even faster. In other cases, memory constraints might rule out the use of balanced parameters, in which case one has to settle for a lower space and update complexity and it would be interesting to know the best time complexity that can be achieved for a given space complexity. Finding optimal tradeoffs between the different costs of ANN is therefore essential for achieving the best performance in different contexts.

Smooth tradeoffs for asymmetric ANN. Various works have analyzed tradeoffs for ANN, among others using multi-probing in LSH to reduce the memory complexity at the cost of a higher query complexity [AIL+15, AMM09, LJW+07, Pan06]. However, most existing techniques either describe one particular tradeoff between the costs, or do not offer provable asymptotic bounds on the query and update exponent as the parameters increase. Recently Kapralov [Kap15] showed how to obtain smooth and provable asymptotic tradeoffs for Euclidean ANN, but as the exponents for the balanced setting are a factor 2 above the lower bound $\rho \geq 1/(2c^2 - 1)$ for large c, it may be possible to improve upon these techniques not only for symmetric but also for asymmetric ANN.

1.1 Contributions.

In this work we extend the symmetric ANN technique of [BDGL16] for dense regimes to asymmetric ANN on the sphere for both sparse and dense regimes, showing how to obtain smooth and significantly improved tradeoffs between the query and update complexities compared to e.g. [Kap15, Pan06] in both the small c and large c regimes. For sparse settings, the tradeoff between the query complexity $n^{\rho_{\mathrm{q}}}$ and the update complexity $n^{\rho_{\mathrm{u}}}$ can essentially be summarized by the non-negative solution pairs $(\rho_{\mathrm{u}}, \rho_{\mathrm{q}}) \in \mathbb{R}^2$ to the following equation, which can be expressed either in terms of θ (left) or c (right) by substituting $\cos\theta = 1 - 1/c^2$.

$$\sqrt{\rho_{\mathrm{q}}} + (\cos\theta)\sqrt{\rho_{\mathrm{u}}} = \sin\theta, \qquad c^2\sqrt{\rho_{\mathrm{q}}} + (c^2-1)\sqrt{\rho_{\mathrm{u}}} = \sqrt{2c^2-1}. \tag{1}$$

The space complexity for the preprocessed data, as well as the time for preprocessing the data, are both $\tilde{O}(n^{1+\rho_{\mathrm{u}}})$. The resulting tradeoffs for the random case for small and large c are illustrated in Table 1 and Figure 2, and can be derived from (1) by substituting $\rho_{\mathrm{u}} = 0$ (minimize the space), $\rho_{\mathrm{q}} = \rho_{\mathrm{u}}$ (balanced ANN), or $\rho_{\mathrm{q}} = 0$ (minimize the time), and computing Taylor expansions around $c \in \{1, \infty\}$.

Small approximation factors. In the regime of small $c = 1 + \varepsilon$, as described in Table 1 we obtain an update complexity $n^{o(1)}$ and space complexity $n^{1+o(1)}$ with query complexity $n^{1-4\varepsilon^2+O(\varepsilon^3)}$. This improves upon results of Kapralov, where a sublinear query complexity and a quasi-linear space complexity could only be achieved for approximation factors $c > 1.73$ [Kap15]. Balancing the complexities leads to asymptotic exponents $\rho_{\mathrm{q}} = \rho_{\mathrm{u}} = 1/(2c^2-1)$, which means that both exponents scale as $1 - O(\varepsilon)$ for small $c > 1$. These exponents match the asymptotic complexities previously obtained by [AR15a, AIL$^+$15] and the lower bounds from [AR15b, Dub10]. A query complexity $n^{o(1)}$ can be achieved for arbitrary c with an update complexity $n^{4/\varepsilon^2+O(1/\varepsilon)}$, matching the asymptotic lower bounds of [AIP06, PTW10][1] and the constructions of [IM98, KOR00] with a smaller leading constant in the exponent[2]. This also improves upon [Kap15], achieving a query complexity $n^{o(1)}$ only for $c > 1.73$.

Large approximation factors. For large c, both ρ_{q} and ρ_{u} are proportional to $1/c^2$, with leading constants $1/2$ in the balanced regime, and leading constant 2 if the other complexity is minimized. This improves upon results from [Kap15], whose leading constants are a factor 2 higher in all cases, and matches the lower bound on the space complexity of $n^{\Omega(1/c^2)}$ of [PTW08] for query complexities $n^{o(1)}$.

High-density regime. Finally, for data sets of size $n = 2^{\Theta(d)}$, we obtain improved tradeoffs between the query and update complexities compared to results obtained using locality-sensitive hashing, even for balanced settings. We show that also for this harder problem we obtain query exponents less than 1 regardless of the tradeoff, while a query exponent 0 is impossible to achieve with our methods.

1.2 Outline.

In Section 2 we describe preliminary notation and results regarding spherical filters. Section 3 describes asymptotic tradeoffs for the sparse regime of $n = 2^{o(d)}$, and Section 4 then discusses the application of these techniques to the dense regime of $n = 2^{\Theta(d)}$. Section 5 concludes with a discussion on extending our methods to slightly different problems, and open problems for future work.

2 Preliminaries

2.1 Subsets of the unit sphere.

We first recall some preliminary notation and results on geometric objects on the unit sphere, similar to [BDGL16]. Let μ denote the canonical Lebesgue measure over $\mathbb{R}^d$, and let us write $\langle \cdot, \cdot \rangle$ for the standard

[1] The constant 1/4 in the exponent even matches the lower bound for single-probe schemes of [PTW10, Theorem 1.5].

[2] Explicit leading constants are not stated in [IM98, KOR00] but appear to be 9 and 9/2 respectively, compared to our 1/4.

	General expressions	**Small** $c = 1 + \varepsilon$	**Large** $c \to \infty$
Minimize space $(\beta = \cos\theta)$	$\rho_{\mathrm{q}} = \mathbf{(2c^2 - 1)/c^4}$ $\rho_{\mathrm{u}} = \mathbf{0}$	$\rho_{\mathrm{q}} = 1 - 4\varepsilon^2 + O(\varepsilon^3)$ $\rho_{\mathrm{u}} = 0$	$\rho_{\mathrm{q}} = 2/c^2 + O(1/c^4)$ $\rho_{\mathrm{u}} = 0$
Balance costs $(\beta = 1)$	$\rho_{\mathrm{q}} = \mathbf{1/(2c^2 - 1)}$ $\rho_{\mathrm{u}} = \mathbf{1/(2c^2 - 1)}$	$\rho_{\mathrm{q}} = 1 - 4\varepsilon + O(\varepsilon^2)$ $\rho_{\mathrm{u}} = 1 - 4\varepsilon + O(\varepsilon^2)$	$\rho_{\mathrm{q}} = 1/(2c^2) + O(1/c^4)$ $\rho_{\mathrm{u}} = 1/(2c^2) + O(1/c^4)$
Minimize time $(\beta = 1/\cos\theta)$	$\rho_{\mathrm{q}} = \mathbf{0}$ $\rho_{\mathrm{u}} = \mathbf{(2c^2 - 1)/(c^2 - 1)^2}$	$\rho_{\mathrm{q}} = 0$ $\rho_{\mathrm{u}} = 1/(4\varepsilon^2) + O(1/\varepsilon)$	$\rho_{\mathrm{q}} = 0$ $\rho_{\mathrm{u}} = 2/c^2 + O(1/c^4)$

Table 1: The extreme points of our asymptotic tradeoffs. Answering a query takes time $\tilde{O}(n^{\rho_{\mathrm{q}}})$, updates take $\tilde{O}(n^{\rho_{\mathrm{u}}})$ operations, and the space/preprocessing complexities are $\tilde{O}(n^{1+\rho_{\mathrm{u}}})$. Lower order terms which tend to 0 as $d, n \to \infty$ are omitted for clarity. The colors match those used in Figure 2.

