Fair Near Neighbor Search: Independent Range Sampling in High Dimensions

Martin Aumüller[1], Rasmus Pagh[1], and Francesco Silvestri[2]

[1]BARC and IT University of Copenhagen, Denmark, {maau,pagh}@itu.dk
[2]University of Padova, Italy, silvestri@dei.unipd.it

Abstract

Similarity search is a fundamental algorithmic primitive, widely used in many computer science disciplines. There are several variants of the similarity search problem, and one of the most relevant is the r-near neighbor (r-NN) problem: given a radius $r > 0$ and a set of points S, construct a data structure that, for any given query point q, returns a point p within distance at most r from q. In this paper, we study the r-NN problem in the light of fairness. We consider fairness in the sense of equal opportunity: all points that are within distance r from the query should have the same probability to be returned. Locality sensitive hashing (LSH), the most common approach to similarity search in high dimensions, does not provide such a fairness guarantee. To address this, we propose efficient data structures for r-NN where all points in S that are near q have the same probability to be selected and returned by the query. Specifically, we first propose a black-box approach that, given any LSH scheme, constructs a data structure for uniformly sampling points in the neighborhood of a query. Then, we develop a data structure for fair similarity search under inner product, which requires nearly-linear space and exploits locality sensitive filters.

1 Introduction

In recent years, there has been an increasing interest in building algorithms that achieve *fairness* under certain technical definitions of fairness [15]. The goal is to remove, or at least minimize, unethical behavior such as discrimination and bias in algorithmic decision making. There is no unique definition of fairness (see [18] and references therein), but different formulations that depend on the computational problem and on the ethical goals we aim for. Fairness goals are often defined in the political context of socio-technical systems [24], and have to be seen in an interdisciplinary spectrum covering many fields outside computer science [28]. In machine learning, algorithmic fairness is often considered with respect to a classification process. The concept of "equal opportunity" requires that people who can achieve a certain advantaged outcome, such as finishing a university degree or paying back a loan, have equal opportunity of being able to get access to it in the first place, such as getting into a university program or getting approval of a loan.

In this paper, we will propose what fairness could mean in the setting of similarity search. Similarity search is an important primitive in many applications in computer science such as machine learning, recommender systems, and data mining. One of the most common formulations of similarity search is the r-near neighbor (r-NN) problem: for a given radius $r > 0$, a distance function $\mathcal{D}(\cdot, \cdot)$ that reflects the (dis)similarity between two data points, and a set S of data points, the r-NN problem requires to construct a data structure that, given a query point $\mathbf{q}$, returns a point $\mathbf{p}$ such that $\mathcal{D}(\mathbf{p}, \mathbf{q}) \leq r$, if such a point exists.

We consider fairness in the sense of equal opportunity. Our goal is to develop a data structure for the r-near neighbor problem where all points within distance r from the given query have the same probability to be returned: if $B_S(\mathbf{q}, r)$ is the ball of input points at distance at most r from a query $\mathbf{q}$, we would like that the each point in $B_S(\mathbf{q}, r)$ is returned as near neighbor of $\mathbf{q}$ with probability $1/|B_S(\mathbf{q}, r)|$. For all constructions presented in this paper, these guarantees hold only in the absence of a failure event that happens with probability at most δ for some small $\delta > 0$. In other words, we aim at solving the following sampling problem:

Definition 1. *Consider a set $S \subseteq \mathcal{X}$ of n points in a metric space $(\mathcal{X}, \mathcal{D})$. The* r-near neighbor sampling problem *(r-NNS) asks to construct a data structure for S to solve the following task with probability at least $1 - \delta$: Given query $\mathbf{q}$, return a point $\mathbf{p}$ uniformly sampled from the set $B_S(\mathbf{q}, r)$.*

To see an example application for such a system, consider a recommender system used by a newspaper to recommend articles to users. Popular recommender systems based on matrix factorization [22] give recommendations by computing the inner product similarity of a user feature vector with all item feature vectors using some efficient similarity search algorithm. It is common practice to recommend those items that have the largest inner product with the user. However, in general it is not clear that it is desirable to recommend the "closest" articles. Indeed, it might be desirable to recommend articles that are on the same topic but are not *too* aligned with the user feature vector, and may provide a different perspective [1]. Knowing a solution to the r-NNS problem allows to make a recommendation slightly further away from the user feature vector, but still within a certain distance threshold, as likely to be returned as the closest feature vectors.

To the best of our knowledge, previous results focused on exact near neighbor sampling in the Euclidean space up to three dimensions [3, 4, 19, 26]. Although these results might be extended to $\mathbb{R}^d$ for any $d > 1$, they suffer the *curse of dimensionality* as the query time increases exponentially with the dimension, making the data structures too expensive in high dimensions. These bounds are unlikely to be significantly improved since several conditional lower bounds show that an exponential dependency on d in query time or space is unavoidable for *exact* near neighbor search (see e.g., [7, 30]).

A common solution to the near neigbor problem in high dimensions is provided by the locality-sensitive hashing (LSH) framework proposed by Indyk and Motwani [20]. In this framework, which is formally introduced in Section 2, data points are hashed into buckets and only colliding points are inspected when answering a query. Locality-sensitive hash functions are designed in such a way that the collision probability between two points is a decreasing function of their distance [11]. As we will show in Section 2, the standard LSH approach is not suitable for solving the r-NNS problem. While the uniformity property required in r-NNS can be trivially achieved by finding *all* r-near neighbor of a query and then randomly selecting one of them, this is computationally inefficient since the query time is a function of the size of the neighborhood. One contribution in this paper is the description of a much more efficient data structure, that still uses LSH in a black-box way.

Observe that the definition above does not require different query results to be independent. If the query algorithm is deterministic and randomness is only used in the construction of the data structure, the returned near neighbor of a query will always be the same. Furthermore, the result of a query $\mathbf{q}$ might influence the result of a different query $\mathbf{q}'$. This motivates us to extend the r-NNS problem to the scenario where we aim at independence.

Definition 2. *Consider a set $S \subseteq \mathcal{X}$ of n points in a metric space $(\mathcal{X}, \mathcal{D})$. The* r-near neighbor independent sampling problem *(r-NNIS) asks to construct a data structure for S that for any sequence of up to n queries $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ satisfies the following properties with probability at least $1 - \delta$:*

1. *For each query $\mathbf{q}_i$, it returns a point $\mathrm{OUT}_{i,\mathbf{q}_i}$ uniformly sampled from $B(\mathbf{q}_i, r)$;*

2. *The point returned for query $\mathbf{q}_i$, with $i > 1$, is independent of previous query results. That is, for any $\mathbf{p} \in B(\mathbf{q}_i, r)$ and any sequence $\mathbf{p}_1, \ldots, \mathbf{p}_{i-1}$, we have that*

$$\Pr[\mathrm{OUT}_{i,\mathbf{q}}{=}\mathbf{p} \mid \mathrm{OUT}_{i-1,\mathbf{q}_{i-1}}{=}\mathbf{p}_{i-1}, \ldots, \mathrm{OUT}_{1,\mathbf{q}_1}{=}\mathbf{p}_1] = 1/|B_S(\mathbf{q}, r)|.$$

We note that in the low-dimensional setting [19, 4, 3], the r-near neighbor independent sampling problem is usually called *independent range sampling* (IRS).

1.1 Results

In this paper we propose several solutions to the r-NN(I)S problem under different settings. We hope that our fairness view on independent range sampling gives a new perspective to fair similarity search. On the technical side, the paper contributes the following methods:

- Section 3 describes a solution to the r-near neighbor sampling problem with running time guarantees matching those of standard LSH up to polylog factors. The data structure uses an independent permutation of the data points and inspects buckets according to the order of points under this permutation.

- Section 4 shows how to solve the independent sampling case, using a more involved data structure. The query time still matches that of standard LSH up to poly-logarithmic factors. Each bucket is equipped with a count-sketch and the algorithm works by repeatedly sampling points within a certain window from the permutation.

- Lastly, in Section 5 we introduce an easy-to-implement nearly-linear space data structure based on the LSH filter approach put forward in [8, 13]. As each input point appears once in the data structure, the data structure can be easily adapted to solve the NNIS problem. The data structure is described for similarity search under inner product, but it can be adapted to other metrics with standard techniques.

1.2 Previous work

Independent Range Sampling The IRS problem is a variant of a more general problem known as *random sampling queries*, introduced in the '80s for computing statistics from a database: given a database and a query (e.g., range, relational operator), return a random sample of the query result set. The random sample can be used for estimating aggregate queries (e.g., sum or count) or for query optimization. We refer to the survey [25] for a more detailed overview. One of the first results on range sampling is by Olken and Rotem [26], who proposed a data structure based on a R-tree for sampling points within a region described by a union of polygons. Hu et al. introduced in [19] the IRS problem, where the focus is on extracting *independent* random samples. The paper presents a data structure for the unweighted, dynamic version in one dimension. Later, Afshani and Wei [4] proposed a data structure that solves the weighted case in one dimension, and the unweighted case in three dimensions for half-space queries. In a very recent manuscript, Afshani and Phillips [3] extended this line of work and provide a lower bound for the worst-case query time with nearly-linear space.

Applications of Independent Range Sampling Since near-neighbor search in the context of recommender systems works usually on medium- to high-dimensional data sets, we see IRS as a natural primitive to introduce a concept of fairness into recommender systems. As mentioned in [3], IRS is a useful statistical tool in data analysis and has been used in the database community for many years. Instead of obtaining the set of all r-near neighbors, which could be a very costly operation, an analyst (or algorithm) might require only a small sample of "typical" data points sampled independently from a range to provide statistical properties of the data set. Another useful application is diversity maximization in a recommender system context. As described by Adomavicius and Kwon in [2], recommendations can be made more diverse by sampling k items from a larger top-ℓ list of recommendations at random. Our data structures could replace the final near neighbor search routine employed in such systems. Finally, we observe that IRS can have applications even in the context of discrimination discovery. For instance Luong et al. [23] used k nearest neighbor search for detecting significant differences of treatment among users with similar, legally admissible, characteristics. Our data structure can be used in this context for speed up the procedure by sampling a subset of points.

