New directions in nearest neighbor searching with applications to lattice sieving

Anja Becker*[1], Léo Ducas†[2], Nicolas Gama[3], and Thijs Laarhoven[4]

[1] *EPFL, Lausanne, Switzerland,* anja.becker@epfl.ch
[2] *CWI, Amsterdam, The Netherlands,* leo.ducas@cwi.nl
[3] *UVSQ, Versailles, France,* nicolas.gama@prism.uvsq.fr
[4] *TUe, Eindhoven, The Netherlands,* mail@thijs.com

Abstract

To solve the approximate nearest neighbor search problem (NNS) on the sphere, we propose a method using locality-sensitive filters (LSF), with the property that nearby vectors have a higher probability of surviving the same filter than vectors which are far apart. We instantiate the filters using spherical caps of height $1-\alpha$, where a vector survives a filter if it is contained in the corresponding spherical cap, and where ideally each filter has an independent, uniformly random direction.

For small α, these filters are very similar to the spherical locality-sensitive hash (LSH) family previously studied by Andoni et al. For larger α bounded away from 0, these filters potentially achieve a superior performance, provided we have access to an efficient oracle for finding relevant filters. Whereas existing LSH schemes are limited by a performance parameter of $\rho \geq 1/(2c^2-1)$ to solve approximate NNS with approximation factor c, with spherical LSF we potentially achieve smaller asymptotic values of ρ, depending on the density of the data set. For sparse data sets where the dimension is super-logarithmic in the size of the data set, we asymptotically obtain $\rho = 1/(2c^2-1)$, while for a logarithmic dimensionality with density constant κ we obtain asymptotics of $\rho \sim 1/(4\kappa c^2)$.

To instantiate the filters and prove the existence of an efficient decoding oracle, we replace the independent filters by filters taken from certain structured random product codes. We show that the additional structure in these concatenation codes allows us to decode efficiently using techniques similar to lattice enumeration, and we can find the relevant filters with low overhead, while at the same time not significantly changing the collision probabilities of the filters.

We finally apply spherical LSF to sieving algorithms for solving the shortest vector problem (SVP) on lattices, and show that this leads to a heuristic time complexity for solving SVP in dimension n of $(3/2)^{n/2+o(n)} \approx 2^{0.292n+o(n)}$. This asymptotically improves upon the previous best algorithms for solving SVP which use spherical LSH and cross-polytope LSH and run in time $2^{0.298n+o(n)}$. Experiments with the GaussSieve validate the claimed speedup and show that this method may be practical as well, as the polynomial overhead is small.

1 Introduction

Nearest neighbor searching (NNS). The nearest neighbor search problem (NNS) is an important algorithmic problem in various fields, such as machine learning, coding theory, pattern recognition, and data compression [DHS00, SDI05, Bis06, Dub10]. Given an n-dimensional data set of size N, the problem is to preprocess a data structure such that, given a query vector later, we can quickly identify nearby vectors in time $O(N^\rho)$ for $\rho < 1$.

Locality-sensitive hashing (LSH). One well-known technique for solving NNS is locality-sensitive hashing (LSH) [IM99]. Using locality-sensitive hash functions, which have 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 of nearby vectors. A query is answered by going through all vectors which have at least one hash in common with the target vector, and searching these candidates for a near neighbor.

Approximate NNS. In case the nearest point in the data set is known to be a factor c closer than all other points in the data set, or when the returned

*Supported by the Swiss National Science Foundation, grant numbers 200021-126368 and 200020-153113

†Supported by an NWO Free Competition Grant.

point is allowed to be at most a factor c further away than the nearest neighbor, recent techniques of Andoni et al. [AINR14, AR15a, AIL+15] have shown how to answer queries in time $O(N^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1} + o(1)$$

where the order term vanishes in high dimensions. Within the class of LSH algorithms, these results are essentially optimal [Dub10, OWZ14].[1]

Lattices. One recent application of LSH is to speed up algorithms for solving the shortest vector problem (SVP) on lattices. Given a set $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^n$ of n linearly independent vectors, the lattice associated to B is the set of all integer linear combinations of the basis vectors:

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{n} \lambda_i \mathbf{b}_i : \lambda_i \in \mathbb{Z} \right\}.$$

Given a basis of a lattice, the shortest vector problem asks to find a shortest (w.r.t. the Euclidean norm) non-zero vector in this lattice. Estimating the computational hardness of SVP is particularly relevant for estimating the security of and selecting parameters for lattice-based cryptography [LP11, vdPS13].

Lattice sieving. A recent class of algorithms for solving SVP is lattice sieving [AKS01, NV08, MV10], which are algorithms running in time and space $2^{O(n)}$. Heuristic sieving algorithms are currently the fastest algorithms known for solving SVP in high dimensions, and various recent work has shown how these algorithms can be sped up with NNS techniques [BGJ15, BL15, Laa15, LdW15]. The fastest heuristic algorithms to date for solving SVP in high dimensions are based on spherical LSH [AR15a, LdW15] and cross-polytope LSH [AIL+15, BL15] and achieve time complexities of $2^{0.298n+o(n)}$.

1.1 Contributions and outline.

After introducing some preliminary notation, terminology, and describing some useful lemmas about geometric objects on the sphere in Section 2, the paper is organized as follows.

Locality-sensitive filtering (LSF). In Section 3 we introduce the concept of locality-sensitive filtering (LSF), which in short corresponds to locality-sensitive hashing where only few vectors are actually assigned to buckets. Conceptually, this is similar to approaches of e.g. [Dub10, MO15]. We analyze its properties, its relation with LSH, and how this potentially leads to an improved performance over LSH given access to a certain decoding oracle.

Spherical LSF. To instantiate these filters on the sphere, in Section 4 we propose to use filters defined by taking a random unit vector $\mathbf{s}$ and letting a vector $\mathbf{w}$ pass through this filter iff $\langle \mathbf{w}, \mathbf{s} \rangle \geq \alpha$ for some $\alpha \in [0, 1)$. We highlight similarities and differences with spherical LSH [AR15a] and show how this potentially leads to an improved performance over spherical LSH.

Random product codes. All these results depend on the existence of an efficient decoding oracle. To instantiate this oracle, in Section 5 we propose to use spherical cap filters where the random vectors are taken from a certain structured code C over the sphere such that, given a query vector $\mathbf{v}$, we can compute all relevant filters with minimal overhead using list decoding. A crucial issue is to prove that filters from such a code C behave as well as uniformly random and independently chosen filters, which is shown in the appendix.

Practical aspects. While random product codes satisfy all the properties we need to prove that two nearby vectors almost always have a common neighbor among the filter vectors, these codes may not be very efficient in practice, as the individual block codes have subexponential size and are not efficiently decodable. In Section 6 we discuss practical aspects of these codes, and how we may or may not be able to replace these fully random block codes by even more structured codes.

Application to lattice sieving. In Section 7 we apply our method to lattice sieving, and show that we obtain an asymptotic complexity for solving SVP of only $2^{0.292n+o(n)}$, improving upon the $2^{0.298n+o(n)}$ complexity using spherical or cross-polytope LSH. Figure 1 illustrates the asymptotic time-memory tradeoffs of our algorithm and other results from the literature. Experimental results show that the improvement is relevant in moderate dimensions as well.

Relation with May and Ozerov's techniques. Independently of our work, Herold [Her15] studied how the nearest neighbor technique introduced by May and Ozerov for decoding binary codes [MO15] can be converted to angular distances, and what this would lead to for lattice sieving. For the Nguyen-Vidick sieve [NV08], he showed that this leads to the same time complexity of $2^{0.292n+o(n)}$ using very similar, if not equivalent techniques. One important difference between his work and ours is that Herold's result uses the fact that in the Nguyen-Vidick sieve one has to solve *batch-NNS*, i.e. solving many instances of NNS at the same time rather than one at a time.

[1] Recent work [AR15b] suggests that the asymptotic bound $\rho \geq 1/(2c^2 - 1)$ on LSH only holds under certain non-trivial assumptions, such as a low description complexity of the hash regions. As these assumptions do not necessarily hold in high-density settings, it is not clear whether it is possible to achieve smaller values of ρ in the high-density regime.

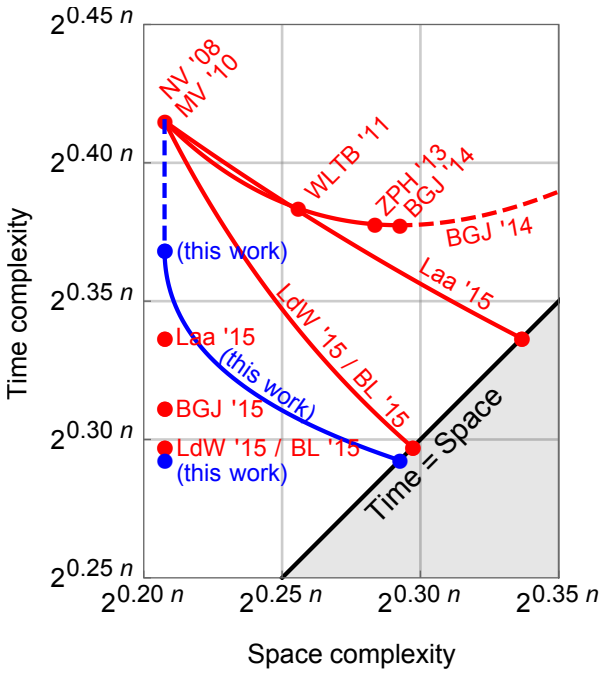

Figure 1: The asymptotic time-memory trade-off for sieving algorithms in high dimensions.

This means that the same techniques do not directly apply to the GaussSieve or to the general nearest neighbor problem without introducing the list-decoding of random product codes.

2 Preliminaries

Geometric objects on the sphere. Let μ be the canonical Lebesgue measure over $\mathbb{R}^n$, and let $\langle \cdot, \cdot \rangle$ denote the standard Euclidean inner product. We denote the unit sphere by $\mathcal{S}^{n-1} := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = 1\}$ and half-spaces by $\mathcal{H}_{\mathbf{v},\alpha} := \{\mathbf{x} \in \mathbb{R}^n : \langle \mathbf{v}, \mathbf{x} \rangle \geq \alpha\}$. For $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$ such that $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$ and $\alpha, \beta \in [0, 1]$ we denote spherical caps and wedges by:

$$\mathcal{C}_{\mathbf{v},\alpha} := \mathcal{S}^{n-1} \cap \mathcal{H}_{\mathbf{v},\alpha},$$
$$\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta} := \mathcal{S}^{n-1} \cap \mathcal{H}_{\mathbf{v},\alpha} \cap \mathcal{H}_{\mathbf{w},\beta}.$$

The following two lemmas estimate the volume of spherical caps and wedges for large n. We denote these quantities as follows, where $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$ and $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$:

$$\mathcal{C}_n(\alpha) := \frac{\mu(\mathcal{C}_{\mathbf{v},\alpha})}{\mu(\mathcal{S}^{n-1})},$$
$$\mathcal{W}_n(\alpha, \beta, \theta) := \frac{\mu(\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta})}{\mu(\mathcal{S}^{n-1})}.$$

Lemma 2.1 is elementary [MV10, Lemma 4.1], while Lemma 2.2 is proved in Appendix A.