Euclidean inner product. We denote the unit sphere in $\mathbb{R}^d$ by $\mathcal{S}^{d-1} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$ and half-spaces by $\mathcal{H}_{\boldsymbol{u},\alpha} := \{\boldsymbol{x} \in \mathbb{R}^d : \langle \boldsymbol{u}, \boldsymbol{x} \rangle \geq \alpha\}$. For constants $\alpha, \alpha_1, \alpha_2 \in (0,1)$ and vectors $\boldsymbol{u}, \boldsymbol{u}_1, \boldsymbol{u}_2 \in \mathcal{S}^{d-1}$ we denote spherical caps and wedges by $\mathcal{C}_{\boldsymbol{u},\alpha} := \mathcal{S}^{d-1} \cap \mathcal{H}_{\boldsymbol{u},\alpha}$ and $\mathcal{W}_{\boldsymbol{u}_1,\alpha_1,\boldsymbol{u}_2,\alpha_2} := \mathcal{S}^{d-1} \cap \mathcal{H}_{\boldsymbol{u}_1,\alpha_1} \cap \mathcal{H}_{\boldsymbol{u}_2,\alpha_2}$.

For analyzing the performance of spherical filters, we would like to know the volumes of these objects in high dimensions. The following asymptotic estimates can be found in [BDGL16, MV10], where $\gamma = \gamma(\alpha_1, \alpha_2, \theta)$ satisfies $\gamma^2 = (\alpha_1^2 + \alpha_2^2 - 2\alpha_1\alpha_2 \cos\theta)/\sin^2\theta$ and θ denotes the angle $\phi(\boldsymbol{u}_1, \boldsymbol{u}_2) := \arccos\langle \boldsymbol{u}_1, \boldsymbol{u}_2 \rangle$ between $\boldsymbol{u}_1, \boldsymbol{u}_2$.

$$\mathcal{C}(\alpha) := \frac{\mu(\mathcal{C}_{\boldsymbol{u},\alpha})}{\mu(\mathcal{S}^{d-1})} = d^{\Theta(1)} \left(\sqrt{1-\alpha^2}\right)^d, \quad \mathcal{W}(\alpha_1, \alpha_2, \theta) := \frac{\mu(\mathcal{W}_{\boldsymbol{u}_1,\alpha_1,\boldsymbol{u}_2,\alpha_2})}{\mu(\mathcal{S}^{d-1})} = d^{\Theta(1)} \left(\sqrt{1-\gamma^2}\right)^d. \tag{2}$$

2.2 Symmetric spherical filters in the dense regime.

We now continue with a brief description of the algorithm of Becker–Ducas–Gama–Laarhoven [BDGL16] for solving dense ANN on the sphere.

Initialization. Let $m = \Theta(\log d)$ and suppose that $m|d$. We partition the d coordinates into m blocks of size d/m, and for each of these m blocks of coordinates we randomly sample $t^{1/m}$ code words from $\mathcal{S}^{d/m-1}$. This results in m *subcodes* $C_1, \dots, C_m \subset \mathcal{S}^{d/m-1}$. Combining one code word from each subcode, we obtain $(t^{1/m})^m = t$ different vectors $\frac{1}{\sqrt{m}}(\boldsymbol{c}_1, \dots, \boldsymbol{c}_m) \in \mathcal{S}^{d-1}$ with $\boldsymbol{c}_i \in C_i$. We denote the resulting set of vectors by the *code* C. The ideas behind this construction are that (1) this code C behaves as a set of t random unit vectors in $\mathcal{S}^{d-1}$, where the difference with a completely random code is negligible for large parameters [BDGL16, Theorem 5.1]; and (2) the additional structure hidden in C allows us to *decode* faster than with a linear search. The parameter t will be specified later.

Preprocessing. Next, given $\mathcal{D} \subset \mathcal{S}^{d-1}$, we consider each point $\boldsymbol{p} \in \mathcal{D}$ one by one and compute its relevant filters $\mathrm{Update}(\boldsymbol{p}) := \{\boldsymbol{c} \in C : \langle \boldsymbol{p}, \boldsymbol{c} \rangle \geq \alpha\}$. Naively finding these filters by a linear search over all filters would cost time $\tilde{O}(t)$, but as described in [BDGL16] this can be done in time $O(|\mathrm{Update}(\boldsymbol{p}_i)|)$ due to the hidden additional structure in the code[3]. Finally, we store all vectors in the respective filter buckets $B_1, \dots, B_t$, where $\boldsymbol{p}$ is stored in B_j iff $\boldsymbol{c}_j \in \mathrm{Update}(\boldsymbol{p})$. The parameter α will be specified later.

Answering a query. To find neighbors for a query vector $\boldsymbol{q}$, we compute its relevant filters $\mathrm{Query}(\boldsymbol{q}) := \{\boldsymbol{c} \in C : \langle \boldsymbol{q}, \boldsymbol{c} \rangle \geq \alpha\}$ in time proportional to the size of this set. Then, we visit all these buckets in our data structure, and compare $\boldsymbol{q}$ to all vectors $\boldsymbol{p}$ in these buckets. The cost of this step is proportional to the

[3] Note that the overhead of the enumeration-based decoding algorithm [BDGL16, Algorithm 1] mainly consists of computing and sorting all blockwise inner products $\langle \boldsymbol{p}_i, \boldsymbol{c}_i \rangle$, which can be done in time $\tilde{O}(t^{1/m}) = n^{o(1)}$.

number of vectors colliding with $\boldsymbol{q}$ in these filters, and the success probability of answering a query depends on the probability that two nearby vectors are found due to a collision.

Updating the data structure (optional). In certain applications, it may further be important that one can efficiently update the data structure when $\mathcal{D}$ is changed. Inserting or removing a vector $\boldsymbol{p}$ from the buckets is done by computing $\mathrm{Update}(\boldsymbol{p})$ in time proportional to $|\mathrm{Update}(\boldsymbol{p})|$, and inserting/removing the vector from the corresponding buckets. Note that by e.g. keeping buckets sorted in lexicographic order, updates in one bucket can be done in time $\tilde{O}(\log n) = d^{O(1)}$.

Correctness. To prove that this filtering construction works for certain parameters α and t, two properties are crucial: the code C needs to be efficiently decodable, and C must be sufficiently smooth on $\mathcal{S}^{d-1}$ in the sense that collision probabilities are essentially equal to those of uniformly random codes $C \subset \mathcal{S}^{d-1}$. These two properties were proved in [BDGL16, Lemma 5.1 and Theorem 5.1] respectively.

3 Asymmetric spherical filters for sparse regimes

To convert the spherical filter construction described above to the low-density regime, we need to make sure that the overhead remains negligible in n. Note that costs $t^{1/m} = t^{1/\log d}$ are considered $n^{o(1)}$ in [BDGL16] as $t = 2^{\Theta(d)}$ and $n = 2^{\Theta(d)}$. In the sparse setting of $n = 2^{\Theta(d/\log d)}$, this may no longer be the case[4]. To overcome this potential issue, we set $m = O(\log^2 d)$, so that $t^{1/m} = n^{o(1)}$ even if $t = 2^{\Theta(d)}$. Increasing m means that the code C becomes less smooth, but a detailed inspection of the proof of [BDGL16, Thm. 5.1] shows that also for $m = \log^{O(1)} d$ the code is sufficiently smooth on $\mathcal{S}^{d-1}$.

To allow for tradeoffs between the query/update complexities, we introduce two parameters α_{q} and α_{u} for querying and updating the database. This means that we redefine $\mathrm{Query}(\boldsymbol{q}) := \{\boldsymbol{c} \in C : \langle \boldsymbol{q}, \boldsymbol{c} \rangle \geq \alpha_{\mathrm{q}}\}$ and $\mathrm{Update}(\boldsymbol{p}) := \{\boldsymbol{c} \in C : \langle \boldsymbol{p}, \boldsymbol{c} \rangle \geq \alpha_{\mathrm{u}}\}$ where $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}} \in (0, 1)$ are to be chosen later. Smaller parameters mean that more filters are contained in these sets, so intuitively $\alpha_{\mathrm{q}} < \alpha_{\mathrm{u}}$ means more time is spent on queries, while $\alpha_{\mathrm{q}} > \alpha_{\mathrm{u}}$ means more time is spent on updates and less time on queries.

