Polytopes, Lattices, and Spherical Codes for the Nearest Neighbor Problem

Thijs Laarhoven
Eindhoven University of Technology, The Netherlands
http://www.thijs.com/
mail@thijs.com

Abstract

We study locality-sensitive hash methods for the nearest neighbor problem for the angular distance, focusing on the approach of first projecting down onto a random low-dimensional subspace, and then partitioning the projected vectors according to the Voronoi cells induced by a well-chosen spherical code. This approach generalizes and interpolates between the fast but asymptotically suboptimal hyperplane hashing of Charikar [STOC 2002], and asymptotically optimal but practically often slower hash families of e.g. Andoni–Indyk [FOCS 2006], Andoni–Indyk–Nguyen–Razenshteyn [SODA 2014] and Andoni–Indyk–Laarhoven–Razenshteyn–Schmidt [NIPS 2015]. We set up a framework for analyzing the performance of any spherical code in this context, and we provide results for various codes appearing in the literature, such as those related to regular polytopes and root lattices. Similar to hyperplane hashing, and unlike e.g. cross-polytope hashing, our analysis of collision probabilities and query exponents is *exact* and does not hide any order terms which vanish only for large d, thus facilitating an easier parameter selection in practical applications.

For the two-dimensional case, we analytically derive closed-form expressions for arbitrary spherical codes, and we show that the equilateral triangle is optimal, achieving a better performance than the two-dimensional analogues of hyperplane and cross-polytope hashing. In three and four dimensions, we numerically find that the tetrahedron and 5-cell (the 3-simplex and 4-simplex) and the 16-cell (the 4-orthoplex) achieve the best query exponents, while in five or more dimensions orthoplices appear to outperform regular simplices, as well as the root lattice families A_k and D_k in terms of minimizing the query exponent. We provide lower bounds based on spherical caps, and we predict that in higher dimensions, larger spherical codes exist which outperform orthoplices in terms of the query exponent, and we argue why using the D_k root lattices will likely lead to better results in practice as well (compared to using cross-polytopes), due to a better trade-off between the asymptotic query exponent and the concrete costs of hashing.

2012 ACM Subject Classification Theory of computation → Nearest neighbor algorithms; Theory of computation → Random projections and metric embeddings; Theory of computation → Computational geometry

Keywords and phrases (approximate) nearest neighbor problem, spherical codes, polytopes, lattices, locality-sensitive hashing (LSH)

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.76

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/1907.04628.

Funding The author is supported by an NWO Veni grant with project number 016.Veni.192.005. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing at the University of California, Berkeley.

1 Introduction

Given a large database of high-dimensional vectors, together with a target data point which does not lie in this database, a natural question to ask is: which item in the database is the

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 76; pp. 76:1–76:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


most similar to the query? And can we somehow preprocess and store the database in a data structure that allows such queries to be answered faster? These and related questions have long been studied in various contexts, such as machine learning, coding theory, pattern recognition, and cryptography [14, 19, 20, 25, 30, 32], under the headers of *similarity search* and *nearest neighbor searching*. Observe that a naive solution might consist of simply storing the data set in a big list, and to search this list in linear time to find the nearest neighbor to a query point. This solution requires an amount of time and space scaling linearly in the size of the data set, and solutions we are interested in commonly require more preprocessed space (and time), but achieve a sublinear query time complexity to find the nearest neighbor.

Depending on the context of the problem, different solutions for these problems have been proposed and studied. For the case where the dimensionality of the original problem is constant, efficient solutions are known to exist [8]. Throughout the remainder of the paper, we will therefore assume that the dimensionality of the data set is superconstant. In the late 1990s, Indyk–Motwani [22] proposed the *locality-sensitive hashing* framework, and until today this method remains one of the most prominent and popular methods for nearest neighbor searching in high-dimensional vector spaces, both due to its asymptotic performance when using theoretically optimal hash families [4, 5], and due to its practical performance when instantiated with truly efficient locality-sensitive hash functions [1,13,16]. And whereas many other methods scale poorly as the dimensionality of the problem increases, locality-sensitive hashing remains competitive even in high-dimensional settings.

Although solutions for both asymptotic and concrete settings have been studied over the years, there is often a separation between both worlds: some methods work well in practice but do not scale optimally when the parameters increase (e.g. Charikar's hash family [16]); while some other methods are known to achieve a superior performance for sufficiently large parameter sizes, but may not be quite as practical in some applications due to large hidden order terms in the asymptotic analysis (e.g. hash families studied in [3–5] and filter families from [6, 11, 17]). Moreover, the latter methods are often not as easy to deploy in practice due to these unspecified hidden order terms, making it hard to choose the scheme parameters that optimize the performance. A key problem in this area thus remains to close the gap between theory and practice, and to offer solutions that interpolate between *quick–and–dirty* simple approaches that might not scale well with the problem size, and more sophisticated methods that only start outperforming these simpler methods as the parameters are sufficiently large.

1.1 Related work

In this paper we will focus on methods for solving the nearest neighbor problem for the *angular distance*, which is closely related to the nearest neighbor problem in various ℓ_p-norms: as shown by Andoni and Razenshteyn [7], a solution for the nearest neighbor problem for the angular distance (or the Euclidean distance on the sphere) can be optimally extended to a solution for the ℓ_2-norm for all of $\mathbb{R}^d$. Solutions for the ℓ_2-norm can further be translated to e.g. solutions for the ℓ_1-norm via appropriate embeddings.

For the angular distance, perhaps the most well-known and widely deployed approach for finding similar items in a large database is to use the hyperplane hash family of Charikar [16]. For spherically-symmetric, random data sets on the sphere of size n, it can find a nearest neighbor at angle at most e.g. $\theta = \frac{\pi}{3}$ in sublinear time $\tilde{O}(n^\rho)$ and space $\tilde{O}(n^{1+\rho})$, with $\rho \approx 0.5850$. Various improvements later showed that smaller query exponents ρ can be achieved in sufficiently high dimensions [3, 5], the most practical of which is based on cross-polytopes or orthoplices [4, 24, 35]: for large d and for the same target angle $\theta = \frac{\pi}{3}$, the query time complexity scales as $n^{\rho+o(1)}$ with $\rho = 1/3$. Note that the convergence to the limit is

rather slow [4, Theorem 1] and depends on both d and n being sufficiently large – for fixed d and large n, the exponent is still larger than $1/3$. In certain practical applications with moderate-sized data sets, hyperplane hashing still appears to outperform cross-polytope hashing and other advanced hashing and filtering schemes [2, 11, 12, 28, 29] due to its low cost for hashing, and the absence of lower order terms in the exponent.