Lemma 2.1. (Volume of a spherical cap) *For arbitrary $\alpha \in (0, 1)$, we have*

$$\mathcal{C}_n(\alpha) = \mathrm{poly}(n) \cdot \left(\sqrt{1 - \alpha^2}\right)^n.$$

Lemma 2.2. (Volume of a wedge) *For arbitrary constants $\alpha, \beta \in (0, 1)$, we have*

$$\mathcal{W}_n(\alpha, \beta, \theta) = \mathrm{poly}(n) \cdot \left(\sqrt{1 - \gamma^2}\right)^n,$$
$$\text{with } \gamma = \sqrt{\frac{\alpha^2 + \beta^2 - 2\alpha\beta\cos\theta}{\sin^2\theta}}.$$

In the special case $\alpha = \beta$, we obtain

$$\mathcal{W}_n(\alpha, \alpha, \theta) = \mathrm{poly}(n) \cdot \left(\sqrt{1 - \frac{2\alpha^2}{1 + \cos\theta}}\right)^n.$$

Nearest neighbor searching (NNS). The nearest neighbor search (NNS) problem is defined as follows [IM99]. Given a list of n-dimensional vectors,

$$L = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N\} \subset \mathbb{R}^n,$$

preprocess L in such a way that, given a query vector $\mathbf{v} \notin L$ later, one can efficiently find an element $\mathbf{w}^* \in L$ which is close(st) to $\mathbf{v}$. In the setting of nearest neighbor search on the sphere, we assume that all vectors lie on the unit sphere, i.e., $L \subset \mathcal{S}^{n-1}$. This special case itself is relevant in various practical applications, but also in theory as the paper [AR15a] shows a reduction from NNS in the entire Euclidean space to NNS on the sphere. Thus an important problem is finding efficient methods for solving NNS on the sphere.

A common relaxation of NNS is approximate NNS: find a "nearby" neighbor in L, which is allowed to be a factor c further away from the target vector than the nearest neighbor. A slight variant of this problem that we will consider here is: given that all vectors in L lie at distance r_2 except for one element at distance $r_1 < r_2$, find this one nearby element. On the unit sphere, a distance r translates to an angle $\theta = \arccos(1 - \frac{1}{2}r^2)$.

Locality-sensitive hashing (LSH). One method for solving high-dimensional NNS relies on the use of locality-sensitive hash functions h sampled from a certain hash function family $\mathcal{H}$. Informally, these functions map vectors $\mathbf{w}$ to low-dimensional sketches $h(\mathbf{w})$, such that nearby vectors $\mathbf{v}, \mathbf{w}$ have a higher probability of having the same sketch (i.e. $h(\mathbf{v}) = h(\mathbf{w})$) than faraway vectors. In other words, these functions are sensitive to how nearby (local) vectors are in space, in assigning equal output values to different vectors.

To use these hash families to solve NNS, one generally uses the following method described in [IM99].

First, choose $t \cdot k$ random hash functions $h_{i,j} \in \mathcal{H}$, and combine k of them at a time through concatenation to build t different hash functions $h_1, \ldots, h_t$ defined by $h_i(\mathbf{v}) = (h_{i,1}(\mathbf{v}), \ldots, h_{i,k}(\mathbf{v}))$. Then, given the list L, we build t different hash tables $T_1, \ldots, T_t$, where for each hash table T_i we insert $\mathbf{w} \in L$ into the bucket labeled $h_i(\mathbf{w})$. Finally, given a vector $\mathbf{v}$, we compute its t images $h_i(\mathbf{v})$, gather all the candidate vectors that collide with $\mathbf{v}$ in at least one of these hash tables as a list of candidates, and search this set of candidates for a nearest neighbor. With a suitable hash function family $\mathcal{H}$ and well-chosen parameters k and t, this may guarantee that nearby vectors will always collide in at least one hash table (except with negligible probability), and faraway vectors almost never collide with $\mathbf{v}$. Computing a query's t hashes and performing comparisons with the colliding vectors may then require less effort than a naive linear search.

Solving approximate NNS with LSH. Let the function p, describing collision probabilities between vectors at angle θ, be defined as follows:

$$p(\theta) := \Pr_{h \sim \mathcal{H}}[h(\mathbf{v}) = h(\mathbf{w}) \mid \mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}, \langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta].$$

As is well-known in LSH literature, the power of an LSH family $\mathcal{H}$ in distinguishing between nearby vectors at angle θ_1 and distant vectors at angle θ_2 can be captured by the performance indicator $\rho = \frac{\log p(\theta_1)}{\log p(\theta_2)}$ as the following lemma illustrates[2].

LEMMA 2.3. *Let $\mathcal{H}$ be an LSH family with collision probability function p. Then we can solve approximate NNS with parameters θ_1, θ_2 in time $\mathcal{Q} = \tilde{O}(N^\rho)$, with parameters*

$$k = \frac{\log N}{\log 1/p(\theta_2)}, \qquad \rho = \frac{\log 1/p(\theta_1)}{\log 1/p(\theta_2)},$$
$$t = \exp\left[\log N \cdot \frac{\log 1/p(\theta_1)}{\log 1/p(\theta_2)}\right],$$

Spherical LSH. Recently an LSH family for the sphere was proposed by Andoni et al. [AINR14, AR15a] which works as follows. First, sample $U = 2^{\Theta(\sqrt{n})}$ vectors $\mathbf{s}_1, \ldots, \mathbf{s}_U \in \mathbb{R}^n$ from an n-dimensional Gaussian distribution with average norm 1. To each $\mathbf{s}_i$ we then associate a hash region as follows:

$$H_{\mathbf{s}_i} := \left(\mathcal{S}^{n-1} \cap \mathcal{H}_{\mathbf{s}_i,\alpha}\right) \setminus \bigcup_{j=1}^{i-1} H_{\mathbf{s}_j}.$$

[2]For simplicity, this assumes that the angle of a query vector with all vectors $\mathbf{w} \in L$ is exactly θ_2, except for one special nearby vector $\mathbf{w}^* \in L$ which has angle θ_1 with $\mathbf{v}$.

With the choices $\alpha = n^{-1/4}$ and $U = 2^{\Theta(\sqrt{n})}$, it is guaranteed that with high probability, at the end the entire sphere is covered by these hash regions and each point can be assigned a hash value between 1 and U. On the other hand, taking $\alpha = n^{-1/4}$ and $U = 2^{\Theta(\sqrt{n})}$ guarantees that computing hashes can trivially be done in $2^{\Theta(\sqrt{n})} = 2^{o(n)}$ time by going through all hash regions until a nearby vector $\mathbf{s}_i$ is found. One set of points corresponds to one hash function h, and sampling $h \sim \mathcal{H}$ corresponds to sampling $\mathbf{s}_1, \ldots, \mathbf{s}_U \in \mathbb{R}^n$ from a Gaussian distribution.

The following result, implicitly stated in [AINR14, Lemma 3.3] and [AR15a, Appendix B.1], describes the probability of collision for this hash family, and the resulting expression for ρ:

$$p(\theta) = \exp\left[-\frac{\sqrt{n}}{2}\tan^2\left(\frac{\theta}{2}\right)(1 + o(1))\right],$$
$$\rho = \frac{\log 1/p(\theta_1)}{\log 1/p(\theta_2)} = \frac{\tan^2(\theta_1/2)}{\tan^2(\theta_2/2)}(1 + o(1)).$$

3 Locality-sensitive filtering (LSF)

Instead of locality-sensitive (hash) functions, we will consider locality-sensitive mappings or filters, where each filter maps a vector to a binary value: either a vector survives the filter, or it does not. Alternatively, a filter f maps an input list L of size N to an output list $L_f \subseteq L$ of points which pass through this filter. We would like a filter to only assign vectors to the same bucket if vectors are nearby in space. In other words, the filters should be chosen such that after applying (a sequence of) filter(s) to an input set L, the output set L' only contains points which are nearby.

To solve the nearest neighbor problem with these filters, we propose the following method. Given a distribution $\mathcal{F}$ of filters, we draw $t \cdot k$ filters $f_{i,j} \in \mathcal{F}$, and combine k at a time to build t filters f_i, where $\mathbf{w}$ passes through the concatenated filter f_i if it passes through all partial filters $f_{i,j}$ for $j = 1, \ldots k$. Then, given the list L, we build t different filtered buckets $L_1, \ldots, L_t$, where a vector $\mathbf{w} \in L$ is inserted into the bucket L_i iff $\mathbf{w}$ survives the concatenated filter f_i. Finally, given a query vector $\mathbf{v}$, we check which of the concatenated filters it passes through, gather all the candidate vectors that pass through at least one of the filters that $\mathbf{v}$ passes through, and search this set of candidates for a nearest neighbor. With a suitable partial filter distribution $\mathcal{F}$ and parameters k and t, this allows us to solve (approximate) NNS.

Performance of LSF. For analyzing the performance of LSF, we assume that we have an efficient oracle $\mathcal{O}$ which identifies the concatenated filters a vector $\mathbf{v}$ passes through (the *relevant* filters) in time $O(F_{\mathbf{v}})$,

where $F_{\mathbf{v}}$ is the number of relevant filters for $\mathbf{v}$ out of all t concatenated filters. This assumption is crucial, as without this we will not obtain an improved performance over LSH. Assuming the distribution $\mathcal{F}$ is spherically symmetric, similar to collision probabilities in LSH we define

$$p(\theta) := \Pr_{f\sim\mathcal{F}}[\mathbf{v},\mathbf{w} \in L_f \mid \mathbf{v},\mathbf{w} \in \mathcal{S}^{n-1}, \langle \mathbf{v},\mathbf{w}\rangle = \cos\theta].$$

Note that the difference with LSH is that a collision is now defined as two vectors passing through the same filter, rather than obtaining the same hash value. Now, $\mathbf{v}$ survives a sequence of k partial filters with probability $p(0)^k$, so $F_{\mathbf{v}} = O(t \cdot p(0)^k)$. On the other hand, a vector $\mathbf{w}$ at angle θ with $\mathbf{v}$ collides with $\mathbf{v}$ in a k-concatenated filter with probability $p(\theta)^k$. As all N vectors (but one) are assumed to lie at angle θ_2 with $\mathbf{v}$, the costs of processing a query with an efficient oracle are

$$\mathcal{Q} = \tilde{\mathcal{O}}(t \cdot p(0)^k + t \cdot p(\theta_2)^k \cdot N).$$

The first term above counts the average number of colliding relevant vectors for the single nearby vector, while the second term counts the number of distant vectors colliding with our query vector.