2 Preliminaries

2.1 Near neighbor search

Let $(\mathcal{X}, \mathcal{D})$ be a high dimensional metric space over the set $\mathcal{X}$ with distance function $\mathcal{D}(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. As an example, we can set $\mathcal{X} = \mathbb{R}^d$ and let $\mathcal{D}(\cdot,\cdot)$ denote the ℓ_p norm, with $p \in \{1, 2, +\infty\}$. Given a set $S \subseteq \mathcal{X}$ of n points in the metric space $(\mathcal{X}, \mathcal{D})$ and a radius $r > 0$, the *r-Near Neighbor* (r-NN) problem requires to construct a data structure that, for any given query point $\mathbf{q} \in \mathcal{X}$, returns a point $\mathbf{p} \in S$ such that $\mathcal{D}(\mathbf{q}, \mathbf{p}) \leq r$, if such a point exists. In search of efficient solutions to the r-NN problem, several works have targeted the approximate version, named *(c, r)-Approximate Near Neighbor* ((c, r)-ANN) where $c > 1$ is the approximation factor: for any given query $\mathbf{q} \in \mathcal{X}$, the data structure can return a point $\mathbf{p}$ with distance $\mathcal{D}(\mathbf{q}, \mathbf{p}) \leq c \cdot r$. We refer to the survey [6] for an overview on techniques for approximate near neighbor search in high dimensions. We say that two points $\mathbf{p}, \mathbf{q}$ are *r-near* if $\mathcal{D}(\mathbf{p}, \mathbf{q}) \leq r$, *$(c, r)$-near* if $r < \mathcal{D}(\mathbf{p}, \mathbf{q}) \leq c \cdot r$, and *far* if $\mathcal{D}(\mathbf{p}, \mathbf{q}) > c \cdot r$ (if r is clear in the context, we will use the term *near* instead of r-near). We use $B_S(\mathbf{q}, r)$ to denote the points in S within the ball of radius r and center $\mathbf{q}$ (i.e. $B_S(\mathbf{q}, r) = \{p \in S \colon \mathcal{D}(\mathbf{p}, \mathbf{q}) \leq r\}$), and let $b_S(\mathbf{q}, r) = |B_S(\mathbf{q}, r)|$.

For the sake of notational simplicity, we assume that evaluating a distance $\mathcal{D}(\cdot,\cdot)$ or an hash function takes $O(1)$ time, and that each entry in $\mathcal{X}$ can be stored in $O(1)$ memory words. If this is not the case and a point in $\mathcal{X}$ requires σ words and τ time for reading a point, computing $\mathcal{D}(\cdot,\cdot)$ or an hash function (with $\sigma \leq \tau$), then it suffices to add the additive term $n \cdot \sigma$ to our space bounds (we store a point once in memory and then refer to it with constant size pointers), and multiply construction and query times by a factor τ (since the time of our algorithms is almost equivalent to the number of distance computations or hash computations). We assume the length of a memory word to be $\Theta(\log n)$ bits, where n is the input size.

2.2 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a common tool for solving the ANN problem and was introduced in [20].

Definition 3. *A distribution $\mathcal{H}$ over maps $h : \mathcal{X} \to U$, for a suitable set U, is called $(r, c \cdot r, p_1, p_2)$-sensitive if the following holds for any $\mathbf{x}$, $\mathbf{y} \in \mathcal{X}$:*

- *if $\mathcal{D}(\mathbf{x}, \mathbf{y}) \leq r$, then $\Pr_h[h(\mathbf{x}) = h(\mathbf{y})] \geq p_1$;*
- *if $\mathcal{D}(\mathbf{x}, \mathbf{y}) > c \cdot r$, then $\Pr_h[h(\mathbf{x}) = h(\mathbf{y})] \leq p_2$.*

The distribution $\mathcal{H}$ is called an LSH family, and has quality $\rho = \rho(\mathcal{H}) = \frac{\log p_1}{\log p_2}$.

For the sake of simplicity, we assume that $p_2 \leq 1/n$: if $p_2 > 1/n$, then it suffices to create a new LSH family $\mathcal{H}_K$ obtained by concatenating $K = \Theta\left(\log_{p_2}(1/n)\right)$ i.i.d. hashing functions from $\mathcal{H}$. The new family $\mathcal{H}_K$ is (r, cr, p_1^K, p_2^K)-sensitive and ρ does not change.

The standard approach to (c, r)-ANN using LSH functions is the following. Let $\ell_1, \ldots, \ell_L$ be L functions randomly and uniformly selected from $\mathcal{H}$. The data structure consists of L hash tables $H_1, \dots H_L$: each hash table H_i contains the input set S and uses the hash function ℓ_i to split the point set into buckets. For each query $\mathbf{q}$, we iterate over the L hash tables: for any hash function, compute $\ell_i(\mathbf{q})$ and compute, using H_i, the set $S_{i,\ell_i(\mathbf{q})} = \{\mathbf{p} : \mathbf{p} \in S, \ell_i(\mathbf{p}) = \ell_i(\mathbf{q})\}$ of points in S with the same hash value; then, compute the distance $\mathcal{D}(\mathbf{q}, \mathbf{p})$ for each point $\mathbf{p} \in S_{i,\ell_i(\mathbf{q})}$. The procedure stops as soon as a (c, r)-near point is found. It stops and returns $\perp$ if there are no remaining points to check or if it found more than $3L$ far points [20]. By setting $L = \Theta\left(p_1^{-1} \log n\right)$, the above data structure returns, with probability at least $1 - 1/n$, a (c, r)-near point of $\mathbf{q}$ in time $O(L) = O(n^\rho \log n)$ and space $O(Ln) = O\left(n^{1+\rho} \log n\right)$, assuming that the hash function can be computed in $O(1)$ time and each hash value requires $O(1)$ space. By avoiding stopping after $3L$ far points have been found, the above data structure can be adapted to return a r-near point of $\mathbf{q}$ in expected time $O(n^\rho \log n + b_S(\mathbf{q}, cr))$.

Standard LSH families do not satisfy the fairness definitions introduced above. Consider the simple case with the data set $S = \{\mathbf{x}, \mathbf{y}\}$ and where $\mathcal{D}(\mathbf{x}, \mathbf{y}) = r$ and query $\mathbf{q} = \mathbf{x}$: with a standard LSH approach, we have that $\mathbf{q}$ collides with $\mathbf{x}$ with probability 1, while $\mathbf{q}$ collides with $\mathbf{y}$ only once in expectation; therefore, it is likely that the first near point of $\mathbf{q}$ found by the data structure is $\mathbf{x}$. This is true even if the order in which the $L = \tilde{O}\left(p_1^{-1}\right)$ hash tables are visited is randomized. However, if performance is not sought, the standard LSH method can be easily adapted to randomly return a point in $B_S(q, r)$: it suffices to find all near neighbors of $\mathbf{q}$ and to randomly select one of them. However, this approach could require expected time $\tilde{O}\left(b_S(\mathbf{q}, r) n^\rho + b_S(\mathbf{q}, cr)\right)$.

2.3 Sketch for distinct elements

In Section 4 we will use sketches for estimating the number of distinct elements. Consider a stream of n elements $x_1, \dots x_m$ in the domain $[n] = \{1, \dots, n\}$ and let F_0 be the number of distinct elements in the stream (i.e., the zeroth-frequency moment). Several papers have studied sketches (i.e., compact data structures) for estimating F_0. For the sake of simplicity we use the simple sketch in [10], which generalizes the seminal result by Flajolet and Martin [17]. The data structure consists of $\Delta = \Theta(\log(1/\delta))$ lists $L_1, \dots L_\Delta$; for $1 \leq w \leq \Delta$, L_w contains the $t = \Theta\left(1/\epsilon^2\right)$ distinct smallest values of the set $\{\psi_w(x_i) : 1 \leq i \leq m\}$, where $\psi_w : [n] \to [n^3]$ is a hash function picked from a pairwise independent family. It is shown in [10] that the median $\hat{F}_0$ of the values $tn^3/v_0, \dots, tn^3/v_\Delta$, where v_w denotes the t-th smallest value in L_w, is an ϵ-approximation to the number of distinct elements in the stream with probability at least $1 - \delta$: that is, $(1 - \epsilon)F_0 \leq \hat{F}_0 \leq (1 + \epsilon)F_0$. The data structure requires $O\left(\epsilon^{-2} \log m \log(1/\delta)\right)$ bits and $O(\log(1/\epsilon) \log m \log(1/\delta))$ query time. A nice property of this sketch is that if we split the stream in k segments and we compute the sketch of each segment, then it is possible to reconstruct the sketch of the entire stream by combining the sketches of individual segments (the cost is linear in the sketch size).

3 r-near neighbor sampling

We start with a simple data structure for the r-near neighbor sampling problem in high dimensions that leverages on LSH: with high probability, for each given query $\mathbf{q}$, the data structure returns a point uniformly sampled in $B_S(\mathbf{q}, r)$. We will then see that, with a small change in the query procedure, the data structure supports independent sampling when the same query point is repeated.

The main idea is quite simple. We initially assign a (random) rank to each point in S using a random permutation, and then construct the standard LSH data structure for solving the r-NN problem on S. For a given query $\mathbf{q}$ and assuming that all points in $B_S(\mathbf{q}, r)$ collide with $\mathbf{q}$ at least once, the data structure returns the point in $B_S(\mathbf{q}, r)$ with the lowest rank in the random permutation. The random permutation, which is independent of the collision probability, guarantees that all points in $B_S(\mathbf{q}, r)$ have the same probability to be returned as near point.

