Sampling a Near Neighbor in High Dimensions

— Who is the Fairest of Them All?

MARTIN AUMÜLLER, IT University of Copenhagen, Denmark
SARIEL HAR-PELED, University of Illinois at Urbana-Champaign, USA
SEPIDEH MAHABADI, Toyota Technological Institute at Chicago, USA
RASMUS PAGH, BARC and University of Copenhagen, Denmark
FRANCESCO SILVESTRI, University of Padova, Italy

Similarity search is a fundamental algorithmic primitive, widely used in many computer science disciplines. Given a set of points S and a radius parameter $r > 0$, the r-near neighbor (r-NN) problem asks for a data structure that, given any 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 individual fairness and providing equal opportunities: all points that are within distance r from the query should have the same probability to be returned. In the low-dimensional case, this problem was first studied by Hu, Qiao, and Tao (PODS 2014). Locality sensitive hashing (LSH), the theoretically strongest approach to similarity search in high dimensions, does not provide such a fairness guarantee.

In this work, we show that LSH based algorithms can be made fair, without a significant loss in efficiency. We propose several efficient data structures for the exact and approximate variants of the fair NN problem. Our approach works more generally for sampling uniformly from a sub-collection of sets of a given collection and can be used in a few other applications. We also develop a data structure for fair similarity search under inner product that requires nearly-linear space and exploits locality sensitive filters. The paper concludes with an experimental evaluation that highlights the unfairness of state-of-the-art NN data structures and shows the performance of our algorithms on real-world datasets.

CCS Concepts: • **Theory of computation → Sketching and sampling**; • **Information systems → Nearest-neighbor search**;

Additional Key Words and Phrases: Similarity search; Near Neighbor; Locality Sensitive Hashing; Fairness; Sampling

ACM Reference Format:
Martin Aumüller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. 2018. Sampling a Near Neighbor in High Dimensions: — Who is the Fairest of Them All?. 1, 1 (January 2018), 39 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In recent years, following a growing concern about the fairness of the algorithms and their bias toward a specific population or feature [25, 40, 48, 57], there has been an increasing interest in building algorithms that achieve (appropriately defined) *fairness* [31]. The goal is to remove, or at least minimize, unethical behavior such as discrimination and bias in

Authors' addresses: Martin Aumüller, IT University of Copenhagen, Copenhagen, Denmark, maau@itu.dk; Sariel Har-Peled, University of Illinois at Urbana-Champaign, Urbana, IL, USA, sariel@illinois.edu; Sepideh Mahabadi, Toyota Technological Institute at Chicago, Chicago, IL, USA, mahabadi@ttic.edu; Rasmus Pagh, BARC and University of Copenhagen, Copenhagen, Denmark, pagh@di.ku.dk; Francesco Silvestri, University of Padova, Padova, Italy, silvestri@dei.unipd.it.

algorithmic decision making, as nowadays, many important decisions, such as college admissions, offering home loans, or estimating the likelihood of recidivism, rely on machine learning algorithms. While algorithms are not inherently biased, nevertheless, they may amplify the already existing biases in the data. Hence, this concern has led to the design of fair algorithms for many different applications, e.g., [6, 16, 19, 23, 24, 30, 32, 49, 59, 63].

There is no unique definition of fairness (see [40] and references therein), but different formulations that depend on the computational problem at hand, and on the ethical goals we aim for. Fairness goals are often defined in the political context of socio-technical systems [58], and have to be seen in an interdisciplinary spectrum covering many fields outside computer science [66]. In particular, researchers have studied both *group fairness*[1] (where demographics of the population are preserved in the outcome), and *individual fairness* (where the goal is to treat individuals with similar conditions similarly) [31]. The latter 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.

Bias in the data used for training machine learning algorithms is a monumental challenge in creating fair algorithms [25, 44, 69, 71]. Here, we are interested in a somewhat different problem of handling the bias introduced by the data structures used by such algorithms. Specifically, data structures may introduce bias in the data stored in them, and the way they answer queries, because of the way the data is stored and how it is being accessed. It is also possible that some techniques for boosting performance, like randomization and approximation that result in non-deterministic behavior, add to the overall algorithmic bias. For instance, some database indexes for fast search might give an (unexpected) advantage to some portions of the input data. Such a defect leads to selection bias by the algorithms using such data structures. It is thus natural to want data structures that do not introduce a selection bias into the data when handling queries. To this end, imagine a data structure that can return, as an answer to a query, an item out of a set of acceptable answers. The purpose is then to return uniformly a random item out of the set of acceptable outcomes, without explicitly computing the whole set of acceptable answers (which might be prohibitively expensive).

The Near Neighbor Problem. In this work, we study similarity search and in particular the near neighbor problem from the perspective of individual fairness. Similarity search is an important primitive in many applications in computer science such as machine learning, recommender systems, data mining, computer vision, and many others; see [8, 67] for an overview. One of the most common formulations of similarity search is the r-near neighbor (r-NN) problem, formally defined as follows. Let $(\mathcal{X}, \mathcal{D})$ be a metric space where the distance function $\mathcal{D}(\cdot, \cdot)$ reflects the (dis)similarity between two data points. Given a set $S \subseteq \mathcal{X}$ of n points and a radius parameter r, the goal of the r-NN problem is to preprocess S and construct a data structure, such that for a query point $q \in \mathcal{X}$, one can report a point $p \in S$, such that $\mathcal{D}(p, q) \leq r$ if such a point exists. As all the existing algorithms for the *exact* variant of the problem have either space or query time that depends exponentially on the ambient dimension of $\mathcal{X}$, people have considered the approximate variant of the problem. In the *c-approximate near neighbor* (ANN) problem, the algorithm is allowed to report a point p whose distance to the query is at most cr if a point within distance r of the query exists, for some prespecified constant $c > 1$.

Fair Near Neighbor. As we will see, common existing data structures for similarity search have a behavior that introduces bias in the output. Our goal is to capture and algorithmically remove this bias from these data structures. Our goal is to develop a data structure for the r-near neighbor problem where we aim to be fair among "all the points"

[1]The concept is denoted as statistical fairness too, e.g., [25].

in the neighborhood, i.e., all points within distance r from the given query have the same probability to be returned. We introduce and study the *fair near neighbor* problem: 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 each point in $B_S(\mathbf{q}, r)$ is returned as near neighbor of $\mathbf{q}$ with the uniform probability of $1/n(\mathbf{q}, r)$ where $n(\mathbf{q}, r) = |B_S(\mathbf{q}, r)|$.

Locality Sensitive Hashing. Perhaps the most prominent approach to get an ANN data structure for high-dimensional data is via the Locality Sensitive Hashing (LSH) framework proposed by Indyk and Motwani [37, 45], which leads to sub-linear query time and sub-quadratic space. In particular, for $\mathcal{X} = \mathbb{R}^d$, by using LSH one can get a query time of $n^{\rho+o(1)}$ and space $n^{1+\rho+o(1)}$ where for the L_1 distance metric $\rho = 1/c$ [37, 45], and for the L_2 distance metric $\rho = 1/c^2 + o_c(1)$ [8]. In the LSH framework, which is formally introduced in Section 5.1, the idea is to hash all points using several hash functions that are chosen randomly, with the property that closer points have a higher probability of collision than the far points. Thus, the collision probability between two points is a decreasing function of their distance [21]. Therefore, the closer points to a query have a higher probability of falling into a bucket being probed than far points. Thus, reporting a random point from a random bucket computed for the query produces a distribution that is biased by the distance to the query: closer points to the query tend to have a higher probability of being chosen. On the other hand, the uniformity property required in fair NN 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 much more efficient data structures that still use LSH in a black-box way.

Applications: When random nearby is better than nearest. The bias mentioned above towards nearer points is usually a good property, but is not always desirable. Indeed, consider the following scenarios:

(I) The nearest neighbor might not be the best if the input is noisy, and the closest point might be viewed as an unrepresentative outlier. Any point in the neighborhood might be then considered to be equivalently beneficial: this is to some extent why k-NN classification [34] is so effective in reducing the effect of noise.

(II) In many cases, k-NN classification works better if k is large: however, computing the k nearest-neighbors is quite expensive if k is large [42]. The classification can be significantly sped up by extracting a random nearby neighbor.

(III) If one wants to estimate the number of items with a desired property within the neighborhood, then the easiest way to do it is via uniform random sampling from the neighborhood. In particular, this is useful for density estimation [52]. More generally, this can be seen as a special case of query sampling in database systems [61], where the goal is to return a random sample of the output of a given query, for efficiently providing statistics on the query. This can for example be used for estimating aggregate queries (e.g., `sum` or `count`), see [60] for more details. Another example for the usefulness is discrimination discovery in existing databases [56]: by performing independent queries to obtain a sample with statistical significance, we can reason about the distribution of attribute types. We could report on discrimination if the population counts grouped by a certain attribute differ much more than we would expect them to.

(IV) We are interested in anonymizing the query [2], thus returning a random near-neighbor might serve as the first line of defense in trying to make it harder to recover the query. Similarly, one might want to anonymize the nearest-neighbor [64], for applications where we are interested in a "typical" data item close to the query, without identifying the nearest item.

(V) As another application, consider a recommender system used by a newspaper to recommend articles to users. Popular recommender systems based on matrix factorization [51] 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]. As described by Adomavicius and Kwon in [3], 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.

(VI) Another natural application is simulating a random walk in the graph where two items are connected if they are in distance at most r from each other. Such random walks are used by some graph clustering algorithms [41].

To the best of our knowledge, previous results focused on exact near neighbor sampling in the Euclidean space up to three dimensions [4, 5, 43, 61]. Although these results might be extended to $\mathbb{R}^d$ for any $d > 1$, they suffer from 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., [9, 70]).

1.1 Problem formulations

In the following we formally define the variants of the fair NN problem that we consider in this paper. For all constructions presented, these guarantees hold only in the absence of a failure event that happens with probability at most δ for some arbitrarily small $\delta > 0$.

Definition 1 (r-near neighbor sampling, i.e., Fair NN with dependence). *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 $B_S(\mathbf{q}, r)$. We also refer to this problem as Fair NN with Dependence.*

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 be correlated with 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 (r-near neighbor independent sampling, i.e., Fair NN). *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 $B_S(\mathbf{q}_i, r)$;

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

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

We also refer to this problem as Fair NN.

We note that in the low-dimensional setting [4, 5, 43], the r-near neighbor independent sampling problem is usually called *independent range sampling* (IRS). Next, motivated by applications, we define two approximate variants of the problem that we study in this work. More precisely, we slightly relax the fairness constraint, allowing the probabilities of reporting a neighbor to be an "almost uniform" distribution.

Definition 3 (Approximately Fair NN). *Consider a set $S \subseteq \mathcal{X}$ of n points in a metric space $(\mathcal{X}, \mathcal{D})$. The* Approximately Fair NN *problem asks to construct a data structure for S that for any query $\mathbf{q}$, returns each point $\mathbf{p} \in B_S(\mathbf{q}, r)$ with probability $\mu_{\mathbf{p}}$ where $\mu_{\mathbf{p}}$ is an approximately uniform probability distribution: $\mathbb{P}(\mathbf{q}, r)/(1+\varepsilon) \leq \mu_{\mathbf{p}} \leq (1+\varepsilon)\mathbb{P}(\mathbf{q}, r)$, where $\mathbb{P}(\mathbf{q}, r) = 1/n(\mathbf{q}, r)$. We again make the same independence assumption as in Definition 2.*

Next, we allow the algorithm to report an almost uniform distribution from an *approximate* neighborhood of the query.