Related to the topic of this paper, various other works have further observed the relation between finding good locality-sensitive hash families for the angular distance and finding "nice" spherical codes that partition the sphere well, and allow for efficient decoding [3,4,11,15,35,36]. The requirements on a spherical code to be a *good* spherical code are somewhat intricate to state, and to date it is an open problem to exactly quantify which spherical codes are the most suited for nearest neighbor searching. It may well be possible to improve upon the state-of-the-art orthoplex (cross-polytope) locality-sensitive hash family [4, 31] in practice with a method achieving the same asymptotic scaling, but with a faster convergence to the limit, and thus potentially a better performance in practice for large problem instances.

1.2 A framework for evaluating spherical codes

As a first contribution of this paper, we provide a framework for analyzing the performance of arbitrary spherical codes in the context of locality-sensitive hashing for the angular distance, where we focus on the approach of (1) projecting down onto a random low-dimensional subspace, and (2) partitioning the resulting subspace according to the Voronoi cells induced by a spherical code. More specifically, we relate the collision probabilities appearing in the analysis of these hash functions to so-called *orthant probabilities* of multivariate normal distributions with non-trivial correlation matrices. Below we informally state this relation for general spherical codes, which provides us a recipe for computing the collision probabilities (and the query exponent) as a function of the set of vertices $\mathcal{C}$ of the corresponding spherical code. Here a hash family being (θ, p_1, p_2)-sensitive means that uniformly random vectors on the sphere collide with probability at most p_2, and target nearest neighbors at angle at most θ from a random query vector collide in a random hash function with probability at least p_1. Further details and a more formal statement can be found in the full version.

▶ **Theorem 1** (Spherical code locality-sensitive hashing). *Let $\mathcal{C} \subset \mathcal{S}^{k-1}$ be a k-dimensional spherical code, and consider a hash family where:*

- *We first project onto a k-dimensional subspace using a random matrix $\mathbf{A} \sim \mathcal{N}(0,1)^{k \times d}$;*
- *We then assign hash values based on which $\mathbf{c} \in \mathcal{C}$ is nearest to the projected vector.*

Then for any $\theta \in (0, \frac{\pi}{2})$, this family is (θ, p_1, p_2)-sensitive, where p_2 can be expressed in terms of the relative volumes of the Voronoi cells induced by $\mathcal{C}$, and p_1 can be expressed as a sum of orthant probabilities $\Pr_{\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_i)}(\boldsymbol{z} \geq \mathbf{0})$, where each $\boldsymbol{\Sigma}_i$ has size $(2k) \times (2k)$.

The locality-sensitive hashing exponent $\rho = \log p_1 / \log p_2$ describes the distinguishing power of a hash family, as for large n this tells us that we can solve the (average-case) nearest neighbor problem with target angle θ in query time $\tilde{O}(n^\rho)$ with space $\tilde{O}(n^{1+\rho})$. The theorem above essentially provides us a recipe which, given any spherical code $\mathcal{C}$ and target angle θ as input, tells us how to compute these probabilities p_1 and p_2 (and thus ρ) exactly.

While the above theorem is somewhat abstract, we show how to explicitly construct the correlation matrices $\boldsymbol{\Sigma}_i$ appearing in the theorem, allowing us to always at least obtain numerical estimates on the performance of different spherical codes in the context of nearest neighbor searching. We further study how to reduce the dimensionality of the problem (and in particular, the sizes of the matrices $\boldsymbol{\Sigma}_i$) when the spherical code exhibits many symmetries, such as being isogonal, and we show how using only the *relevant vectors* of each code word can further simplify the computations. In most cases the resulting orthant probabilities will

still remain too complex to evaluate analytically, but in some cases we can obtain exact expressions this way.

1.3 A survey of known spherical codes

One and two dimensions. Using this new framework, in the full version we then apply it to various spherical codes, starting in very low dimensions. For the one-dimensional case we rederive the celebrated result of Charikar [16] for one-dimensional spherical codes, noting that the collision probability analysis in fact dates back to an old result from the late 1800s [33]. Applying the same framework to two-dimensional codes, among others we establish a general formula for collision probabilities and query exponents for arbitrary polygons, as well as a more compact description of the collision probabilities for regular polygons.

▶ **Theorem 2** (Collision probabilities for regular polygons). *Let $\mathcal{C} \subset \mathcal{S}^1$ consist of the vertices of the regular c-gon, for $c \geq 2$, and let $\theta \in (0, \frac{\pi}{2})$. Then the corresponding project–and–partition hash family $\mathcal{H}$ is (θ, p_1, p_2)-sensitive, with:*

$$p_1 = \frac{1}{c} + c\left(\frac{\pi - \theta}{2\pi}\right)^2 - c\left(\frac{\arccos(-\cos\theta\cos\frac{2\pi}{c})}{2\pi}\right)^2, \qquad p_2 = 1/c. \tag{1}$$

As a direct corollary of the above theorem, we establish that *triangular locality-sensitive hashing* achieves a superior asymptotic performance to hyperplane hashing, and achieves the lowest query exponents ρ among all regular polygons.

▶ **Corollary 3** (Triangular locality-sensitive hashing). *Consider the hash family where we first project onto a random plane using a random projection matrix $\mathbf{A} \sim \mathcal{N}(0,1)^{2\times d}$, and then decode to the nearest corner of a fixed equilateral triangle centered at $(0,0)$. Then, for target angles $\theta \in (0, \frac{\pi}{2})$, this hash family achieves query exponents ρ of the form:*

$$\rho = \ln\left[\frac{1}{3} + 3\left(\frac{\pi - \theta}{2\pi}\right)^2 - 3\left(\frac{\arccos(\frac{1}{2}\cos\theta)}{2\pi}\right)^2\right] \Big/ \ln\left[\frac{1}{3}\right]. \tag{2}$$

Among all such project–and–partition hash families based on regular k-gons, this family achieves the lowest values ρ for any target angle $\theta \in (0, \frac{\pi}{2})$.

Note again that the above statement of ρ is exact, and does not hide any order terms in d or n – in fact, the collision probabilities do not depend on d at all, due to our choice of $\mathbf{A}$ being Gaussian. Further note that the 2-gon in the plane corresponds to the one-dimensional antipodal code of Charikar, and triangular hashing therefore strictly improves upon hyperplane hashing for any θ in terms of the query exponent ρ. For instance, we can find neighbors at angle at most $\frac{\pi}{3}$ in time $\tilde{O}(n^\rho)$ with $\rho \approx 0.56996$, offering a concrete but minor improvement over the hyperplane hashing approach of Charikar [16] with query exponent $\rho \approx 0.5850$. This is the best we can do with any two-dimensional isogonal spherical code, and we numerically predict that this code achieves the lowest values ρ among all (not necessarily isogonal) two-dimensional spherical codes as well.