Next, to guarantee that we will find a nearby vector at angle θ_1 with probability $1-\varepsilon$, we need $1-(1-p(\theta_1)^k)^t = O(t \cdot p(\theta_1)^k) \geq 1-\varepsilon$, or $t = O(1/p(\theta_1)^k)$. We further want to minimize the total cost $\mathcal{Q}$ of processing a query, which corresponds to balancing the two contributions to $\mathcal{Q}$; larger k and t leads to more selective filtering and fewer comparisons, but increases the cost of finding the relevant filters. Equating the two terms in $\mathcal{Q}$, minimizing the overall query cost, translates to $p(0)^k = p(\theta_2)^k \cdot N$ up to subexponential terms. Solving for k, we obtain expressions for k and t, which in turn can be substituted into $\mathcal{Q}$ to find the best parameters for LSF as follows.

THEOREM 3.1. *Let $\mathcal{F}$ be an LSF distribution with collision probability function p, and let $\mathcal{O}$ be an efficient oracle for computing $\mathbf{v}$'s relevant filters in time $O(|F_{\mathbf{v}}|)$. Then we can solve approximate NNS with parameters θ_1 and θ_2 in time $\mathcal{Q} = \tilde{O}(N^\rho)$, with parameters*

$$k = \frac{\log N}{\log p(0)/p(\theta_2)}, \qquad \rho = \frac{\log p(0)/p(\theta_1)}{\log p(0)/p(\theta_2)},$$
$$t = \exp\left[\log N \cdot \frac{\log 1/p(\theta_1)}{\log p(0)/p(\theta_2)}\right].$$

Notice the similarity with LSH and Lemma 2.3, where the only difference is that in some cases a 1 is replaced by $p(0)$. In LSH the function p denotes collision probabilities in the hash table, and as each vector is always assigned to a hash bucket for a hash function h, we have $p(0) = 1$. Indeed, substituting $p(0) = 1$ we obtain the expressions from Lemma 2.3. For LSF however we only get a collision of $\mathbf{v}$ with itself in filter f if $\mathbf{v}$ survives filter f; if $\mathbf{v}$ is filtered out, we will not get a collision between $\mathbf{v}$ and itself. So generally $p(0) < 1$ for LSF, leading to strictly lower (better) values ρ than for LSH, and for LSF we get strictly larger values t.

Remark. The above theorem is given only as an illustration of our approach, as we do not know of any implementation of such an oracle when the set of filters is chosen independently at random from $\mathcal{F}$. In Section 5 we do provide such an oracle for a set of filters that are more structured, yet still ensuring the proper collision probabilities.

4 Spherical LSF

We will instantiate the concept of LSF with the following spherical cap LSF distribution $\mathcal{F}$. A filter is constructed by drawing a random vector $\mathbf{s} \in \mathcal{S}^{n-1}$, and a vector $\mathbf{w}$ passes through this filter if it satisfies $\langle \mathbf{w},\mathbf{s}\rangle \geq \alpha$. In other words, a vector $\mathbf{w}$ passes a filter if it lies in the spherical cap centered at $\mathbf{s}$ of height $1-\alpha$. Comparing this to spherical LSH, this means that for a filter vector $\mathbf{s}$, the corresponding filtered region is:

$$H_{\mathbf{s}} := \mathcal{S}^{n-1} \cap \mathcal{H}_{\mathbf{s},\alpha}.$$

The probability that two vectors survive the same filter is exactly proportional to the volume of a wedge $\mathcal{W}(\alpha,\alpha,\theta)$; $\mathbf{v}$ and $\mathbf{w}$ survive the filter corresponding to $\mathbf{s}$ iff $\mathbf{s}$ lies in the wedge defined by $\mathbf{v}$ and $\mathbf{w}$. By Lemma 2.2 we therefore obtain:

$$p(\theta) = \exp\left[\frac{n}{2}\ln\left(1-\frac{2\alpha^2}{1+\cos\theta}\right)(1+o(1))\right].$$

If we assume we have an efficient oracle for determining a vector's relevant filters, then by Theorem 3.1 we obtain a performance parameter ρ of

$$\rho = \frac{\log\left(1-\alpha^2\right) - \log\left(1-\frac{2\alpha^2}{1+\cos\theta_1}\right)}{\log\left(1-\alpha^2\right) - \log\left(1-\frac{2\alpha^2}{1+\cos\theta_2}\right)}(1+o(1)).$$

Notice that a Taylor series expansion of ρ for $\alpha \approx 0$ gives us $\rho \overset{(\alpha\approx 0)}{\sim} \frac{\tan^2(\theta_1/2)}{\tan^2(\theta_2/2)}$ which is equivalent to the exponent ρ of spherical LSH. In other words, for small α the performance of spherical LSF (provided an oracle $\mathcal{O}$ exists) will be equivalent to the performance of spherical LSH.

Optimizing α and fixing $k = 1$. An intrinsic lower bound on k is given by $k \geq 1$, which implies

$$k = \frac{\log N}{\log p(0)/p(\theta_2)} \geq 1$$
$$\implies \quad \alpha \leq \alpha_0 := \sqrt{1 + \frac{N^{2/n}(\cos\theta_2 - 1)}{2N^{2/n} - \cos\theta_2 - 1}}\,.$$

Depending on N, θ_1, θ_2, as well as on the existence of efficient decoding oracles for given α, this bounds which values α can be used. We further observe that ρ is decreasing in α, which implies one should choose α to be as large as possible. This suggests taking $\alpha = \alpha_0$ is optimal, which corresponds to fixing $k = 1$. In that case, we always only use one filter for each of the t combined filters.

Note that the upper bound α_0 is decreasing with $N^{2/n}$, and so high-density settings with $N = \exp(\kappa n)$ for large density κ are easier to solve than low-density cases. For $\kappa \to \infty$ we further obtain $\alpha_0 \to \sqrt{2}/2$, while for $\kappa \to 0$ the upper bound on α becomes $\alpha_0 = (2\varepsilon + O(\varepsilon^2))^{1/2}$.

Exponent ρ for $r_1 = \frac{1}{c}\sqrt{2}$ and $r_2 = \sqrt{2}$. For general θ_1, θ_2, N, we now have a recipe to choose our parameters α, t, ρ and $k = 1$. To study the performance of spherical LSF and compare it with other results, let us focus on the random case of [AR15a], where $\theta_1 = \arccos(1 - \frac{1}{c^2})$ and $\theta_2 = \frac{\pi}{2}$, so that nearby vectors are a factor c closer than faraway (orthogonal) vectors. In that case we obtain the upper bound

$$\alpha_0 = \sqrt{\frac{N^{2/n} - 1}{2N^{2/n} - 1}} = \sqrt{\frac{e^{2\kappa} - 1}{2e^{2\kappa} - 1}}\,.$$

Figure 2 illustrates the values of ρ for different c and κ.

Performing a Taylor series expansion for ρ for small κ (and fixed $c > 1$), we obtain

$$\rho = \frac{1 - \kappa}{2c^2 - 1} + \frac{\kappa}{(2c^2 - 1)^2} + O(\kappa^2). \qquad (\kappa \to 0)$$

Alternatively, if we look at high-density settings, then for arbitrary c and large κ we obtain:

$$\rho = -\frac{1}{2\kappa}\log\left(1 - \frac{1}{2c^2 - 1}\right) + O\left(\frac{1}{\kappa^2}\right). \quad (\kappa \to \infty)$$

For large approximation factors c this implies that $\rho \sim \frac{1}{2c^2}$ for small κ and $\rho \sim \frac{1}{4\kappa c^2}$ for large κ.

The low-density regime. In the low density case $N = 2^{o(n)}$ or $\kappa = o(1)$, the exponent ρ of spherical LSF tends to the same value as in spherical LSH [AINR14, AR15a]. Nevertheless, spherical LSF could be significantly faster in practice, because ρ tends to its limit from below depending on $\kappa = o(1)$, and the hidden subexponential term may be smaller. Note that the density can always be increased to $\kappa = \Omega(1/\log n)$ via the Johnson-Lindenstrauss transform [JL84].

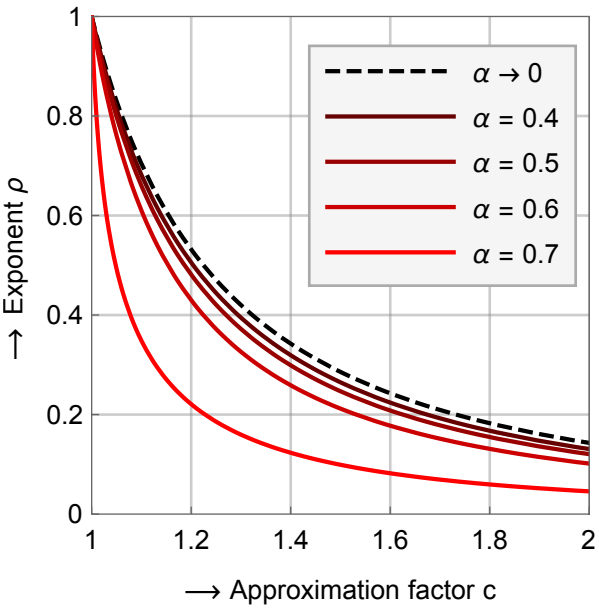

Figure 2: The performance parameter ρ for spherical LSF against the approximation factor c, in the random setting (cf. [AR15a]) and for asymptotically large n. For $\kappa \to 0$, we have $\alpha \leq \alpha_0 \to 0$ and $\rho(c) \to 1/(2c^2 - 1)$.

5 Random product codes

To build an oracle that is able to efficiently determine the set of relevant filters for a given vector in the context of spherical LSF, we will modify the distribution of filters; rather than randomly sampling all of the filters independently, we will sample a code C on the sphere which determines which filters we use, and which admits a fast decoding algorithm for finding the relevant vectors using list decoding. Below we choose $m = O(\text{polylog}\, n)$, and we assume $n = m \cdot b$ for an integral block size b. We further identify vectors in $\mathbb{R}^n$ with tuples of m vectors in $\mathbb{R}^b$, e.g. $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_m) \in (\mathbb{R}^b)^m$.