3.1 Random sparse instances.

For studying the sparse regime of $n = 2^{o(d)}$, we will consider the *random* model of [AR15a, AIL+15] defined as follows.

Definition 1 (Random θ-ANN in the sparse regime). *Given an angle $\theta \in (0, \frac{1}{2}\pi)$, a query $\boldsymbol{q} \in \mathbb{R}^d$, and a data set $\mathcal{D}$ of size $n = 2^{o(d)}$, the θ-ANN problem is defined as the problem of either finding a point $\boldsymbol{p} \in \mathcal{D}$ with $\phi(\boldsymbol{p}, \boldsymbol{q}) \leq \frac{1}{2}\pi$, or concluding that w.h.p. no vector $\boldsymbol{p} \in \mathcal{D}$ exists with $\phi(\boldsymbol{p}, \boldsymbol{q}) \leq \theta$.*

Note that for angles $\psi = \frac{1}{2}\pi - \delta$ where $\delta > 0$ is fixed independently of d and n, a random point $\boldsymbol{u}$ on the sphere *covers* a fraction $(1 - O(\delta^2))^d = 2^{-\Theta(d)}$ of the sphere with points at angle at most ψ from $\boldsymbol{u}$. Together, $n = 2^{o(d)}$ points therefore cover a fraction of at most $t \cdot 2^{-\Theta(d)} = 2^{-\Theta(d)}$ of the sphere. For a query $\boldsymbol{q}$ sampled uniformly at random from the sphere, with high probability it is far away from all n points, i.e. at angle at least ψ from $\mathcal{D}$. In other words, we expect that there is a significant gap between the angle with the (planted) nearest neighbor, and the angle with all other vectors, in which case solving ANN with small approximation factor is actually equivalent to solving the exact NN problem.

If we were to define the notion of "far away" as being at some angle $\psi < \frac{1}{2}\pi$ from $\boldsymbol{q}$, and we somehow expect that a significant part of the data set lies at angle at most ψ from $\boldsymbol{q}$, then the data set and queries are apparently concentrated on one part of the sphere and their distribution is not spherically symmetric. If this is indeed the case, then this knowledge may be exploited using data-dependent ANN techniques [WSSJ14], and such techniques may be preferred over data-independent filters.

[4] For even sparser data sets, we can always first apply a dimension reduction using the Johnson-Lindenstrauss transform [JL84] to transform the points to d'-dimensional vectors with $n = 2^{\Omega(d'/\log d')}$, and without significantly distorting inter-point distances.

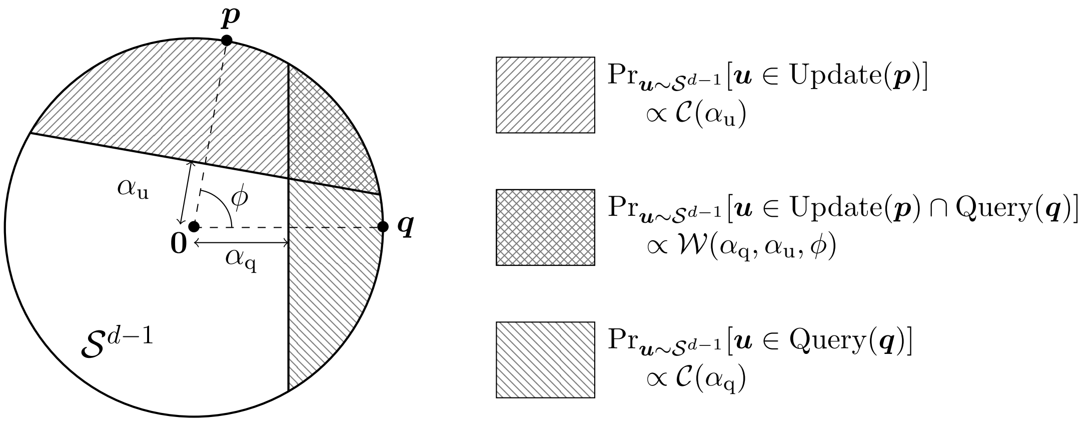
$\boldsymbol{p}$
$\boldsymbol{q}$
$\mathbf{0}$
α_{u}
ϕ
α_{q}
$\mathcal{S}^{d-1}$
$\Pr_{\boldsymbol{u}\sim\mathcal{S}^{d-1}}[\boldsymbol{u} \in \mathrm{Update}(\boldsymbol{p})] \propto \mathcal{C}(\alpha_{\mathrm{u}})$
$\Pr_{\boldsymbol{u}\sim\mathcal{S}^{d-1}}[\boldsymbol{u} \in \mathrm{Update}(\boldsymbol{p}) \cap \mathrm{Query}(\boldsymbol{q})] \propto \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \phi)$
$\Pr_{\boldsymbol{u}\sim\mathcal{S}^{d-1}}[\boldsymbol{u} \in \mathrm{Query}(\boldsymbol{q})] \propto \mathcal{C}(\alpha_{\mathrm{q}})$

Figure 1: The geometry of spherical filters. A vector $\boldsymbol{p}$ is inserted into/deleted from a filter $\boldsymbol{u}$ with probability proportional to $\mathcal{C}(\alpha_{\mathrm{u}})$, over the randomness of sampling $\boldsymbol{u}$ at random from $\mathcal{S}^{d-1}$; a filter $\boldsymbol{u}$ is queried for nearest neighbors for $\boldsymbol{q}$ with probability $\mathcal{C}(\alpha_{\mathrm{q}})$; and a vector $\boldsymbol{p}$ at angle ϕ from $\boldsymbol{q}$ is found as a candidate nearest neighbor in one of the filters with probability proportional to $\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \phi)$.

3.2 Main result.

Before describing the analysis that leads to the optimized parameter choices, we state the main result for the random, sparse setting described above, in terms of the nearby angle θ.

Theorem 1. *Let $\theta \in (0, \frac{1}{2}\pi)$ and let $\beta \in [\cos\theta, 1/\cos\theta]$. Then using parameters $\alpha_{\mathrm{q}} = \beta\sqrt{(2\log n)/d}$ and $\alpha_{\mathrm{u}} = \sqrt{(2\log n)/d}$ we can solve the θ-ANN problem on the sphere with query/update exponents:*

$$\rho_{\mathrm{q}} = \left(\frac{1-\beta\cos\theta}{\sin\theta}\right)^2 + O\left(\frac{1}{\log d}\right), \qquad \rho_{\mathrm{u}} = \left(\frac{\beta-\cos\theta}{\sin\theta}\right)^2 + O\left(\frac{1}{\log d}\right). \tag{3}$$

The resulting algorithm has a query time complexity $\tilde{O}(n^{\rho_{\mathrm{q}}})$, an update time complexity $\tilde{O}(n^{\rho_{\mathrm{u}}})$, a preprocessing time complexity $\tilde{O}(n^{1+\rho_{\mathrm{q}}})$, and a total space complexity of $\tilde{O}(n^{1+\rho_{\mathrm{q}}})$.

This result can equivalently be expressed in terms of c, by replacing $\theta = \arccos(1 - 1/c^2)$. In that case, $\theta \in (0, \frac{1}{2}\pi)$ translates to $c \in (1, \infty)$, the interval for β becomes $\beta \in [\frac{c^2-1}{c^2}, \frac{c^2}{c^2-1}]$, and we get

$$\rho_{\mathrm{q}} = \frac{(\beta(1-c^2)+c^2)^2}{2c^2-1} + O\left(\frac{1}{\log d}\right), \qquad \rho_{\mathrm{u}} = \frac{(1-c^2+\beta c^2)^2}{2c^2-1} + O\left(\frac{1}{\log d}\right). \tag{4}$$

Due to the simple dependence of these expressions on β, we can easily compute β as a function of ρ_{u} and θ (or c), and substitute this expression for β into ρ_{q} to express ρ_{q} in terms of ρ_{u} and θ (or c):

$$\sqrt{\rho_{\mathrm{q}}} = \sin\theta - \sqrt{\rho_{\mathrm{u}}} \cdot \cos\theta + O\left(\frac{1}{\log d}\right) = \frac{1}{c^2}\left(\sqrt{2c^2-1} - \sqrt{\rho_{\mathrm{u}}} \cdot (c^2-1)\right) + O\left(\frac{1}{\log d}\right). \tag{5}$$

From these expressions we can derive both Table 1 and Figure 2 by substituting appropriate values for $\rho_{\mathrm{q}}, \rho_{\mathrm{u}}, \beta$ and computing Taylor expansions around $c = 1$ and $c = \infty$.