Three and four dimensions. For three-dimensional codes, through numerical integration of the resulting orthant probabilities we conclude that the *tetrahedron* appears to minimize the query exponent ρ out of all three-dimensional codes, beating the three-dimensional analogues of e.g. the cross-polytope and hypercube, as well as various sphere packings and other regular polytopes appearing in the literature, such as the Platonic and Archimedean solids. In four

dimensions an interesting phenomenon occurs: the so-called 5-*cell* (4-simplex) and 16-*cell* (4-orthoplex) are optimal in different regimes, with the 5-cell inducing a more coarse-grained partition of the space and achieving a better performance when the nearest neighbor lies relatively far away from the data set, and the 16-cell inducing a more fine-grained partition, and working better when the nearest neighbor lies relatively close to the target vector. This crossover effect is also visualized in Figure 1, and it strengthens our intuition that as the dimensionality goes up, or as the nearest neighbor lies closer to the query, more fine-grained partitions are necessary to obtain the best performance. An overview of some of the exponents ρ for various low-dimensional spherical codes, for different target angles $\theta \in \frac{\pi}{12}\{1,2,3,4,5\}$, is given in Table 1. The best exponents ρ are highlighted in bold.

Five and more dimensions. For higher-dimensional spherical codes, we obtain further improvements in the query exponents ρ through the use of suitable spherical codes, as shown in Table 1 and Figures 1–4. For dimensions 5 and 6, the corresponding orthoplices achieve a better performance than the simplices, and the exotic polytopes 1_{21} and 2_{21}, related to the root lattice E_6, seem useful in the context of nearest neighbor searching as well.

In the full version we further study the performance of the following five non-trivial infinite families of spherical codes. Color codes below correspond to the same colors used in Table 1 and Figures 1–4 to differentiate these families of codes.

- The **simplices** S_k on $k+1$ vertices;
- The **orthoplices** O_k on $2k$ vertices (also known as cross-polytopes [4, 36]);
- The **hypercubes** C_k on 2^k vertices (as studied in [27]);
- The **expanded simplices** A_k on $k(k+1)$ vertices;
- The **rectified orthoplices** D_k on $2k(k-1)$ vertices.

The latter two families are connected to the root lattices A_k and D_k, while the first three are related to the lattice $\mathbb{Z}^k$. We conjecture that, should other nice families of dense lattices be found (e.g. the recent [37]), these may give rise to suitable spherical codes in nearest neighbor applications as well. For each of the above five families we give closed-form expressions on the correlation matrices $\mathbf{\Sigma}$, but the resulting orthant probabilities that need to be evaluated for computing ρ do not appear to admit simple closed-form expressions. Apart from the family of hypercubes, these are all asymptotically optimal (with $\rho \to (1-\cos\theta)/(1+\cos\theta)$ as $k \to \infty$), although the convergence to the limit may differ for each family.

1.4 Lower bounds via spherical caps

Although this work tries to be exhaustive in covering as many (families of) spherical codes as possible, better spherical codes may exist, achieving even lower query exponents ρ. As these codes may be hard to find, and as it may be difficult to rule out the existence of other, better spherical codes, the next best thing one might hope for is a somewhat tight lower bound on the performance of any k-dimensional spherical code in our framework, which hopefully comes close to the performance of the spherical codes we have considered in this survey.

As has been established in several previous works on nearest neighbor searching on the sphere [3, 5, 6, 11, 17, 26], ideally we would like the hash regions induced by the partitions to take the shape of a spherical cap. Such a shape minimizes the angular radius, given that the region has a fixed volume, and in a sense it is the most natural and smoothest shape that a region on a sphere can take. So in a utopian world, one might hope that a spherical code partitions the sphere into c regions, and each region corresponds exactly to a spherical cap of volume $\mathrm{Vol}(\mathcal{S}^{k-1})/c$. Clearly such spherical codes do not exist for $k > 2$ and $c > 2$, but such an extremal example does give us an indication on the limits of what might be achievable in dimension k, and how the optimal ρ decreases with k. Note that random spherical codes might approach this utopian setting in high dimensions.

Following the above reasoning, and using a classic result of Baernstein–Taylor [10], we state a formal lower bound on the performance parameter ρ of any k-dimensional spherical code of size c, and by minimizing over c for given k, one obtains a lower bound for any spherical code living in k dimensions. Here $I_x(a, b)$ denotes the regularized incomplete beta function, which comes from computing the volume of a sphere in k dimensions, while $\|\cdot\|$ denotes the Euclidean norm.

▶ **Theorem 4** (Spherical cap lower bounds). *Let $\mathcal{C} \subset \mathcal{S}^{k-1}$ be a spherical code, and let $\mathcal{H}$ be the associated project–and–partition hash family. Then the parameter ρ for $\mathcal{C}$, for target angle $\theta \in (0, \frac{\pi}{2})$, must satisfy:*

$$\rho(\theta) \geq \rho_k(\theta) := \min_{c \geq 2} \rho_k(c, \theta), \tag{3}$$

where, with $\alpha_k(c)$ denoting the solution α to $\frac{1}{2} \cdot I_{1-\alpha^2}(\frac{k-1}{2}, \frac{1}{2}) = \frac{1}{c}$, $\rho_k(c, \theta)$ is given by:

$$\rho_k(c, \theta) := \log \left\{ \Pr_{\boldsymbol{x}, \boldsymbol{y} \sim \mathcal{N}(0,1)^k} \left(\frac{x_1}{\|\boldsymbol{x}\|} \geq \alpha_k(c), \frac{x_1 \cos\theta + y_1 \sin\theta}{\|\boldsymbol{x}\cos\theta + \boldsymbol{y}\sin\theta\|} \geq \alpha_k(c) \right) \right\} \Big/ \log\left(\frac{1}{c}\right).$$

The above minimization over $c \geq 2$ concerns the possible code sizes, and $\rho_k(c, \theta)$ describes the parameter ρ one would obtain when indeed, the code consisted of c equivalent regions of equal volume, and each region was shaped like a spherical cap. Equivalently, one could state that any spherical code of size exactly c must satisfy $\rho(\theta) \geq \rho_k(c, \theta)$.

Numerical evaluation of these expressions $\rho_k(c, \theta)$, and the resulting minimization over c, leads to the values in Table 1 in the rows indicated by *spherical caps*. The superscripts denote the values c that numerically solve the minimization problems. These results in low dimensions suggest that, especially for small angles, the optimal code size should increase superlinearly with the dimension. This inspires the following conjecture, stating that the use of orthoplices is likely not optimal in higher dimensions.