Definition 4 (Approximately Fair ANN). *Consider a set $S \subseteq \mathcal{X}$ of n points in a metric space $(\mathcal{X}, \mathcal{D})$. The* Approximately Fair ANN *problem asks to construct a data structure for S that for any query $\mathbf{q}$, returns each point $\mathbf{p} \in S'$ with probability $\mu_{\mathbf{p}}$ where $\varphi/(1+\varepsilon) \leq \mu_{\mathbf{p}} \leq (1+\varepsilon)\varphi$, where S' is a point set such that $B_S(\mathbf{q}, r) \subseteq S' \subseteq B_S(\mathbf{q}, cr)$, and $\varphi = 1/|S'|$. We again assume the same independence assumption as in Definition 2.*

1.2 Our results

We propose several solutions to the different variants of the Fair NN problem. Our solutions make use of the LSH framework [45] and we denote by $\mathcal{S}(n, c)$ the space usage and by $\mathcal{Q}(n, c)$ the running time of a standard LSH data structure that solves the c-ANN problem in the space $(\mathcal{X}, \mathcal{D})$.[2]

- Section 5.2 describes a solution to the Fair NN problem with dependence with expected running time $\widetilde{O}(\mathcal{Q}(n, c) + n(\mathbf{q}, cr) - n(\mathbf{q}, r))$ and space $\mathcal{S}(n, c) + O(n)$. The data structure uses an independent permutation of the data points on top of a standard LSH data structure and inspects buckets according to the order of points under this permutation. See Theorem 1 for the exact statement.
- In Section 5.3 we provide a data structure for Approximately Fair ANN that uses space $\mathcal{S}(n, c)$ and whose query time is $\widetilde{O}(\mathcal{Q}(n, c))$, both in expectation and also with high probability (using slightly different bounds). See Theorem 2 for the exact statement.
- Section 5.4 shows how to solve the Fair NN problem in expected query time $\widetilde{O}(\mathcal{Q}(n, c) + n(\mathbf{q}, cr)/n(\mathbf{q}, r))$ and space usage $O(\mathcal{S}(n, c))$. Each bucket is equipped with a count-sketch and the algorithm works by repeatedly sampling points within a certain window from the permutation. See Theorem 3 for the exact statement.
- In Section 6 we introduce an easy-to-implement nearly-linear space data structure based on the locality-sensitive filter approach put forward in [11, 26]. As each input point appears once in the data structure, the data structure can be easily adapted to solve the Fair NN problem. While conceptually simple, it does not use LSH as a black-box and works only for some distances: we describe it for similarity search under inner product, although it can be adapted to some other metrics (like Euclidean and Hamming distances) with standard techniques. See Theorem 5 for the exact statement.
- Lastly, in Section 7 we present an empirical evaluation of (un)fairness in traditional recommendation systems on real-world datasets, and we then analyze the additional computational cost for guaranteeing fairness. More

[2] Technically, the LSH data structure uses a factor $\Theta(\log n)$ more repetitions to obtain high probability bounds on the event that every near point collides with the query.

Result reference	Fairness definition	Indep. queries	Exact neighbor	Exact prob.	Expected query time
Theorem 1	Fair NN with dependence (Definition 1)	no	yes	yes	$O\left((n^\rho + n(\mathbf{q}, cr) - n(\mathbf{q}, r)) \log^2 n\right)$
Theorem 2	Approximately Fair ANN (Definition 4)	yes	no	no	$O(n^\rho \log n \log(n/\varepsilon))$
Lemma 19 (follows from Theorem 2)	Approximately Fair NN (Definition 3)	yes	yes	no	$O\left(n^\rho \frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)} \log n \log \frac{n}{\varepsilon}\right)$
Theorem 3	Fair NN (Definition 2)	yes	yes	yes	$O\left(\left(n^\rho + \frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\right) \log^5 n\right)$

Table 1. The table summarizes the main results in this paper. For each result, we report the version of fairness addressed and the respective query complexity. The table also recalls some features of the addressed definition: if *independent queries* are supported; if the sample is from the *exact neighborhood of near points*; if the sample *probability is exact*. Additionally, Theorem 5 provides another data structure for Definition 2 that uses linear space for points on the unit sphere; the notation slightly differs from the one in the table, and we refer to Section 6 for more details.

precisely, we compare the performance of our algorithms with the algorithm that uniformly picks a bucket and reports a random point, on five different datasets using both Euclidean distance and Jaccard similarity. Our empirical results show that while the standard LSH algorithm fails to fairly sample a point in the neighborhood of the query, our algorithms produce empirical distributions which are much closer to the uniform distribution. We further include a case study highlighting the unfairness that might arise in special situations when considering Approximately Fair ANN.

We remark that for the approximate variants, the dependence of our algorithms on ε is only $O(\log(1/\varepsilon))$. While we omitted the exact poly-logarithmic factors in the list above, they are generally lower for the approximate versions. Furthermore, these methods can be embedded in the existing LSH method to achieve unbiased query results in a straightforward way. On the other hand, the exact methods will have higher logarithmic factors and use additional data structures.

1.3 Data structure for sampling from a sub-collection of sets

In order to obtain our results, we first study a more generic problem in Section 3: given a collection $\mathcal{F}$ of sets from a universe of n elements, a query is a sub-collection $\mathcal{G} \subseteq \mathcal{F}$ of these sets and the goal is to sample (almost) uniformly from the union of the sets in this sub-collection. We also show how to modify the data structure to handle outliers in Section 4 as the sampling algorithm needs to ignore such points and to not report them in the sample. We also show how to modify the data structure to handle outliers in Section 4, as it is the case for LSH, as the sampling algorithm needs to ignore such points once they are reported as a sample. This will allow us to derive most of the results concerning variants of Fair NN in Section 5 as corollaries from these more abstract data structures.

Applications. Here are a few examples of applications of a data structure that provides uniform samples from a union of sets:

(A) Given a subset A of vertices in the graph, randomly pick (with uniform distribution) a neighbor to one of the vertices of A. This can be used in simulating disease spread [47].
(B) To implement Fair NN as shown in Sections 5 and 6. Intuitively, we build on the LSH approach and define $\mathcal{F}$ as the collection of buckets in the LSH data structure. Then for a given query point q, we define $\mathcal{G}$ as the buckets containing q. The Fair NN problem then reduces to uniform sampling on this family $\mathcal{G}$.

(C) Uniform sampling for range searching [4, 5, 43]. Indeed, consider a set of points, stored in a data structure for range queries. Using the above, we can support sampling from the points reported by several queries, even if the reported answers are not disjoint.

Being unaware of any previous work on this problem, we believe this data structure is of independent interest.

1.4 Discussion of Fairness Assumptions

In the context of our problem definition we assume—as do many papers on fairness-related topics—an implicit world-view described by Friedler *et al.* [36] as "what you see is what you get". WYSIWYG means that a certain distance between individuals in the so-called "construct space" (the true merit of individuals) is approximately represented by the feature vectors in "observed space". As described in their paper, one has to subscribe to this world-view to achieve certain fairness conditions. Moreover, we acknowledge that our problem definition requires to set a threshold parameter r which might be internal to the dataset. This problem occurs frequently in the machine learning community, e.g., when score thresholding is applied to obtain a classification result. Kannan *et al.* discuss the fairness implications of such threshold approaches in [46].

We stress that the r-near neighbor independent sampling problem might not be ***the*** fairness definition in the context of similarity search. Instead, we think of it as a suitable starting point for discussion, and acknowledge that the application will often motivate a suitable fairness property. For example, in the case of a recommender system, we might want to consider a weighted case where closer points are more likely to be returned. As discussed earlier, and exemplified in the experimental evaluation, a standard LSH approach does not have such guarantees despite its monotonic collision probability function. We leave the weighted case as an interesting direction for future work.

1.5 Previous versions of this work

The different problem variants discussed in Section 1.1 were first introduced in [39] (for approximate fair near neighbor search) and [15] (for fair near neighbor search). Moreover, [39] also introduced the more abstract problem of sampling uniformly from a sub-collection of sets. Both papers present algorithms that achieve running time bounds that are roughly similar to the main statements presented in Section 1.2. In this paper, we extend the work in [15] by describing variants that work in more abstract setting of sampling from a sub-collection. Furthermore, we discuss a high probability bound on the running time of the query algorithm in Section 5.4, and simplify the algorithms for approximate fair near neighbors that were described in [39]. Lastly, this paper presents a unified experimental view on the problem of sampling a fair near neighbor. A succinct summary of the technical contributions of this paper was published in [14].

2 PRELIMINARIES

Set representation. Let $\mathcal{U}$ be an underlying ground set of n objects (i.e., elements). In this paper, we deal with sets of objects. Assume that such a set $A \subseteq \mathcal{U}$ is stored in some reasonable data structure, where one can insert delete, or query an object in constant time. Querying for an object $o \in \mathcal{U}$ requires deciding if $o \in A$. Such a representation of a set is straightforward to implement using an array to store the objects, and a hash table. This representation allows random access to the elements in the set, or uniform sampling from the set.

If hashing is not feasible, one can just use a standard dictionary data structure – this would slow down the operations by a logarithmic factor.

Subset size estimation. We need the following standard estimation tool, [18, Lemma 2.8].

Lemma 1. *Consider two sets $B \subseteq U$, where $n = |U|$. Let $\xi, \gamma \in (0,1)$ be parameters, such that $\gamma < 1/\log n$. Assume that one is given access to a membership oracle that, given an element $x \in U$, returns whether or not $x \in B$. Then, one can compute an estimate s, such that $(1-\xi)|B| \leq s \leq (1+\xi)|B|$, and computing this estimates requires $O((n/|B|)\xi^{-2}\log\gamma^{-1})$ oracle queries. The returned estimate is correct with probability $\geq 1-\gamma$.*

Weighted sampling. We need the following standard data structure for weighted sampling.

Lemma 2. *Given a set of objects $\mathcal{H} = \{o_1, \ldots, o_t\}$, with associated weights $w_1, \ldots, w_t$, one can preprocess them in $O(t)$ time, such that one can sample an object out of $\mathcal{H}$. The probability of an object o_i to be sampled is $w_i / \sum_{j=1}^{t} w_j$. In addition the data structure supports updates to the weights. An update or sample operation takes $O(\log t)$ time.*

Proof. Build a balanced binary tree T, where the objects of $\mathcal{G}$ are stored in the leaves. Every internal node u of T, also maintains the total weight $w(u)$ of the objects in its subtree. The tree T has height $O(\log t)$, and weight updates can be carried out in $O(\log t)$ time, by updating the path from the root to the leaf storing the relevant object.

Sampling is now done as follows – we start the traversal from the root. At each stage, when at node u, the algorithm considers the two children u_1, u_2. It continues to u_1 with probability $w(u_1)/w(u)$, and otherwise it continues into u_2. The object sampled is the one in the leaf that this traversal ends up at. □

Sketch for distinct elements. In Section 3.5 we will use sketches for estimating the number of distinct elements. Consider a stream of m elements $x_1, \ldots, x_m$ in the domain $[n] = \{1, \ldots, n\}$ 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 [17], which generalizes the seminal result by Flajolet and Martin [35]. The data structure consists of $\Delta = \Theta(\log(1/\delta))$ lists $L_1, \ldots L_\Delta$; 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 [17] that the median $\widehat{F_0}$ of the values $tn^3/v_0, \ldots, tn^3/v_\Delta$, where v_w denotes the tth 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, $(1-\epsilon)F_0 \leq \widehat{F_0} \leq (1+\epsilon)F_0$. The data structure requires $O\left(\epsilon^{-2}\log m \log(1/\delta)\right)$ bits and $O\left(\log(1/\epsilon)\log m \log(1/\delta)\right)$ 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 DATA STRUCTURE FOR SAMPLING FROM THE UNION OF SETS

Problem definition. Assume you are given a data structure that contains a large collection $\mathcal{F}$ of sets of objects. In total, there are $n = |\bigcup \mathcal{F}|$ objects. The sets in $\mathcal{F}$ are not necessarily disjoint. The task is to preprocess the data structure, such that given a sub-collection $\mathcal{G} \subseteq \mathcal{F}$ of the sets, one can quickly pick uniformly at random an object from the set $\bigcup \mathcal{G} := \bigcup_{A \in \mathcal{G}} A$. As notation, we define $m = ||\mathcal{G}|| := \sum_{A \in \mathcal{G}} |A|$, $g = |\mathcal{G}|$, and $N = |\bigcup \mathcal{G}|$.

In the same way as there were multiple fairness definitions in Section 1.1, we can wish for a one-shot sample from $\bigcup \mathcal{G}$ (allowing for *dependence*) or for *independent* results. Moreover, sample probabilities can be *exact* or *approximate*. Since all elements are valid elements to return, there is no difference between an *exact* or *approximate* neighborhood. This will be the topic of Section 4.

Preprocessing. For each set $A \in \mathcal{F}$, we build the set representation mentioned in the preliminaries section. In addition, we assume that each set is stored in a data structure that enables easy random access or uniform sampling on this set

(for example, store each set in its own array). Thus, for each set A, and an element, we can decide if the element is in A in constant time.