Construction. Let $\mathcal{H}$ be an $(r, c \cdot r, p_1, p_2)$-sensitive LSH family with $p_2 = O(1/n)$ (see Section 2.2). Let $\ell_1, \ldots, \ell_L$ be $L = \Theta\left(p_1^{-1} \log n\right)$ hash functions selected independently and uniformly at random from $\mathcal{H}$. The n points in S are randomly permuted (possibly, with an universal hash family over maps $\mathcal{X} \to [n^3]$) and let $r(\mathbf{p})$ denote the rank of point $\mathbf{p} \in S$ after the permutation. For each ℓ_i, we partition the input points S according to the hash values of ℓ_i and denote with $S_{i,j} = \{\mathbf{p} : \mathbf{p} \in S, \ell_i(\mathbf{p}) = j\}$ the set of points with hash value j under ℓ_i. We store points in each $S_{i,j}$ sorted by increasing ranks.

Query. Let $\mathbf{q}$ be the query point. The query procedure extracts from the set $S_{\mathbf{q}} = \bigcup_{i=1}^{L} S_{i,\ell_i(\mathbf{q})}$ (i.e., points colliding with $\mathbf{q}$ under the L hash functions) the r-near point of q with lowest rank. This is done as follows. Initialize $r_{\min} = +\infty$ and $\mathbf{x}_{\min} = \perp$, where $\perp$ denotes a special symbol meaning "no near neighbor". For each i in $\{1, \ldots L\}$, scan $S_{i,j}$ until an r-near point $\mathbf{x}_i$ is found or the end of the array is reached: in the first case and if $r_{\min} > r(\mathbf{x}_i)$, we set $r_{\min} = r(\mathbf{x}_i)$ and $\mathbf{x}_{\min} = \mathbf{x}_i$ (we note that $\mathbf{x}_i$ is the r-near point in $S_{i,j}$ with lowest rank). After scanning the L buckets, we return $\mathbf{x}_{\min}$.

Theorem 1. *With high probability $1 - 1/n$, the above data structure solves the r-NNS problem: given a query $\mathbf{q}$, each point $\mathbf{p} \in B_S(\mathbf{q}, r)$ is returned as near neighbor of $\mathbf{q}$ with probability $1/b_S(\mathbf{q}, r)$. The data structure requires $\Theta\left(n^{1+\rho} \log n\right)$ words, $\Theta\left(n^{1+\rho} \log n\right)$ construction time, and the expected query time is*

$$O\left(\left(n^{\rho} + \frac{b_S(\mathbf{q}, cr)}{b_S(\mathbf{q}, r)}\right) \log n\right).$$

Proof. Since $L = \Theta\left(p_1^{-1} \log n\right)$, all points in $B_S(\mathbf{q}, r)$ collide with $\mathbf{q}$ at least once (i.e., $B_S(\mathbf{q}, r) \subseteq \bigcup_{i=1}^{L} S_{i,\ell_i(\mathbf{q})}$) with probability at least $1 - 1/n$. The initial random permutation guarantees that each point in $B_S(\mathbf{q}, r)$ has probability $1/b_S(\mathbf{q}, r)$ to have the smallest rank in $B_S(\mathbf{q}, r)$, and hence to be returned as output.

The space complexity and construction time follow since the algorithm builds and stores L tables with n references to points in S. To upper bound the expected query time, we introduce the random variable $X_{\mathbf{p},i}$ for each point $\mathbf{p}$ with $\mathcal{D}(\mathbf{p}, \mathbf{q}) > r$: $X_{\mathbf{p},i}$ takes value 1 if $\mathbf{p}$ collides with $\mathbf{q}$ under ℓ_i and if it has a rank smaller than all points in $B_S(\mathbf{q}, r)$; $X_{\mathbf{p},i}$ is 0 otherwise. Points $\mathbf{p}$ and $\mathbf{q}$ collide with probability $\Pr[\ell_i(\mathbf{q}) = \ell_i(\mathbf{p})]$ and $\mathbf{p}$ has rank smaller than points in $B_S(\mathbf{q}, r)$ with probability $1/(b_S(\mathbf{q}, r) + 1)$; thus we have that $\mathrm{E}[X_{\mathbf{p},i}] = \Pr[\ell_i(\mathbf{q}) = \ell_i(\mathbf{p})] / (b_S(\mathbf{q}, r) + 1)$ (note that the random bits used for LSH construction and for the initial permutation are independent). Let μ be the number of points inspected by the query algorithm over all repetitions. By linearity of expectation, we get:

$$\begin{aligned} \mathrm{E}[\mu] &\leq L + \mathrm{E}\left[\sum_{\mathbf{p} \in S \setminus B_S(\mathbf{q}, r)} \sum_{i=1}^{L} X_{\mathbf{p},i}\right] \leq L + L \cdot \sum_{\mathbf{p} \in S \setminus B_S(\mathbf{q}, r)} \frac{\Pr[\ell_1(\mathbf{q}) = \ell_1(\mathbf{p})]}{b_S(\mathbf{q}, r)} \\ &\leq L + L \frac{b_S(\mathbf{q}, cr) p_1 + n p_2}{b_S(\mathbf{q}, r)} = O\left(\left(n^{\rho} + \frac{b_S(\mathbf{q}, cr)}{b_S(\mathbf{q}, r)}\right) \log n\right), \end{aligned}$$

since $p_2 = O(1/n)$. □

We observe that our data structure for r-NNS automatically gives a data structure for r-NN that improves the standard LSH approach: the standard approach incurs a $\tilde{O}(n^{\rho} + b_S(\mathbf{q}, cr))$ expected query time for worst-case data sets, while our data structure is never worse than $\tilde{O}(n^{\rho} + b_S(\mathbf{q}, cr)/b_S(\mathbf{q}, r))$ in expectation. This is consequence of the initial random permutation that breaks long chains of consecutive (c, r)-near points. We remark that all of our methods have an additional running time term that scales with $b(\mathbf{q}, cr)/b(\mathbf{q}, r)$. This makes running time data-dependent in the following sense: If c is increased (by the user), the ρ-value of the LSH will decrease (decreasing the running time), but we pay for it with a possible increase in $b(\mathbf{q}, cr)/b(\mathbf{q}, r)$.

3.1 Sampling with a repeated query

If we repeat the same query $\mathbf{q}$ several times, the above data structure always returns the same point. If we let OUT_i denote the output at the i-th repetition of query $\mathbf{q}$, we have that $\Pr[OUT_i = \mathbf{p} | OUT_1 = \mathbf{p}_1]$ is 1 if $\mathbf{p} = \mathbf{p}_1$ and 0 otherwise. We would like to extend the above data structure to get

$$\Pr[OUT_i = \mathbf{p} | OUT_{i-1} = \mathbf{p}_{i-1}, \dots OUT_1 = \mathbf{p}_1] = \Pr[OUT_i = \mathbf{p}] = \frac{1}{b_S(\mathbf{q}, r)} \tag{1}$$

for a given query $\mathbf{q}$ and $1 < i \leq \Theta(n)$ (i.e., there is a linear number of queries), and thus to solve the r-NNIS problem in the case when only one query is repeated. The main idea is to select a near neighbor $\mathbf{p}$ of $\mathbf{q}$ as in the data structure of Theorem 1 and, just before returning $\mathbf{p}$, to apply a small random perturbation to ranks for "destroying" any relevant information that can be collected by repeating the query. The perturbation is obtained by applying a random swap, similar to the one in the Fisher-Yates shuffle [21]: we random select a point $\mathbf{p}' \in S$ (in fact, a suitable subset of S) and exchange the ranks of $\mathbf{p}$ and $\mathbf{p}'$. In this way, we get a data structure satisfying Equation (1) with expected query time $O\left((n^\rho + b_S(\mathbf{q}, cr) - b_S(\mathbf{q}, r)) \log^2 n\right)$. For lack of space, we refer to Appendix A for more details.

We remark that the rerandomization technique is only restricted to single element queries: indeed, over time all elements in the ball $B_S(\mathbf{q}, r)$ get higher and higher ranks. This means that for another query $\mathbf{q}'$ such that $B_S(\mathbf{q}, r)$ and $B_S(\mathbf{q}', r)$ have a non-empty intersection, the elements in $B_S(\mathbf{q}', r) \setminus B_S(\mathbf{q}, r)$ become more and more likely to be returned. The next section will provide a slightly more involved data structure that guarantees independence among queries.

3.2 Sampling k points with/without replacement

The data structure for r-NNS can be easily adapted to return k points uniformly sampled *without* replacement from $B_S(\mathbf{q}, r)$: by assuming for well-definedness that $k \leq b_S(\mathbf{q}, r)$, it suffices to return the k points in $S_\mathbf{q}$ with the smallest ranks. On the other hand, a set of k points uniformly sampled *with* replacement from $B_S(\mathbf{q}, r)$ can be obtained by performing k times the query $\mathbf{q}$ in the data structure for repeated queries, as described in the previous subsection.

4 r-near neighbor independent sampling

In this section, we present a data structure that solves the r-NNIS problem with high probability. Let S be the input set of n points and let Λ be the sequence of the n input points after a random permutation; the rank of a point in S is its position in Λ. We first highlight the main idea of the query procedure, then we describe its technical limitations and how to solve them.

Let $k \geq 1$ be a suitable value that depends on the query point $\mathbf{q}$, and assume that Λ is split into k segments Λ_i, with $i \in \{0, \dots, k-1\}$. (We assume for simplicity that n and k are powers of two.) Each segment Λ_i contains the n/k points in Λ with rank in $[i \cdot n/k, (i+1) \cdot n/k)$ We denote with $\lambda_{\mathbf{q},i}$ the number of near neighbors of $\mathbf{q}$ in Λ_i, and with $\lambda \geq \max_i\{\lambda_{\mathbf{q},i}\}$ an upper bound on the number of near neighbors of $\mathbf{q}$ in each segment. By the initial random permutation, we have that each segment contains at most $\lambda = \Theta\left((b_S(\mathbf{q}, r)/k) \log n\right)$ near neighbors with probability at least $1 - 1/n^2$. The query algorithm works in three steps: A) Select uniformly at random an integer h in $\{0, \dots, k-1\}$ (i.e., select a segment Λ_h); B) With probability $\lambda_{\mathbf{q},h}/\lambda$ move to step

C, otherwise repeat step A; C) Return a point uniformly sampled among the near neighbors of q in Λ_h. All random choices are independent.