3.3 Cost analysis.

We now proceed with a proof of Theorem 1, by analyzing the costs of different steps of the filtering process in terms of the spherical cap heights α_{q} and α_{u} and the angle of nearby vectors θ, and optimizing the parameters accordingly. The analysis will be done in terms of θ.

Updating the data structure. The probability that a filter is considered for updates is equal to the probability that $\langle \boldsymbol{p}, \boldsymbol{c} \rangle \geq \alpha_{\mathrm{u}}$ for random $\boldsymbol{c}$, which is proportional to $\mathcal{C}(\alpha_{\mathrm{u}})$ (cf. Figure 1). The size of $\mathrm{Update}(\boldsymbol{p}) \subseteq C$ and the time required to compute this set with efficient decoding [BDGL16, Algorithm 1] are of the order $t \cdot \mathcal{C}(\alpha_{\mathrm{u}})$. The total preprocessing time comes down to repeating this procedure n times, and the total space complexity is also equal to $n \cdot t \cdot \mathcal{C}(\alpha_{\mathrm{u}})$. (We only store non-empty buckets.)

Answering a query. The probability that a filter is considered for query $\boldsymbol{q}$ is of the order $\mathcal{C}(\alpha_{\mathrm{q}})$ (cf. Figure 1), and the size of $\mathrm{Query}(\boldsymbol{p})$ is of the order $t \cdot \mathcal{C}(\alpha_{\mathrm{q}})$. After finding the relevant buckets, we go through a number of collisions with distant vectors before (potentially) finding a near neighbor. The probability that distant vectors collide in a filter is proportional to $\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \frac{1}{2}\pi)$ (cf. Figure 1), so the number of comparisons for all t filters and all n distant vectors is $\tilde{O}(n \cdot t \cdot \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \frac{1}{2}\pi))$.

Choosing the number of filters. Note that the probability that a nearby vector at angle at most θ from a query $\boldsymbol{q}$ collides with $\boldsymbol{q}$ in a random filter is proportional to $\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$. By the union bound, the probability that two nearby vectors collide in at least one filter is at least $t \cdot \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$. To make sure that nearby vectors are found with constant probability (say 90%), we set $t \propto 1/\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$. With the choice of t fixed in terms of $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta$ and the above cost analysis in mind, the following table gives an overview of the asymptotic costs of spherical filtering in random, sparse settings.

Quantity	**Costs for general** $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta$
Time: Finding relevant filters for a query	$\mathcal{C}(\alpha_{\mathrm{q}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Comparing a query with colliding vectors	$n \cdot \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \frac{1}{2}\pi) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Finding relevant filters for an update	$\mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Preprocessing the data	$n \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Space: Storing all filter entries	$n \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$

3.4 Balancing the query costs.

Next, note that answering a query consists of two steps: find the α_{q}-relevant filters, and go through all candidate near neighbors in these buckets. To obtain an optimal balance between these costs (with only a polynomial difference in d in the time complexities), we must have $\mathcal{C}(\alpha_{\mathrm{q}}) = d^{O(1)} \cdot n \cdot \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \frac{1}{2}\pi)$. Raising both sides to the power $2/d$, this is equivalent to $1 - \alpha_{\mathrm{u}}^2 = d^{O(1/d)} n^{2/d} (1 - \alpha_{\mathrm{q}}^2 - \alpha_{\mathrm{u}}^2)$. Isolating α_{u} and noting that $n^{1/d} = \exp O(\frac{1}{\log d})$, this leads to:

$$\alpha_{\mathrm{u}} = d^{O(1/d)} \sqrt{\frac{n^{2/d} - 1}{n^{2/d}}} = \sqrt{\frac{2 \log n}{d}} \left(1 + O\left(\frac{1}{\log d}\right)\right). \tag{6}$$

This choice of α_{u} guarantees that the query costs are balanced. As α_{q} will have a similar scaling to α_{u}, we set $\alpha_{\mathrm{q}} = \beta \cdot \alpha_{\mathrm{u}}$ for β to be chosen later. Note that $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}} = o(1)$ implies that the corresponding spherical caps (cf. Figure 1) are *almost-hemispheres*, similar to spherical LSH [AR15a]. However, in our case the parameters scale as $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}} = O(1/\sqrt{\log d})$, compared to $\alpha = O(1/\sqrt[4]{d})$ in [AR15a].

Explicit costs. With α_{u} fixed and the relation between α_{q} and α_{u} expressed in terms of β, we now evaluate the costs for large d and $n = \exp O(\frac{d}{\log d})$ in terms of β and θ. Using Taylor expansions we get:

$$\frac{\log \mathcal{C}(\alpha_{\mathrm{q}})}{\log n} = -\beta^2 + O\left(\frac{1}{\log d}\right), \qquad \frac{\log \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)}{\log n} = -\frac{1 + \beta^2 - 2\beta \cos\theta}{\sin^2 \theta} + O\left(\frac{1}{\log d}\right), \tag{7}$$

$$\frac{\log \mathcal{C}(\alpha_{\mathrm{u}})}{\log n} = -1 + O\left(\frac{1}{\log d}\right), \qquad \frac{\log \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \frac{1}{2}\pi)}{\log n} = -1 - \beta^2 + O\left(\frac{1}{\log d}\right). \tag{8}$$

Combining these expressions, we can derive asymptotics for all of the costs related to the filtering algorithm. For the query/update exponents we then obtain the expressions given in Theorem 1.

3.5 Optimal parameter range.

Note that the best complexities are obtained by choosing $\beta \in [\cos\theta, 1/\cos\theta]$; beyond this range, complexities are strictly worse. Taking inverses in this range, we get:

$$\beta = \frac{1 - \sqrt{\rho_{\mathrm{q}}} \cdot \sin\theta}{\cos\theta} + O\left(\frac{1}{\log d}\right) = \cos\theta + \sqrt{\rho_{\mathrm{u}}} \cdot \sin\theta + O\left(\frac{1}{\log d}\right). \tag{9}$$

Isolating $\sqrt{\rho_{\mathrm{q}}}$ then leads to (1), while (9) also shows how to choose β to achieve given complexities.

4 Asymmetric spherical filters for dense regimes

We now revisit the dense regime of data sets of size $n = 2^{\Theta(d)}$, as previously analyzed in [BDGL16] for symmetric ANN. We will again use two parameters $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}} \in [0, 1]$ where the optimization now leads to a slightly different, more refined result, depending on the chosen tradeoff.

4.1 Random dense instances.

To study the dense regime, we consider the following model.

Definition 2 (Random θ-ANN in the dense regime)**.** *Given an angle $\theta \in (0, \frac{1}{2}\pi)$, a query $\boldsymbol{q} \in \mathbb{R}^d$, and a data set $\mathcal{D}$ of $n = 2^{\Theta(d)}$ points sampled uniformly at random from $\mathcal{S}^{d-1}$, the random θ-ANN problem is defined as the problem of either finding a point $\boldsymbol{p} \in \mathcal{D}$ with $\phi(\boldsymbol{p}, \boldsymbol{q}) \leq \theta$, or concluding that with high probability no vector $\boldsymbol{p} \in \mathcal{D}$ exists with $\phi(\boldsymbol{p}, \boldsymbol{q}) \leq \theta$.*

At first sight, the above definition does not seem to correspond to an *approximate*, but to an *exact* nearest neighbor instance. However, we made a critical assumption on $\mathcal{D}$ here: we assume that these points are sampled uniformly at random from the sphere. This seems to be a natural assumption in various applications (see e.g. [Her15, Laa15, MO15]), and this implies that in fact many of the points in the data set lie at angle approximately $\frac{1}{2}\pi$ from $\boldsymbol{q}$. As a result the problem is significantly easier than e.g. the worst-case ANN setting with $c \approx 1$, where the entire data set might lie at angle $\theta + \delta$ from $\boldsymbol{q}$.