Solutions. The naive solution is to take the sets under consideration (in $\mathcal{G}$), compute their union, and sample directly from the union set $\bigcup\mathcal{G}$. Our purpose is to do (much) better – in particular, the goal is to get a query time that depends logarithmically on the total size of all sets in $\mathcal{G}$.

The approaches discussed in the following use two ideas: (i) using a random permutation of the universe to introduce a natural order of the elements and (ii) using rejection sampling to introduce randomness during the query to guarantee (approximately) equal output probability.

The first approach in Section 3.1 uses only the random permutation and results in one-shot uniform sample, lacking independence. The data structures in Section 3.2–Section 3.4 build on top of rejection sampling. They provide independence, but introduce approximate probabilities. Finally, the data structure in Section 3.5 makes use of both ideas to produce an independent sample with exact probabilities.

3.1 Uniform sampling with dependence

We start with a simple data structure for sampling a uniform point from a collection of sets, i.e., given a sub-collection $\mathcal{G}$, sample a point in $\bigcup\mathcal{G}$ uniformly at random. Since all randomness is in the preprocessing of the data structure, this variant does not guarantee independence regarding multiple queries.

The main idea is quite simple. We initially assign a (random) rank to each of the n objects in $\bigcup\mathcal{F}$ using a random permutation. We sort all sets in $\mathcal{F}$ according to the ranks of their objects. For a given query collection $\mathcal{G}$, we iterate through each set and keep track of the element with minimum rank in $\bigcup\mathcal{G}$. This element will be returned as answer to the query. The random permutation guarantees that all points in $\mathcal{G}$ have the same chance of being returned.

Lemma 3. *Let $N = |\bigcup\mathcal{G}|$, $g = |\mathcal{G}|$, and $m = \sum_{X\in\mathcal{G}} |X|$. The above algorithm samples an element $x \in \bigcup\mathcal{G}$ according to the uniform distribution in time $O(g)$.*

Proof. The algorithm keeps track of the element of minimum rank among the g different sets. Since all of them are sorted during preprocessing, this takes time $O(1)$ per set. The sample is uniformly distributed because each element has the same chance of being the smallest under the random permutation. □

If we repeat the same query on the same sub-collection $\mathcal{G}$, the above data structure always returns the same point: if we let OUT_i denote the output of the ith sample from $\mathcal{G}$, we have that $\Pr[OUT_i = x | OUT_1 = x_1]$ is 1 if $x = x_1$ and 0 otherwise. We now extend the above data structure to get independent samples when we repeat the query on the *same* sub-collection $\mathcal{G}$. That is,

$$\Pr[OUT_i = x | OUT_{i-1} = x_{i-1}, \ldots OUT_1 = x_1] = \Pr[OUT_i = x] = 1/N.$$

We add to each set in $\mathcal{F}$ a priority queue which supports key updates, using ranks as key. For each point $x \in \bigcup\mathcal{F}$, we keep a pointer to all sets (and their respective priority queue) containing x. At query time, we search the point in sets $\mathcal{G}$ with minimum rank, as in the previous approach. Then, just before returning the sample, we 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 [50]: let r_x be the rank of x; we randomly select a rank r in $\{r_x, \ldots, n\}$ and let y be the point with rank r; then we swap the ranks of x and y and update accordingly the priority queues. We have the following lemma, where δ denotes the maximum number of sets in $\mathcal{F}$ containing a point in $\bigcup\mathcal{F}$.

Lemma 4. *Let $n = |\bigcup \mathcal{F}|$, $N = |\bigcup \mathcal{G}|$, $g = |\mathcal{G}|$, and $\delta = \max_{x \in \bigcup \mathcal{F}} |\{A \in \mathcal{F} | x \in A\}|$. Assume that we repeat k times the above query procedure on the sub-collection $\mathcal{G}$, and let OUT_i denote the output of the ith iteration with $1 \leq i \leq k$. Then, for any $x \in \bigcup \mathcal{G}$, we have that $\Pr[OUT_1 = x] = 1/N$ and $\Pr[OUT_i = x | OUT_{i-1} = x_{i-1}, \ldots OUT_1 = x_1] = \Pr[OUT_1 = x] = 1/N$ for any $i > 1$. Each query requires $O((g + \delta) \log n)$ time.*

Proof. Let L be the set of points in $\bigcup \mathcal{F}$ with ranks larger than r_x. We have that $|L| = n - r_x$ and $\bigcup \mathcal{G} \setminus \{x\} \subseteq L$. Before the swap, the ranks of points in L are unknown and all permutations of points in L are equally likely. After the swap, each point in $L \bigcup \{x\}$ has rank r_x with probability $1/(n - r_x + 1)$, independently of previous random choices. Moreover, each point in $\bigcup \mathcal{G}$ has probability $1/N$ to be the point in $\bigcup \mathcal{G}$ with smaller rank after the swap.

By assuming that each priority queue can be updated in $O(\log n)$ time, the point with minimum rank can be extracted in $O(g \log n)$ time, and the final rank shuffle requires $O(\delta \log n)$ time as we need to update the priority queues of the at most 2δ sets containing point x or point y. □

We remark that the rerandomization technique is only restricted to single element queries: over time all elements in $\mathcal{G}$ get higher and higher ranks: this means that for another collection $\mathcal{G}'$, which intersects $\mathcal{G}$, the elements in $\bigcup \mathcal{G} \setminus \bigcup \mathcal{G}'$ become more and more likely to be returned. The next sections provide slightly more involved data structures that guarantee independence even among different queries.

3.2 Uniform sampling via exact degree computation

The query is a family $\mathcal{G} \subseteq \mathcal{F}$. The ***degree*** of an element $x \in \bigcup \mathcal{G}$, is the number of sets of $\mathcal{G}$ that contain it – that is, $\mathrm{d}_{\mathcal{G}}(x) = \left|\mathrm{D}_{\mathcal{G}}(x)\right|$, where $\mathrm{D}_{\mathcal{G}}(x) = \{A \in \mathcal{G} \mid x \in A\}$. The algorithm repeatedly does the following:

(I) Picks one set from $\mathcal{G}$ with probabilities proportional to their sizes. That is, a set $A \in \mathcal{G}$ is picked with probability $|A| / m$.
(II) It picks an element $x \in A$ uniformly at random.
(III) Computes the degree $\mathrm{d} = \mathrm{d}_{\mathcal{G}}(x)$.
(IV) With probability $1/\mathrm{d}$, output x and stop. Otherwise, continues to the next iteration.

Lemma 5. *Let $N = |\bigcup \mathcal{G}|$, $g = |\mathcal{G}|$, and $m = \sum_{X \in \mathcal{G}} |X|$. The above algorithm samples an element $x \in \bigcup \mathcal{G}$ according to the uniform distribution. The algorithm takes in expectation $O(gm/N) = O(g^2)$ time. The query time is $O(g^2 \log N)$ with high probability.*

Proof. Observe that an element $x \in \bigcup \mathcal{G}$ is picked by step (II) with probability $\alpha = \mathrm{d}(x)/m$. The element x is output with probability $\beta = 1/\mathrm{d}(x)$. As such, the probability of x to be output by the algorithm in this round is $\alpha\beta = 1/m$. This implies that the output distribution is uniform on all the elements of $\bigcup \mathcal{G}$.

The probability of success in a round is N/m, which implies that in expectation m/N rounds are used, and with high probability $O((m/N) \log N)$ rounds. Computing the degree $\mathrm{d}_{\mathcal{G}}(x)$ takes $O(|\mathcal{G}|)$ time, which implies the first bound on the running time. As for the second bound, observe that an element can appear only once in each set of $\mathcal{G}$, which readily implies that $\mathrm{d}(y) \leq |\mathcal{G}|$, for all $y \in \bigcup \mathcal{G}$. □

3.3 Almost uniform sampling via degree approximation

The bottleneck in the above algorithm is computing the degree of an element. We replace this by an approximation.

Definition 5. *Given two positive real numbers x and y, and a parameter $\varepsilon \in (0, 1)$, the numbers x and y are ε-***approximation** *of each other, denoted by $x \approx_\varepsilon y$, if $x/(1+\varepsilon) \leq y \leq x(1+\varepsilon)$ (or, equivalently, $y/(1+\varepsilon) \leq x \leq y(1+\varepsilon)$).*

In the approximate version, given an item $x \in \bigcup \mathcal{G}$, we can approximate its degree and get an improved runtime for the algorithm.

Lemma 6. *The input is a family of sets $\mathcal{F}$ that one can preprocess in linear time. Let $\mathcal{G} \subseteq \mathcal{F}$ be a sub-family and let $N = |\bigcup \mathcal{G}|$, $g = |\mathcal{G}|$, and $\varepsilon \in (0,1)$ be a parameter. One can sample an element $x \in \bigcup \mathcal{G}$ with almost uniform probability distribution. Specifically, the probability of an element to be output is $\approx_\varepsilon 1/N$. After linear time preprocessing, the query time is $O\big(g\varepsilon^{-2}\log N\big)$ in expectation, and the query succeeds with high probability.*

Proof. Let $m = ||\mathcal{G}||$. Since $\mathrm{d}(x) = \big|\mathrm{D}_{\mathcal{G}}(x)\big|$, it follows that we need to approximate the size of $\mathrm{D}_{\mathcal{G}}(x)$ in $\mathcal{G}$. Given a set $A \in \mathcal{G}$, we can in constant time check if $x \in A$, and as such decide if $A \in \mathrm{D}_{\mathcal{G}}(x)$. It follows that we can apply the algorithm of Lemma 1, which requires $W(x) = O\big(\frac{g}{\mathrm{d}(x)}\varepsilon^{-2}\log N\big)$ time, where the algorithm succeeds with high probability. The query algorithm is the same as before, except that it uses the estimated degree.

For $x \in \bigcup \mathcal{G}$, let $\mathcal{E}_x$ be the event that the element x is picked for estimation in a round, and let $\mathcal{E}'_x$ be the event that it was actually output in that round. Clearly, we have $\mathbb{P}\big[\mathcal{E}'_x \bigm| \mathcal{E}_x\big] = 1/d$, where d is the degree estimate of x. Since $d \approx_\varepsilon \mathrm{d}(x)$ (with high probability), it follows that $\mathbb{P}\big[\mathcal{E}'_x \bigm| \mathcal{E}_x\big] \approx_\varepsilon 1/\mathrm{d}(x)$. Since there are $\mathrm{d}(x)$ copies of x in $\mathcal{G}$, and the element for estimation is picked uniformly from the sets of $\mathcal{G}$, it follows that the probability of any element $x \in \bigcup \mathcal{G}$ to be output in a round is

$$\mathbb{P}\big[\mathcal{E}'_x\big] = \mathbb{P}\big[\mathcal{E}'_x \bigm| \mathcal{E}_x\big]\,\mathbb{P}[\mathcal{E}_x] = \mathbb{P}\big[\mathcal{E}'_x \bigm| \mathcal{E}_x\big]\,\frac{\mathrm{d}(x)}{m} \approx_\varepsilon 1/m,$$

as $\mathcal{E}'_x \subseteq \mathcal{E}_x$. As such, the probability of the algorithm terminating in a round is $\alpha = \sum_{x \in \bigcup \mathcal{G}} \mathbb{P}\big[\mathcal{E}'_x\big] \approx_\varepsilon N/m \geq N/2m$. As for the expected amount of work in each round, observe that it is proportional to

$$W = \sum_{x \in \bigcup \mathcal{G}} \mathbb{P}[\mathcal{E}_x] W(x) = \sum_{x \in \bigcup \mathcal{G}} \frac{\mathrm{d}(x)}{m}\,\frac{g}{\varepsilon^2 \mathrm{d}(x)}\log N = O\Big(\frac{ng}{m}\varepsilon^{-2}\log N\Big).$$

Intuitively, since the expected amount of work in each iteration is W, and the expected number of rounds is $1/\alpha$, the expected running time is $O(W/\alpha)$. This argument is not quite right, as the amount of work in each round affects the probability of the algorithm to terminate in the round (i.e., the two variables are not independent). We continue with a bit more care – let L_i be the running time in the ith round of the algorithm if it was to do an ith iteration (i.e., think about a version of the algorithm that skips the experiment in the end of the iteration to decide whether it is going to stop), and let Y_i be a random variable that is 1 if the (original) algorithm had not stopped at the end of the first i iterations of the algorithm.