Definition 5.1. (Random product codes) *The distribution $\mathcal{R}_{n,m,B}$ on subsets of $\mathbb{R}^n$ of size $M = B^m$ is defined as the distribution of codes C of the form*

$$C = Q \cdot (C_1 \times C_2 \times \cdots \times C_m),$$

where Q is a uniformly random rotation over $\mathbb{R}^n$ and the subcodes $C_i \subset \sqrt{1/m} \cdot \mathcal{S}^{b-1}$ for $i = 1, \ldots, m$ are sets of B uniformly random and independently sampled vectors over the sphere $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$.

We need two properties for random product codes $C \sim \mathcal{R}_{n,m,B}$ to be useful for our purposes: the code must be efficiently decodable, and it must behave (almost) as good as a fully random code over the sphere, when considering the probabilities of collision between two vectors on the sphere.

5.1 List-decodability of random product codes.

We first describe an efficient list decoding method for the above random product codes in the regime where the list L has exponential size in n. A short description is given in the following proof.

LEMMA 5.1. *There exists an algorithm that, given the description $Q, C_1, \ldots, C_m$ of a random product code $C \sim \mathcal{R}_{n,m,B}$ and a target vector $\mathbf{t}$, returns the set $S = C \cap \mathcal{C}_{\mathbf{t},\alpha}$ in average time*

$$\mathcal{T}_{LD}(M, \alpha) = O(nB + mB \log B + m \cdot M \cdot \mathcal{C}_n(\alpha))$$

over the randomness of $C \sim \mathcal{R}_{n,m,B}$.

Proof. The algorithm receives as input a code $C = Q \cdot (C_1 \times \cdots \times C_m)$ where $|C_j| = B$; a target vector $\mathbf{t} \in \mathcal{S}^{n-1}$; and a parameter $\alpha < 1$. We first compute $\mathbf{v} = Q^{-1}\mathbf{t}$ and parse $\mathbf{v}$ as $(\mathbf{v}_1, \ldots, \mathbf{v}_m) \in (\mathbb{R}^b)^m$.

For each set of elements in C_j we compute all $\langle \mathbf{v}_i, \mathbf{c}_{i,j} \rangle$ and sort them into lists L_j, hence obtaining m lists of size B. We now wish to identify all vectors $\mathbf{c} = (\mathbf{c}_{i,j})_{i \in [1,B], j \in [1,m]} \in C_1 \times \cdots \times C_m$ for which $\langle \mathbf{v}, \mathbf{c} \rangle \geq \alpha \sum_{i=1}^{m} \|\mathbf{v}_i\|$. To do so, we visit the enumeration tree in a depth-first manner. Its nodes at level $k \leq m$ are labeled by $C_1 \times \cdots \times C_k$, and the parenthood is defined by the direct prefix relation. We use the sorted lists L_j to define in which order to visit siblings. Because of this ordering, if a node has no solution in its descendants, then we know that all its next siblings will not lead to a solution either. This allows to prune the enumeration tree and guarantees that the number of visited nodes is no larger than $2m|S|$, where $|S|$ is the number of solutions.

The overall running time is the sum of the three following terms: $m \cdot B$ dot products of dimension b, followed by $O(m \cdot B \log B)$ operations for the sorting step, and finally the visit of $O(m \cdot |S|)$ nodes for the pruned enumeration, where $|S| = O(M \cdot \mathcal{C}_n(\alpha))$. □

Efficient decoding algorithm. The algorithm outlined in the proof of Lemma 5.1 is explicitly described as Algorithm 1. It is inspired by the lattice enumeration algorithm of Fincke, Pohst, and Kannan [Kan83, FP83], with some additional precomputations exploiting the structure of the code, which largely shrinks the enumeration tree. In this algorithm we denote $\mathbf{c}_{i,j}$ for $j \in [1, B]$ the elements of C_i after the sort,

Algorithm 1 EfficientListDecoding$(C, \mathbf{t}, \alpha)$

Require: The description $Q \in \mathbb{R}^{n \times n}$ and $C_1, \ldots, C_m \subset \mathbb{R}^b$ of the code C; a target vector $\mathbf{t} \in \mathbb{R}^n$; and $\alpha < 1$.
Ensure: Return all α-close code words $S = C \cap \mathcal{C}_{\mathbf{t},\alpha}$.

```
1: Sort each C_i by decreasing dot-product with t_i.
2: Precompute m bounds R_i = α − Σ_{k=i+1}^{m} d_{k,1}.
3: Initialize an empty output set S ← ∅.
4: for each j_1 ∈ [1, B] s.t. d_{i,j_1} ≥ R_1 do
5:     for each j_2 ∈ [1, B] s.t. d_{i,j_2} ≥ R_2 − d_{1,j_1} do
6:         [...]
7:         for each j_m ∈ [1, B] s.t. d_{i,j_m} ≥ R_m − d_{1,j_1} −
       d_{2,j_2} − ··· − d_{m−1,j_{m−1}} do
8:             S ← S ∪ {(c_{1,j_1}, ..., c_{m,j_m})}
9:         end for
10:        [...]
11:    end for
12: end for
13: return S
```

and $d_{i,j} = \langle \mathbf{c}_{i,j}, \mathbf{t}_i \rangle$ their dot product with part of the target vector $\mathbf{t}$.

For simplicity, the core of the pseudo-code is described as m imbricated for-loops. If m is a variable and not a fixed parameter, we let the reader replace the m loops with its equivalent recursive or while-based construction. The sorting phase in steps 1,2 requires $O(mB)$ dot-product computations and runs in $\tilde{O}(mB)$ comparisons. Then the subsets of indexes in each for-loop are contiguous and easy to compute, since each list is already sorted by decreasing $d_{i,j}$. An equivalent way of presenting the k-th for-loop would be "for $j_k \in [1, p]$" where p is the smallest index s.t. $d_{i,p} \leq R_i - \sum_{l=1}^{k-1} d_{l,j_l}$, which can quickly be found by binary search.

If an index j_k is rejected at the k-th for-loop, we know that the partial vector $(\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{k,j_k})$ cannot be extended as a neighbor, since even after adding all maximum partial dot products $d_{l,1}$ for $l \geq k+1$, the overall dot product remains smaller than α. This, combined with the fact that the condition in the m-th for-loop is exactly $\langle \mathbf{t}, (\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{m,j_m}) \rangle \geq \alpha$, proves that the algorithm enumerates $C \cap \mathcal{C}_{\mathbf{t},\alpha}$, i.e., all code words which are neighbor to $\mathbf{t}$.

Furthermore, unlike classical enumeration methods, this additional property proves that there is no dead branch during enumeration: each time we enter the k-th for-loop on index j_k, we are guaranteed that at least the neighbor $(\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{k,j_k}, \mathbf{c}_{k+1,1}, \ldots, \mathbf{c}_{m,1})$ will be added to the list S. Thus, the overall complexity of the for-loop parts is proportional to m times the size of $C \cap \mathcal{C}_{\mathbf{t},\alpha}$.

Efficient list-decodability regime. If the parameters ensure that the average output size $M \cdot \mathcal{C}_n(\alpha)$ is larger than $B \log B$, then we are in the regime of effi-

cient list decoding: the running time is essentially proportional to the output size. This is trivially the case when $M = t = 2^{\Omega(n)}$, $\alpha = \Omega(1)$ and $m = \log n$ for the dense case of Section 4. In the sparse case, relying on the Johnson-Lindenstrauss transform [JL84] to ensure $N = 2^{\Theta(n/\log n)}$ this is easily adjusted to $m = \log^2 n$.

5.2 Randomness of random product codes. On average over the randomness of the code, for two vectors $\mathbf{v}, \mathbf{w}$ at angle θ we expect exactly $M \cdot \mathcal{W}_n(\alpha, \alpha, \theta)$ code words $\mathbf{c}$ to simultaneously fulfill $\langle \mathbf{v}, \mathbf{c} \rangle \geq \alpha$ and $\langle \mathbf{w}, \mathbf{c} \rangle \geq \alpha$. But it could be the case that the set $I = C \cap \mathcal{W}_{\mathbf{w},\alpha,\mathbf{v},\beta}$ is empty most of the time, and very large in some cases; this is in particular the case if all the points of C are concentrated in a small region of the space, or are in some other way not well-distributed over the sphere.

To ensure that the code is useful for our task, we need not only consider how large $M \cdot \mathcal{W}_n(\alpha, \beta, \theta)$ is, but also make sure that C behaves similarly to a random code with respect to intersections with random wedges. The following theorem states that the probability of collision for random product codes does not deviate much from the probability of collision for completely random codes.

THEOREM 5.1. (RANDOM BEHAVIOR OF RPC) *For large n, suppose that $M \cdot \mathcal{W}_n(\alpha, \beta, \theta) \to 0$ or $M \cdot \mathcal{W}_n(\alpha, \beta, \theta) \geq 2^{\tilde{\mathcal{O}}(\sqrt{n})}$. Then, for $\mathbf{v}, \mathbf{w} \in \mathcal{S}^{n-1}$ at angle θ, over the choice $C \sim \mathcal{R}_{n,m,B}$, the probability q that a code word $\mathbf{c} \in C$ lies in the wedge $\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta}$ satisfies:*

$$q \geq \min\left\{ M \cdot \mathcal{W}_n(\alpha, \beta, \theta) \cdot 2^{\tilde{\mathcal{O}}(\sqrt{n})}\,,\, 1 - \mathrm{negl}(n) \right\},$$
$$q \leq \min\{\, M \cdot \mathcal{W}_n(\alpha, \beta, \theta)\,,\, 1 \,\}.$$

The proof of Theorem 5.1 is detailed in Appendix C. Intuitively, the proof relies on the fact that if $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$ and m is reasonably small, then with high probability the block-wise dot products satisfy $\langle \mathbf{v}_i, \mathbf{w}_i \rangle \approx \frac{1}{m}\cos\theta$ for $i = 1, \ldots, m$, and with high probability $\|\mathbf{v}_i\|, \|\mathbf{w}_i\| \approx 1/\sqrt{m}$. This means that the total, n-dimensional wedge can be well-approximated by a cartesian product of m wedges of dimension $b = n/m$:

$$\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta} \approx \prod_{i=1}^{m} \frac{1}{\sqrt{m}}\, \mathcal{W}_{\sqrt{m}\mathbf{v}_i,\alpha,\sqrt{m}\mathbf{w}_i,\beta}.$$

The proof consists in formalizing this approximation, showing that the losses in this approximation are small, and using this approximation to compute collision probabilities for random product codes.