▶ **Conjecture 5** (Orthoplices are suboptimal for large k). *For arbitrary $\theta \in (0, \frac{\pi}{2})$, there exists a dimension $k_0 \in \mathbb{N}$ such that, for all dimensions $k \geq k_0$, there exist spherical codes $\mathcal{C} \subset \mathcal{S}^{k-1}$ whose query exponents ρ in the project–and–partition framework are smaller than the exponents ρ of the k-orthoplex.*

Actually finding such spherical codes, or finding families of spherical codes that outperform orthoplices may again be closely related to the problem of finding nice families of dense lattices with efficient decoding algorithms. We informally conjecture that in high dimensions, and for sufficiently large code sizes c, random spherical codes may be close to optimal. If we care only about minimizing ρ, then a further study might focus on (1) getting a better grip of the optimal scaling of $c = c(k)$ with k, and (2) estimating the asymptotic performance of using random spherical codes of size $c(k)$, for large k. We predict this will lead to better values ρ than those obtained with cross-polytope hashing.

1.5 Selecting spherical codes in practice

Although the exponent ρ, and therefore the probabilities p_1 and p_2, directly imply the main performance parameters to assess the asymptotic performance of a locality-sensitive hash family, in practice we are always dealing with concrete, non-asymptotic values n, d, and θ – if the convergence to the optimal asymptotic scaling is slow, or if the hash functions are too expensive to evaluate in practice, then hash families with lower query exponents ρ may actually be less practical for small, concrete values of n and d than fast hash families with

larger ρ. For example, the hash family from [3] may be "optimal", but appears to be only of theoretical interest. In practice one needs to find a balance between decreasing ρ and using hash functions which are fast to evaluate.

As a more concrete example, consider how hyperplane hashing [16] allows us to partition a sphere in 2^k regions with a single random projection matrix $\mathbf{A} \in \mathbb{R}^{k \times d}$ (or equivalently k matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k \in \mathbb{R}^{1 \times d}$ merged into one large matrix). For cross-polytope hashing [4] a k-dimensional projection would only divide the k-dimensional sphere into $2k$ regions. As the total required number of hash buckets in the data structure is often roughly the same, regardless of the chosen hash family,[1] this means that if we wish to divide the sphere into 2^k hash regions with cross-polytope hashing, we would need $m = k/\log_2(2k)$ independent random projection matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m \in \mathbb{R}^{k \times d}$ to hash a vector to one of 2^k buckets. So even though the resulting partitions for cross-polytope hashing generate smaller exponents ρ, in practice this improvement may not offset the additional costs of computing the projections/rotations, which is almost a linear factor $O(d)$ more than for hyperplane hashing. So especially for data sets of small/moderate sizes, hyperplane hashing may be more practical than cross-polytope hashing, as also observed in e.g. [12, 25, 28, 29].

We explicitly quantify the trade-off between the complexity of the hash functions and the asymptotic query exponents ρ in Figure 3, where on the vertical axis we plotted the query exponents ρ as computed in this paper, and on the horizontal axis we plotted the number of bits extracted per row of a projection matrix; for hyperplane hashing we extract 1 bit per projection, while e.g. for cross-polytope hashing in dimension k we only extract $\log(2k)/k$ bits per projection. Depending on the application, it may be desirable to choose hash families with slightly larger values ρ, if this means the cost of the hashing becomes less. Figure 3 makes a case for using hashes induced by the root lattices D_k, as well as those induced by exotic polytopes such as the 2_{21}-polytope; compared to e.g. the 5-orthoplex, the D_{10} lattice achieves lower values ρ and extracts more bits per projection, while the 2_{21}-polytope further improves upon D_{10} by extracting more hash bits per projection.

To further illustrate how these trade-offs might look in a real-world setting, Figure 4 describes a case study for average-case nearest neighbor searching with different sizes n for the data sets. For small data sets, hyperplane hashing is quite competitive, and the best other spherical codes are those induced by the D_k lattices (for small k), and spherical codes generated by the polytopes $1_{21}, 1_{31}$ and 2_{21}. This case study matches the recommendations from Figure 3. For large n, the role of ρ becomes more prominent, and higher-dimensional orthoplices and D_k lattices attain the best performance. In the full version we further describe how one might choose the best codes in practice.

1.6 Summary and open problems

With the project–and–partition framework outlined in this paper, we can now easily formalize and analyze the performance of any spherical code in this framework, and analyze the collision probabilities and query exponents either analytically (by attempting to simplify the corresponding multivariate orthant probabilities) or numerically (by evaluating these multivariate integrals with mathematical software, such as the mvtnorm package [21]). We observed that already in two dimensions, it is possible to improve upon hyperplane hashing with a smaller exponent ρ by using triangular partitions, and we described closed-form

[1] While a hash family with lower query exponents ρ does require a smaller number of hash buckets per hash table, this effect is marginal compared to the increase in the number of projections/rotations required by e.g. cross-polytope hashing compared to hyperplane hashing.

formulas for the resulting parameters ρ. In three and four dimensions the improvements in the exponent ρ become more significant, with e.g. the tetrahedron, the 5-cell, and the 16-cell achieving the best exponents ρ in these dimensions. For higher dimensions the simplices and orthoplices seem to achieve the best performance in theory, thus making a strong case for the use of these partitions as advertised in [4, 24, 35, 36], and as observed in nearest neighbor benchmarks [9, 13, 31]. The family of D_k lattices and the generalized m-max hash functions however seem to offer better practical trade-offs between the asymptotic performance in terms of ρ and the costs of computing hashes, as discussed in the full version.

Finding better spherical codes. An important open problem, both for our project–and–partition framework and for arbitrary hash families for the angular distance, is to find (if they exist) other nice families of spherical codes which achieve an even better performance than the family of orthoplices. Numerics in the full version suggest that as k increases, the optimal code size c should increase faster than the linear scaling of $c = 2k$ offered by orthoplices. Finding the optimal scaling of c as a function of k, and finding nicer spherical codes closely matching this appropriate scaling of c is left for future work.

Faster pseudorandom projections. To make e.g. hyperplane and cross-polytope hashing more practical, various ideas were proposed in e.g. [1, 4] to work with sparse projections and pseudorandom rotation matrices. Similar ideas can be used for any hash family, to reduce the cost of multiplication by a random Gaussian matrix $\mathbf{A}$. Concretely, one can often replace it with a sparse, "sufficiently random" pseudorandom matrix $\mathbf{A}$ while still achieving good query exponents ρ in practice. Depending on how these pseudorandom projections are instantiated, this may lead to a different practical evaluation than what we described in our practical case analyses. In particular, as this reduces the relative cost of the hashing, this will further favor schemes which reduce ρ at the cost of increasing the (naive) complexity of hashing.