For comparing this problem with the sparse setting of Section 3, observe that this dense problem is *harder*: uniformly random points on the sphere (of which roughly half has angle less than $\frac{1}{2}\pi$ with $\boldsymbol{q}$) are more likely to cause collisions than orthogonal points to $\boldsymbol{q}$. The number of collisions with distant vectors will therefore increase, and we expect the query and update exponents to be larger. This was also observed in e.g. [BDGL16, BL15, Laa15, LdW15], where the exponent for lattice sieving with other ANN techniques would have been smaller if one could only assume that far away means orthogonal.

Note that we could also extend the analysis of Section 3 to the dense regime simply by fixing the distant angle at $\psi = \frac{\pi}{2}$. In that case, similar to [BDGL16], both the query and update exponents will become smaller compared to the low-density regime as the problem becomes easier. However, such an assumption would imply that the data set is not spherically symmetric and is concentrated on one part of the sphere, in which case data-dependent methods may be preferred [LRU14, WSSJ14].

4.2 Density reduction and the critical density.

As $\mathcal{D}$ is sampled at random from $\mathcal{S}^{d-1}$, a point $\boldsymbol{p} \in \mathcal{D}$ is close to $\boldsymbol{q}$ with probability proportional to $(\sin\theta)^d$. With n points, we expect to find approximately $n \cdot (\sin\theta)^d$ nearby neighbors $\boldsymbol{p} \in \mathcal{D}$. For $n \ll (\sin\theta)^{-d}$, nearby vectors are rare, and we are essentially solving the exact (decisional) nearest neighbor problem with high probability. On the other hand, if $n \gg (\sin\theta)^{-d}$, then we expect there to be many ($n^{O(1)}$) solutions $\boldsymbol{p} \in \mathcal{D}$ with $\phi(\boldsymbol{p}, \boldsymbol{q}) \leq \theta$.

In our analysis we will focus on the case where $n = \tilde{O}((\sin\theta)^{-d})$: there might not be any near neighbors in $\mathcal{D}$ at all, but if there is one, we want to find it. For the regime $n \gg (\sin\theta)^{-d}$, we can reduce this problem to a regime with a lower density $n' = \tilde{O}((\sin\theta)^{-d})$ through a simple transformation:

- Randomly select a subset $\mathcal{D}' \subset \mathcal{D}$ of size $n' = \tilde{O}((\sin\theta)^{-d})$.
- Run the (approximate) nearest neighbor algorithm on this subset $\mathcal{D}'$.

By choosing the hidden factor inside n' sufficiently large, with high probability there will be a solution in this smaller subset $\mathcal{D}'$ as well, which our algorithm will find with high probability. This means that in our cost analysis, we can then simply replace n by n' to obtain the asymptotic complexities after this density reduction step. We denote the regime $n \propto (\sin\theta)^{-d}$ as the *critical density.*

Note that if we are given a random data set of size n, then we expect the nearest neighbor to a random query $\boldsymbol{q} \in \mathcal{S}^{d-1}$ to lie at angle $\theta \approx \arcsin(n^{-1/d})$ from $\boldsymbol{q}$; for larger angles we will find many solutions, while for smaller angles w.h.p. there are no solutions at all (except for *planted* near neighbor instances (*outliers*) which are not random on the sphere). This further motivates why the critical density is important, as $\theta \approx \arcsin(n^{-1/d})$ commonly corresponds to solving the *exact* nearest neighbor problem for random data sets. Setting $\theta \ll \arcsin(n^{-1/d})$ then corresponds to searching for outliers.

4.3 Main result.

We first state the main result for the random, dense setting described above without making any assumptions on the density. A derivation of Theorem 2 can be found in Appendix B.

Theorem 2. *Let $\theta \in (0, \frac{1}{2}\pi)$ and let $\beta \in [\cos\theta, 1/\cos\theta]$. Then using parameters $\alpha_{\mathrm{q}} = \beta\sqrt{1 - n^{-2/d}}$ and $\alpha_{\mathrm{u}} = \sqrt{1 - n^{-2/d}}$ we can solve the dense θ-ANN problem on the sphere with exponents:*

$$\rho_{\mathrm{q}} = \frac{-d}{2\log n} \log\left[1 - \left(1 - n^{-2/d}\right)\frac{1 + \beta^2 - 2\beta\cos\theta}{\sin^2\theta}\right] + \frac{d}{2\log n}\log\left[1 - \left(1 - n^{-2/d}\right)\beta^2\right], \tag{10}$$

$$\rho_{\mathrm{u}} = \frac{-d}{2\log n} \log\left[1 - \left(1 - n^{-2/d}\right)\frac{1 + \beta^2 - 2\beta\cos\theta}{\sin^2\theta}\right] - 1. \tag{11}$$

Note that in the limit of $n^{1/d} \to 1$, corresponding to sparse data sets, we obtain the expressions from Theorem 1. This matches our intuition that taking n points uniformly at random from the sphere with $n = 2^{o(d)}$ roughly means that all points have angle $\psi = \frac{1}{2}\pi$ with a query $\boldsymbol{q}$, as described in Section 3.1.

4.4 Critical densities.

As a special case of the above result, we focus on the regime of $n \propto (\sin\theta)^{-d}$. The following result shows the complexities obtained after substituting this density into Theorem 2.

Corollary 1. *Let $\theta \in (0, \frac{1}{2}\pi)$, let $\beta \in [\cos\theta, 1/\cos\theta]$, and let $n = (1/\sin\theta)^d$. Then using parameters $\alpha_{\mathrm{q}} = \beta\cos\theta$ and $\alpha_{\mathrm{u}} = \cos\theta$, the complexities in Theorem 2 reduce to:*

$$n^{\rho_{\mathrm{q}}} = \left(\frac{\sin^2\theta\,(\beta\cos\theta + 1)}{\beta\cos\theta - \cos 2\theta}\right)^{d/2}, \qquad n^{\rho_{\mathrm{u}}} = \left(\frac{\sin^2\theta}{1 - \cot^2\theta\,(\beta^2 - 2\beta\cos\theta + 1)}\right)^{d/2}. \tag{12}$$

To obtain further intuition into the above results, let us consider the limits obtained by setting $\beta \in \{\cos\theta, 1, 1/\cos\theta\}$. For $\beta = \cos\theta$, we obtain exponents of the order:

$$n \propto (\sin\theta)^{-d}, \quad \beta = \cos\theta, \qquad \Longrightarrow \qquad \rho_{\mathrm{u}} = 0, \quad \rho_{\mathrm{q}} = \frac{\log 2 - \log(3 + 2\cos(2\,\theta))}{2\log(\sin\theta)}. \tag{13}$$

Balancing the complexities is done by setting $\beta = 1$, in which case we obtain the asymptotic expressions:

$$n \propto (\sin\theta)^{-d}, \quad \beta = 1, \qquad \Longrightarrow \qquad \rho_{\mathrm{q}} = \rho_{\mathrm{u}} = \frac{2\log(\tan\frac{\theta}{2}) + \log(2\cos\theta + 1)}{2\log(\sin\theta)} - 1. \tag{14}$$

To minimize the query complexity, we would ideally set $\beta = 1/\cos\theta$, as then the query exponent approaches 0 for large n. However, one can easily verify that substituting $\beta = 1/\cos\theta$ into (12) leads to a denominator of 0, i.e. the update costs and the space complexities blow up as β approaches $1/\cos\theta$. To get an idea of how the update complexity scales in terms of the query complexity, we set $\rho_{\mathrm{q}} = \delta$ and we compute a Taylor expansion around $\delta = 0$ for ρ_{u} to obtain:

$$n \propto (\sin\theta)^{-d}, \quad \beta = 1 \qquad \Longrightarrow \qquad \rho_{\mathrm{q}} = \delta, \quad \rho_{\mathrm{u}} = \frac{\log\left(8\,\delta\log(1/\sin\theta)\tan^2\theta\right)}{2\log(\sin\theta)} - 1 + O(\delta). \tag{15}$$

In other words, for fixed angles θ, to achieve $\rho_{\mathrm{q}} = \delta$ the parameter ρ_{u} scales as $\log(1/\delta)$. Note that except for the latter result, we can substitute $\cos\theta = 1 - 1/c^2$ and $c = 1 + \varepsilon$, and compute a Taylor series expansion of these expressions around $\varepsilon = 0$ to obtain the expressions in Table 1 for small $c = 1 + \varepsilon$. This matches our intuition that $\theta \to \frac{1}{2}\pi$ for random data sets corresponds to $\theta \approx \frac{1}{2}\pi$ for sparse settings.