The above procedure guarantees that the returned near neighbor point is uniformly sampled in $B_S(\mathbf{q}, r)$. Indeed, a point $\mathbf{p} \in B_S(\mathbf{q}, r)$ in segment Λ_h is sampled and returned at step C with probability $\Pr[OUT = \mathbf{p}] = \sum_{j=1}^{+\infty} p_j$, where p_j is the probability of returning $\mathbf{p}$ at the j-th iteration (i.e., after repeating j times step A). We have:

$$p_j = \left(\sum_{i=0}^{k-1} \frac{1 - \lambda_{\mathbf{q},i}/\lambda}{k} \right)^{j-1} \frac{\lambda_{\mathbf{q},h}}{k\lambda} \frac{1}{\lambda_{\mathbf{q},h}} = \left(1 - \frac{b_S(\mathbf{q}, r)}{k\lambda} \right)^{j-1} \frac{1}{k\lambda}.$$

The term with exponent $j - 1$ is the probability that step A is repeated $j - 1$ times; the second term is the probability of selecting segment Λ_h during the j-th iteration and then to move to step C; the third term is the probability of returning point $\mathbf{p}$ in step C. Then:

$$\Pr[OUT = \mathbf{p}] = \sum_{j=1}^{+\infty} p_j = \sum_{j=1}^{+\infty} \left(1 - \frac{b_S(\mathbf{q}, r)}{k\lambda} \right)^{j-1} \frac{1}{k\lambda} = \frac{1}{b_S(\mathbf{q}, r)}.$$

As all random choices are taken independently at query time, the solution guarantees the independence among output points required by Definition 2.

The above approach however cannot be efficiently implemented. The first problem is that each segment might contain a large number of points with distance larger than r: these points are not used by the query algorithm, but still affect the running time as the entire segment must be read to find all near neighbors. We solve this problem by filtering out far points with LSH: at query time, we use LSH to retrieve only the near neighbors (and a small number of (c, r)-near and far points) that are in the selected segment. A second issue is due to segment size: to improve performance, segments should be small and with at least one near neighbor of $\mathbf{q}$; thus, the number k of segments should be set to $b_S(\mathbf{q}, r)$. However, $b_S(\mathbf{q}, r)$ is not known at query time. Hence, we initially set $k = 2\hat{s_{\mathbf{q}}}$, where $\hat{s_{\mathbf{q}}}$ is a 1/2-approximation of the number $s_{\mathbf{q}}$ of points colliding with $\mathbf{q}$; such an estimate can be computed with the count distinct sketch from Section 2. As $\hat{s_{\mathbf{q}}}$ can be much larger than $b_S(\mathbf{q}, r)$, we expect to sample several segments without near neighbors: thus, for every $\Sigma = \Theta\left(\log^2 n\right)$ sampled segments with no near neighbors of $\mathbf{q}$, we repeat the procedure with a smaller value of k (i.e., we set $k = k/2$). We are now ready to fully describe our data structure.

Construction Let $\ell_1, \ldots, \ell_L$ be $L = \Theta\left(p_1^{-1} \log n\right)$ hash functions selected independently and uniformly at random from an (r, cr, p_1, p_2)-sensitive LSH family $\mathcal{H}$, with $p_2 = O(1/n)$. Let $r(\mathbf{p})$ be the rank of point $\mathbf{p} \in S$ after a random permutation of the n data points. For each ℓ_i, we partition the input points S according to the hash values of ℓ_i and denote with $S_{i,j} = \{\mathbf{p} : \mathbf{p} \in S, \ell_i(\mathbf{p}) = j\}$ the set of points with hash value j under ℓ_i. We associate to each non-empty bucket $S_{i,j}$: 1) an index (e.g., a balance binary tree) for efficiently retrieving all points in $S_{i,j}$ with ranks in a given range; 2) a count distinct sketch of $S_{i,j}$ (see Section 2.3) with $\epsilon = 1/2$ and $\delta = 1/n^3$. (For buckets containing less that $\Theta(\log n)$ points of S, we do not store a count distinct sketch since it requires more space than the points; we generate the sketch from $S_{i,j}$ every time it is required.) We observe that any segment Λ_i can be constructed by collecting all points in every $S_{i',j}$ with ranks in $[i \cdot n/k, (i + 1) \cdot n/k)$.

Query Consider a query $\mathbf{q}$ and let $S_{\mathbf{q}} = \bigcup_{i=1}^{L} S_{i,\ell_i(\mathbf{q})}$ and $S_{\mathbf{q},r} = S_{\mathbf{q}} \cap B(\mathbf{q}, r)$. A 1/2-approximation $\hat{s_{\mathbf{q}}}$ of $s_{\mathbf{q}} = |S_{\mathbf{q}}|$ follows by merging the count distinct sketches associated with buckets $S_{i,\ell_i(\mathbf{q})}$ for each $i \in \{1, \ldots, L\}$. We note that $s_{\mathbf{q},r} = |S_{\mathbf{q},r}|$ cannot be estimated with sketches since they cannot distinguish near and far points. The query algorithm is the following (for simplicity, we assume n to be a power of two):

1. Merge all count distinct sketches of buckets $S_{i,\ell_i(\mathbf{q})}$, for each $i \in \{1, \ldots, L\}$, and compute a 1/2-approximation $\hat{s_\mathbf{q}}$ of $s_\mathbf{q}$, such that $s_\mathbf{q}/2 \leq \hat{s_\mathbf{q}} \leq 1.5 s_\mathbf{q}$.

2. Set k to the smallest power of two larger than or equal to $2\hat{s_\mathbf{q}}$; let $\lambda = \Theta(\log n)$, $\sigma_{\text{fail}} = 0$ and $\Sigma = \Theta\left(\log^2 n\right)$.

3. Repeat the following steps until successful or $k < 2$:

   (a) Assume the input sequence Λ to be split into k segments Λ_i of size n/k, where Λ_i contains the points in S with ranks in $[i \cdot n/k, (i+1) \cdot n/k)$.

   (b) Randomly select an index h uniformly at random from $\{0, \ldots, k-1\}$. Using the index in each bucket, retrieve the set $\Lambda'_{\mathbf{q},h} = S_{\mathbf{q},r} \cap \Lambda_h$, that is all points with rank in $[h \cdot n/k, (h+1) \cdot n/k)$ and with distance at most r from $\mathbf{q}$. Set $\lambda_{\mathbf{q},h} = |\Lambda'_{\mathbf{q},h}|$.

   (c) Increment σ_{fail}. If $\sigma_{\text{fail}} = \Sigma$, then set $k = k/2$ and $\sigma_{\text{fail}} = 0$.

   (d) With probability $\lambda_{\mathbf{q},h}/\lambda$, declare success.

4. If the previous loop ended with success, return a near neighbor of $\mathbf{q}$ sampled uniformly at random in $\Lambda'_{\mathbf{q},h}$; otherwise return $\perp$.

Theorem 2. *With probability at least $1-1/n$, the above data structure solves the r-near neighbor independent sampling problem. The data structure requires $\Theta\left(n^{1+\rho}\log n\right)$ words, $O\left(n^{1+\rho}\log^2 n\right)$ construction time, and the expected query time is:*[1]

$$O\left(\left(n^\rho + \frac{b_S(\mathbf{q}, c\cdot r)}{b_S(\mathbf{q}, r)}\right)\log^5 n\right).$$

Proof. Let $\mathbf{q}$ be a query and assume that $b_S(\mathbf{q}, r) \geq 1$. We start by bounding the initial failure probability of the data structure. By a union bound, we have that the following three events hold simultaneously with probability at least $1 - 1/(2n^2)$:

1. All near neighbors in $B_S(\mathbf{q}, r)$ collide with $\mathbf{q}$ under at least one LSH function. By setting the constant in $L = \Theta\left(p_1^{-1}\log n\right)$, the claim holds with probability at least $1 - 1/(6n^2)$.

2. Count distinct sketches provide a 1/2-approximation of $s_\mathbf{q}$. By setting $\delta = 1/(6n^2)$ in the count distinct sketch construction (see Section 2.3), the approximation guarantee holds with probability at least $1 - 1/(6n^2)$.

3. When $k = 2^{\lceil \log s_{\mathbf{q},r}\rceil}$, every segment of size n/k contains no more than $\lambda = \Theta(\log n)$ points from $S_{\mathbf{q},r}$. As points are initially randomly permuted, the claim holds with probability at least $1 - 1/(6n^2)$ by suitably setting the constant in $\lambda = \Theta(\log n)$.

From now on assume that these events are true. We proceed to show that, with probability at least $1 - 1/(2n^2)$, the algorithm returns a point uniformly sampled in $B_S(\mathbf{q}, r)$. In other words, for $\mathbf{p} \in B_S(\mathbf{q}, r)$, we show that $\Pr[OUT = \mathbf{p}] = 1/b_S(\mathbf{q}, r)$.

Let us first discuss the additional failure event. Even when $b_S(\mathbf{q}, r) \geq 1$, there exists a non-zero probability that the algorithm returns $\perp$: the probability of this event is upper bounded by the probability p' that a near neighbor is not returned in the Σ iteration where $k = 2^{\lceil \log s_{\mathbf{q},r}\rceil}$ (the actual probability is even lower, since a near neighbor can be returned in an iteration where $k > 2^{\lceil \log s_{\mathbf{q},r}\rceil}$). By suitably setting constants in $\lambda = \Theta(\log n)$ and $\Sigma = \Theta\left(\log^2 n\right)$, we get:

$$p' = \left(1 - \frac{b_S(\mathbf{q}, r)}{k\lambda}\right)^\Sigma \leq e^{-\Sigma b_S(\mathbf{q},r)/(k\lambda)} \leq e^{\Theta(-\Sigma/\log n)} \leq \frac{1}{2n^2}.$$

[1] For simplicity we assume that $b_S(q, r) > 0$. When $b_S(\mathbf{q}, r) = 0$, it suffices to replace $b_S(\mathbf{q}, r)$ with $b_S(\mathbf{q}, r) + 1$ in the query complexity.