Using orthogonal projection matrices. In our framework, we focused on projecting down onto a low-dimensional subspace using Gaussian matrices $\mathbf{A} \sim \mathcal{N}(0,1)^{k \times d}$. For $k \ll d$, using Gaussian matrices or orthogonal matrices (i.e. $\mathbf{A}\mathbf{A}^T = I_k$) is essentially equivalent [18, 23], but for large k it may be beneficial to use proper rotations induced by orthogonal matrices. For instance, recent work [27] showed that for hypercube hashing in the ambient space, there is a clear gap between using random or orthogonal matrices; using orthogonal matrices generally works better than using random Gaussian projection matrices.

The main issue when analyzing the same framework with orthogonal matrices is that the dependence on d then does not disappear; the distribution of $\mathbf{A}\boldsymbol{x}$ then relies on both k and d, rather than only on $\boldsymbol{k}$. Our framework is in a sense dimensionless, as the performance can be computed without knowledge of d, and for $k = 2$ this even allowed us to obtain explicit analytic expressions for the collision probabilities for arbitrary k-gons. Using orthogonal matrices, the collision probabilities will likely become complicated functions of d, and comparing different spherical codes may then become a more difficult task. We leave it as an open problem to adjust the above framework to the setting where $\mathbf{A}$ is orthogonal, and to see how much can be gained by using orthogonal rather than Gaussian matrices[2].

The remainder of the paper can be found in the full version at `[arXiv:1907.04628]`.

[2] Note that one obtains different asymptotics when analyzing the hypercube for Gaussian [16] and orthogonal projection/rotation matrices [27]. For constant $k = O(1)$ and large d both approaches are asymptotically equivalent, but for large $k = O(d)$ we expect orthogonal matrices to give better results.

Table 1 Parameters ρ for various spherical codes. Sphere packings are from [34]. Rows labeled "spherical caps" are lower bounds (Theorem 4). Superscripts refer to the number of caps minimizing ρ_θ. Bold values indicate the best values ρ encountered up to this dimension. Results for $k = 1, 2, d$, were obtained analytically; for $k = 3, 4, 5, 6$ most results were obtained through numerical integration and Monte Carlo simulation.

k	spherical code	c	$\rho(\frac{\pi}{12})$	$\rho(\frac{\pi}{6})$	$\rho(\frac{\pi}{4})$	$\rho(\frac{\pi}{3})$	$\rho(\frac{5\pi}{12})$
1	*spherical caps*		$0.1255^{(2)}$	$0.2630^{(2)}$	$0.4150^{(2)}$	$0.5850^{(2)}$	$0.7776^{(2)}$
	hyperplane	2	**0.1255**	**0.2630**	**0.4150**	**0.5850**	**0.7776**
2	*spherical caps*		$0.1194^{(3)}$	$0.2518^{(3)}$	$0.4005^{(3)}$	$0.5700^{(3)}$	$0.7666^{(3)}$
	triangle (S_2)	3	**0.1194**	**0.2518**	**0.4005**	**0.5700**	**0.7666**
	square (O_2, C_2, D_2)	4	0.1255	0.2630	0.4150	0.5850	0.7776
	pentagon	5	0.1343	0.2788	0.4346	0.6040	0.7905
	hexagon (A_2)	6	0.1438	0.2954	0.4544	0.6222	0.8022
3	*spherical caps*		$0.1117^{(5)}$	$0.2389^{(4)}$	$0.3846^{(4)}$	$0.5548^{(4)}$	$0.7561^{(4)}$
	tetrahedron (S_3)	4	**0.1155**	**0.2445**	**0.3910**	**0.5600**	**0.7592**
	sphere packing	5	0.1170	0.2481	0.3965	0.5664	0.7644
	octahedron (O_3)	6	0.1159	0.2465	0.3952	0.5661	0.7649
	sphere packing	7	0.1207	0.2554	0.4065	0.5772	0.7725
	cube (C_3)	8	0.1255	0.2630	0.4150	0.5850	0.7776
	sphere packing	9	0.1217	0.2591	0.4129	0.5850	0.7786
	icosahedron	12	0.1255	0.2678	0.4255	0.5983	0.7883
	cuboctahedron (A_3, D_3)	12	0.1294	0.2728	0.4301	0.6017	0.7900
	dodecahedron	20	0.1509	0.3077	0.4692	0.6360	0.8112
4	*spherical caps*		$0.1050^{(7)}$	$0.2285^{(6)}$	$0.3730^{(6)}$	$0.5433^{(5)}$	$0.7466^{(5)}$
	5-cell (S_4)	5	0.1126	0.2392	0.3840	**0.5527**	**0.7537**
	sphere packing	6	0.1128	0.2401	0.3861	0.5555	0.7564
	sphere packing	7	0.1120	0.2392	0.3852	0.5553	0.7567
	16-cell (O_4)	8	**0.1107**	**0.2368**	**0.3822**	0.5528	0.7553
	sphere packing	10	0.1136	0.2426	0.3909	0.5623	0.7622
	sphere packing	13	0.1133	0.2439	0.3939	0.5666	0.7663
	tesseract (C_4)	16	0.1255	0.2630	0.4150	0.5850	0.7776
	runcinated 5-cell (A_4)	20	0.1216	0.2586	0.4128	0.5855	0.7795
	octacube (D_4)	24	0.1202	0.2577	0.4140	0.5877	0.7823
5	*spherical caps*		$0.0997^{(13)}$	$0.2203^{(10)}$	$0.3630^{(8)}$	$0.5342^{(7)}$	$0.7415^{(6)}$
	5-simplex (S_5)	6	0.1105	0.2354	0.3785	0.5469	0.7493
	5-orthoplex (O_5)	10	**0.1076**	**0.2299**	**0.3733**	**0.5433**	**0.7483**
	1_{21}-polytope	16	0.1080	0.2330	0.3794	0.5516	0.7554
	expanded 5-simplex (A_5)	30	0.1167	0.2494	0.4007	0.5735	0.7713
	5-hypercube (C_5)	32	0.1255	0.2630	0.4150	0.5850	0.7776
	rectified 5-orthoplex (D_5)	40	0.1139	0.2471	0.4009	0.5757	0.7739
6	*spherical caps*		$0.0955^{(18)}$	$0.2135^{(15)}$	$0.3552^{(11)}$	$0.5263^{(9)}$	$0.7357^{(8)}$
	6-simplex (S_6)	7	0.1089	0.2319	0.3742	0.5422	0.7458
	6-orthoplex (O_6)	12	0.1065	0.2260	**0.3670**	**0.5361**	**0.7431**
	2_{21}-polytope	27	**0.1038**	**0.2258**	0.3712	0.5442	0.7510
	1_{31}-polytope	32	0.1062	0.2314	0.3788	0.5520	0.7564
	expanded 6-simplex (A_6)	42	0.1133	0.2427	0.3917	0.5642	0.7647
	rectified 6-orthoplex (D_6)	60	0.1107	0.2404	0.3915	0.5661	0.7673
	hypercube (C_6)	64	0.1255	0.2630	0.4150	0.5850	0.7776
d	*lower bound*		*0.0173*	*0.0718*	*0.1716*	*0.3333*	*0.5888*
	simplex (S_d)	$d+1$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	orthoplex (O_d)	$2d$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	expanded simplex (A_d)	$d(d+1)$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	rectified orthoplex (D_d)	$2d(d-1)$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	hypercube (C_d; **A** orth.)	2^d	0.0799	0.1800	0.3151	0.5201	0.7686
	hypercube (C_d; **A** Gaussian)	2^d	0.1255	0.2630	0.4150	0.5850	0.7776