5.3 Application to LSF. Equipped with this code we may now replace, in the construction of Sections 3 and 4, the set of t independent filters, by a set of filters defined by a code $C \sim \mathcal{R}_{n,m,(t^{1/m})}$. Algorithm 1 provides the efficient oracle that computes the set of relevant filters for a given target vector. Theorem 5.1 ensures that the probabilities of collisions (and hence, the complexity analysis) presented in Sections 3 and 4 also hold when the filters are not chosen independently but according to a random product code.

6 Practical aspects

While spherical LSF with random product codes as described in the previous sections achieves small asymptotic exponents ρ, at first sight this scheme looks very similar to spherical LSH [AR15a], which is known to be theoretically optimal for low-density settings but seems less useful in practice due to the high sub-exponential cost of computing hash values: without imposing any structure on the set of $U = 2^{\Theta(\sqrt{n})}$ hash vectors $\mathbf{s}_1, \ldots, \mathbf{s}_U$ in spherical LSH, decoding cannot be done faster than in time $2^{\Theta(\sqrt{n})}$ by simply going through all these vectors one by one to find the first one that is nearby. Although the subcodes $C_1, \ldots, C_m$ in spherical LSF have sub-exponential size as well, the cost of computing all inner products and sorting them to find the good ones may be costly. Indeed, preliminary experiments show that the sorting of the blockwise inner products is one of the main bottlenecks of answering a query.

Structured subcodes. A natural way to try to improve upon the sub-exponential costs is to make the subcodes C_i more structured. Ultimately, we would like the subcodes C_i to be:

(1) of the appropriate size $t^{1/m}$;

(2) *smooth* on the unit sphere;

(3) *efficiently* decodable.

Smoothness. In (2), "smooth" means that if we have a subcode of size proportional to $1/\mathcal{W}_{n/m}(\alpha, \beta, \theta)$, then we know that *on average* a random wedge on the sphere with parameters α, β, θ contains 1 point from the subcode, but this is not enough; we want the distribution to be strongly concentrated around its mean. For instance, subcodes for which all code words are clustered on one part of the sphere still have a good average number of code words in a random wedge, but in many cases a random wedge will be empty, meaning that with high probability we will not find a collision in the filters between nearby vectors.

Decoding complexity. For (3), "efficiently" decodable could mean various things. A smooth code which we can decode slightly faster than with a brute force search over all code words could already lead to big

savings compared to the current, naive approach of using fully random subcodes and decoding in linear time. In other words, any decoding time better than $O(|C_i|)$ for the subcodes could already be interesting. If we can go much further than this, and we can construct smooth spherical codes of the appropriate size for which decoding can be done in sublinear time (or even logarithmic time), then note that we do not need to decompose the code into subcodes at all. This decomposition is purely to make the decoding time subexponential in n.

Designing smooth subcodes. To make sure that a subcode is smooth, intuitively one would like the points on the sphere to be as equally spaced as possible, such that the maximum distance from a point on the sphere to a code word is minimized. The problem of finding suitable codes then seems closely related to designing efficiently decodable spherical codes and spherical packings or coverings. Finding smooth subcodes which lead to a significant practical decoding advantage is left for future work, although we mention that using hypercross-polytopes or hypersimplices does not seem to work as these are not smooth. Note that subcodes related to spherical coverings may also be of interest for designing an LSF-based nearest neighbor scheme with no false negatives [Pag16]; if we have subcodes for which each wedge on the sphere is non-empty, then we can guarantee that nearby vectors will always collide in at least one of the filters. In practice, our code implementation does some heuristic effort towards smoothness, by starting from a random subcode, and having them push each other away other when they are too close on the sphere. This seems to lead to non-negligible savings of roughly $20\% - 50\%$ in the explored parameter set.

Pruning the tree. Besides structuring the subcodes, one could try to save on the costs of computing and sorting blockwise inner products by noting that if $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$ and m is reasonably small, then with high probability the block-wise dot products satisfy $\langle \mathbf{v}_i, \mathbf{w}_i \rangle \approx \frac{1}{m} \cos\theta$. In particular, the blockwise dot products will be large if two vectors are nearby. This means that instead of computing and sorting *all* partial inner products, one could just store only those code words from the subcode with a sufficiently large inner product with the query vector. As most of the blockwise dot products will be concentrated around 0, this may reduce the size of the sublists significantly.

Reusing subcodes. An easy way to slightly save on the space complexity of storing all the filter vectors is to reuse the subcodes and set $C_1 = C_2 = \cdots = C_m$. This is also what is done in our implementation in the following section. Unless there is a spherically asymmetric structure in the data set, this extra condition on the subcodes should not make the scheme any worse and slightly more practical. This also means that finding one nice subcode in dimension $b = n/m$ suffices to construct a suitable product code C.

Decoding subcodes separately. Finally, observe that the proof of smoothness of random product codes relies on approximating a wedge on an n-dimensional sphere by the Cartesian product of $m = O(\log n)$ subwedges on (n/m)-dimensional unit spheres. This suggests that rather than using list-decoding, and searching for code words $\mathbf{c} \in C$ such that $\langle \mathbf{v}, \mathbf{c} \rangle \geq \alpha$, one could also decode subcodes directly and look for tuples $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_m) \in C_1 \times \cdots \times C_m$ such that $\langle \mathbf{v}_i, \mathbf{c}_i \rangle \geq \alpha/m$ for each $i = 1, \ldots, m$. This would also make decoding significantly easier; decode each block separately, and take all combinations of solutions for each subcode.

However, this modification significantly affects the practical performance of the scheme, as in that case the collision probabilities of the entire, concatenated code are roughly given by the product of the collision probabilities of the subcodes[3]. This means that for instance the performance parameter ρ, which is not affected by raising both the subcode collision probabilities $\tilde{p}(\theta_1)$ and $\tilde{p}(\theta_2)$ to the same power m, is almost exactly the same as the parameter $\tilde{\rho}$ for the subcodes. As a result, decoding each subcode separately is not any better than decoding based only on one subcode, and ignoring all other subcodes! Note that in theory the dimension of the subcodes $n/m = O(n/\log n)$ is almost as big as the dimension n of the entire code, and so decoding each subcode separately is sufficient to achieve the same asymptotic performance. In practice however, the correlated list-decoding where all blocks are jointly decoded is crucial for obtaining a superior performance over just using one $O(n/\log n)$-dimensional code for decoding.

7 Application to lattice sieving

Let us now consider an explicit application of the proposed framework: to speed up the search for nearby vectors inside lattice sieving algorithms for solving the shortest vector problem in high dimensions. The application of nearest neighbor searching techniques to sieving has previously been described in [BGJ15, BL15, Laa15, LdW15].

Classical sieving. Given a basis B of a lattice $\mathcal{L}(B)$, we can easily sample reasonably long lattice vectors using Klein's nearest plane algorithm [Kle00]

[3] This is slightly inaccurate, as it assumes that the events $\langle \mathbf{v}_i, \mathbf{c}_i \rangle \geq \alpha/m$ are independent for different i, where $\mathbf{v}$ and $\mathbf{c}$ lie on the unit sphere in $\mathbb{R}^n$. However, using concentration inequalities on the blockwise norms (see e.g. Lemma C.2) we can almost consider these events as independent, in which case the probabilities for the subcodes multiply.

according to a distribution which is statistically close to a discrete Gaussian distribution over the lattice of large variance. Then, given an input list of lattice vectors, a sieve performs a polynomial number of times a sieving or reduction step, where we use the basic idea that if $\mathbf{v}, \mathbf{w} \in \mathcal{L}(B)$, then also $\mathbf{v} \pm \mathbf{w} \in \mathcal{L}$, and if our list of lattice vectors is long enough, then we will find many pairs $\mathbf{v}, \mathbf{w}$ such that $\|\mathbf{v} \pm \mathbf{w}\| \leq \max\{\|\mathbf{v}\|, \|\mathbf{w}\|\}$. So by simply comparing pairs of vectors in a large list, we can build a list of shorter lattice vectors just by looking at sums/differences of pairs of vectors.

For analyzing these sieving algorithms, an assumption which is commonly made is that if we scale the input lists of each of these sieving applications to lie on the unit sphere, then these vectors are uniformly distributed on the sphere. With this heuristic assumption, the resulting complexities seem to be much closer to reality; the best provable bounds on sieving [PS09] only show that one can solve SVP in time $2^{2.465n+o(n)}$, while heuristic analyses and experimental results suggest the current best time complexity for sieving may be around $2^{0.30n+o(n)}$ in high dimensions n.

Nearest neighbor speed-ups. A naive way to perform pairwise comparisons is to compare all possible pairs of vectors and see if their sum or difference results in a shorter lattice vector. Observe that if two vectors cannot reduce one another, then clearly it must hold that their pairwise angle is larger than 60°. Similarly, if two vectors can reduce each other, then their angle is close to 60°, so a test for pairwise reductions is roughly equivalent to the test if the angle between two vectors of (almost) similar length is at most $\theta_1 = \pi/3$. To guarantee that the output list of shorter vectors is large enough to perform further reductions in the following iterations, Nguyen and Vidick [NV08] showed that N must be of the order $(4/3)^{n/2+o(n)}$. A quadratic search of pairs in each step then leads to a time complexity of the order $(4/3)^{n+o(n)} \approx 2^{0.415n+o(n)}$. However, replacing the quadratic search by nearest neighbor techniques, it is actually possible to perform these searches in sub-quadratic time. Various improvements were suggested over the last few years [BGJ14, BGJ15, Laa15, WLTB11, ZPH13], with the current best time exponent standing at $2^{0.298n+o(n)}$ using spherical LSH [AR15a, LdW15] or cross-polytope LSH [AIL+15, BL15].

Asymmetric choice of α and β. In the reduction phase of a lattice sieve, we are interested in finding pairs of vectors $\mathbf{v}, \mathbf{w}$ at an angle at most $\theta_1 = \pi/3$ if they exist. These vectors allow us to obtain a shorter vector $\mathbf{v} \pm \mathbf{w}$. As described in the previous sections, we fix $k = 1$, and here we make an a priori asymmetric choice of the parameters α and β in spherical LSF:

- $\alpha \in (0, 1)$ represents the **query** parameter for finding the relevant vectors of a given target vector;
- $\beta \in (0, 1)$ represents the **insertion** parameter for finding all filters that a vector is inserted in.