By the above, we have that $y_i = \mathbb{P}[Y_i = 1] = \mathbb{P}[Y_i = 1 \mid Y_{i-1} = 1]\,\mathbb{P}[Y_{i-1} = 1] \leq (1-\alpha)y_{i-1} \leq (1-\alpha)^i$, and $\mathbb{E}[L_i] = O(W)$. Importantly, L_i and Y_{i-1} are independent (while L_i and Y_i are dependent). We clearly have that the running time of the algorithm is $O\big(\sum_{i=1}^{\infty} Y_{i-1}L_i\big)$ (here, we define $Y_0 = 1$). Thus, the expected running time of the algorithm is proportional to

$$\begin{aligned}\mathbb{E}\Big[\sum_i Y_{i-1}L_i\Big] &= \sum_i \mathbb{E}[Y_{i-1}L_i] = \sum_i \mathbb{E}[Y_{i-1}]\,\mathbb{E}[L_i] \leq W\sum_i y_{i-1} \leq W\sum_{i=1}^{\infty}(1-\alpha)^{i-1} = \frac{W}{\alpha}\\ &= O(g\varepsilon^{-2}\log N),\end{aligned}$$

because of linearity of expectations, and since L_i and Y_{i-1} are independent. □

Remark 1. *The query time of Lemma 6 deteriorates to $O\Big(g\varepsilon^{-2}\log^2 T\Big)$ if one wants the bound to hold with high probability, where T is some (rough) upper bound on N. This follows by restarting the query algorithm if the query time exceeds (say by a factor of two) the expected running time. A standard application of Markov's inequality implies that this*

process would have to be restarted at most $O(\log T)$ times, with high probability. Here, one can set T to be $n \cdot g$ as a rough upper bound on N.

Remark 2. *The sampling algorithm is independent of whether or not we fully know the underlying family $\mathcal{F}$ and the sub-family $\mathcal{G}$. This means the past queries do not affect the sampled object reported for the query $\mathcal{G}$. Therefore, the almost uniform distribution property holds in the presence of several queries and independently for each of them.*

3.4 Almost uniform sampling via simulation

It turns out that one can avoid the degree approximation stage in the above algorithm, and achieve only a polylogarithmic dependence on ε^{-1}. To this end, let x be the element picked. We need to simulate a process that accepts x with probability $1/\mathsf{d}(x)$.

We start with the following natural idea for estimating $\mathsf{d}(x)$ – probe the sets randomly (with replacement), and stop in the ith iteration if it is the first iteration where the probe found a set that contains x. If there are g sets, then the distribution of i is geometric, with probability $p = \mathsf{d}(x)/g$. In particular, in expectation, $\mathbb{E}[i] = g/\mathsf{d}(x)$, which implies that $\mathsf{d}(x) = g/\mathbb{E}[i]$. As such, it is natural to take g/i as an estimation for the degree of x. Thus, to simulate a process that succeeds with probability $1/\mathsf{d}(x)$, it would be natural to return 1 with probability i/g and 0 otherwise. Surprisingly, while this seems like a heuristic, it does work, under the right interpretation, as testified by the following.

Lemma 7. *Assume we have g urns, and exactly $\mathsf{d} > 0$ of them, are non-empty. Furthermore, assume that we can check if a specific urn is empty in constant time. Then, there is a randomized algorithm, that outputs a number $Y \geq 0$, such that $\mathbb{E}[Y] = 1/\mathsf{d}$. The expected running time of the algorithm is $O(g/\mathsf{d})$.*

Proof. The algorithm repeatedly probes urns (uniformly at random), until it finds a non-empty urn. Assume it found a non-empty urn in the ith probe. The algorithm outputs the value i/g and stops.

Setting $p = \mathsf{d}/g$, and let Y be the output of the algorithm. we have that

$$\mathbb{E}\big[Y\big] = \sum_{i=1}^{\infty} \frac{i}{g}(1-p)^{i-1}p = \frac{p}{g(1-p)} \sum_{i=1}^{\infty} i(1-p)^i = \frac{p}{g(1-p)} \cdot \frac{1-p}{p^2} = \frac{1}{pg} = \frac{1}{\mathsf{d}},$$

using the formula $\sum_{i=1}^{\infty} ix^i = x/(1-x)^2$.

The expected number of probes performed by the algorithm until it finds a non-empty urn is $1/p = g/\mathsf{d}$, which implies that the expected running time of the algorithm is $O(g/\mathsf{d})$. □

The natural way to deploy Lemma 7, is to run its algorithm to get a number y, and then return 1 with probability y. The problem is that y can be strictly larger than 1, which is meaningless for probabilities. Instead, we backoff by using the value y/Δ, for some parameter Δ. If the returned value is larger than 1, we just treat it at zero. If the zeroing never happened, the algorithm would return one with probability $1/(\mathsf{d}(x)\Delta)$ – which we can use for our purposes via, essentially, amplification. Instead, the probability of success is going to be slightly smaller, but fortunately, the loss can be made arbitrarily small by taking Δ to be sufficiently large.

Lemma 8. *There are g urns, and exactly $\mathsf{d} > 0$ of them are not empty. Furthermore, assume one can check if a specific urn is empty in constant time. Let $\gamma \in (0,1)$ be a parameter. Then one can output a number $Z \geq 0$, such that $Z \in [0,1]$, and $\mathbb{E}[Z] \in I = \left[\frac{1}{\mathsf{d}\Delta} - \gamma, \frac{1}{\mathsf{d}\Delta}\right]$, where $\Delta = \lceil \ln \gamma^{-1} \rceil + 4 = \Theta(\log \gamma^{-1})$. The expected running time of the algorithm is $O(g/\mathsf{d})$.*

Alternatively, the algorithm can output a bit X, such that $\mathbb{P}[X = 1] \in I$.

Proof. We modify the algorithm of Lemma 7, so that it outputs $i/(g\Delta)$ instead of i/g. If the algorithm does not stop in the first $g\Delta + 1$ iterations, then the algorithm stops and outputs 0. Observe that the probability that the algorithm fails to stop in the first $g\Delta$ iterations, for $p = \mathrm{d}/g$, is $(1-p)^{g\Delta} \leq \exp\left(-\frac{\mathrm{d}}{g} g\Delta\right) \leq \exp(-\mathrm{d}\Delta) \leq \exp(-\Delta) \ll \gamma$.

Let Z be the random variable that is the number output by the algorithm. Arguing as in Lemma 7, we have that $\mathbb{E}[Z] \leq 1/(\mathrm{d}\Delta)$. More precisely, we have $\mathbb{E}[Z] = \frac{1}{\mathrm{d}\Delta} - \sum_{i=g\Delta+1}^{\infty} \frac{i}{g\Delta}(1-p)^{i-1}p$. Let

$$\sum_{i=gj+1}^{g(j+1)} \frac{i}{g}(1-p)^{i-1}p \leq (j+1) \sum_{i=gj+1}^{g(j+1)} (1-p)^{i-1}p = (j+1)(1-p)^{gj} \sum_{i=0}^{g-1} (1-p)^i p$$
$$\leq (j+1)(1-p)^{gj} \leq (j+1)\left(1 - \frac{\mathrm{d}}{g}\right)^{gj} \leq (j+1)\exp(-\mathrm{d}j).$$

Let $g(j) = \frac{j+1}{\Delta}\exp(-\mathrm{d}j)$. We have that $\mathbb{E}[Z] \geq \frac{1}{\mathrm{d}\Delta} - \beta$, where $\beta = \sum_{j=\Delta}^{\infty} g(j)$. Furthermore, for $j \geq \Delta$, we have

$$\frac{g(j+1)}{g(j)} = \frac{(j+2)\exp(-\mathrm{d}(j+1))}{(j+1)\exp(-\mathrm{d}j)} \leq \left(1 + \frac{1}{\Delta}\right)e^{-\mathrm{d}} \leq \frac{5}{4}e^{-\mathrm{d}} \leq \frac{1}{2}.$$

As such, we have that

$$\beta = \sum_{j=\Delta}^{\infty} g(j) \leq 2g(\Delta) \leq 2\frac{\Delta+1}{\Delta}\exp(-\mathrm{d}\Delta) \leq 4\exp(-\Delta) \leq \gamma,$$

by the choice of value for Δ. This implies that $\mathbb{E}[Z] \geq 1/(\mathrm{d}\Delta) - \beta \geq 1/(\mathrm{d}\Delta) - \gamma$, as desired.

The alternative algorithm takes the output Z, and returns 1 with probability Z, and zero otherwise. □

Lemma 9. *The input is a family of sets $\mathcal{F}$ that one preprocesses in linear time. Let $\mathcal{G} \subseteq \mathcal{F}$ be a sub-family and let $N = |\bigcup \mathcal{G}|$, $g = |\mathcal{G}|$, and let $\varepsilon \in (0,1)$ be a parameter. One can sample an element $x \in \bigcup \mathcal{G}$ with almost uniform probability distribution. Specifically, the probability of an element to be output is $\approx_\varepsilon 1/N$. After linear time preprocessing, the query time is $O(g\log(g/\varepsilon))$, in expectation, and the query succeeds, with probability $\geq 1 - 1/g^{O(1)}$.*

Proof. The algorithm repeatedly samples an element x using steps (I) and (II) of the algorithm of Section 3.2. The algorithm returns x if the algorithm of Lemma 8, invoked with $\gamma = (\varepsilon/g)^{O(1)}$ returns 1. We have that $\Delta = \Theta(\log(g/\varepsilon))$. Let $\alpha = 1/(\mathrm{d}(x)\Delta)$. The algorithm returns x in this iteration with probability p, where $p \in [\alpha - \gamma, \alpha]$. Observe that $\alpha \geq 1/(g\Delta)$, which implies that $\gamma \ll (\varepsilon/4)\alpha$, it follows that $p \approx_\varepsilon 1/(\mathrm{d}(x)\Delta)$, as desired. The expected running time of each round is $O(g/\mathrm{d}(x))$.

Arguing as in Lemma 6, this implies that each round takes $O(Ng/m)$ time in expectation, where $m = ||\mathcal{G}||$. Similarly, the expected number of rounds, in expectation, is $O(\Delta m/N)$. Again, arguing as in Lemma 6, implies that the expected running time is $O(g\Delta) = O(g\log(g/\varepsilon))$. □

Remark 3. *Similar to Remark 1, the query time of Lemma 9 can be made to work with high probability with an additional logarithmic factor. Thus with high probability, the query time is $O(g\log(g/\varepsilon)\log N)$.*

3.5 Uniform sampling using random ranks

In this section, we present a data structure that samples an element uniformly at random from $\bigcup \mathcal{G}$ using both ideas from the previous subsections: we assign a random rank to each object as in Section 3.1, and use rejection sampling as in Sections 3.2–3.4 to provide independent and uniform output probabilities. Let Λ be the sequence of the $n = |\bigcup \mathcal{F}|$ input elements after a random permutation; the rank of an element is its position in Λ. We first highlight the main idea of the query procedure.

Let $k \geq 1$ be a suitable value that depends on the collection $\mathcal{G}$, and assume that Λ is split into k segments Λ_i, with $i \in \{0, \ldots, k-1\}$. (We assume for simplicity that n and k are powers of two.) Each segment Λ_i contains the n/k elements in Λ with rank in $[i \cdot n/k, (i+1) \cdot n/k)$. We denote with $\lambda_{\mathcal{G},i}$ the number of elements from $\bigcup \mathcal{G}$ in Λ_i, and with $\lambda \geq \max_i\{\lambda_{\mathcal{G},i}\}$ an upper bound on the number of these elements in each segment. By the initial random permutation, we have that each segment contains at most $\lambda = \Theta\left((N/k) \log n\right)$ elements from $\bigcup \mathcal{G}$ with probability at least $1 - 1/n^2$. (Of course, N is *not* known at query time.)

The query algorithm works in the following three steps in which all random choices are independent.

(I) Select uniformly at random an integer h in $\{0, \ldots, k-1\}$ (i.e., select a segment Λ_h);
(II) With probability $\lambda_{\mathcal{G},h}/\lambda$ move to step (III), otherwise repeat step (I);
(III) Return an element uniformly sampled among the elements in $\bigcup \mathcal{G}$ in Λ_h.