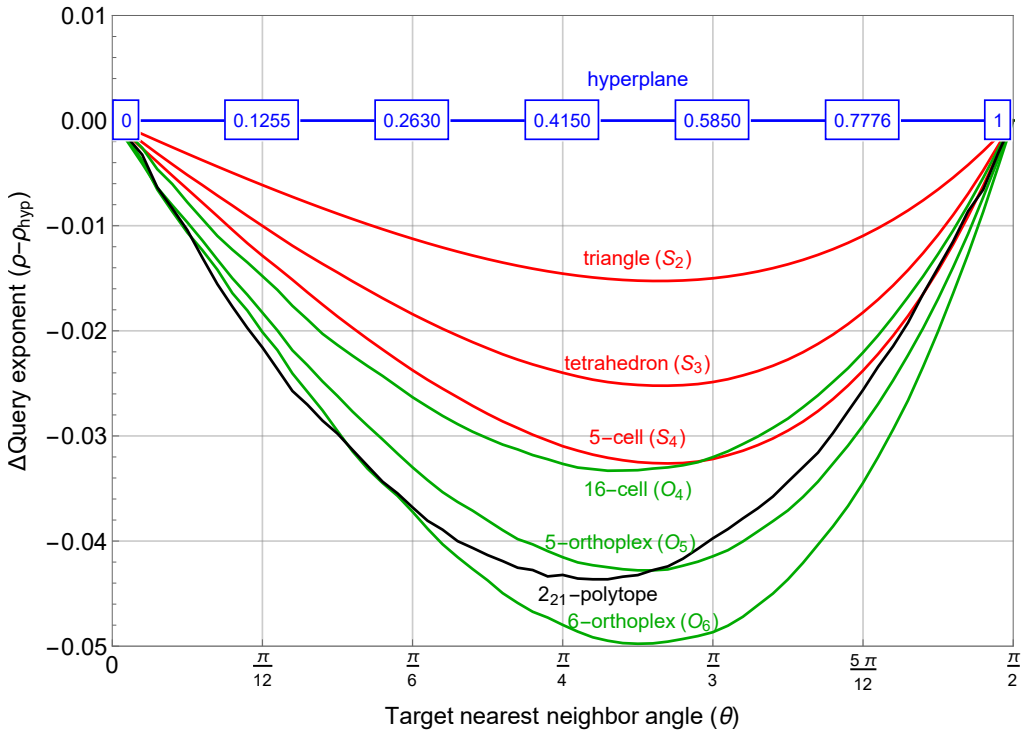

Figure 1 Comparison of the improvements in the query exponent ρ over hyperplane hashing [16] with query exponents ρ_{hyp}. The horizontal blue line denotes the baseline hyperplane hashing approach, with $\rho - \rho_{\text{hyp}} \equiv 0$. Lower curves denote improvements to ρ_{hyp} using various spherical codes. The measurements at the five vertical gridlines correspond to the values in Table 1. For instance, at $\theta = \pi/6$ the triangle has query exponent approximately 0.011 lower than hyperplane hashing (which has exponent 0.2630), leading to $\rho \approx 0.2520$; from Table 1 we get the more precise estimate $\rho \approx 0.2518$; while the theory in the full version allows us to compute ρ exactly through a closed-form expression. The polytopes in this figure are those achieving the lowest exponents ρ in their respective dimensions at one of these grid lines, as highlighted in boldface in Table 1.

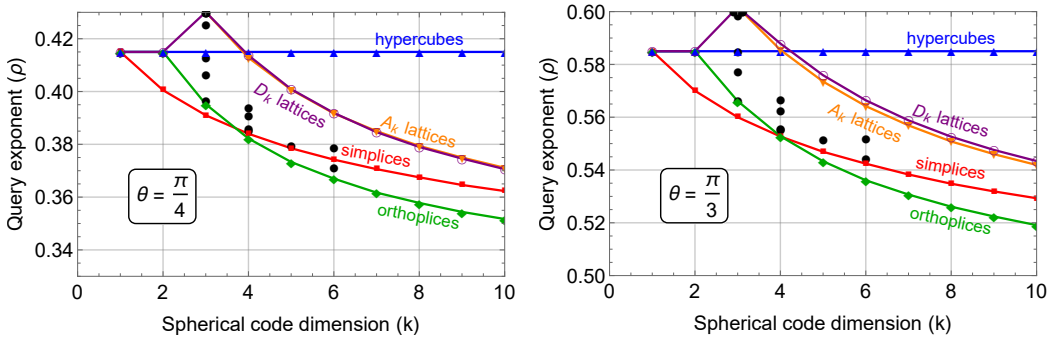

Figure 2 Query exponents ρ for project–and–partition hash families based on the spherical codes listed in Table 1. Lower query exponents ρ are generally better, and for these hash families the best exponents ρ are achieved by the simplex for $k \leq 3$ and by the orthoplex for $k \geq 5$. For $k = 4$ the simplex and orthoplex are close, and which of the two achieves a better performance depends on the target nearest neighbor angle θ.