Larger α correspond to more selective querying, which means the querying will be cheaper as fewer buckets are visited. For fixed β, this comes at the cost of having to use more filters to make sure the query is successful, and so more space is required. On the other hand, large β correspond to more selective insertion in the database. In particular, if $\alpha < \beta$, then more effort is spent on constructing the database (preprocessing) than on answering a query. This could be compared to probing techniques of e.g. [Pan06].

Answering a query. Now, given two parameters α and β, we can describe the costs of answering a query (computing the relevant vectors, and comparing to then be computed as follows.

THEOREM 7.1. (COSTS OF ONE QUERY) *Let $k = 1$. Given N points which are uniformly distributed on the sphere and indexed by t spherical filters with parameters α, β, the time to answer a query is:*

$$\mathcal{T}_{query} = \tilde{\mathcal{O}}(t \cdot \mathcal{C}(\alpha) \cdot [1 + N \cdot \mathcal{C}(\beta)]).$$

Proof. First, to answer a query, we compute the $O(t \cdot \mathcal{C}(\alpha))$ relevant filters with minimal overhead using list-decodable random product codes. As we assume the vectors in L are uniformly distributed on the sphere, and all filters cover an equal portion of the sphere, each filter bucket will roughly have the same size. In total each list vector has been inserted in $O(t \cdot \mathcal{C}(\beta))$ filters, leading to $O(N \cdot t \cdot \mathcal{C}(\beta))$ total entries in the filter database, and $O(N \cdot \mathcal{C}(\beta))$ vectors per filter. The cost of computing relevant filters is therefore $\tilde{\mathcal{O}}(t \cdot \mathcal{C}(\alpha))$, and the cost of comparing the vector to all other vectors in these filters is $\tilde{\mathcal{O}}(t \cdot \mathcal{C}(\alpha) \cdot N \cdot \mathcal{C}(\beta))$. □

Total costs of sieving. Besides minor initialization costs of the lattice sieve, the algorithm's complexity can be described by poly(n) applications of the sieve, where each sieve performs N queries to the database. The overall cost of the sieve can thus be summarized as $\tilde{\mathcal{O}}(N \cdot t \cdot \mathcal{C}(\alpha) \cdot [1 + N \cdot \mathcal{C}(\beta)])$. To further analyze the sieve complexity, to make sure that we do not miss nearby vectors at angle 60°, we need to choose the number of filters sufficiently large. In particular, the probability that two vectors at angle 60° are found through a collision is $\tilde{\mathcal{O}}(t \cdot \mathcal{W}_n(\alpha, \beta, \frac{\pi}{3}))$ and must be close to 1. This means that we need to choose t as

$$t = \tilde{\mathcal{O}}\left(1 \Big/ \mathcal{W}_n(\alpha, \beta, \tfrac{\pi}{3})\right).$$

Together with the previous analysis and the bound $N = (4/3)^{n/2+o(n)}$, this means that the total costs of answering queries in sieving can be summarized as:

$$\begin{aligned}\mathcal{T}_1 &= \tilde{\mathcal{O}}\left(\frac{N \cdot \mathcal{C}(\alpha) \cdot [1 + N \cdot \mathcal{C}(\beta)]}{\mathcal{W}_n(\alpha, \beta, \frac{\pi}{3})}\right) \\ &= \tilde{\mathcal{O}}\left(\left(\tfrac{4(1-\alpha^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\right)^{\frac{n}{2}}\left[1 + \left(\tfrac{4(1-\beta^2)}{3}\right)^{\frac{n}{2}}\right]\right).\end{aligned}$$

Besides these costs, in sieving the construction of the database is part of the overall complexity as well. Filling the database with list points means that for each of the N lattice vectors we need to compute the $t{\cdot}\mathcal{C}(\beta)$ relevant filters. This leads to an added "preprocessing" cost of

$$\mathcal{T}_2 = \tilde{\mathcal{O}}\left(\frac{N \cdot \mathcal{C}(\beta)}{\mathcal{W}_n(\alpha, \beta, \frac{\pi}{3})}\right) = \tilde{\mathcal{O}}\left(\left(\tfrac{4(1-\beta^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\right)^{\frac{n}{2}}\right).$$

Note that the space complexity is given by essentially having to store N vectors and $t \cdot N \cdot \mathcal{C}(\beta)$ filter entries:

$$\begin{aligned}\mathcal{S} &= \tilde{\mathcal{O}}\left(N + \frac{N \cdot t \cdot \mathcal{C}(\beta)}{\mathcal{W}_n(\alpha, \beta, \frac{\pi}{3})}\right) \\ &= \tilde{\mathcal{O}}\left(\left(\tfrac{4}{3}\right)^{\frac{n}{2}} + \left(\tfrac{4(1-\beta^2)}{3-4(\alpha^2+\beta^2-\alpha\beta)}\right)^{\frac{n}{2}}\right).\end{aligned}$$

What remains is an optimization over α and β to minimize the time $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$ and the space $\mathcal{S}$. Three options stand out:

- For $\alpha = \beta = \frac{1}{2}$ we obtain the best time complexity:
$$\begin{aligned}\mathcal{T} &= (3/2)^{n/2+o(n)} \approx 2^{0.292n+o(n)}, \\ \mathcal{S} &= (3/2)^{n/2+o(n)} \approx 2^{0.292n+o(n)}.\end{aligned}$$
- For $\alpha = \frac{1}{4}$ and $\beta = \frac{1}{2}$, we obtain the best time complexity without increasing the space complexity:
$$\begin{aligned}\mathcal{T} &= (5/3)^{n/2+o(n)} \approx 2^{0.368n+o(n)}, \\ \mathcal{S} &= (4/3)^{n/2+o(n)} \approx 2^{0.208n+o(n)}.\end{aligned}$$
- For $\alpha \in (\frac{1}{4}, \frac{1}{2})$ and $\beta = \frac{1}{2}$, we obtain the best time complexities for given space complexities, as illustrated by the blue curve Figure 1.

As introduced in [BGJ15] and later also applied in [Laa15, LdW15], it is also possible to obtain the best running time while maintaining a memory complexity of $(4/3)^{n/2+o(n)} = 2^{0.208n+o(n)}$ as indicated in Figure 1. Unfortunately, to obtain this space complexity, one would have to use the Nguyen-Vidick sieve [NV08], which performs quite poorly in practice compared to the GaussSieve [MV10]. For the GaussSieve, the curve in Figure 1 is the best time/memory trade-off we can obtain with this method.

Experimental results. To show the practicability of our proposed sieve algorithm, we implemented the LSF acceleration in the GaussSieve algorithm [MV10]. We ran experiments on an Intel Quad-Core(TM) Q9550 at 2.83GHz with 4GB RAM. Our implementation is not vectorized or parallelized. As input bases, we chose LLL-reduced bases of the SVP lattice challenge [SGBN] in dimensions 50 to 72. We chose $\alpha \in \{0.44, 0.47\}$ as $\alpha = \beta = 0.50$ appears to be slightly worse in practice, even though it is optimal asymptotically. Figure 3 compares the running time of our new algorithm with the GaussSieve.

We observe that the acceleration matches predictions from the theoretical analysis. For example, with $\alpha = 0.44$, we predict an asymptotic time complexity of approximately $2^{0.307n+o(n)}$ or a speed-up of $2^{0.108n+o(n)}$ compared to the GaussSieve algorithm. This closely matches the observed speed-up of $2^{(0.513-0.405)n+o(n)} = 2^{0.108n+o(n)}$. Note that the estimated time complexities do not exactly match the quadratic and sub-quadratic estimates of $2^{0.415n+o(n)}$ and $2^{0.307n+o(n)}$, which is consistent with various previous experiments performed using the GaussSieve [BL15, Laa15, MV10]. This is most likely caused by having to reduce a vector $2^{o(n)}$ times, which leads to a $o(n)$ term in the exponent which for a least-squares fit in low dimensions distorts the leading constant approximation. However, as we only modified the search routine, these factors cancel out when comparing the complexities of different GaussSieve-based algorithms.

Acknowledgments

The authors thank Daniel Dadush, Ilya Razenshteyn, Benne de Weger, and the anonymous reviewers for valuable suggestions and comments. The authors also thank Gottfried Herold and Elena Kirshanova for enlightening discussions in Bochum, and for pointing us to the relation with May and Ozerov's techniques.

References

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

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

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

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

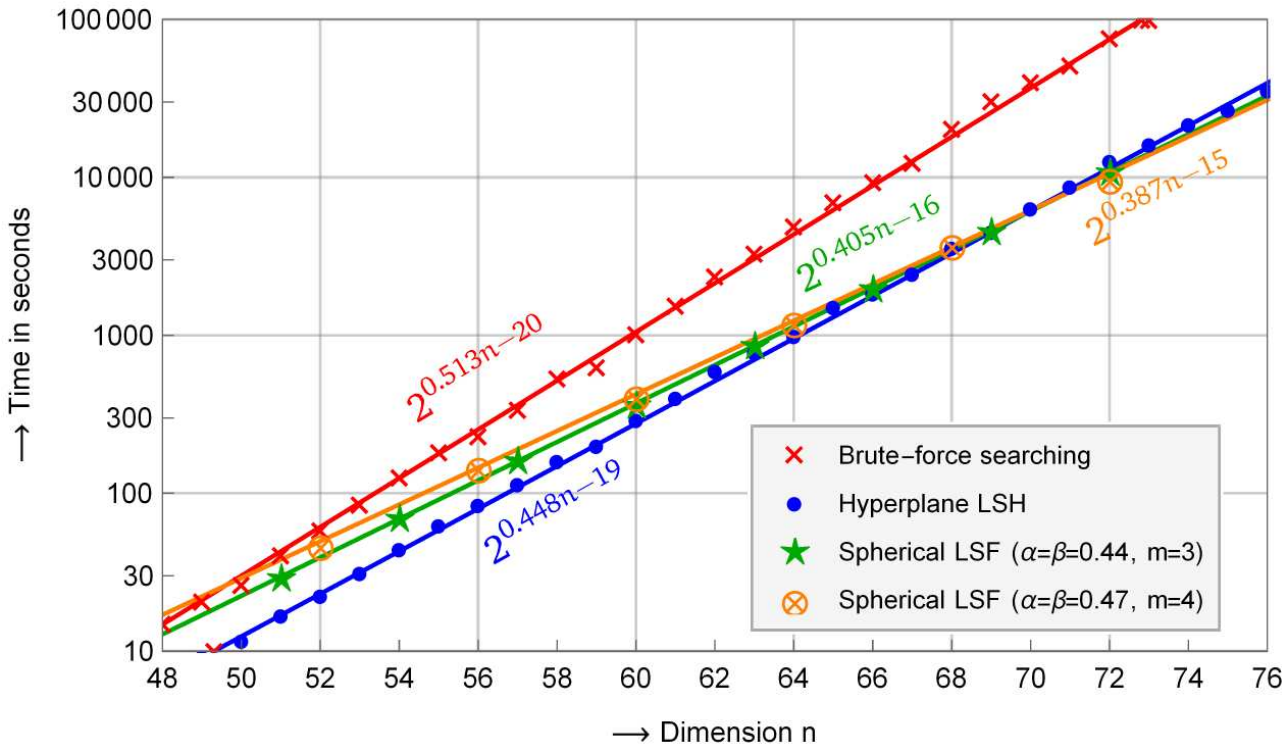

Figure 3: The running times of the basic GaussSieve algorithm (red), the GaussSieve with hyperplane LSH [Cha02, Laa15] (blue), and the GaussSieve with spherical LSF with different parameters (green and orange). Points indicate experimental data, lines indicate least-squares fits of the form 2^{an+b} for constants a, b. For simplicity we have only performed experiments for dimensions which are divisible by the number of blocks m.