We note that in Step (II), $\lambda_{\mathcal{G},h}$ can be computed by iterating through the g sets and collection points using a range query on segment Λ_h: since elements in each set are sorted by their rank, the range query can be carried out by searching for rank hn/k using a binary search and then enumerating all elements with rank smaller than $(h+1)n/k$.

Since each object in $\bigcup \mathcal{G}$ has a probability of $1/(k\lambda)$ of being returned in Step (III), the result is a uniform sample of $\bigcup \mathcal{G}$. The algorithm described above works for all choices of k, but a good choice has to depend on $\mathcal{G}$ for the following reasons. On the one hand, the segments should be small, because otherwise Step (III) will take too long. On the other hand, they have to contain at least one element from $\bigcup \mathcal{G}$, otherwise we sample many "empty" segments in Step (I). We will see that the number k of segments should be roughly set to N to balance the trade-off. However, the number N of distinct elements in $\bigcup \mathcal{G}$ is not known. Thus, we set $k = 2\widehat{s}_{\mathcal{G}}$, where $\widehat{s}_{\mathcal{G}}$ is a $1/2$-approximation of N. Such an estimate can be computed by storing a count distinct sketch for each set in $\mathcal{F}$. To compute $\widehat{s}_{\mathcal{G}}$ we merge the count distinct sketches of all g sets of $\mathcal{G}$. To compute $\lambda_{\mathcal{G},h}$ efficiently, we assume that, at construction time, the elements in each set in $\mathcal{F}$ are sorted by their rank.

Lemma 10. *Let $N = |\bigcup \mathcal{G}|$, $g = |\mathcal{G}|$, $m = \sum_{X \in \mathcal{G}} |X|$, and $n = |\bigcup \mathcal{F}|$. With probability at least $1 - 1/n^2$, the algorithm described above returns an element $x \in \bigcup \mathcal{G}$ according to the uniform distribution. The algorithm has an expected running time of $O(g \log^2 n)$.*

Proof. We start by bounding the initial failure probability of the data structure. By a union bound, we have that the following two events hold simultaneously with probability at least $1 - 1/n^2$:

(1) Count distinct sketches provide a $1/2$-approximation of N. By setting $\delta = 1/(2n^2)$ in the count distinct sketch construction (see Section 2), the approximation guarantee holds with probability at least $1 - 1/(2n^2)$.
(2) Every segment of size n/k contains no more than $\lambda = \Theta(\log n)$ elements from $\bigcup \mathcal{G}$. As elements are initially randomly permuted, the claim holds with probability at least $1 - 1/(2n^2)$ by suitably setting the constant in $\lambda = \Theta(\log n)$.

From now on assume that these events are true.

Each element in $\bigcup \mathcal{G}$ has the same probability $1/(k\Lambda)$ of being returned in Step (III), so all points are equally likely to be sampled. Note also that the guarantees are independent of the initial random permutation as soon as the two events above hold. This means that the data structure returns a uniform sample from a union-of-sets.

We now focus on the time complexity of the query algorithm. In Step (II), the computation of $\lambda_{\mathcal{G},h}$ takes time $O(g \log n)$: the binary search and the enumeration requires $O(\log n + o)$ time for each set, where o is the output size; Since each segment contains $O(\log n)$ elements from $\bigcup \mathcal{G}$, one iteration of Step (II) takes time $O(g \log n)$. By our choice

of k, we have that $\lambda_{\mathcal{G},h}/\lambda = \Theta(1/\log n)$, thus we expect to carry out $\Theta(\log n)$ iterations before reaching (III). Thus, we expect to spend time $O(g \log^2 n)$ before reaching Step (III). Step (III) again takes time $O(g \log n)$, with the same analysis as above. □

Remark 4. *The query time of Lemma 10 can be made to work with high probability with an additional logarithmic factor. Thus with high probability, the query time is* $O\left(g \log^3(n)\right)$.

4 HANDLING OUTLIERS

Imagine a situation where we have a marked set of outliers O that is provided as input together with the sub-collection $\mathcal{G}$. We are interested in sampling from $\bigcup \mathcal{G} \setminus O$. We assume that the total degree of the outliers in the query is at most m_O for some prespecified parameter m_O. More precisely, we have $\mathrm{d}_{\mathcal{G}}(O) = \sum_{x \in O} \mathrm{d}_{\mathcal{G}}(x) \leq m_O$.

4.1 Sampling with Dependence

We run a variant of the original algorithm from Section 3.1. We use a priority queue PQ to keep track of the point with smallest rank in $\bigcup \mathcal{G}$. Initially, for each $G \in \mathcal{G}$ we add the pair (x, G) to the priority queue, where x is the element with the smallest rank in G. As long as the element with the smallest rank in PQ is not in $\bigcup \mathcal{G} \setminus O$, we iterate the following: Let (x, G) be the entry extracted by an `extractMin` operation on PQ. Let y be the element in G with the next largest rank. Insert (y, G) into PQ.

Lemma 11. *The input is a family of sets $\mathcal{F}$ that one can preprocess in linear time, and a query is a sub-family $\mathcal{G} \subseteq \mathcal{F}$ and a set of outliers O. Let $N = |\bigcup \mathcal{G} \setminus O|$ and $g = |\mathcal{G}|$. The above approach samples uniformly at random an element $x \in \bigcup \mathcal{G} \setminus O$. The query time is $O((g + \mathrm{d}_{\mathcal{G}}(O)/(N+1)) \log g)$ in expectation, and $O((g + \mathrm{d}_{\mathcal{G}}(O)) \log g)$ in the worst case.*

Proof. For each $o \in O$ and each $1 \leq i \leq g$, define the random variable $X_{o,i}$ that is 1 if o is present in the ith collection of $\mathcal{G}$ and has a rank smaller than all elements $\bigcup \mathcal{G} \setminus O$. By the initial random permutation, the probability that an outlier $o \in O$ has a smaller rank than the N elements in $\bigcup \mathcal{G} \setminus O$ is exactly $1/(N+1)$. Let R be the number of rounds carried out by the query algorithm. By linearity of expectation, we get:

$$\mathbb{E}[R] \leq g + \mathbb{E}\left[\sum_{o \in O} \sum_{i=1}^{g} X_{o,i}\right] = g + \frac{\mathrm{d}_{\mathcal{G}}(O)}{N+1}.$$

The lemma follows because each round takes time $O(\log g)$ for the priority queue operations. Since an outlier cannot be sampled twice, the algorithm stops after $\mathrm{d}_{\mathcal{G}}(O)$ rounds in the worst case, and the query time is $O((g + \mathrm{d}_{\mathcal{G}}(O)) \log g)$ in this case. □

Similarly to Lemma 4, we can extend the above data structure to support output independence if the same query $\mathcal{G}$ is repeated several times. It suffices to repeat the process until a point in $\bigcup \mathcal{G} \setminus O$ is found, and to apply the swap to the returned point. Note that to efficiently perform swaps each set in $\mathcal{F}$ should store points in a priority queue with ranks as keys. We get the following lemma.

Lemma 12. *The input is a family of sets $\mathcal{F}$ that one can preprocess in linear time. A query is a sub-family $\mathcal{G} \subseteq \mathcal{F}$, and a set of outliers O. Let $n = |\bigcup \mathcal{F}|$, $N = |\bigcup \mathcal{G} \setminus O|$, $g = |\mathcal{G}|$ and $\delta = \max_{x \in \bigcup \mathcal{F}} |\{A \in \mathcal{F} | x \in A\}|$. Assume to repeat k times the above query procedure on the sub-collection $\mathcal{G}$, and let OUT_i denote the output of the ith iteration with $1 \leq i \leq k$. Then, we have that $OUT_i \in \bigcup \mathcal{G} \setminus O$ for any $1 \leq i \leq k$ and, for any $x \in \bigcup \mathcal{G} \setminus O$,* $\Pr[OUT_1 = x] =$

$1/N$ *and* $\Pr[OUT_i = x | OUT_{i-1} = x_{i-1}, \ldots OUT_1 = x_1] = \Pr[OUT_1 = x] = 1/N$ *for* $i > 1$. *The expected query time is* $O\left(d_{\mathcal{G}}(O) \log n + (g+\delta)\log n\right)$ *time.*

Proof. Initially, we need $O(g \log n)$ time to find the point in $\bigcup \mathcal{G}$ with smaller rank. Then we need to repeat the procedure $\mathrm{d}_{\mathcal{G}}(O)/(N+1)$ times in expectation since the probability that an outlier $o \in O$ has a smaller rank than the N elements in $\bigcup \mathcal{G} \setminus O$ is $1/(N+1)$. Since each repetition costs $O(\log n)$ and the final swap takes $O(\delta \log n)$ time, the expected running time follows. The probabilities follows from Lemma 4. □

4.2 Almost uniform sampling with outliers

Definition 6. *For a set* T, *and a parameter* $\varepsilon \in [0,1)$, *a sampling algorithm that outputs a sample* $x \in T$ *generates* ε**-uniform distribution**, *if for any* $y \in T$, *we have that* $\frac{1}{(1+\varepsilon)|T|} \leq \mathbb{P}[x = y] \leq \frac{1+\varepsilon}{|T|}$.

Lemma 13. *The input is a family of sets* $\mathcal{F}$ *that one can preprocess in linear time. A query is a sub-family* $\mathcal{G} \subseteq \mathcal{F}$, *a set of outliers* O, *a parameter* m_O, *and a parameter* $\varepsilon \in (0,1)$. *One can either*

(A) Sample an element $x \in \bigcup \mathcal{G} \setminus O$ *with* ε*-uniform distribution.*

(B) Alternatively, report that $\mathrm{d}_{\mathcal{G}}(O) > m_O$.

The expected query time is $O\big(m_O + g\log(g/\varepsilon)\big)$, *and the query succeeds, with probability* $\geq 1 - 1/g^{O(1)}$, *where* $g = |\mathcal{G}|$.

Proof. The main modification of the algorithm of Lemma 9 is that whenever we encounter an outlier (the assumption is that one can check if an element is an outlier in constant time), then we delete it from the set A where it was discovered. If we implement sets as arrays, this can be done by moving an outlier object to the end of the active prefix of the array, and decreasing the count of the active array. We also need to decrease the (active) size of the set. If the algorithm encounters more than m_O outliers then it stops and reports that the number of outliers is too large.

Otherwise, the algorithm continues as before. The only difference is that once the query process is done, the active count (i.e., size) of each set needs to be restored to its original size, as is the size of the set. This clearly can be done in time proportional to the query time. □

4.3 Uniform sampling with outliers

We run a variant of the original algorithm from Section 3.5. In the same way as before, we use the count distinct sketches to obtain an upper bound $\widehat{s}_{\mathcal{G}}$ on the number of distinct elements in $\mathcal{G}$. Because of the presence of outliers, this bound will not necessarily be close to N, but could be much larger. Thus, we run the algorithm at most $\log n$ rounds to find a suitable value of k. In round i, we use the value $k_i = 2\widehat{s}_{\mathcal{G}}/2^i$. Moreover, a single round is iterated for $\Sigma = \Theta(\log^2 n)$ steps. If $k < 2$, we report that $\bigcup \mathcal{G} \setminus O$ is empty. The precise algorithm is presented in the following. As before, it takes an integer parameter m_O controlling the number of outliers.

(A) Merge all count distinct sketches of the g sets in $\mathcal{G}$, and compute a 1/2-approximation $\widehat{s}_{\mathcal{G}}$ of $s_{\mathcal{G}} = |\bigcup \mathcal{G}|$, such that $s_{\mathcal{G}}/2 \leq \widehat{s}_{\mathcal{G}} \leq 1.5 s_{\mathcal{G}}$.

(B) Set k to the smallest power of two larger than or equal to $2\widehat{s}_{\mathcal{G}}$; let $\lambda = \Theta(\log n)$, $\sigma_{\text{fail}} = 0$ and $\Sigma = \Theta\left(\log^2 n\right)$.

(C) Repeat the following steps until successful or $k < 2$:

  (I) Assume the input sequence Λ to be split into k segments Λ_i of size n/k, where Λ_i contains the points in $\bigcup \mathcal{F} \setminus O$ with ranks in $[i \cdot n/k, (i+1) \cdot n/k)$. Denote with λ_i the size of Λ_i.

  (II) Select uniformly at random an integer h in $\{0, \ldots, k-1\}$ (i.e., select a segment Λ_h);

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

(IV) Compute $\lambda_{\mathcal{G},h}$ and count the number of outliers inspected on the way. If there are more than m_O outliers, report that $\mathrm{d}_{\mathcal{G}}(O) > m_O$. Otherwise, with probability $\lambda_{\mathcal{G},h}/\lambda$, declare success.