Finally, we observe that substituting $\theta = \frac{1}{3}\pi$, for a minimum space complexity we obtain $\rho_{\mathrm{q}} = \log(\frac{5}{4})/\log(\frac{4}{3})$, while balancing both costs leads to $\rho_{\mathrm{q}} = \rho_{\mathrm{u}} = \log(\frac{9}{8})/\log(\frac{4}{3})$. These results match those derived in [BDGL16] for the application to sieving for finding shortest lattice vectors.

5 Discussion and open problems

We conclude this work with a brief discussion on how the described methods can possibly be extended and modified to solve other problems and to obtain a better performance in practical applications.

Probing sequences. In LSH, a common technique to reduce the space complexity, or to reduce the number of false negatives, is to use probing [AIL+15, LJW+07, Pan06]: one does not only check *exact* matches in the hash tables for reductions, but also *approximate* matches which are still more likely to contain near neighbors than random buckets. Efficiently being able to define a *probing sequence* of all buckets, in order of likelihood of containing near neighbors, can be useful both in theory and in practice.

For spherical filters, an optimal probing sequence is obtained by sorting the filters (code words) according to their inner products with the target vector. Due to the large number of buckets t, computing and sorting all filter buckets is too costly, but for practical applications we can do the following. We first choose a sequence $1 = \alpha_0 > \alpha_1 > \cdots > \alpha_T$, and then given a target $\boldsymbol{t}$ we apply our decoding algorithm to find all code words $\boldsymbol{c} \in C$ with $\langle \boldsymbol{c}, \boldsymbol{t}\rangle \in (\alpha_1, \alpha_0]$. The corresponding buckets are the most likely to contain nearby vectors. If this does not result in a nearest neighbor, we apply our decoding algorithm to find code words $\boldsymbol{c} \in C$ with $\langle \boldsymbol{c}, \boldsymbol{t}\rangle \in (\alpha_2, \alpha_1]$, and we repeat the above procedure until e.g. we are convinced that no solution exists. For constant T, the overhead of this repeated decoding is small.

To implement the search for finding code words $\boldsymbol{c} \in C$ with $\langle \boldsymbol{c}, \boldsymbol{t}\rangle \in (\alpha_{\mathrm{low}}, \alpha_{\mathrm{u}}]$ efficiently, we can use Algorithm 1 in Appendix C. The most costly step of this decoding algorithm is computing and sorting all blockwise inner products $\langle \boldsymbol{c}_{k,j}, \boldsymbol{t}_k\rangle$, but note that these computations have to be performed only once; later calls to this function with different intervals $(\alpha_{i+1}, \alpha_i]$ can reuse these sorted lists.

Ruling out false negatives. An alternative to using probing sequences to make sure that there are no false negatives, is to construct a scheme which guarantees that there are never any false negatives at all (see e.g. [Pag16]). In the filtering framework, this corresponds to using codes C such that it is guaranteed that nearby vectors always collide in one of the filters. In other words, for each pair of points $\boldsymbol{p}, \boldsymbol{q}$ on the sphere at angle θ, the corresponding wedge $\mathcal{W}_{\boldsymbol{p},\alpha_{\mathrm{u}},\boldsymbol{q},\alpha_{\mathrm{q}}}$ must contain a code word $\boldsymbol{c} \in C$. Note that with our random construction we can only show that with high probability, this is the case.

For spherical filters, codes guaranteeing this property correspond to spherical codes such that all possible wedges $\mathcal{W}_{\boldsymbol{p},\alpha_{\mathrm{u}},\boldsymbol{q},\alpha_{\mathrm{q}}}$ for $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{S}^{d-1}$ contain at least one code word $\boldsymbol{c} \in C$. For $\alpha_{\mathrm{q}} = \alpha_{\mathrm{u}} = \alpha$, note that at the middle of such a wedge lies a point $\boldsymbol{y} = (\boldsymbol{p} + \boldsymbol{q})/\|\boldsymbol{p} + \boldsymbol{q}\|$ at angle $\theta/2$ from both $\boldsymbol{p}$ and $\boldsymbol{q}$. If a code is not covering and allows for false negatives, then there are no code words at angle $\frac{1}{2}\theta - \arccos\alpha$ from $\boldsymbol{y}$. In particular, the *covering radius* of the code (the smallest angle ψ such that spheres of angle ψ around all code words cover the entire sphere) is therefore larger than $\frac{1}{2}\theta - \arccos\alpha$. Equivalently, being able to construct spherical codes of low cardinality with covering radius at most $\frac{1}{2}\theta - \arccos \alpha$ implies being able to construct a spherical filtering scheme without false negatives.

As we make crucial use of concatenated codes $C = C_1 \times \cdots \times C_m$ to allow for efficient decoding, covering codes without efficient decoding algorithms cannot be used for C. Instead, one might aim at using such covering codes for the subcodes: if all subcodes C_i have covering radius at most $\frac{1}{2}\theta - \arccos \frac{\alpha}{m}$, then the concatenated code $C = C_1 \times \cdots \times C_m$ has a covering radius of at most $\frac{1}{2}\theta - \arccos \alpha$. Finding tight bounds on the size of a spherical code with covering radius $\frac{1}{2}\theta - \arccos \frac{\alpha}{m}$ (where θ is defined by the problem setting, and α may be chosen later) would directly lead to an upper bound on the number of filters needed to guarantee that there are no false positives.

Sparse codes for efficiency. As described in e.g. [Ach01, BDGL16, LHC06], it is sometimes possible to use sparse, rather than fully random codes without losing on the performance of a nearest neighbor scheme. Using sparse subcodes C_i might further reduce the overhead of decoding (computing blockwise inner products). For this we could either use randomly sampled sparse subcodes, but one might also consider using codes which are guaranteed to be "smooth" on the sphere and have a small covering radius. Similar to Leech lattice LSH [AI06], one might consider using vectors from the Leech lattice in 24 dimensions to define the subcodes. Asymptotically in our construction the block size d/m needs to scale with d, and fixing $d/m = 24$ would invalidate the proof of smoothness of [BDGL16, Theorem 5.1], but in practice both n and d are fixed, and only practical assessments can show whether predetermined spherical codes can be used to obtain an even better performance.

Optimality of the tradeoff. An interesting open problem for future work is determining precise bounds on the best tradeoff that one can possibly hope to achieve in the sparse regime. Since our tradeoff matches various known bounds in the regimes of small and large approximation factors c [AIP06, And09, KOR00, PTW08, PTW10] and no LSH scheme can improve upon our results in the balanced setting [AR15b, Dub10], and since the tradeoff can be described through a remarkably simple relation (especially when described in terms of θ), we conjecture that this tradeoff is optimal.

Conjecture 1. *Any algorithm for sparse θ-ANN on the sphere must satisfy* $\sqrt{\rho_{\mathrm{q}}} + \sqrt{\rho_{\mathrm{u}}} \cdot \cos\theta \geq \sin\theta$.

As a first step, one might try to (dis)prove this conjecture within the LSH framework, similar to various other works focusing on lower bounds for schemes that fall in this category [MNP07, OWZ14].