[AR15b] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. 2015.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. In *ANTS*, pages 49–70, 2014.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. `http://eprint.iacr.org/`.

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

[BL15] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. *Cryptology ePrint Archive, Report 2015/823*, 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-Interscience, 2000.

[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, 2010.

[FP83] Ulrich Fincke and Michael Pohst. A procedure for determining algebraic integers of given norm. *Computer algebra*, pages 194–202, 1983.

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

[IM99] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1999.

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

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

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

[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.

[LM00] Béatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000.

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

[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.

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

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

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

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

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

[SDI05] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. Nearest-neighbor methods in learning and vision: Theory and practice. *MIT Press*, 2005.

[SGBN] Michael Schneider, Nicolas Gama, P. Baumann, and P. Nobach. SVP challenge: `http://www.latticechallenge.org/svp-challenge/`.

[vdPS13] Joop van de Pol and Nigel P. Smart. Estimating key sizes for high dimensional lattice-based systems. In *IMACC*, pages 290–303, 2013.

[WLTB11] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *ASIACCS*, pages 1–9, 2011.

[ZPH13] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In *SAC*, pages 29–47, 2013.

A Asymptotics of the volume of a wedge

We restate the preliminary lemma and give its proof.

LEMMA A.1. (RESTATEMENT OF LEMMA 2.2) *For arbitrary constants $\alpha, \beta \in (0,1)$, we have*

$$\mathcal{W}_n(\alpha,\beta,\theta) = \mathrm{poly}(n)\cdot\left(\sqrt{1-\gamma^2}\right)^n,$$
$$\text{with } \gamma = \sqrt{\frac{\alpha^2+\beta^2-2\alpha\beta\cos\theta}{\sin^2\theta}}.$$

In the special case $\alpha=\beta$, we obtain

$$\mathcal{W}_n(\alpha,\alpha,\theta) = \mathrm{poly}(n)\cdot\left(\sqrt{1-\frac{2\alpha^2}{1+\cos\theta}}\right)^n.$$

Let us compute the volume of a wedge with parameters (α,β,θ), which is the volume of the intersection of the spherical caps centered at $\mathbf{v}=(1,0,\ldots,0)$ (defined by $\langle\mathbf{v},\mathbf{x}\rangle\geq\alpha$) and at $\mathbf{w}=(\cos\theta,\sin\theta,0,\ldots,0)$ (defined by $\langle\mathbf{w},\mathbf{x}\rangle\geq\beta$).

Let f denote the orthogonal projection from the n-dimensional unit sphere to the two-dimensional plane spanned by the vectors $\mathbf{v}$ and $\mathbf{w}$. For any measurable subset U of the two-dimensional circle, the volume of the preimage $f^{-1}(U)$ is given by:

$$\int_{x,y\in U}\frac{\mu(\mathcal{S}^{n-3})}{\mu(\mathcal{S}^{n-1})}\left(\sqrt{1-x^2-y^2}\right)^{n-4}dx\,dy.$$

Alternatively, if U is described in terms of radial coordinates r and ϕ:

$$\int_{r,\phi\in U}\frac{\mu(\mathcal{S}^{n-3})}{\mu(\mathcal{S}^{n-1})}\left(\sqrt{1-r^2}\right)^{n-4}r\,dr\,d\phi.$$

For all $r\in[0,1]$, let us write $g(r)=\int_{\phi:(r,\phi)\in U}d\phi\in[0,2\pi]$. If U is the projection of a cap intersection, then $g(r)$ is a continuous function from $[0,1]$ to $[0,2\pi]$. We make use of the following lemma:

LEMMA A.2. *Let $\gamma\in(0,1)$, and let $g(r)$ be a continuous function on $(\gamma,1)$ equivalent to $(r-\gamma)^\nu$ for some positive real number $\nu>0$. Then, as $n\to\infty$,*

$$\int_\gamma^1 g(r)\left(\sqrt{1-r^2}\right)^n dr \sim A\cdot\frac{\left(\sqrt{1-\gamma^2}\right)^n}{n^{\nu+1}},$$

where $A=\left(\frac{\gamma}{1-\gamma^2}\right)^{\nu+1}\Gamma(\nu+1)$ does not depend on n.

Proof. We write $g(x)=h(x)(x-\gamma)^\nu$ where $h(x)$ is continuous over $(\gamma,1)$, has limit $h(x)\to 1$ as $x\to\gamma$, and is bounded by $H>0$. Then, using the change of variable $t=n(r-\gamma)$, the integral rewrites as

$$\int_{t=0}^{n(1-\gamma)} h\left(\gamma+\tfrac{t}{n}\right)\left(\tfrac{t}{n}\right)^\nu\left(1-\gamma^2-\tfrac{2\gamma t}{n}-\tfrac{t^2}{n^2}\right)^{n/2}\frac{dt}{n}$$
$$=\frac{(1-\gamma^2)^{n/2}}{n^{\nu+1}}\int_{t=0}^{\infty}\chi_{\{t\leq(1-\gamma)n\}}h\left(\gamma+\tfrac{t}{n}\right)t^\nu$$
$$\times\left(1-\tfrac{2\gamma t}{(1-\gamma^2)n}-\tfrac{t^2}{(1-\gamma^2)n^2}\right)^{n/2}dt.$$

Obviously, the term inside the integral converges to $t^\nu\exp(-\frac{\gamma t}{1-\gamma^2})$ for all $t\geq 0$, and is bounded by

$Ht^\nu \exp(-\frac{\gamma t}{1-\gamma^2})$, which is integrable over $\mathbb{R}^+$. By the dominated convergence theorem, the integral term converges to $\int_0^\infty t^\nu \exp(-\frac{\gamma t}{1-\gamma^2})$, which is a standard Laplace integral equal to the expression for A. This concludes the proof. □

In the case of the wedge, the set U is simply a rounded triangle, defined by the three inequalities $\langle \mathbf{v}, \mathbf{x}\rangle \geq \alpha$, $\langle \mathbf{w}, \mathbf{x}\rangle \geq \beta$, and $\|\mathbf{x}\| = 1$. The point of smallest norm in this rounded triangle is the vector $\mathbf{c}$ satisfying $\langle \mathbf{v}, \mathbf{c}\rangle = \alpha$ and $\langle \mathbf{w}, \mathbf{c}\rangle = \beta$, and its norm is γ. In this case, the function $g(r)$ is null over $[0, \gamma]$, continuous and increasing over $[\gamma, 1]$, and admits the equivalent in $\Theta(r - \gamma)$ when r is close to γ. Thus, applying Lemma A.2, the volume of the wedge is a $\Theta(\frac{1}{n^2}(\sqrt{1-\gamma^2})^n)$ for large n, completing the proof.

Note that the main difference between a wedge and a cap is that for a cap of parameter γ, the function $g(r)$ would be proportional to $\Theta((r-\gamma)^{1/2})$, so the volume of the cap is $\Theta(\frac{1}{n^{3/2}}(\sqrt{1-\gamma^2})^n)$ for large n. Thus, asymptotically, up to some $\sqrt{n}$ factor, the cap of parameter γ and the wedge of parameter α, β, θ have the same volume.

B Random behavior of random product codes

The proof of Theorem 5.1 is based on the technical Lemma C.4 given in the next section.

Proof. The second inequality $q \leq M \cdot \mathcal{W}_n(\alpha, \beta, \theta)$ is straightforward: for any $\mathbf{c} \in C_1 \times \cdots \times C_m$, over the randomness of Q, the probability that $\mathbf{c}$ falls in $\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta}$ is exactly $\mathcal{W}_n(\alpha, \beta, \theta)$. Using the union bound over the $M = B^m$ code words, the result follows.

The first inequality requires more care. We list the two geometric facts required to proceed, facts detailed and proved as Lemma C.4. In the following, we parse $Q\mathbf{v}$ as $(\mathbf{v}_1, \ldots, \mathbf{v}_m)$ and $Q\mathbf{w}$ as $(\mathbf{w}_1, \ldots, \mathbf{w}_m)$.

1. The wedge $\mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta}$ contains $Q^{-1}\Pi$
$$Q^{-1}\Pi \subset \mathcal{W}_{\mathbf{v},\alpha,\mathbf{w},\beta},$$
where Q defines a random rotation and Π is a product of m sub-wedges:
$$\Pi = \prod_{i=1}^{m} \frac{1}{\sqrt{m}} \mathcal{W}_{\sqrt{m}\cdot\mathbf{v}_i,\alpha,\sqrt{m}\cdot\mathbf{w}_i,\beta}.$$

2. The aforementioned sub-wedges have parameters close to the original wedges except with negligible probability over the choice of Q. That is, for all i and for $\epsilon = \tilde{\mathcal{O}}(n^{-1/2})$,
$$\begin{aligned}\frac{\mu(\mathcal{W}_{\sqrt{m}\mathbf{v}_i,\alpha,\sqrt{m}\mathbf{w}_i,\beta})}{\mu(\mathcal{S}^{b-1})} &\geq \mathcal{W}_b(\alpha-\epsilon, \beta-\epsilon, \theta-\epsilon) \\ &\geq \mathcal{W}_n^{1/m}(\alpha, \beta, \theta) \Big/ 2^{\tilde{\mathcal{O}}(\sqrt{n})}.\end{aligned}$$
Because of the inclusion (item 1), the probability p that $C \cap \mathcal{W}_{\mathbf{w},\alpha,\mathbf{v},\beta}$ is not empty must be greater than the probability that each $C_i \cap \mathcal{W}_{\sqrt{m}\mathbf{v}_i,\alpha,\sqrt{m}\mathbf{w}_i,\beta}$ is non-empty. Since all codes C_i are perfectly random and uniformly independent, we have
$$\begin{aligned}q_i &= 1 - \left(1 - \frac{\mu(\mathcal{W}_{\sqrt{m}\mathbf{v}_i,\alpha,\sqrt{m}\mathbf{w}_i,\beta})}{\mu(\mathcal{S}^{b-1})}\right)^B, \\ q &\geq q_1 q_2 \ldots q_m - \mathrm{negl}(n).\end{aligned}$$
For conciseness, we set $W = \mathcal{W}_n(\alpha, \beta, \theta)$. Now, from the second item, we deduce that $q_i = 1 - \left(1 - W^{1/m}/2^{\tilde{\mathcal{O}}(\sqrt{n})}\right)^B$. We now discuss the two cases:

- If $B \cdot W^{1/m} \to 0$, then $q_i = \frac{W^{1/m}\cdot B}{2^{\tilde{\mathcal{O}}(\sqrt{n})}}$, so
$$q \geq \left(\frac{W^{1/m}\cdot B}{2^{\tilde{\mathcal{O}}(\sqrt{n})}}\right)^m \geq \frac{W \cdot B^m}{2^{\tilde{\mathcal{O}}(\sqrt{n})}} = \frac{W \cdot M}{2^{\tilde{\mathcal{O}}(\sqrt{n})}}.$$
- If $B \cdot W^{1/m} \geq 2^{\tilde{\mathcal{O}}(\sqrt{n})}$, then $q_i = 1 - \mathrm{negl}(n)$, and we conclude that $q \geq 1 - \mathrm{negl}(n)$.