(D) If the previous loop ended with success, return an element uniformly sampled among the elements in $\bigcup \mathcal{G} \setminus O$ in Λ_h, otherwise return $\perp$.

Lemma 14. *The input is a family of sets $\mathcal{F}$ that one can preprocess in linear time. A query is a sub-family $\mathcal{G} \subseteq \mathcal{F}$, a set of outliers O, and a parameter m_O. With high probability, one can either:*

(A) Sample a uniform element $x \in \bigcup \mathcal{G} \setminus O$, or

(B) Report that $\mathrm{d}_{\mathcal{G}}(O) > m_O$.

The expected time is $O((g + m_O) \log^4 n)$, where $N = |\bigcup \mathcal{G} \setminus O|$, $g = |\mathcal{G}|$, and $n = ||\mathcal{F}||$.

The proof will follow along the lines of the proof of Lemma 10. We provide a self-contained version for completeness and to highlight the challenges of introducing outliers.

Proof. We start by bounding the initial failure probability of the data structure. By a union bound, we have that the following two events hold simultaneously with probability at least $1 - 1/n^2$:

(1) Count distinct sketches provide a $1/2$-approximation of $|\bigcup \mathcal{G}|$. By setting $\delta = 1/(4n^2)$ in the count distinct sketch construction (see Section 2), the approximation guarantee holds with probability at least $1 - 1/(4n^2)$.

(2) When $k = 2^{\lceil \log N \rceil}$, every segment of size n/k contains no more than $\lambda = \Theta(\log N)$ points from $\bigcup \mathcal{G} \setminus O$. As points are initially randomly permuted, the claim holds with probability at least $1 - 1/(4n^2)$ by suitably setting the constant in $\lambda = \Theta(\log n)$.

From now on assume that these events are true.

We will first discuss the additional failure event: $N \geq 1$, but the algorithm reports $\perp$. The probability of this event is upper bounded by the probability p' that no element is returned in the Σ iterations where $k = 2^{\lceil \log N \rceil}$ (the actual probability is even lower, since an element can be returned in an iteration where $k > 2^{\lceil \log N \rceil}$). By suitably setting constants in $\lambda = \Theta(\log n)$ and $\Sigma = \Theta\left(\log^2 n\right)$, we get:

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

By a union bound, with probability at least $1 - 1/n^2$ none of these three events are true. To show that the returned element is uniformly sampled in $\bigcup \mathcal{G} \setminus O$, recall that each element in $\bigcup \mathcal{G} \setminus O$ has the same probability of $1/(k\lambda)$ of being output.

For the running time, first focus on the round where $k = 2^{\lceil \log N \rceil}$. In this round, we carry out $\Theta(\log^2 n)$ iterations. In each iteration, we extract the points with rank in $[hn/k, (h+1)n/k]$ from each of the g sets, counting all outlier points that we retrieve on the way. For each set, we expect to find $N/k = O(1)$ points in $\bigcup \mathcal{G} \setminus O$. If we retrieve more than m_O outliers, we report that $\mathrm{d}_{\mathcal{G}}(O) > m_O$. Reporting points with a given rank costs $O(\log n + o)$ in each bucket (where o is the output size). Thus, one iteration is expected to take time $O((g + m_O) \log n)$. The expected running time of all $\Sigma = \Theta(\log^2 n)$ iterations is bounded by $O((g + m_O) \log^3 n)$. Observe that for all the rounds carried out before, k is only larger and thus the segments are smaller. This means that we may multiply our upper bound with $\log n$, which completes the proof. □

5 IN THE SEARCH FOR A FAIR NEAR NEIGHBOR

In this section, we employ the data structures developed in the previous sections to show the results on fair near neighbor search listed in Section 1.2.

First, let us briefly give some preliminaries on LSH. We refer the reader to [37] for further details. Throughout the section, we assume that our metric space $(\mathcal{X}, \mathcal{D})$ admits an LSH data structure.

5.1 Background on LSH

Locality Sensitive Hashing (LSH) is a common tool for solving the ANN problem and was introduced in [45].

Definition 7. *A distribution $\mathcal{H}$ over maps $h\colon \mathcal{X} \to U$, 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 $\mathcal{D}$ denote the data structure constructed by LSH, and let c denote the approximation parameter of LSH. Each $\mathcal{D}$ consists of

$$\mathsf{L} = n^{\rho} \tag{1}$$

hash functions $\ell_1, \ldots, \ell_{\mathsf{L}}$ randomly and uniformly selected from $\mathcal{H}$. The performance of the LSH data structure is determined by this parameter L, and one tries to minimize the value of ρ (and thus L) by picking the "right" hash functions. The data structure $\mathcal{D}$ contains L hash tables $H_1, \ldots H_{\mathsf{L}}$: 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

$$H_i(\mathbf{p}) = \{\mathbf{p} : \mathbf{p} \in S, \ell_i(\mathbf{p}) = \ell_i(\mathbf{q})\} \tag{2}$$

of points in S with the same hash value; then, compute the distance $\mathcal{D}(\mathbf{q}, \mathbf{p})$ for each point $\mathbf{p} \in H_i(\mathbf{q})$. 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 $3\mathsf{L}$ far points [45].

Definition 8. *For a query point $\mathbf{q} \in S$, an* **outlier** *for an LSH data structure is a point $\mathbf{p} \in S \setminus B_S(\mathbf{q}, cr)$, such that $\mathbf{p} \in S_{\mathbf{q}} = \bigcup_i H_i(\mathbf{q})$. An LSH data-structure is good if there are not too many outliers for the query point. Specifically, the LSH data-structure is* **useful** *if the number of outliers for $\mathbf{q}$ is at most $3L$.*

We summarize the guarantees in the following two lemmas [37].

Lemma 15. *Consider an LSH data structure as above. For a given query point $\mathbf{q}$, and a point $\mathbf{p} \in B_S(\mathbf{q}, r)$, with probability $\geq 1 - 1/e - 1/3$, we have that $\mathbf{p} \in S_{\mathbf{q}}$ and this data structure is useful (i.e., $S_{\mathbf{q}}$ contains at most $3L$ outliers for $\mathbf{q}$).*

This is not quite strong enough for our purposes. We build an LSH data-structure that uses $O(\mathsf{L} \log n)$ hash functions (instead of L). The probability of the query point to collide with a point of $B_S(\mathbf{q}, r)$ is $\geq 1 - 1/n^{O(1)}$ in the new LSH

structure, while the expected number of outliers grows linearly with the number of hash functions. We thus have the following.

Lemma 16. *Consider an LSH data structure as above, using $O(\mathsf{L}\log n)$ hash functions. For a given query point $\mathbf{q}$, we have that (i) $B_S(\mathbf{q}, r) \subseteq S_{\mathbf{q}}$ with probability ≥ 0.9, (ii) $S_{\mathbf{q}}$ contains in expectation $O(\mathsf{L}\log n)$ outliers for $\mathbf{q}$, and (iii) $S_{\mathbf{q}}$ contains in expectation $O\left((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r))\log n\right)$ points with distance larger than r.*

The main technical issue is that before scanning $S_{\mathbf{q}}$, during the query process, one can not tell whether the set $S_{\mathbf{q}}$ is large because there are many points in $B_S(\mathbf{q}, r)$, or because the data structure contains many outliers. However, the algorithm can detect if the LSH data structure is useless if the query process encounters too many outliers. By using $\Theta(\log n)$ independent LSH data structures, we can guarantee with high probability that there is a data structure that does not have too many outliers (here the answer returned is not sampled, so there is no fairness guarantee).

Lemma 17. *Let the query point be $\mathbf{q}$. Let $\mathcal{D}_1, \ldots, \mathcal{D}_t$ be $t = \Theta(\log n)$ independent LSH data structures of Lemma 16. Then, with high probability, for a constant fraction of indices $j \in [\![t]\!] = \{1, \ldots, t\}$, we have that (i) $B(\mathbf{q}, r) \subseteq S_{j,\mathbf{q}} = \bigcup_i H_i^j(q)$ and (ii) the number of outliers is $\left|S_{j,\mathbf{q}} \setminus B_S(\mathbf{q}, cr)\right| = O(\mathsf{L}\log n)$, where $S_{j,\mathbf{q}}$ is the set of all points in buckets that collide with $\mathbf{q}$. The space used is $\mathcal{S}(n, c) = O(n\mathsf{L}\log^2 n)$, and the expected query time is $\mathcal{Q}(n, c) = O(\mathsf{L}\log n)$.*

Proof. For a random $i \in [\![t]\!]$, the data structure $\mathcal{D}_i$ has the desired properties with probability ≥ 0.9, by Lemma 16. By Chernoff's inequality, as $t = \Theta(\log n)$, at least a constant fraction of these data-structures have the desired property.

As for the query time, given a query, the data structure starts with $i = 1$. In the ith iteration, the algorithm uses from $\mathcal{D}_i$, and computes the $O(\mathsf{L}\log n)$ lists that contains the elements of $S_{j,\mathbf{q}}$. The algorithm scans these lists – if it encounters more than $O(\mathsf{L}\log n)$ outliers, it increases i and move on to the next data-structure. As soon as the algorithm encounters a near point, in these lists, it stops and returns it. □

Remark 5. *In the above, we ignored the dependency on the dimension d. In particular, the $O(\cdot)$ in hides a factor of d.*

In the following, we present data structures that solve the problems defined in Section 1.1. For most of the problem variants (all except Lemma 19) that require to return an r-near neighbor (but not a cr-near neighbor), we require that the LSH data structure behaves well on points in $B(\mathbf{q}, cr) \setminus B(\mathbf{q}, r)$. Note that Definition 7 does not specify the behavior of the LSH on such points. In particular, we assume that the collision probability function of the LSH is monotonically decreasing and that points at distance at least r collide with probability $O(p_1^k)$. Such an LSH has the property that the query is expected to collide with $O((n(\mathbf{q}, cr) - n(\mathbf{q}, r))\log n)$ points within distance r to cr. We note that most LSH data structures have this property naturally by providing a closed formula for the CPF of the LSH.

5.2 Exact Neighborhood with Dependence

We will next present an algorithm that solves Fair NN with dependence as defined in Definition 1.

Lemma 18. *Given a set S of n points and a parameter r, we can preprocess it such that given a query q, one can report a point $p \in S$ with probability $1/n(\mathbf{q}, r)$ (points returned by subsequent queries might not be independent). The algorithm uses space $O(n\mathsf{L}\log n)$ and has expected query time $O\left(\left(\mathsf{L} + \frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\right)\log^2 n\right)$.*

Proof. Let $\mathcal{D}$ be a data structure constructed of Lemma 16. Let $\mathcal{F}$ be the set of all buckets in the data structure. For a query point $\mathbf{q}$, consider the family $\mathcal{G}$ of all buckets containing the query, and thus $g = |\mathcal{G}| = O(\mathsf{L}\log n)$. We consider as outliers $\mathcal{O}$ the set of points that are farther than r from q. By Lemma 11, we have that a query requires

$O((\mathsf{L} + \mathrm{d}_{\mathcal{G}}(O)/n(\mathbf{q}, r)) \log \mathsf{L})$ expected time (the expectation is over the random choices of the sampling data structure). The expected number of outliers is $O\left((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r)) \log n\right)$ (the expectation is over the random choices of LSH), since by Lemma 16 there are at most $O(\mathsf{L} \log n)$ points at distance at least cr and $O\left((n(\mathbf{q}, cr) - n(\mathbf{q}, r)) \log n\right)$ points at distance between r and cr. □

The above results will always return the same point if we repeat the same query. By using the technique in Lemma 12, we get the following result if the same query is repeated:

Theorem 1. *Given a set S of n points and a parameter $r > 0$, we can preprocess it such that by repeating k times a query $\mathbf{q}$, we have with high probability $1 - 1/n$:*

(1) $\mathbf{p} \in B_S(\mathbf{q}, r)$ is returned as near neighbor of $\mathbf{q}$ with probability $1/n(\mathbf{q}, r)$.