Extension to Euclidean spaces. As mentioned in the introduction, Andoni and Razenshteyn showed how to reduce ANN in Euclidean spaces to (sparse) random ANN on the sphere in the symmetric case, using LSH techniques [AR15a]. An important open problem for future work is to see whether the techniques and the reduction described in [AR15a] are compatible with locality-sensitive filters, and with asymmetric nearest neighbor techniques such as those presented in this paper. If this is possible, then our results may also be applicable to all of ℓ_2^d, rather than only to the angular distance on $\mathbb{R}^d$ or to Euclidean distances on the unit sphere $\mathcal{S}^{d-1}$.

Combination with cross-polytope LSH. Finally, the recent paper [AIL$^+$15] showed how cross-polytope hashing (previously introduced by Terasawa and Tanaka [TT07]) is asymptotically equally suitable for solving Euclidean nearest neighbor problems on the sphere (and for the angular distance) as the approach of spherical LSH of using large, completely random codes on the sphere [AR15a]. Advantages of cross-polytope LSH over spherical LSH are that they have a much smaller size (allowing for faster decoding), and that cross-polytope hash functions can be efficiently rerandomized using sparse and fast random projections such as Fast Hadamard Transforms [AIL$^+$15]. In that sense, cross-polytope LSH offers a significant practical improvement over spherical LSH.

The approach of using spherical filters is very similar to spherical LSH: large, random (sub)codes are used to define regions on the sphere. A natural question is therefore whether ideas analogous to cross-polytope hashing can be used in combination with spherical filters, to reduce the subexponential overhead in d for decoding to an overhead which is only polynomial in d. This is also left as an open problem for further research.

References

[Ach01] Dimitris Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001.

[AHL01] Helmut Alt and Laura Heinrich-Litan. Exact L_∞ nearest neighbor search in high dimensions. In *SOCG*, pages 157–163, 2001.

[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*, 2015.

[AIP06] Alexandr Andoni, Piotr Indyk, and Mihai Pătraşcu. On the optimality of the dimensionality reduction method. In *FOCS*, pages 449–458, 2006.

[AMM09] Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM*, 57(1):1:1–1:54, November 2009.

[AMN$^+$94] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *SODA*, pages 573–582, 1994.

[And09] Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible.* PhD thesis, Massachusetts Institute of Technology, 2009.

[AR15a] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015.

[AR15b] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. *Manuscript*, pages 1–15, 2015.

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

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, 2006.

[BL15] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. *Cryptology ePrint Archive, Report 2015/823*, pages 1–25, 2015.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition).* Wiley, 2000.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SOCG*, pages 253–262, 2004.

[Dub10] Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, Aug 2010.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[Her15] Gottfried Herold. Applications of nearest neighbor search techniques to the BKW algorithm (draft). 2015.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

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

[Kap15] Michael Kapralov. Smooth tradeoffs between insert and query complexity in nearest neighbor search. In *PODS*, pages 329–342, 2015.

[KOR00] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015.

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

[LHC06] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD*, pages 287–296, 2006.

[LJW$^+$07] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[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.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480, 2010.

[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.

[Pag16] Rasmus Pagh. Locality-sensitive hashing without false negatives. In *SODA*, 2016.

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

[PTW08] Rina Panigrahy, Kunal Talwar, and Udi Wieder. A geometric approach to lower bounds for approximate near-neighbor search and partial match. In *FOCS*, pages 414–423, 2008.

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

[SDI05] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005.

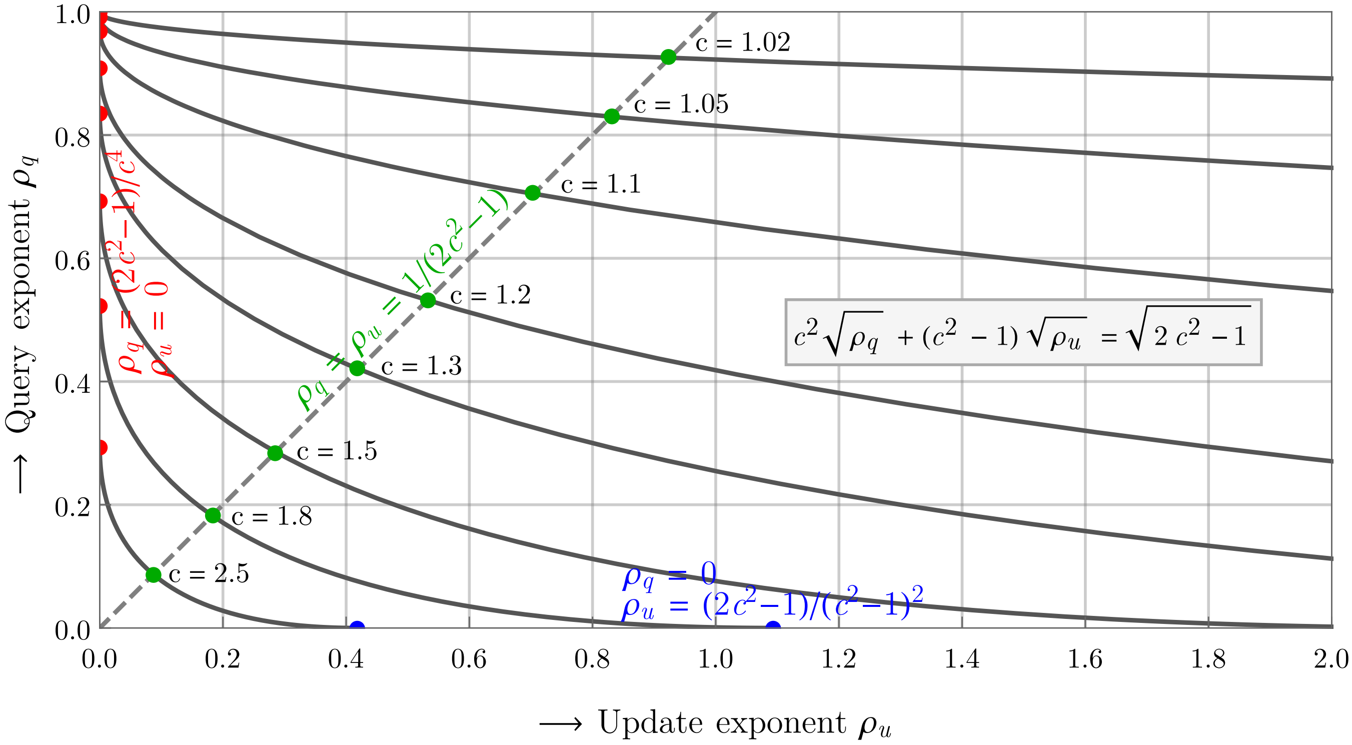

Figure 2: Tradeoffs between the query and update complexities, for various c. Informally, the x-axis represents the space and the y-axis the time (for queries). The diagonal represents symmetric ANN.

[SSLM14] Ludwig Schmidt, Matthew Sharifi, and Ignacio Lopez-Moreno. Large-scale speaker identification. In *ICASSP*, pages 1650–1654, 2014.

[STS$^+$13] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *VLDB*, 6(14):1930–1941, 2013.

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

[WSSJ14] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv:1408.2927 [cs.DS]*, pages 1–29, 2014.

A Tradeoff figure in the sparse regime

Figure 2 describes asymptotic tradeoffs for different values of c. Note that the query exponent is always smaller than 1, regardless of $c > 1$, and note that the update exponent is smaller than 1 (and the query exponent less than 1/2) for all $c > \sqrt{2 + \sqrt{2}} \approx 1.85$ corresponding to $\theta = \frac{1}{4}\pi$.

B Analysis for dense regimes

To derive the complexities for the dense regime of $n = 2^{\Theta(d)}$, we follow the same approach as for the sparse regime of $n = 2^{o(d)}$, but without making the assumption that e.g. $n^{1/d} = 1 + o(1)$.

B.1 General expressions.