This shows that with probability at least $1 - 1/n^2$, the initial three events hold and the algorithm returns a near neighbor. As each point in $B_S(\mathbf{q}, r)$ has the same probability $1/(k\lambda)$ to be returned during an iteration of step 3, we have that all points in $B_S(\mathbf{q}, r)$ are equally likely to be sampled. By applying a union bound, we have that the claim also holds with probability at least $1 - 1/n$ for any sequence of n queries. Moreover, as soon as the aforementioned events 1,2 and 3 hold, the output probabilities are not affected by the random choices used for generating the initial permutation, LSH construction and count distinct sketches. Our data structure then solves the r-NNIS problem.

We now focus on the space and time complexity of the data structure. For each bucket $S_{i,j}$, we use $O(|S_{i,j}|)$ memory words for storing the index and the count distinct sketch. Therefore, the space complexity of our data structure is dominated by the L LSH tables, that is $\Theta\left(n^{1+\rho}\log n\right)$ memory words. The construction time is $\Theta\left(n^{1+\rho}\log^2 n\right)$, where the additional multiplicative $\log n$ factor is due to sketch construction.

The expected query time can be upper bounded by assuming that the algorithm returns a point in $B_S(\mathbf{q}, r)$ only when k is equal to the smallest power of two larger than $b_S(\mathbf{q}, r)$ (i.e., $k = 2^{\lceil \log s_{\mathbf{q},r} \rceil}$). The cost of each iteration of step 3 is dominated by step 3.b, where the set $\Lambda'_{\mathbf{q},h}$ is constructed: the set is obtained by first extracting all points with rank in $[hn/k, (h+1)n/k)$ from buckets $S_{i,\ell_i(\mathbf{q})}$, for each $i \in \{1, \ldots, L\}$, and then by discarding those points with distance larger than r from $\mathbf{q}$. For each bucket, we expect to find $b_S(\mathbf{q}, r)/k = O(1)$ near neighbors of $\mathbf{q}$, $b_S(\mathbf{q}, c\cdot r)p_1/k = O(b_S(\mathbf{q}, c\cdot r)p_1/b_S(\mathbf{q}, r))$ (c, r)-near neighbors, and not more than $np_2 = O(1)$ far points. Since there are $L = \Theta\left(p_1^{-1}\log n\right)$ buckets and the rank query for reporting points within a given rank costs $O(\log n + o)$ in each bucket (where o is the output size), the cost of each iteration of step 3 is $O\left((n^\rho + b_S(\mathbf{q}, c\cdot r)/b_S(\mathbf{q}, r))\log^2 n\right)$. This bound extends to all $k > 2^{\lceil \log s_{\mathbf{q},r} \rceil}$, since the denominator in the calculations is only larger in this case. When $k \sim b_S(\mathbf{q}, r)$, we expect $O(\log n)$ iterations of step 3 before returning a near neighbor of $\mathbf{q}$ (recall that step 3.d is successful with probability $O(1/\log n)$).

Finally, we bound the cost of all iterations where $k > 2^{\lceil \log s_{\mathbf{q},r} \rceil}$. Since $\hat{s_{\mathbf{q}}} \leq 2n$, we observe that the k value is adapted $O(\log n)$ times in step 3.d. For each fixed value of k, $O\left(\log^2 n\right)$ iterations of step 3 are carried out. These factors yield the claimed bound on the expected running time. □

5 Independent sampling in nearly-linear space

In this section we study another approach to obtain a data structure for the r-NNIS problem. The method uses nearly-linear space by storing each data point exactly once in each of $\Theta(\log n)$ independent data structures. The presented approach is simpler than the solution found in the previous section since it avoids any additional data structure on top of the basic filtering approach described in [13]. It can be seen as a particular instantiation of the more general space-time trade-off data structures that were recently described in [9, 13]. It can also be seen as a variant of the empirical approach discussed in [16] with a theoretical analysis. Compared to [9, 13], it provides much easier parameterization and a simpler way to make it efficient. We provide here a succinct description of the data structure. All proofs can be found in Appendix B.

In this section it will be easier to state bounds on the running time with respect to inner product similarity on unit length vectors in $\mathbb{R}^d$. We define the (α, β)-NN problem analogously to (c, r)-NN, replacing the distance thresholds r and cr with α and β such that $-1 < \beta < \alpha < 1$. This means that the algorithm guarantees that if there exists a point $\mathbf{p}$ with inner product at least α with the query point, the data structure returns a point $\mathbf{p}^*$ with inner product at least β with constant probability. The reader is reminded that for unit length vectors we have the relation $\|\mathbf{p} - \mathbf{q}\|_2^2 = 2 - 2\langle \mathbf{p}, \mathbf{q}\rangle$. We will use the notation $B_S(\mathbf{q}, \alpha) = \{\mathbf{p} \in S \mid \langle \mathbf{p}, \mathbf{q}\rangle \geq \alpha\}$ and $b_S(\mathbf{q}, \alpha) = |B_S(\mathbf{q}, \alpha)|$. We define the α-NNIS problem analogously to r-NNIS with respect to inner product similarity.

5.1 Description of the data structure

Construction Given $m \geq 1$ and $\alpha < 1$, let $t = \lceil 1/(1-\alpha^2) \rceil$ and assume that $m^{1/t}$ is an integer. First, choose $tm^{1/t}$ random vectors $\mathbf{a}_{i,j}$, for $i \in [t], j \in [m^{1/t}]$, where each $\mathbf{a}_{i,j} = (a_1, \ldots, a_d) \sim \mathcal{N}(0,1)^d$ is a vector of d independent and identically distributed standard normal Gaussians. Next, consider a point $\mathbf{p} \in S$. For $i \in [t]$, let j_i denote the index maximizing $\langle \mathbf{p}, \mathbf{a}_{i,j_i} \rangle$. Then we map the index of $\mathbf{p}$ in S to the bucket $(j_1, \ldots, j_t) \in [m^{1/t}]^t$, and use a hash table to keep track of all non-empty buckets. Since a reference to each data point in S is stored exactly once, the space usage can be bounded by $O(n + tm^{1/t})$.

Query Given the query point $\mathbf{q}$, evaluate the dot products with all $tm^{1/t}$ vectors $\mathbf{a}_{i,j}$. For $\varepsilon \in (0,1)$, let $f(\alpha, \varepsilon) = \sqrt{2(1-\alpha^2)\ln(1/\varepsilon)}$. For $i \in [t]$, let $\Delta_{\mathbf{q},i}$ be the value of the largest inner product of $\mathbf{q}$ with the vectors $\mathbf{a}_{i,j}$ for $j \in [m^{1/t}]$. Furthermore, let $\mathcal{I}_i = \{j \mid \langle \mathbf{a}_{i,j}, \mathbf{q} \rangle \geq \alpha\Delta_{\mathbf{q},i} - f(\alpha, \varepsilon)\}$. The query algorithm checks the points in all buckets $(i_1, \ldots, i_t) \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_t$, one bucket after the other. If a bucket contains a close point, return it, otherwise return $\perp$.

Theorem 3. *Let $S \subseteq \mathcal{X}$ with $|S| = n$, $-1 < \beta < \alpha < 1$, and let $\varepsilon > 0$ be a constant. Let $\rho = \frac{(1-\alpha^2)(1-\beta^2)}{(1-\alpha\beta)^2}$. There exists $m = m(\alpha, \beta, n)$ such that the data structure described above solves the (α, β)-NN problem with probability at least $1 - \varepsilon$ in linear space and expected time $n^{\rho + o(1)}$.*

We remark that this result is equivalent to running time statements found in [13] for the linear space regime, but the method is considerably simpler. The analysis connects storing data points in the list associated with the largest inner product with well-known bounds on the order statistics of a collection of standard normal variables as discussed in [14].

5.2 Solving α-NNIS

Construction Set $L = \Theta(\log n)$ and build L independent data structures $\mathrm{DS}_1, \ldots, \mathrm{DS}_L$ as described above. For each $p \in S$, store a reference from p to the L buckets it is stored in.

Query For query $\mathbf{q}$, evaluate all $tm^{1/t}$ filters in each individual DS_ℓ. Let $\mathcal{I}$ be the set of buckets $(i_{\ell,1}, \ldots, i_{\ell,t})$ above the query threshold, for each $\ell \in [L]$, and set $T = |\mathcal{I}|$. Enumerate the buckets from 1 to T in an arbitrary order over all repetitions. Let k_i be the number of data points in bucket $i \in [T]$, and set $K = \sum k_i$. First, check for the existence of a near neighbor by running the standard query algorithm described above on every individual data structure. If no near neighbor exists, output $\perp$ and return. Otherwise, perform the following steps until success is declared:

A. Choose a random integer i in $\{1, \ldots, T\}$, where the probability of choosing i equals k_i/K.

B. Choose a random point $\mathbf{p}$ from bucket i uniformly at random. Using the L references of $\mathbf{p}$ to its buckets, compute $c_\mathbf{p}$, $0 \leq c_\mathbf{p} \leq L$, the number of times $\mathbf{p}$ occurs in the T buckets, in time $O(L)$.

C. If $\mathbf{p}$ is a near neighbor, report $\mathbf{p}$ and declare success with probability $1/c_\mathbf{p}$.

D. If $\mathbf{p}$ is a far point, remove $\mathbf{p}$ from the bucket and decrement k_i and K.

After a point $\mathbf{p}$ has been reported, move all far points removed during the process into their bucket again. We assume that removing and inserting a point takes constant time in expectation.