In both cases, that provides the desired result. □

C Approximation of wedges by subwedges

This subsection is devoted to the proof of our main technical Lemma C.4, given at the end of the section.

Let $\mathcal{U}_n$ denote the uniform distribution over $\mathcal{S}^{n-1}$, and let $\mathcal{N}_{\sigma^2}$ denote the centered normal distribution over $\mathbb{R}$ of variance σ^2. We mainly use the normal distribution as a proxy to study the uniform distribution, as the normal distribution is simpler to project on subspaces. We recall that the distribution of the squared norm of a vector $\mathbf{v}$, sampled from a $\mathcal{N}_1^n$ distribution, is called the chi-squared distribution with n degrees of freedom, and is denoted χ_n^2.

Lemma C.1. (χ^2 Concentration [LM00]) *For $\mathbf{v}$ sampled as $\mathbf{v} \sim \mathcal{N}_1^n$, we have*
$$\begin{aligned}\Pr(\|\mathbf{v}\|^2 - n \geq 2\sqrt{nt} + 2t) &\leq \exp(-t), \\ \Pr(\|\mathbf{v}\|^2 - n \leq 2\sqrt{nt}) &\leq \exp(-t).\end{aligned}$$
In particular, for $t = \log^2 n$ we have $\frac{1}{n}\|\mathbf{v}\|^2 = 1 + \tilde{O}(n^{-1/2})$ except with negligible probability in n.

In the following lemmas we assume that $m = \mathrm{polylog}(n)$, so that blockwise norms and dot products are strongly concentrated around their means.

Lemma C.2. (Blockwise norms) *Let $\mathbf{v}$ be sampled as $\mathbf{v} \sim \mathcal{U}_n$, and let us write $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_m)$ with $\mathbf{v}_i \in \mathbb{R}^b$ for $i = 1, \ldots, m$. Then, except with negligible probability, we have that for all $i = 1, \ldots, m$:*
$$\|\mathbf{v}_i\|^2 = \frac{1}{m}\left(1 + \tilde{O}\big(n^{-1/2}\big)\right).$$

Proof. The distribution $\mathcal{U}_n$ can be sampled by drawing $\mathbf{v}' \leftarrow \mathcal{N}_{1/n}^n$ and taking $\mathbf{v} = \mathbf{v}'/\|\mathbf{v}'\|$. Writing $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_m)$ and $\mathbf{v}' = (\mathbf{v}'_1, \ldots, \mathbf{v}'_m)$, we then have $\mathbf{v}_i = \mathbf{v}'_i/\|\mathbf{v}'\|$. We conclude the proof by applying Lemma C.1 on each $\|\mathbf{v}'_i\|$ and on $\|\mathbf{v}'\|$. □

LEMMA C.3. (BLOCKWISE INNER PRODUCTS) *Let $\mathbf{v}, \mathbf{w} \sim \mathcal{U}_n$ be independent uniform samples conditioned on $\langle \mathbf{v}, \mathbf{w} \rangle = 0$. Then, for all i, we have $|\langle \mathbf{w}_i, \mathbf{v}_i \rangle| \leq \tilde{O}(n^{-1/2})$ except with negligible probability.*

Proof. The distribution of $(\mathbf{v}, \mathbf{w})$ may be sampled by applying Gram-Schmidt orthogonalization to independent normal vectors as follows, where $\mathbf{v}', \mathbf{w}' \sim \mathcal{N}_{1/n}^n$.

$$\mathbf{v} = \frac{\mathbf{v}'}{\|\mathbf{v}'\|}, \quad \mathbf{w} = \frac{\mathbf{w}' - \langle \mathbf{w}', \mathbf{v} \rangle \mathbf{v}}{\|\mathbf{w}' - \langle \mathbf{w}', \mathbf{v} \rangle \mathbf{v}\|}.$$

By Lemma C.1, we have that except with negligible probability, $\frac{3}{4} \leq \|\mathbf{v}\|, \|\mathbf{w}\| \leq \frac{5}{4}$. Additionally, $\langle \mathbf{w}', \mathbf{v} \rangle$ is distributed according to $\mathcal{N}_{1/n}$ under the randomness of $\mathbf{w}'$, so we have $|\langle \mathbf{w}', \mathbf{v} \rangle| \leq (\log n)/\sqrt{n}$ except with negligible probability. First, this implies that $\|\mathbf{w}' - \langle \mathbf{w}', \mathbf{v} \rangle \mathbf{v}\| \geq \frac{1}{2}$, and we derive:

$$|\langle \mathbf{w}_i, \mathbf{v}_i \rangle| \leq 2 \left(\langle \mathbf{w}'_i, \mathbf{v}_i \rangle - \langle \mathbf{w}', \mathbf{v} \rangle \|\mathbf{v}_i\| \right).$$

Again, under the randomness of $\mathbf{w}'$, the inner products $\langle \mathbf{w}'_i, \mathbf{v}_i \rangle$ are distributed according to $\mathcal{N}_{\|\mathbf{v}_i\|^2/n}$, so with overwhelming probability we have:

$$\langle \mathbf{w}_i, \mathbf{v}_i \rangle| \leq 4 \cdot \|\mathbf{v}_i\| \cdot (\log n)/\sqrt{n}.$$

Finally, we invoke Lemma C.2 to conclude that, for all i, we have $|\langle \mathbf{w}_i, \mathbf{v}_i \rangle| \leq O((\log n)/\sqrt{n})$ except with negligible probability. □

Using the previous lemmas, we are now ready to prove the main technical result.

LEMMA C.4. (APPROXIMATION BY WEDGES) *Let $\mathbf{v}, \mathbf{w}$ be independent uniformly random samples from $\mathcal{S}^{n-1}$ conditioned on the fact that $\langle \mathbf{v}, \mathbf{w} \rangle = \cos\theta$. Then, except with negligible probability, for some $\epsilon = \tilde{O}(n^{-1/2})$ the following holds for all i:*

$$\frac{\mu(\mathcal{W}_{\sqrt{m}\mathbf{v}_i, \alpha, \sqrt{m}\mathbf{w}_i, \beta})}{\mu(\mathcal{S}^{b-1})} \geq \mathcal{W}_b(\alpha - \epsilon, \beta - \epsilon, \theta - \epsilon).$$

Additionally, the wedge product

$$\Pi = \prod_{i=1}^{m} \frac{1}{\sqrt{m}} \mathcal{W}_{\sqrt{m}\mathbf{v}_i, \alpha, \sqrt{m}\mathbf{w}_i, \beta}$$

is included in $\mathcal{W}_{\mathbf{v}, \alpha, \mathbf{w}, \beta}$.

Proof. Let us start by proving the inclusion $\Pi \subset \mathcal{W}_{\mathbf{v}, \alpha, \mathbf{w}, \beta}$. Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \in \Pi$, that is, each $\mathbf{x}_i$ belongs to $\frac{1}{\sqrt{m}}\mathcal{S}^{b-1}$ and satisfies $\langle \mathbf{x}_i, \mathbf{v}_i \rangle \geq \frac{\alpha}{m}$ and $\langle \mathbf{x}_i, \mathbf{w}_i \rangle \geq \frac{\beta}{m}$. Summing over all i, we obtain $\|\mathbf{x}\|^2 = 1$, $\langle \mathbf{x}, \mathbf{v} \rangle \geq \alpha$ and $\langle \mathbf{x}, \mathbf{w} \rangle \geq \beta$, which concludes the proof of the inclusion.

We now move to the proof of the main result. First note that $\mathcal{W}_{\mathbf{a}, \alpha, \mathbf{b}, \beta}$ has volume $\mathcal{W}(\frac{\alpha}{\|\mathbf{a}\|}, \frac{\beta}{\|\mathbf{b}\|}, \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\|\|\mathbf{b}\|})$, so it suffices to prove that $\|\mathbf{v}_i\|^2 = \frac{1}{m}(1 + \tilde{O}(n^{-1/2}))$, $\|\mathbf{w}_i\|^2 = \frac{1}{m}(1 + \tilde{O}(n^{-1/2}))$ and $\langle \mathbf{v}_i, \mathbf{w}_i \rangle = \frac{\cos\theta}{m}(1 + \tilde{O}(n^{-1/2}))$. The first two statements follow from Lemma C.2 with overwhelming probability. For the last one, write $\mathbf{w}' = \cos\theta \cdot \mathbf{v} + \sin\theta \cdot \mathbf{w}'$ where $\mathbf{v}, \mathbf{w}'$ are sampled uniformly on the sphere conditioned on being orthogonal. Then:

$$\begin{aligned} \langle \mathbf{v}_i, \mathbf{w}_i \rangle &= \cos\theta \cdot \|\mathbf{v}_i\|^2 + \langle \mathbf{v}_i, \mathbf{w}'_i \rangle \\ &= \frac{\cos\theta}{m} \left(1 + \tilde{O}\left(\frac{1}{\sqrt{n}} \right) \right) + O\left(\frac{\log n}{\sqrt{n}} \right), \end{aligned}$$

where the second term follows from Lemma C.3. This concludes the proof. □