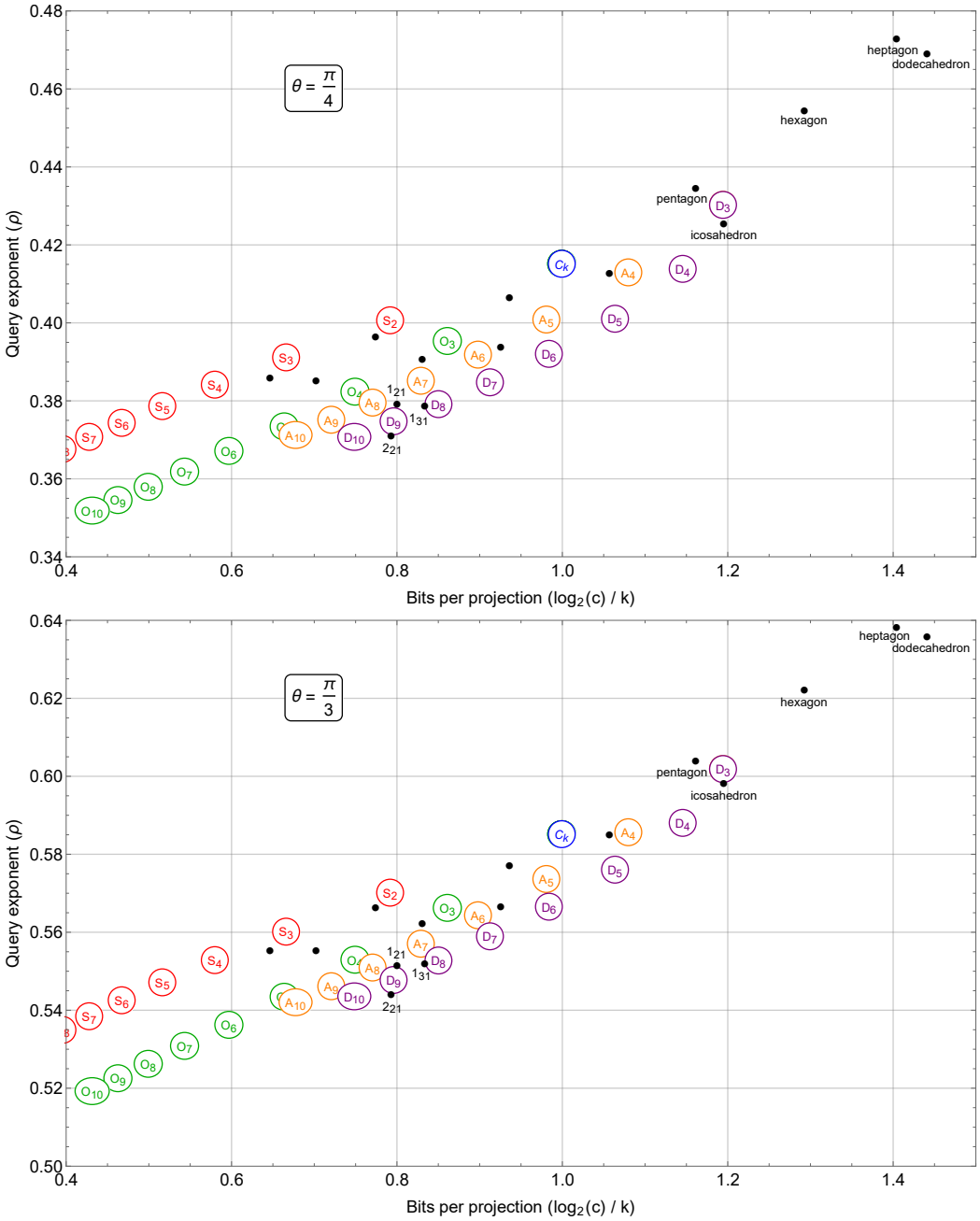

Figure 3 A comparison between different spherical codes in terms of the query exponent ρ (vertical axis) and the bits of information extracted from each row of the projection matrix $\mathbf{A}$ (horizontal axis). The top figure corresponds to target angle $\theta = \frac{\pi}{4}$ (or approximation factor $c = \sqrt{2+\sqrt{2}}$), the bottom figure to target angle $\theta = \frac{\pi}{3}$ (or $c = \sqrt{2}$). Codes further down generate hash functions with a more discriminative power (a lower value ρ), while codes further to the right extract more bits per projection, therefore requiring fewer inner product computations to compute hash values. So for our purposes, the best codes would be as far to the right and as far down as possible. Hypercubes C_k all achieve the same value ρ and the same value $\log_2(c)/k = 1$.