Theorem 4. *Let $S \subseteq \mathcal{X}$ with $|S| = n$ and $-1 < \beta < \alpha < 1$. The data structure described above solves the α-NNIS problem in nearly-linear space and expected time $n^{\rho+o(1)} + O((b_S(\mathbf{q}, \beta)/b_S(\mathbf{q}, \alpha)) \log^2 n)$.*

Proof. Set $L = \Theta(\log n)$ such that with probability at least $1 - 1/n^2$, all points in $B_S(\mathbf{q}, \alpha)$ are found in the T buckets. Let $\mathbf{p}$ be an arbitrary point in $B_S(\mathbf{q}, \alpha)$. We show that $\mathbf{p}$ is returned by the query algorithm with probability $1/b_S(\mathbf{q}, \alpha)$. This statement follows by the more general observation that the point $\mathbf{p}$ picked in step B is a weighted uniform point among all points present in the buckets after i iterations of the algorithm. That means that if there are K' points in the buckets, the probability of choosing $\mathbf{p}$ is $c_{\mathbf{p}}/K'$. If $\mathbf{p}$ resides in bucket i, the probability of reporting $\mathbf{p}$ is $k_i/K' \cdot 1/k_i \cdot 1/c_{\mathbf{p}} = 1/(K'c_{\mathbf{p}})$. Since $\mathbf{p}$ is stored in $c_{\mathbf{p}}$ different buckets, the probability of reporting $\mathbf{p}$ is $1/K'$. Since this property holds in each round, each near neighbor has equal chance of being reported by the algorithm.

We proceed to proving the running time statement. Observe that evaluating all filters, checking for the existence of a near neighbor, removing far points, and putting far points back into the buckets contributes $n^{\rho+o(1)}$ to the expected running time (see Appendix B for details). We did not account for repeatedly carrying out steps A–C yet for rounds in which we choose a non-far point. We next find a lower bound on the probability that the algorithm declares success in a single such round. First, observe that there are $O(b_S(\mathbf{q}, \beta) \log n)$ non-far points in the T buckets (with repetitions). Fix a point $\mathbf{p} \in B_S(\mathbf{q}, \alpha)$. With probability $\Omega(c_{\mathbf{p}}/(b_S(\mathbf{q}, \beta) \log n))$, $\mathbf{p}$ is chosen in step B. Thus, with probability $\Omega(1/(b_S(\mathbf{q}, \beta) \log n))$, success is declared for point $\mathbf{p}$. Summing up probabilities over all points in $B_S(\mathbf{q}, \alpha)$, we find that the probability of declaring success in a single round is $\Omega(b_S(\mathbf{q}, \alpha)/(b_S(\mathbf{q}, \beta) \log n))$. This means that we expect $O(b_S(\mathbf{q}, \beta) \log n/b_S(\mathbf{q}, \alpha))$ rounds until the algorithm declares success. Each round takes time $O(\log n)$ for computing $c_{\mathbf{p}}$ (which can be done by marking all buckets that are enumerated), so we expect to spend time $O((b_S(\mathbf{q}, \beta)/b_S(\mathbf{q}, \alpha)) \log^2 n)$ for these iterations, which concludes the proof. □

6 Conclusion

In this paper, we have investigated a possible definition of fairness in similarity search by connecting the notion of "equal opportunity" to independent range sampling. An interesting open question is to investigate the applicability of our data structures for problems like discrimination discovery [23], diversity in recommender systems [2], privacy preserving similarity search [27], and estimation of kernel density [12].

The techniques presented here require a manual trade-off between the performance of the LSH part and the additional running time contribution from finding the near points among the non-far points. From a user point of view, we would much rather prefer a parameterless version of our data structure that finds the best trade-off with small overhead, as discussed in [5] in another setting.

References

[1] Serge Abiteboul, Marcelo Arenas, Pablo Barceló, Meghyn Bienvenu, Diego Calvanese, Claire David, Richard Hull, Eyke Hüllermeier, Benny Kimelfeld, Leonid Libkin, Wim Martens, Tova Milo, Filip Murlak, Frank Neven, Magdalena Ortiz, Thomas Schwentick, Julia Stoyanovich, Jianwen Su, Dan Suciu, Victor Vianu, and Ke Yi. Research directions for principles of data management (abridged). *SIGMOD Rec.*, 45(4):5–17, 2017.

[2] Gediminas Adomavicius and YoungOk Kwon. Optimization-based approaches for maximizing aggregate recommendation diversity. *INFORMS Journal on Computing*, 26(2):351–369, 2014.

[3] Peyman Afshani and Jeff M. Phillips. Independent range sampling, revisited again. Arxiv:1903.08014, 2019.

[4] Peyman Afshani and Zhewei Wei. Independent range sampling, revisited. In *Proc. 25th Annual European Symposium on Algorithms (ESA)*, pages 3:1–3:14, 2017.

[5] Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh. Parameter-free locality sensitive hashing for spherical range reporting. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 239–256, 2017.

[6] Ilya P. Razenshteyn Alexandr Andoni, Piotr Indyk. Approximate nearest neighbor search in high dimensions. In *Proc. International Congress of Mathematicians (ICM)*, page 3271–3302, 2018.

[7] Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *Proc. IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–150, 2015.

[8] Alexandr Andoni, Thijs Laarhoven, Ilya P. Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 47–66, 2017.

[9] Alexandr Andoni, Thijs Laarhoven, Ilya P. Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 47–66, 2017.

[10] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. 6th International Workshop Randomization and Approximation Techniques (RANDOM)*, pages 1–10, 2002.

[11] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of 34th ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002.

[12] Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1032–1043, 2017.

[13] Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 31–46, 2017.

[14] Herbert Aron David and Haikady Navada Nagaraja. Order statistics. *Encyclopedia of Statistical Sciences*, 2004.

[15] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, pages 214–226, 2012.

[16] Kave Eshghi and Shyamsundar Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proc. 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 221–229. ACM, 2008.

[17] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[18] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Proc. Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3315–3323, 2016.

[19] Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 246–255, 2014.

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, 1998.

[21] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley, 1997.

[22] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[23] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. k-nn as an implementation of situation testing for discrimination discovery and prevention. In *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 502–510, 2011.

[24] Executive Office of the President. Big data: A report on algorithmic systems, opportunity, and civil rights., 2016.

[25] Frank Olken and Doron Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, 1995.

[26] Frank Olken and Doron Rotem. Sampling from spatial databases. *Statistics and Computing*, pages 43–57, 1995.

[27] M. Sadegh Riazi, Beidi Chen, Anshumali Shrivastava, Dan S. Wallach, and Farinaz Koushanfar. Sub-Linear Privacy-Preserving Near-Neighbor Search with Untrusted Server on Large-Scale Datasets. ArXiv:1612.01835, 2016.

[28] Andrew D. Selbst, danah boyd, Sorelle Friedler, Suresh Venkatasubramanian, and Janet Vertesi. Fairness and abstraction in sociotechnical systems. In *ACM Conference on Fairness, Accountability, and Transparency (FAT*)*, 2019.

[29] Stanislaw J. Szarek and Elisabeth Werner. A nonsymmetric correlation inequality for gaussian measure. *Journal of Multivariate Analysis*, 68(2):193 – 211, 1999.

[30] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2):357–365, 2005.

A Sampling with a repeated query

We first explain why two simple adaptations of the data structure for r-NNS in section 3 don't work with a repeated query. The first adaptation consists in returning the point with the k-th smallest rank in $S_{\mathbf{q}} = \bigcup_{i=1}^{L} S_{i,\ell_i(\mathbf{q})}$ where k is an integer in $[1, S_{\mathbf{q}}]$ randomly and independently selected at query time. Although this approach gives Equations 1, the query time is $O\left(n^\rho b_S(\mathbf{q}, r) + b_S(\mathbf{q}, cr)\right)$ with constant probability, as in the trivial solution described at the beginning. Indeed, since there are point repetitions among buckets, we have to scan almost all buckets $S_{i,\ell_i(\mathbf{q})}$ for every i, getting the same asymptotic query time of the trivial LSH solution. Another approach is to random select an integer value r in $[r_m, r_M]$, where r_m and r_M are the smallest and largest ranks in $S_{\mathbf{q}}$, and then to return the point in $S_{\mathbf{q}}$ with the closest rank to r. However, this approach doesn't guarantee independence among the outputs: Indeed, before the first query the initial permutation is unknown and thus all points are equally likely to be returned; subsequent queries however reveal some details on the initial distribution, conditioning subsequent queries and exposing which points are more likely to be reported. We now describe our solution.

Construction The construction is as in the one in section 3, with the exception that points in each bucket $S_{i,j}$ are stored in a priority queue, where ranks give the priority. We assume that each queue supports the extraction of the point with *minimum* rank and the insertion, deletion, and update of a point given the (unique) rank.

Query The algorithm starts as in the previous data structure by searching for the point $\mathbf{x}$ in $B_S(\mathbf{q}, r) \cap \bigcup_{i=1}^{L} S_{i,\ell_i(\mathbf{q})}$ with lowest rank. However, just before returning the sampled point, we update the rank of $\mathbf{x}$ as follows. Let $r_{\mathbf{x}}$ be the rank of $\mathbf{x}$; the algorithm selects uniformly at random a value r in $\{r_{\mathbf{x}}, \ldots n\}$ and let $\mathbf{y}$ be the point in S with rank r; then the algorithm swaps the rank of $\mathbf{x}$ and $\mathbf{y}$ and updates all data structures (i.e., the priority queues of buckets $S_{i,\ell_i(\mathbf{x})}$ and $S_{i,\ell_i(\mathbf{y})}$ with $1 \le i \le L$).

Theorem 5. *With high probability $1 - 1/n$, the above data structure solves the r-NNIS problem with one single repeated query: for each query point $\mathbf{q}$ and for each point $\mathbf{p} \in S$, we have:*

1. *$\mathbf{p}$ is returned as near neighbor of $\mathbf{q}$ with probability $1/b_S(\mathbf{q}, r)$.*
2. *For each $1 < i \le n$, $\Pr\left[OUT_i = \mathbf{p} | OUT_{i-1} = \mathbf{p}_{i-1}, \ldots OUT_1 = \mathbf{p}_1\right] = 1/b_S(\mathbf{q}, r)$.*

Furthermore, the data structure requires $\Theta\left(n^{1+\rho} \log n\right)$ space and the expected query time is

$$O\left(\left(n^\rho + b_S(\mathbf{q}, cr) - b_S(\mathbf{q}, r)\right) \log^2 n\right).$$

Proof. We claim that after every query $\mathbf{q}$, it is not possible to distinguish how the $B_S(\mathbf{q}, r)$ near neighbors of $\mathbf{q}$ are distributed among the ranks $\{r_{\mathbf{x}}, \ldots n\}$. Before the rank shuffle, the position of $\mathbf{x}$ is known while the positions of points in $B_S(\mathbf{q}, r)/\{\mathbf{x}\}$ in $\{r_{\mathbf{x}}+1, \ldots n\}$ still remain unknown. After swapping $\mathbf{x}$ with a random point with rank $\{r_{\mathbf{x}}, n\}$, $\mathbf{x}$ can be in each rank with probability $1/(n - r_{\mathbf{x}} + 1)$ and all distributions of points in $B_S(\mathbf{q}, r)$ in ranks $\{r_{\mathbf{x}}, n\}$ are thus equally likely. As a consequence, properties 1 and 2 in the statement follow.