(2) $\Pr\left[OUT_i = \mathbf{p} | OUT_{i-1} = \mathbf{p}_{i-1}, \ldots OUT_1 = \mathbf{p}_1\right] = 1/n(\mathbf{q}, r)$ for each $1 < i \leq n$.

where OUT_i denotes the output of the ith iteration with $1 \leq i \leq k$. The data structure requires $O\left(n\mathsf{L} \log n\right)$ space and the expected query time is $O\left((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r)) \log^2 n\right)$.

Proof. The claim follows by using the data structure in Lemma 12, where each query requires expected time $O\left(d_{\mathcal{G}}(O) \log n + (g + \delta) \log n\right)$, where the $d_{\mathcal{G}}(O) = O\left((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r)) \log n\right)$, n is the total number of points, and $\delta = g = \mathsf{L}$. □

5.3 Approximately Fair ANN

In this section we will first provide an algorithm that solves Approximately Fair ANN (Definition 4), and then modify it to resample after sampling a cr-near but not r-near point to solve Approximately Fair NN (Definition 3).

Theorem 2. *Given a set S of n points, and a parameter r, we can preprocess it in time and space $\mathcal{S}(n, c) = O(n\mathsf{L} \log^2 n)$, see Eq. (1). Here, given a query $\mathbf{q}$, one can report a point $\mathbf{p} \in T$, with ε-uniform distribution (see Definition 6), where T is a set that depends on the query point $\mathbf{q}$, and $B_S(\mathbf{q}, r) \subseteq T \subseteq B_S(\mathbf{q}, cr)$. The expected query time is $O(\mathsf{L} \log n \log(n/\varepsilon))$. The query time is $O(\mathsf{L} \log^2 n \log(n/\varepsilon))$ with high probability. The above guarantees hold with high probability.*

Proof. We construct the data-structures $\mathcal{D}_1, \ldots, \mathcal{D}_t$ of Lemma 17, with $t = O(\log n)$. Here, all the values that are mapped to a single bucket by a specific hash function are stored in its own set data-structure (i.e., an array), that supports membership queries in $O(1)$ time (by using hashing).

Starting with $i = 1$, the algorithm looks up the $g = O(\mathsf{L} \log n)$ buckets of the points colliding with the query point in the LSH data structure $\mathcal{D}_i$, and let $\mathcal{G}$ be this set of buckets. Let T be the union of the points stored in the buckets of $\mathcal{G}$. We have with constant probability that $||\mathcal{G} \setminus B_S(\mathbf{q}, cr)|| \leq m_O$, where $m_O = O(\mathsf{L} \log n)$. Thus, we deploy the algorithm of Lemma 13 to the sets of $\mathcal{G}$. With high probability $B_S(\mathbf{q}, r) \subseteq T$, which implies that with constant probability (close to 0.9), we sample the desired point, with the ε-uniform guarantee. If the algorithm of Lemma 13 returns that there were too many outliers, the algorithm increases i, and try again with the next data structure, till success. In expectation, the algorithm would only need to increase i a constant number of times till success, implying the expected bound on the query time. With high probability, the number of rounds till success is $O(\log n)$.

Since $g \approx \mathsf{L} = n^{\Omega(1)}$, all the high probability bounds here hold with probability $\geq 1 - 1/n^{O(1)}$. □

Remark 6. *The best value of L that can be used depends on the underlying metric. For the L_1 distance, the runtime of our algorithm is $\widetilde{O}(n^{1/c + o(1)})$ and for the L_2 distance, the runtime of our algorithm is $\widetilde{O}(n^{1/c^2 + o(1)})$. These match the runtime of the standard LSH-based near neighbor algorithms up to polylog factors.*

Exact neighborhood. One can even sample ε-uniformly from the r-near-neighbors of the query point. Two such data structures are given in the following Lemma 19 and Remark 7. The query time guarantee is somewhat worse.

Lemma 19. *Given a set S of n points, and a parameter r, we can preprocess it in time and space $\mathcal{S}(n,c) = O(n\mathsf{L}\log^2 n)$, see Eq. (1). Here, given a query $\mathbf{q}$, one can report a point $\mathbf{p} \in B_S(\mathbf{q}, r)$, with ε-uniform distribution (see Definition 6). The expected query time is $O\left(\mathsf{L}\frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\log n \log\frac{n}{\varepsilon}\right)$. The query time is $O\left(\mathsf{L}\frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\log^2 n \log\frac{n}{\varepsilon}\right)$ with high probability.*

Proof. Construct and use the data structure of Theorem 2. Given a query point, the algorithm repeatedly gets a neighbor $\mathbf{p}_i$, by calling the query procedure. This query has a ε-uniform distribution on some set T_i such that $B_S(\mathbf{q}, r) \subseteq T_i \subseteq B_S(q, cr)$. As such, if the distance of $\mathbf{p}_i$ from $\mathbf{q}$ is at most r, then the algorithm returns it as the desired answer. Otherwise, the algorithm increases i, and continues to the next round.

The probability that the algorithm succeeds in a round is $\rho = n(\mathbf{q}, r)/n(\mathbf{q}, cr)$, and as such the expected number of rounds is $1/\rho$, which readily implies the result. □

Remark 7. *We remark that the properties in Lemma 19 are independent of the behavior of the LSH with regard to points in $B(\mathbf{q}, cr) \setminus B(\mathbf{q}, r)$. If the LSH behaves well on these points as discussed in Section 5.1, we can build the same data structure as in Theorem 2 but regard all points outside of $B(\mathbf{q}, r)$ as outliers. This results in an expected query time of $O\big((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r))\log n \log\frac{n}{\varepsilon}\big)$. With high probability the query time is $O\Big((\mathsf{L} + n(\mathbf{q}, cr) - n(\mathbf{q}, r))\log^2 n \log\frac{n}{\varepsilon}\Big)$.*

5.4 Exact Neighborhood (Fair NN)

To solve Fair NN (Definition 2), we use the algorithm described in Section 4.3 with all points at distance more than r from the query marked as outliers.

Theorem 3. *Given a set S of n points and a parameter r, we can preprocess it such that given a query q, one can report a point $p \in B_S(\mathbf{q}, r)$ with probability $1/n(\mathbf{q}, r)$. The algorithm uses space $O(n\mathsf{L}\log n)$ and has expected query time $O\left(\left(\mathsf{L} + \frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\right)\log^5 n\right)$.*

Proof. For $t = O(\log n)$, let $\mathcal{D}_1, \ldots, \mathcal{D}_t$ be data structures constructed by LSH. Let $\mathcal{F}$ be the set of all buckets in all data structures. For a query point $\mathbf{q}$, consider the family $\mathcal{G}$ of all buckets containing the query, and thus $g = |\mathcal{G}| = O(\mathsf{L}\log n)$. Moreover, we let O to be the set of outliers, i.e., the points that are farther than r from $\mathbf{q}$. We proceed to bound the number of outliers that we expect to see in Step (IV) of the algorithm described in Section 4.3.

By Lemma 16, we expect at most $O(g)$ points at distance at least cr. Moreover, we set up the LSH such that the probability of colliding with a cr-near point is at most $1/g$ in each bucket. By the random ranks that we assign to each point, we expect to see $n(\mathbf{q}, cr)/k$ cr-near points in segment h. Since, $k = \Theta(n(\mathbf{q}, r))$, we expect to retrieve $O\left(g\frac{n(\mathbf{q},cr)}{gn(\mathbf{q},r)}\right) = O\left(\frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\right)$ cr-near points in one iteration. With the same line of reasoning as in Lemma 14, we can bound the expected running time of the algorithm by $O\left(\left(\mathsf{L} + \frac{n(\mathbf{q},cr)}{n(\mathbf{q},r)}\right)\log^5 n\right)$. □

Remark 8. *With the same argument as above, we can solve Fair NN with an approximate neighborhood (in which we are allowed to return points within distance cr) in expected time $O\big(\mathsf{L}\log^5 n\big)$.*

We now turn our focus on designing an algorithm with a high probability bound on its running time. We note that we cannot use the algorithm from above directly because with constant probability more than $n(\mathbf{q}, cr) + \mathsf{L}$ points are outliers. The following lemma, similar in nature to Lemma 17, proves that by considering independent repetitions, we can guarantee that there exists an LSH data structure with a small number of non-near neighbors, i.e., points at distance larger than r, colliding with the query.

Lemma 20. *Let the query point be* $\mathbf{q}$. *Let* $\mathcal{D}_1, \ldots, \mathcal{D}_t$ *be* $t = \Theta(\log n)$ *independent LSH data structures, each consisting of* $\Theta(\mathsf{L} \log n)$ *hash functions. Then, with high probability, there exists* $j \in \{1, \ldots, t\}$ *such that (i)* $B(\mathbf{q}, r) \subseteq S_{j,\mathbf{q}} = \bigcup_i H_i^j(\mathbf{q})$ *and (ii) the number of non-near points colliding with the query (with duplicates) in all* $\Theta(\mathsf{L} \log n)$ *repetitions is* $O((n(\mathbf{q}, cr) + \mathsf{L}) \log n)$.

Proof. Property (i) can be achieved for all t data structures simultaneously by setting the constant in $\Theta(\mathsf{L} \log n)$ such that each near point has a probability of at least $1 - 1/(n^2 \log n)$ to collide with the query. A union bound over the $\log n$ independent data structure and the $n(\mathbf{q}, r) \leq n$ near points then results in (i). To see that (ii) is true, observe that in a single data structure, we expect not more than $\mathsf{L} \log n$ far points and $n(\mathbf{q}, cr) \log n$ cr-near points to collide with the query. (Recall that each cr-near point is expected to collide at most once with the query.) By Markov's inequality, with probability at most $1/3$ we see more than $3(\mathsf{L} + n(\mathbf{q}, cr)) \log n$ such points. Using $t = \Theta(\log n)$ independent data structures, with high probability, there exists a data structure that has at most $3(\mathsf{L} + n(\mathbf{q}, cr)) \log n$ points at distance larger than r colliding with the query. □

Lemma 21. *Given a set* S *of* n *points and a parameter* r, *we can preprocess it such that given a query* q, *one can report a point* $p \in S$ *with probability* $1/n(\mathbf{q}, r)$. *The algorithm uses space* $O(n\mathsf{L} \log^2 n)$ *and has query time* $O\big((\mathsf{L} + n(\mathbf{q}, cr)) \log^6 n\big)$ *with high probability.*

Proof. We build $\Theta(\log n)$ independent LSH data structures, each containing $\Theta(\mathsf{L} \log n)$ repetitions. For a query point $\mathbf{q}$, consider the family $\mathcal{G}_i$ of all buckets containing the query in data structure $\mathcal{D}_i$. By Lemma 20, in each data structure, all points in $B(\mathbf{q}, r)$ collide with the query with high probability. We assume this is true for all data structures. We change the algorithm presented in Section 4.3 as follows:

(I) We start the algorithm by setting $k = n$.

(II) Before carrying out Step (IV), we calculate the work necessary to compute $\lambda_{\mathcal{G}_i,h}$ in each data structure $i \in \{1, \ldots, t\}$. This is done by carrying out range queries for the ranks in segment h in each bucket to obtain the number of points that we have to inspect. We use the data structure with minimal work to carry out Step (IV).

Since all points in $B(\mathbf{q}, r)$ collide with the query in each data structure, the correctness of the algorithm follows in the same way as in the proof of Lemma 14.

To bound the running time, we concentrate on the iteration in which $k = 2^{\lceil \log n(\mathbf{q},r) \rceil}$, e.g., $k = \Theta(n(\mathbf{q}, r))$. As in the proof of Lemma 14, we have to focus on the time it takes to compute $\lambda_{\mathcal{G},h}$, the number of near points in all $\Theta(\mathsf{L} \log n)$ repetitions in segment h of the chosen data structure. By Lemma 17, with high probability there exists a partition in which the number of outlier points is $O((n(\mathbf{q}, cr) + \mathsf{L}) \log n)$. Assume from here on that it exists. We picked the data structure with the least amount of work in segment h. The data structure with $O((n(\mathbf{q}, cr) + \mathsf{L}) \log n)$ outlier points was a possible candidate, and with high probability there were $O((n(\mathbf{q}, cr) + \mathsf{L}) \log^2 n)$ collisions in this data structure for the chosen segment. (There could be $\mathsf{L} \cdot n(\mathbf{q}, cr) \log n$ colliding near points, but with high probability we see at most a fraction of $\Theta(\log n / n(\mathbf{q}, r))$ of those in segment h because of the random ranks. On the other hand, we cannot say anything about how many *distinct* non-near points collide with the query.) In total, we spend time $O((n(\mathbf{q}, cr) + \mathsf{L}) \log^2 n)$ to compute $\lambda_{\mathcal{G},h}$ in the chosen data structure, and time $O(\mathsf{L} \log^3 n)$ to find out which data structure has the least amount of work. This means that we can bound the running time of a single iteration by $O((n(\mathbf{q}, cr) + \mathsf{L}) \log^3 n)$ with high probability. The lemma statement follows by observing that the segment size k is adapted at most $\log n$ times, and for each k we carry out $O(\log^2 n)$ rounds. □

6 FAIR NN USING NEARLY-LINEAR SPACE

In this section, we describe a data structure that solves the Fair NN problem (Definition 2) using the exact degree computation algorithm from Section 3.2. The bottleneck of that algorithm was the computation of the degree of an element which takes time $O(g)$. However, if we are to use a data structure that has at most $O(\log n)$ repetitions, this bottleneck will be alleviated.