First, we observe that the cost analysis described in Section 3 applies in the dense setting as well, with the modification that we no longer assume that $\boldsymbol{q}$ is orthogonal to all of $\mathcal{D}$ (except for a potential nearest neighbor). The update and query costs remain the same as before in terms of t and the volumes $\mathcal{C}(\cdot)$ and $\mathcal{W}(\cdot)$, but to obtain the number of collisions with distant vectors, we take a different angle. First, observe that each vector is added to $\tilde{O}(t \cdot \mathcal{C}(\alpha_{\mathrm{u}}))$ filters, and that we have n vectors, leading to $\tilde{O}(n \cdot t \cdot \mathcal{C}(\alpha_{\mathrm{u}}))$ total entries in the filters or $\tilde{O}(n \cdot t \cdot \mathcal{C}(\alpha_{\mathrm{u}}))$ entries in each filter[5]. For a given vector $\boldsymbol{q}$, we then query $\tilde{O}(t \cdot \mathcal{C}(\alpha_{\mathrm{q}}))$ buckets for nearest neighbors. In total, we therefore expect to find $\tilde{O}(n \cdot t \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \cdot \mathcal{C}(\alpha_{\mathrm{q}}))$ colliding vectors for a query vector. Again setting $t \propto 1/\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$ to make sure that nearby vectors are found with constant probability, we obtain the following updated table of the asymptotic costs of spherical filtering in random, dense settings.

Quantity	**Costs for** $\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta$
Time: Finding relevant filters for a query	$\mathcal{C}(\alpha_{\mathrm{q}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Comparing a query with colliding vectors	$n \cdot \mathcal{C}(\alpha_{\mathrm{q}}) \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Finding relevant filters for an update	$\mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Time: Preprocessing the data	$n \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$
Space: Storing all filter entries	$n \cdot \mathcal{C}(\alpha_{\mathrm{u}}) \ / \ \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)$

B.2 Balancing the query costs.

Next, to make sure that the query costs are balanced, and not much more time is spent on looking for relevant filters rather than actually doing comparisons, we again look for parameters such that these costs are balanced. In this case we want to solve the asymptotic equation $\mathcal{C}(\alpha_{\mathrm{q}}) = n \cdot \mathcal{C}(\alpha_{\mathrm{q}}) \cdot \mathcal{C}(\alpha_{\mathrm{u}})$ or $\mathcal{C}(\alpha_{\mathrm{u}}) = (1 - \alpha_{\mathrm{u}}^2)^{d/2} = 1/n$. Solving for α_{u} leads to $\alpha_{\mathrm{u}} = \sqrt{1 - n^{-2/d}}$ leading to the parameter choice described in Theorem 2. For now we again set $\alpha_{\mathrm{q}} = \beta \cdot \alpha_{\mathrm{u}}$ with β to be chosen later.

B.3 Explicit costs.

We now evaluate the costs for large d and $n = 2^{\Theta(d)}$, in terms of the ratio β between the two parameters α_{q} and α_{u}, and the nearby angle θ. This leads to $\mathcal{C}(\alpha_{\mathrm{u}}) = 1/n$ and:

$$\mathcal{C}(\alpha_{\mathrm{q}}) = \left(1 - (1 - n^{-2/d})\, \beta^2\right)^{d/2}, \qquad \mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta) = \left(1 - (1 - n^{-2/d})\, \frac{1 + \beta^2 - 2\beta \cos\theta}{\sin^2\theta}\right)^{d/2}. \tag{16}$$

Combining these expressions, we can then derive asymptotic estimates for all of the costs of the algorithm. For the query and update exponents $\rho_{\mathrm{q}} = \log[\mathcal{C}(\alpha_{\mathrm{q}})/\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)]/\log n$ and $\rho_{\mathrm{u}} = \log[\mathcal{C}(\alpha_{\mathrm{u}})/\mathcal{W}(\alpha_{\mathrm{q}}, \alpha_{\mathrm{u}}, \theta)]/\log n$ we then obtain:

$$\rho_{\mathrm{q}} = \frac{-d}{2 \log n} \log\left[1 - \left(1 - n^{-2/d}\right) \frac{1 + \beta^2 - 2\beta\cos\theta}{\sin^2\theta}\right] + \frac{d}{2\log n} \log\left[1 - \left(1 - n^{-2/d}\right)\beta^2\right], \tag{17}$$

$$\rho_{\mathrm{u}} = \frac{-d}{2 \log n} \log\left[1 - \left(1 - n^{-2/d}\right) \frac{1 + \beta^2 - 2\beta\cos\theta}{\sin^2\theta}\right] - 1. \tag{18}$$

These are also the expressions given in Theorem 2.

B.4 Optimal parameter range.

We observe that again the best exponents ρ_{q} and ρ_{u} are obtained by choosing $\beta \in [\cos\theta, 1/\cos\theta]$; beyond this range, the complexities are strictly worse. This completes the derivation of Theorem 2.

[5] Here a crucial property of the filters is that each filter covers a region of exactly the same size on the sphere.

Algorithm 1 EfficientIntervalDecoding$(C, \boldsymbol{t}, \alpha_{\text{low}}, \alpha_{\text{high}})$

```
Require: The description $C_1, \dots, C_m$ of the code $C$; a target vector $\boldsymbol{t} \in \mathbb{R}^d$; and $0 \leq \alpha_{\text{low}} < \alpha_{\text{high}} \leq 1$.
Ensure: Return all code words $\boldsymbol{c} \in C$ with $\langle \boldsymbol{t}, \boldsymbol{c} \rangle \in (\alpha_{\text{low}}, \alpha_{\text{high}}]$
Sort each list $C_k$ by decreasing dot-products $d_{k,j} = \langle \boldsymbol{t}_k, \boldsymbol{c}_{k,j} \rangle$ with $\boldsymbol{t}_k$.
Precompute $m$ bounds $L_k = \alpha_{\text{low}} - \sum_{i=k+1}^{m} d_{i,t^{1/m}}$.
Precompute $m$ bounds $U_k = \alpha_{\text{high}} - \sum_{i=k+1}^{m} d_{i,1}$.
Initialize an empty output set $S \leftarrow \emptyset$.
Compute the lower bound $\ell_1 = \min\{j_1 : d_{1,j_1} > L_1\}$.          ▷ do a binary search over $[1, t^{1/m}]$
Compute the upper bound $u_1 = \max\{j_1 : d_{1,j_1} \leq U_1\}$.          ▷ do a binary search over $[1, t^{1/m}]$
for each $j_1 \in \{\ell_1, \dots, u_1\}$ do
    Compute the lower bound $\ell_2 = \min\{j_2 : d_{2,j_2} > L_2 - d_{1,j_1}\}$.
    Compute the upper bound $u_2 = \max\{j_2 : d_{2,j_2} \leq U_2 - d_{1,j_1}\}$.
    for each $j_2 \in \{\ell_2, \dots, u_2\}$ do
        [...]
        Compute the lower bound $\ell_m = \min\{j_m : d_{m,j_m} > L_m - \sum_{k=1}^{m-1} d_{k,j_k}\}$.
        Compute the upper bound $u_m = \max\{j_m : d_{m,j_m} \leq U_m - \sum_{k=1}^{m-1} d_{k,j_k}\}$.
        for each $j_m \in \{\ell_m, \dots, u_m\}$ do
            Add the code word $\boldsymbol{c} = (\boldsymbol{c}_{1,j_1}, \dots, \boldsymbol{c}_{m,j_m})$ to $S$.
        end for
        [...]
    end for
end for
return $S$
```

C Interval decoding

Algorithm 1 describes how to perform list-decoding for intervals, which may be relevant in practice for e.g. computing probing sequences as described in Section 5. The algorithm is based on [BDGL16, Algorithm 1], where now two sets of bounds are maintained to make sure that we only consider solutions which lie within the given range, rather than above a threshold. The bounds L_k and U_k indicate the minimum and maximum sum of inner products that can still be obtained in the last $m - k$ sorted lists of vectors and inner products; if in the nested for-loops, the current sum of inner products $\sum_i d_{i,j_i}$ is not in the interval $(L_k, U_k]$, then there are no solutions anymore in the remaining part of the tree. Conversely, if this sum of inner products does lie in the interval, then there must be at least one solution.