With high probability, the query algorithm finds $O\left(\left(b_S(\mathbf{q}, cr) - b_S(\mathbf{q}, r) + L\right) \log n\right)$ points with distance larger than r before finding a r-near neighbor. The final rank shuffle requires $O\left(L \log n\right)$ time as we need to update the at most $2L$ buckets and priority queues with points $\mathbf{x}$ and $\mathbf{y}$. The theorem follows.[2] □

B A linear space near-neighbor data structure

We will split up the analysis of the data structure from Section 5 into two parts. First, we describe and analyse a query algorithm that ignores the cost of storing and evaluating the m random vectors. Next, we will describe and analyze the changes necessary to obtain an efficient query method as the one described in Section 5.

B.1 Description of the Data Structure

Construction To set up the data structure for a point set $S \subseteq \mathbb{R}^d$ of n data points and two parameters $\beta < \alpha$, choose $m \ge 1$ random vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ where each $\mathbf{a} = (a_1, \ldots, a_d) \sim \mathcal{N}(0,1)^d$ is a vector of d independent and identically distributed standard normal Gaussians. For each $i \in \{1, \ldots, m\}$, let L_i contain all data points $\mathbf{x} \in S$ such that $\langle \mathbf{a}_i, \mathbf{x} \rangle$ is largest among all vectors $\mathbf{a}$.

[2] We observe that if *all* points touched at query time are updated with the rank-swap approach, then we get $O\left(\left(1 + b_S(\mathbf{q}, cr)/b_S(\mathbf{q}, r)\right) n^\rho\right)$ expected query time. However, this bound is better than the one stated in Theorem 5 only when $b_S(\mathbf{q}, cr) = O\left(b_S(\mathbf{q}, r)\right)$.

Query For a query point $\mathbf{q} \in \mathbb{R}^d$ and for a choice of $\varepsilon \in (0,1)$ controlling the success probability of the query, define $f(\alpha, \varepsilon) = \sqrt{2(1-\alpha^2)\ln(1/\varepsilon)}$. Let $\Delta_{\mathbf{q}}$ be the largest inner product of $\mathbf{q}$ over all $\mathbf{a}$. Let $L'_1, \ldots, L'_K$ denote the lists associated with random vectors $\mathbf{a}$ satisfying $\langle \mathbf{a}, \mathbf{q} \rangle \geq \alpha \Delta_q - f(\alpha, \varepsilon)$. Check all points in $L'_1, \ldots, L'_K$ and report the first point $\mathbf{x}$ such that $\langle \mathbf{q}, \mathbf{x} \rangle \geq \beta$. If no such point exists, report $\perp$.

The proof of the theorem below will ignore the cost of evaluating $\mathbf{a}_1, \ldots, \mathbf{a}_m$. An efficient algorithm for evaluating these vectors is provided in Section B.4.

Theorem 6. *Let $-1 < \beta < \alpha < 1$, $\varepsilon \in (0,1)$, and $n \geq 1$. Let $\rho = \frac{(1-\alpha^2)(1-\beta^2)}{(1-\alpha\beta)^2}$. There exists $m = m(n, \alpha, \beta)$ such that the data structure described above solves the (α, β)-NN problem with probability at least $1 - \varepsilon$ using space $O(m + n)$ and expected query time $n^{\rho + o(1)}$.*

We split the proof up into multiple steps. First, we show that for every choice of m, inspecting the lists associated with those random vectors $\mathbf{a}$ such that their inner product with the query point $\mathbf{q}$ is at least the given query threshold guarantees to find a close point with probability at least $1 - \varepsilon$. The next step is to show that the number of far points in these lists is $n^{\rho + o(1)}$ in expectation.

B.2 Analysis of Close Points

Lemma 1. *Given m and α, let $\mathbf{q}$ and $\mathbf{x}$ such that $\langle \mathbf{q}, \mathbf{x} \rangle = \alpha$. Then we find $\mathbf{x}$ with probability at least $1 - \varepsilon$ in the lists associated with vectors that have inner product at least $\alpha \Delta_{\mathbf{q}} - f(\alpha, \varepsilon)$ with $\mathbf{q}$.*

Proof. By spherical symmetry we may assume that $\mathbf{x} = (1, 0, \ldots, 0)$ and $\mathbf{q} = (\alpha, \sqrt{1-\alpha^2}, 0, \ldots, 0)$ [13]. The probability of finding $\mathbf{x}$ when querying the data structure for $\mathbf{q}$ can be bounded as follows from below. Let $\Delta_{\mathbf{x}}$ be the largest inner product of $\mathbf{x}$ with vectors $\mathbf{a}$ and let $\Delta_{\mathbf{q}}$ be the largest inner product of $\mathbf{q}$ with these vectors. Given these thresholds, finding $\mathbf{x}$ is then equivalent to the statement that for the vector $\mathbf{a}$ with $\langle \mathbf{a}, \mathbf{x} \rangle = \Delta_{\mathbf{x}}$ we have $\langle \mathbf{a}, \mathbf{q} \rangle \geq \alpha \Delta_{\mathbf{q}} - f(\alpha, \varepsilon)$. We note that $\Pr\left[\max\{\langle \mathbf{a}, \mathbf{q} \rangle\} = \Delta\right] = 1 - \Pr\left[\forall i : \langle \mathbf{a}_i, \mathbf{q} \rangle < \Delta\right]$.

Thus, we may lower bound the probability of finding $\mathbf{x}$ for arbitrary choices $\Delta_{\mathbf{x}}$ and $\Delta_{\mathbf{q}}$ as follows:

$$\begin{aligned} \Pr\left[\text{find } \mathbf{x}\right] \geq \Pr&\left[\langle \mathbf{a}, \mathbf{q} \rangle \geq \alpha \Delta_{\mathbf{q}} - f(\alpha, \varepsilon) \mid \langle \mathbf{a}, \mathbf{x} \rangle = \Delta_{\mathbf{x}} \text{ and } \langle \mathbf{a}', \mathbf{q} \rangle = \Delta_{\mathbf{q}}\right] \\ &- \Pr\left[\forall i : \langle \mathbf{a}_i, \mathbf{x} \rangle < \Delta_{\mathbf{x}}\right] - \Pr\left[\forall i : \langle \mathbf{a}_i, \mathbf{q} \rangle < \Delta_{\mathbf{q}}\right]. \end{aligned} \tag{2}$$

Here, we used that $\Pr\left[A \cap B \cap C\right] = 1 - \Pr\left[\overline{A} \cup \overline{B} \cup \overline{C}\right] \geq \Pr\left[A\right] - \Pr\left[\overline{B}\right] - \Pr\left[\overline{C}\right]$. We will now obtain bounds for the three terms on the right-hand side of (2) separately, but we first recall the following lemma from [29]:

Lemma 2 ([29])**.** *Let Z be a standard normal random variable. Then, for every $t \geq 0$, we have that*

$$\frac{1}{\sqrt{2\pi}} \frac{1}{t+1} e^{-t^2/2} \leq \Pr(Z \geq t) \leq \frac{1}{\sqrt{\pi}} \frac{1}{t+1} e^{-t^2/2}.$$

Bounding the first term Since $\mathbf{x} = (1, 0, \ldots, 0)$ and $\mathbf{q} = (\alpha, \sqrt{1-\alpha^2}, 0, \ldots, 0)$, the condition $\langle \mathbf{a}, \mathbf{x} \rangle = \Delta_{\mathbf{x}}$ means that the first component of $\mathbf{a}$ is $\Delta_{\mathbf{x}}$. Thus, we have to bound the probability that a standard normal random variable Z satisfies the inequality $\alpha \Delta_{\mathbf{x}} + \sqrt{1-\alpha^2} Z \geq \alpha \Delta_{\mathbf{q}} - f(\alpha, \varepsilon)$. Reordering terms, we get

$$Z \geq \frac{\alpha \Delta_{\mathbf{q}} - f(\alpha, \varepsilon) - \alpha \Delta_{\mathbf{x}}}{\sqrt{1-\alpha^2}}.$$

Choose $\Delta_{\mathbf{q}} = \Delta_{\mathbf{x}}$. In this case, we bound the probability that Z is larger than a negative value. By symmetry of the standard normal distribution and using Lemma 2, we may compute

$$\Pr\left[Z \geq -\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right] = 1 - \Pr\left[Z < -\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right] = 1 - \Pr\left[Z \geq \frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right]$$
$$\geq 1 - \frac{\operatorname{Exp}\left(-\frac{(f(\alpha,\varepsilon))^2}{2(1-\alpha^2)}\right)}{\sqrt{2\pi}\left(\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}+1\right)} \geq 1-\varepsilon. \tag{3}$$

Bounding the second term and third term We first observe that

$$\Pr\left[\forall i : \langle \mathbf{a}_i, \mathbf{x}\rangle < \Delta_{\mathbf{x}}\right] = \Pr\left[\langle \mathbf{a}_1, \mathbf{x}\rangle < \Delta_{\mathbf{x}}\right]^m = \left(1 - \Pr\left[\langle \mathbf{a}_1, \mathbf{x}\rangle \geq \Delta_{\mathbf{x}}\right]\right)^m$$
$$\leq \left(1 - \frac{\operatorname{Exp}[-\Delta_{\mathbf{x}}^2/2]}{\sqrt{2\pi}(\Delta_x + 1)}\right)^m.$$

Setting $\Delta_{\mathbf{x}} = \sqrt{2\log m - \log(4\kappa\pi\log(m))}$ upper bounds this term by $\operatorname{Exp}[-\sqrt{\kappa}]$. Thus, by setting $\kappa \geq \log^2(1/\delta)$ the second term is upper bounded by $\delta \in (0,1)$. The same thought can be applied to the third summand of (2), which is only smaller because of the negative offset $f(\alpha,\varepsilon)$.