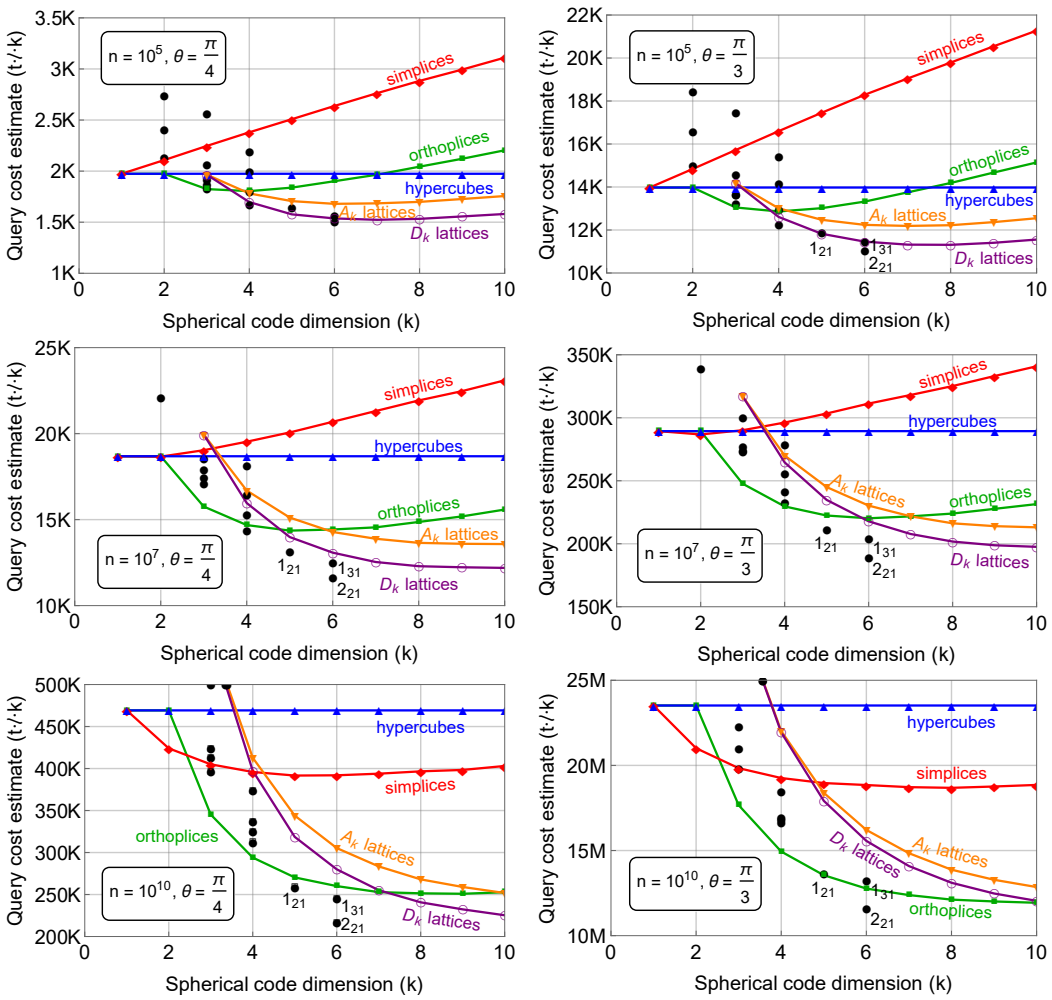

Figure 4 Comparison of the query cost estimates $t \cdot \ell \cdot k$ for different parameters $n \in \{10^5, 10^7, 10^{10}\}$ and $\theta \in \{\frac{\pi}{4}, \frac{\pi}{3}\}$, when using $t = n^\rho$ hash tables and a hash length $\ell = \log(n)/\log(1/p_2)$. The curves correspond to the regular convex polytope families and the root lattice families A_k and D_k, while single black points for $k \leq 6$ correspond to spherical codes described in Table 1. As n increases, the role of a smaller ρ becomes bigger, and so larger spherical codes with smaller values ρ become more suitable choices than those with bigger values ρ but with lower hash costs. The cost of the projections increases linearly with k, while ρ decreases rather slowly with k, suggesting that for each n and θ there is an optimal spherical code dimension $k \ll d$ to project down to, and an optimal spherical code in this dimension to use. Note that other spherical codes than those from the five families of codes sometimes achieve better query cost estimates; in particular, the 2_{21}-polytope (which is connected to the E_6 lattice) might be useful in practice as well, and we cannot rule out the existence of other exotic spherical codes with good properties for nearest neighbor searching.
One of the conclusions one might draw from these figures is that indeed orthoplices (cross-polytopes) perform very well [4], but depending on the parameters it may be better to use the D_k lattices instead – the trends suggest that as k increases further, the query costs of using the D_k lattices will be smaller than those of the orthoplices.

References

1 Dimitris Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001. doi:10.1145/375551.375608.

2 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.

3 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

4 Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015. URL: https://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance.

5 Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014. doi:10.1137/1.9781611973402.76.

6 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. doi:10.1137/1.9781611974782.4.

7 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015. doi:10.1145/2746539.2746553.

8 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. URL: http://dl.acm.org/citation.cfm?id=314464.314652.

9 Martin Aumueller, Erik Bernhardsson, and Alexander Faithfull. ANN benchmarks – available online at http://sss.projects.itu.dk/ann-benchmarks/, 2017. URL: http://sss.projects.itu.dk/ann-benchmarks/.

10 Albert Baernstein and B.A. Taylor. Spherical rearrangements, subharmonic functions, and $*$ -functions in n -space. *Duke Math. J.*, 43(2):245–268, June 1976. doi:10.1215/S0012-7094-76-04322-2.

11 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. doi:10.1137/1.9781611974331.ch2.

12 Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016. doi:10.1007/978-3-319-31517-1_1.

13 Erik Bernhardsson. ANN benchmarks – available online at https://github.com/erikbern/ann-benchmarks, 2016. URL: https://github.com/erikbern/ann-benchmarks.

14 Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006. URL: https://www.springer.com/us/book/9780387310732.

15 Karthekeyan Chandrasekaran, Daniel Dadush, Venkata Gandikota, and Elena Grigorescu. Lattice-based locality sensitive hashing is optimal. In *ITCS*, 2018.

16 Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. doi:10.1145/509907.509965.

17 Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017. doi:10.1137/1.9781611974782.3.

18 Persi Diaconis and David Freedman. A dozen de Finetti-style results in search of a theory. *Annales de l'IHP Probabilités et statistiques*, 23(2):397–423, 1987.

19 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, August 2010. doi:10.1109/TIT.2010.2050814.

20 Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2000. URL: https://dl.acm.org/citation.cfm?id=954544.

21 Alan Genz, Frank Bretz, Tetsuhisa Miwa, Xuefei Mi, Friedrich Leisch, Fabian Scheipl, and Torsten Hothorn. *mvtnorm: Multivariate normal and t distributions*, 2019. R package version 1.0-10. URL: https://CRAN.R-project.org/package=mvtnorm.

22 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. doi:10.1145/276698.276876.

23 Tiefeng Jiang. How many entries of a typical orthogonal matrix can be approximated by independent normals? *The Annals of Probability*, 34(4):1497–1529, 2006. doi:10.1214/009117906000000205.

24 Christopher Kennedy and Rachel Ward. Fast cross-polytope locality-sensitive hashing. In *ITCS*, pages 1–16, 2017. doi:10.4230/LIPIcs.ITCS.2017.53.

25 Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015. doi:10.1007/978-3-662-47989-6_1.

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

27 Thijs Laarhoven. Hypercube LSH for approximate near neighbors. In *MFCS*, pages 1–20, 2017. doi:10.4230/LIPIcs.MFCS.2017.7.

28 Artur Mariano, Thijs Laarhoven, and Christian Bischof. Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In *ICPP*, pages 590–599, 2015. doi:10.1109/ICPP.2015.68.

29 Artur Mariano, Thijs Laarhoven, and Christian Bischof. A parallel variant of LDSieve for the SVP on lattices. In *PDP*, pages 23–30, 2017. doi:10.1109/PDP.2017.60.

30 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015. doi:10.1007/978-3-662-46800-5_9.

31 Ilya Razenshteyn and Ludwig Schmidt. FALCONN – available online at https://falconn-lib.org/, 2016. URL: https://falconn-lib.org/.

32 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005. URL: http://ttic.uchicago.edu/~gregory/annbook/book.html.

33 William F. Sheppard. On the application of the theory of error to cases of normal distribution and normal correlation. *Philosophical Transactions of the Royal Society A*, 192:101–167, 1899.

34 Neil J.A. Sloane. Spherical codes: nice arrangements of points on a sphere in various dimensions. URL: http://neilsloane.com/packings/.

35 Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.

36 Kengo Terasawa and Yuzuru Tanaka. Approximate nearest neighbor search for a dataset of normalized vectors. In *IEICE Transactions on Information and Systems*, volume 92(9), pages 1609–1619, 2009. URL: http://search.ieice.org/bin/summary.php?id=e92-d_9_1609.

37 Serge Vladut. Lattices with exponentially large kissing numbers. *Moscow J. Comb. Number Th.*, 8:163–177, 2019.