Putting everything together Putting the bounds obtained for all three summands together shows that we can find $\mathbf{x}$ with probability at least $1-\varepsilon'$ by choosing ε and δ such that $\varepsilon' \geq \varepsilon + 2\delta$. □

B.3 Analysis of Far Points

Lemma 3. *Let $-1 < \beta < \alpha < 1$. There exists $m = m(n, \alpha, \beta)$ such that the expected number of points $\mathbf{x}$ with $\langle \mathbf{x}, \mathbf{q}\rangle \leq \beta$ in $L'_1, \ldots, L'_K$ where $K = |\{i \mid \langle \mathbf{a}_i, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon)\}|$ is $n^{\rho+o(1)}$.*

Proof. We will first focus on a single far-away point $\mathbf{x}$ with inner product at most β. Again, let $\Delta_{\mathbf{q}}$ be the largest inner product of $\mathbf{q}$. Let $\mathbf{x}$ be stored in L_i. Then we find $\mathbf{x}$ if and only if $\langle \mathbf{a}_i, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon)$. By spherical symmetry, we may assume that $\mathbf{x} = (1, 0, \ldots, 0)$ and $\mathbf{q} = (\beta, \sqrt{1-\beta^2}, 0, \ldots, 0)$.

We first derive values $t_{\mathbf{q}}$ and $t_{\mathbf{x}}$ such that with high probability $\Delta_{\mathbf{q}} \geq t_{\mathbf{q}}$ and with high probability $\Delta_{\mathbf{x}} \leq t_{\mathbf{x}}$. From the proof of Lemma 1, we know that

$$\Pr\left[\max\{\langle \mathbf{a}, \mathbf{q}\rangle\} \geq t\right] \leq 1 - \left(1 - \frac{\operatorname{Exp}\left(-t^2/2\right)}{\sqrt{2\pi}(t+1)}\right)^m.$$

Setting $t_{\mathbf{q}} = \sqrt{2\log(m/\log(n)) - \log(4\pi\log(m/\log n))}$ shows that with high probability we have $\Delta_{\mathbf{q}} \geq t_{\mathbf{q}}$. Similarily, the choice $t_{\mathbf{x}} = \sqrt{2\log(m\log(n)) - \log(4\pi\log(m\log n))}$ is with high probability at least $\Delta_{\mathbf{x}}$. In the following, we condition on the event that $\Delta_{\mathbf{q}} \geq t_{\mathbf{q}}$ and $\Delta_{\mathbf{x}} \leq t_{\mathbf{x}}$.

We may bound the probability of finding $\mathbf{x}$ as follows:

$$\Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = \Delta_{\mathbf{x}}\right] \leq \Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = t_{\mathbf{x}}\right]$$
$$\leq \Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = t_{\mathbf{x}}\right].$$

Given that $\langle \mathbf{a}, \mathbf{x}\rangle$ is $t_{\mathbf{x}}$, the condition $\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha t_{\mathbf{q}} - f(\alpha,\varepsilon)$ is equivalent to the statement that for a standard normal variable Z we have $Z \geq \frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})}{\sqrt{1-\beta^2}}$. Using Lemma 2, we have

$$\begin{aligned}
\Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = t_{\mathbf{x}}\right] &\leq \frac{\operatorname{Exp}\left(-\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})^2}{2(1-\beta^2)}\right)}{\sqrt{\pi}\left(\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})}{\sqrt{1-\beta^2}} + 1\right)} \\
&\leq \operatorname{Exp}\left(-\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})^2}{2(1-\beta^2)}\right) \\
&\overset{(1)}{=} \operatorname{Exp}\left(-\frac{(\alpha-\beta)^2 t_{\mathbf{x}}^2}{2(1-\beta^2)}\left(1 + O(1/\log\log n)\right)\right) \\
&= \left(\frac{1}{m}\right)^{\frac{(\alpha-\beta)^2}{1-\beta^2} + o(1)}, \qquad (4)
\end{aligned}$$

where (1) follows from the observation that $t_{\mathbf{q}}/t_{\mathbf{x}} = 1 + O(1/\log\log n)$ and $f(\alpha,\varepsilon)/t_{\mathbf{x}} = O(1/\log\log n)$ if $m = \Omega(\log n)$.

Next, we want to balance this probability with the expected cost for checking all lists where the inner product with the associated vector $\mathbf{a}$ is at least $\alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon)$. By linearity of expectation, we expect not more than

$$m \cdot \operatorname{Exp}\left(-(\alpha t_{\mathbf{q}})^2\left(1/2 - f(\alpha,\varepsilon)/(\alpha t_{\mathbf{q}}) + f(\alpha,\varepsilon)^2/(2(\alpha t_{\mathbf{q}})^2)\right)\right)$$

lists to be checked, which is $m^{1-\alpha^2+o(1)}$ using the value of $t_{\mathbf{q}}$ set above. This motivates to set (4) equal to $m^{1-\alpha^2}/n$, taking into account that there are at most n far-away points. Solving for m, we get

$$m = n^{\frac{1-\beta^2}{(1-\alpha\beta)^2} + o(1)}$$

and this yields $m^{1-\alpha^2+o(1)} = n^{\rho+o(1)}$. □

B.4 Efficient Evaluation

The previous subsections assumed that we can evaluate (and store) m filters for free. However, this requires space and time $n^{(1-\beta^2)/(1-\alpha\beta)^2}$, which is much higher than the work we expect from checking the points in all filters above the threshold. We solve this problem by using the tensoring approach, which can be seen as a simplified version of the general approach proposed in [13].

Construction Let $t = \lceil 1/(1-\alpha^2)\rceil$ and assume that $m^{1/t}$ is an integer. Consider t independent data structures $\mathrm{DS}_1, \ldots, \mathrm{DS}_t$, each using $m^{1/t}$ random vectors $\mathbf{a}_{i,j}$, for $i \in \{1,\ldots,t\}, j \in [m^{1/t}]$. Each DS_i is instantiated as described above. During preprocessing, consider each $\mathbf{x} \in S$. If $\mathbf{a}_{1,i_1}, \ldots, \mathbf{a}_{t,i_t}$ are the random vectors that achieve the largest inner product with $\mathbf{x}$ in DS_1, …, DS_t, map the index of $\mathbf{x}$ in S to the bucket $(i_1,\ldots,i_t) \in [m^{1/t}]^t$. Use a hash table to keep track of all non-empty buckets. Since each data point in S is stored exactly once, the space usage can be bounded by $O(n + tm^{1/t})$.

Query Given the query point $\mathbf{q}$, evaluate all $tm^{1/t}$ filters. For $i \in \{1,\ldots,t\}$, let $\mathcal{I}_i = \{j \mid \langle \mathbf{a}_{i,j}, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q},i} - f(\alpha,\varepsilon)\}$ be the set of all indices of filters that are above the individual query threshold in DS_i. Check all buckets $(i_1,\ldots,i_t) \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_t$. If there is a bucket containing a close point, return it, otherwise return $\perp$.

Theorem 7. *Let $S \subseteq X$ with $|S| = n$ and $-1 < \beta < \alpha < 1$. The tensoring data structure solves the (α,β)-NN problem in linear space and expected time $n^{\rho+o(1)}$.*

Before proving the theorem, we remark that efficient evaluation comes at the price of lowering the success probability from a constant p to $p^{1/(1-\alpha^2)}$. Thus, for $\delta \in (0,1)$ repeating the construction $\ln(1/\delta)p^{1-\alpha^2}$ times yields a success probability of at least $1-\delta$.

Proof. Observe that with the choice of m as in the proof of Lemma 3, we can bound $m^{1/t} = n^{(1-\alpha^2)(1-\beta^2)/(1-\alpha\beta)^2+o(1)} = n^{\rho+o(1)}$. This means that preprocessing takes time $n^{1+\rho+o(1)}$. Moreover, the additional space needed for storing the $tm^{1/t}$ random vectors is $n^{\rho+o(1)}$ as well. For a given query point $\mathbf{q}$, we expect that each $\mathcal{I}_i$ is of size $m^{(1-\alpha^2)/t+o(1)}$. Thus, we expect to check not more than $m^{1-\alpha^2+o(1)} = n^{\rho+o(1)}$ buckets in the hash table, which shows the stated claim about the expected running time.

Let $\mathbf{x}$ be a point with $\langle \mathbf{q}, \mathbf{x} \rangle \geq \alpha$. The probability of finding $\mathbf{x}$ is the probability that the vector associated with $\mathbf{x}$ has inner product at least $\alpha\Delta_{\mathbf{q},i} - f(\alpha,\varepsilon)$ in DS_i, for all $i \in \{1,\ldots,t\}$. This probability is p^t, where p is the probability of finding $\mathbf{x}$ in a single data structure DS_i. By Theorem 6 and since α is a constant, this probability is constant and can be bounded from below by $1-\delta$ via a proper choice of ε as discussed in the proof of Lemma 1.

Let $\mathbf{y}$ be a point with $\langle \mathbf{q}, \mathbf{y} \rangle < \beta$. Using the same approach in the proof of Lemma 3, we observe that the probability of finding $\mathbf{y}$ in an individual DS_i is $(1/m)^{1/t\cdot(\alpha-\beta)^2/(1-\beta^2)+o(1)}$. Thus the probability of finding $\mathbf{y}$ in a bucket inspected for $\mathbf{q}$ is at most $(1/m)^{(\alpha-\beta)^2/(1-\beta^2)+o(1)}$. Setting parameters as before shows that we expect at most $n^{\rho+o(1)}$ far points in buckets inspected for query $\mathbf{q}$, which completes the proof. □