The following approach uses only the basic filtering approach described in [26], but no other data structures as was necessary for solving uniform sampling with exact probabilities in the previous sections. It can be seen as a particular instantiation of the more general space-time trade-off data structures that were described in [11, 26]. It can also be seen as a variant of the empirical approach discussed in [33] with a theoretical analysis. Compared to [11, 26], it provides much easier parameterization and a simpler way to make it efficient.

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 $n(\mathbf{q}, \alpha) = |B_S(\mathbf{q}, \alpha)|$. We define the α-NNIS problem analogously to r-NNIS with respect to inner product similarity.

We start with a succinct description of the linear space near-neighbor data structure. Next, we will show how to use this data structure to solve the Fair NN problem under inner product similarity.

6.1 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.[3] 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 4. *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 the running time statements found in [26] 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 [29]. The analysis is presented in Appendix A.

[3] As tradition in the literature, we assume that a memory word suffices for reaching the desired precision. See [21] for a discussion.

6.2 Solving α-NNIS

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

Query. We run the rejection sampling approach from Section 3.2 on the data structure described above. For query $\mathbf{q}$, evaluate all $tm^{1/t}$ filters in each individual $\mathcal{D}_\ell$. Let $\mathcal{G}$ be the set of buckets $(i_{\ell,1}, \ldots, i_{\ell,t})$ above the query threshold, for each $\ell \in [L]$, and set $m = |\mathcal{G}|$. First, check for the existence of a near neighbor by running the standard query algorithm described above on every individual data structure. This takes expected time $n^{\rho+o(1)} + O\left(\frac{n(\mathbf{q},\beta)}{n(\mathbf{q},\alpha)+1} \log n\right)$, assuming points in a bucket appear in random order. If no near neighbor exists, output $\perp$ and return. Otherwise, the algorithm performs the following steps until success is declared:

(I) Picks one set from $\mathcal{G}$ with probabilities proportional to their sizes. That is, a set $A \in \mathcal{G}$ is picked with probability $|A|/m$.
(II) It picks a point $\mathbf{p} \in A$ uniformly at random.
(III) Computes the degree $\mathsf{d} = \mathsf{d}_{\mathcal{G}}(\mathbf{p})$.
(IV) If $\mathbf{p}$ is a far point, remove $\mathbf{p}$ from the bucket update the cardinality of A. Continue to the next iteration.
(V) If $\mathbf{p}$ is a near point, outputs $\mathbf{p}$ and stop with probability $1/\mathsf{d}$. Otherwise, continue to the next iteration.

After a point $\mathbf{p}$ has been reported, move all far points removed during the process into their bucket again. As discussed in Section 2, we assume that removing and inserting a point takes constant time in expectation.

Theorem 5. *Let $S \subseteq \mathcal{X}$ with $|S| = n$ and $-1 < \beta < \alpha < 1$. The data structure described above solves the α-NNIS problem in space $O(n \log n)$ and expected time $n^{\rho+o(1)} + O((n(\mathbf{q},\beta)/(n(\mathbf{q},\alpha)+1)) \log^2 n)$.*

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 $B_S(\mathbf{q},\alpha)$. The output is uniform by the arguments given in the proof of the original variant in Lemma 5.

We proceed to prove the running time statement in the case that there exists a point in $B_S(\mathbf{q},\alpha)$. (See the discussion above for the case $n(\mathbf{q},\alpha) = 0$.) 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 A for details). We did not account for repeatedly carrying out steps A–C yet for rounds in which we choose a non-far point. To this end, we next find a lower bound on the probability that the algorithm declares success in a single such round. First, observe that there are $O(n(\mathbf{q},\beta) \log n)$ non-far points in the T buckets (with repetitions). Fix a point $\mathbf{p} \in B_S(\mathbf{q},\alpha)$. With probability $\Omega(c_{\mathbf{p}}/(n(\mathbf{q},\beta) \log n))$, $\mathbf{p}$ is chosen in step B. Thus, with probability $\Omega(1/(n(\mathbf{q},\beta) \log n))$, 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 $\Omega(n(\mathbf{q},\alpha)/(n(\mathbf{q},\beta) \log n))$. This means that we expect $O(n(\mathbf{q},\beta) \log n/n(\mathbf{q},\alpha))$ 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 $O((n(\mathbf{q},\beta)/n(\mathbf{q},\alpha)) \log^2 n)$ for these iterations, which concludes the proof. □

7 EXPERIMENTAL EVALUATION

This section presents a principled experimental evaluation that sheds light on the general fairness implications of our problem definitions. The aim of this evaluation is to complement the theoretical study with a case study focusing on the fairness implications of solving variants of the near-neighbor problem. The evaluation contains both

a validation of the (un)fairness of traditional approaches in a recommendation setting on real-world datasets, an empirical study of unfairness in approximate approaches, an evaluation of the average query time of different methods in this paper, and a short discussion of the additional cost introduced by solving the exact neighborhood problem. We implemented all methods and additional tools in Python 3, and also re-implemented some special cases in C ++ for running time observations. The code, raw result files, and the experimental log containing more details are available at https://github.com/alfahaf/fair-nn. Moreover, the repository contains all scripts and a Docker build script necessary to reproduce and verify the plots presented here.

Datasets and Query Selection. We run our experiments on five different datasets which are either standard benchmarks in a recommendation system setting or in a nearest neighbor search context (see [12]):

(I) MovieLens: a dataset mapping 2112 users to 65536 unique movies. We obtain a set representation by mapping each user to movies rated 4 or higher by the user, resulting in an average set size of 178.1 ($\sigma = 187.5$).
(II) Last.FM: a dataset with 1892 users and 19739 unique artists. We obtain a set representation by mapping each user to their top-20 artists, resulting in an average set size of 19.8 ($\sigma = 1.78$).
(III) MNIST: a random subset of 10K points in the MNIST training data set [53]. The full data set contains 60K images of hand-written digits, where each image is of size 28 by 28. Therefore, each of our points lie in a 784 dimensional Euclidean space and each coordinate is in $[0, 255]$.
(IV) SIFT: We take a random subset of 10K vectors of the SIFT1M image descriptors that contains 1M 128-dimensional points.
(V) GloVe: Finally, we take a random subset of 10K words from the GloVe data set [62]. GloVe is a data set of 1.2M word embeddings in 100-dimensional space.

All datasets are processed automatically by our experimental framework. For the first two datasets, we measure the similarity of two user sets $\mathbf{X}$ and $\mathbf{Y}$ by their Jaccard similarity $J(\mathbf{X}, \mathbf{Y}) = |\mathbf{X} \cap \mathbf{Y}|/|\mathbf{X} \cup \mathbf{Y}|$. For the latter three datasets, we measure distance by using Euclidean distance/L_2 norm.

For each dataset, we pick a set of "interesting queries" to guarantee that the output size is not too small. More specifically, we consider all data points as potential queries for which the 40th nearest neighbor is above a certain distance threshold. Among those points, we choose 50 data points at random as queries and remove them from the data set.

Algorithms. Two different distance measures made it necessary to implement two different LSH families. For Jaccard similarity, we implemented LSH using standard MinHash [20] and applying the 1-bit scheme of Li and König [55]. The implementation takes two parameters k and L, as discussed in Section 5.1. We choose k and L such that the average false negative rate (the ratio of near points not colliding with the queries) is not more than 10%. In particular, k is set such that we expect no more than 5 points with Jaccard similarity at most 0.1 to have the same hash value as the query in a single repetition. Both for Last.FM and MovieLens we used $k = 8$ and $L = 100$ with a similarity threshold of 0.2 and 0.25, respectively.

For L_2 Euclidean distance, we use the LSH data structure from [27]. Each of the L hash functions g_i is the concatenation of k unit hash functions $h_i^1 \circ \cdots \circ h_i^k$. Each of the unit hash functions h_i^j is chosen by selecting a point in a random direction (by choosing every coordinate from a Gaussian distribution with parameters $(0, 1)$). Then all the points are projected onto this one-dimensional direction, and we put a randomly shifted one-dimensional grid of length w along this direction. The cells of this grid are considered as buckets of the unit hash function. For tuning the parameters of LSH, we follow the method described in [27], and the manual of E2LSH library [10], as follows.

For MNIST, the average distance of a query to its nearest neighbor in our data set is around 1250. Thus, we choose the near neighbor radius $r = 1275$. With this choice, the r-neighborhood of all but one query is non-empty. We tune the value of w and set it to 3750. As before, we tune k and L so that the false negative rate is less than 10%, and moreover the cost of hashing (proportional to L) balances out the cost of scanning. This results in the parameter choices $k = 15$ and $L = 100$. This also agrees with the fact that L should be roughly the square root of the total number of points. We use the same method for the other two data sets. For SIFT, we use $R = 270$, $w = 870$, $k = 15$, $L = 100$, and for GLOVE we use $R = 4.7$, $w = 15.7$, $k = 15$, and $L = 100$.

Algorithms. Given a query point $\mathbf{q}$, we retrieve all L buckets corresponding to the query. We then implement the following algorithms and compare their performance in returning a uniform neighbor of the query point.

- **Uniform/Uniform (not fair)**: Picks bucket uniformly at random and picks a random point in bucket.
- **Weighted/Uniform (not fair)**: Picks bucket according to its size, and picks uniformly random point inside bucket.
- **Exact degree (fair)**: Picks bucket according to size, and then picks uniformly random point p inside bucket. Then it computes p's degree *exactly* and rejects p with probability $1 - 1/d(p)$. This is the algorithm discussed in Section 5.3 using the exact degree computation from Section 3.2.
- **Approximate degree (approx. fair)**: Picks bucket according to size, and picks uniformly random point p inside bucket. It approximates p's degree and rejects p with probability $1 - 1/d'(p)$. This is the algorithm discussed in Section 5.3.
- **Rank perturbation (fair, but no query independence)**: Picks the point with minimal rank among all buckets and perturbs the rank afterwards. This is the algorithm discussed in Section 5.2.

Each method removes non-close points that might be selected from the bucket. We remark that the variant Uniform/Uniform closely resembles a straight-forward, but incorrect sampling approach based on the LSH data structure. From the implemented methods, its output distribution should be furthest away from a uniform distribution, and it is one central objective to measure this distance in a real-world setting. Weighted/Uniform takes the different bucket sizes into account, but disregards the individual frequency of a point. Thus, the output is again *not expected* to be uniform, but might be closer in distribution to the uniform distribution.

Degree approximation method. We use the algorithm of Section 3.4 for the degree approximation: we implement a variant of the sampling algorithm which repeatedly samples a bucket uniformly at random and checks whether p belongs to the bucket. If the first time this happens is at iteration i, then it outputs the estimate as $d'(p) = L/i$.

Objectives of the Experiments. Our experiments are tailored to answer the following questions:

(Q1) How (un)fair is the output of different query algorithms in a real-world scenario?
(Q2) What is the extra cost term for solving the exact neighborhood problem?
(Q3) How quickly can the different approaches answer queries?
(Q4) How fair is the output of an algorithm solving the approximate neighborhood version?

7.1 Output Distribution of Different Approaches (Q1)

We start by building the LSH data structure with the parameters detailed in the previous subsection. For each query $\mathbf{q}$, we first compute the number $M(\mathbf{q})$ of near neighbors that collide with the query point. Next, we repeat the query $\mathbf{q}$ $100M(\mathbf{q})$ times and collect the reported points.

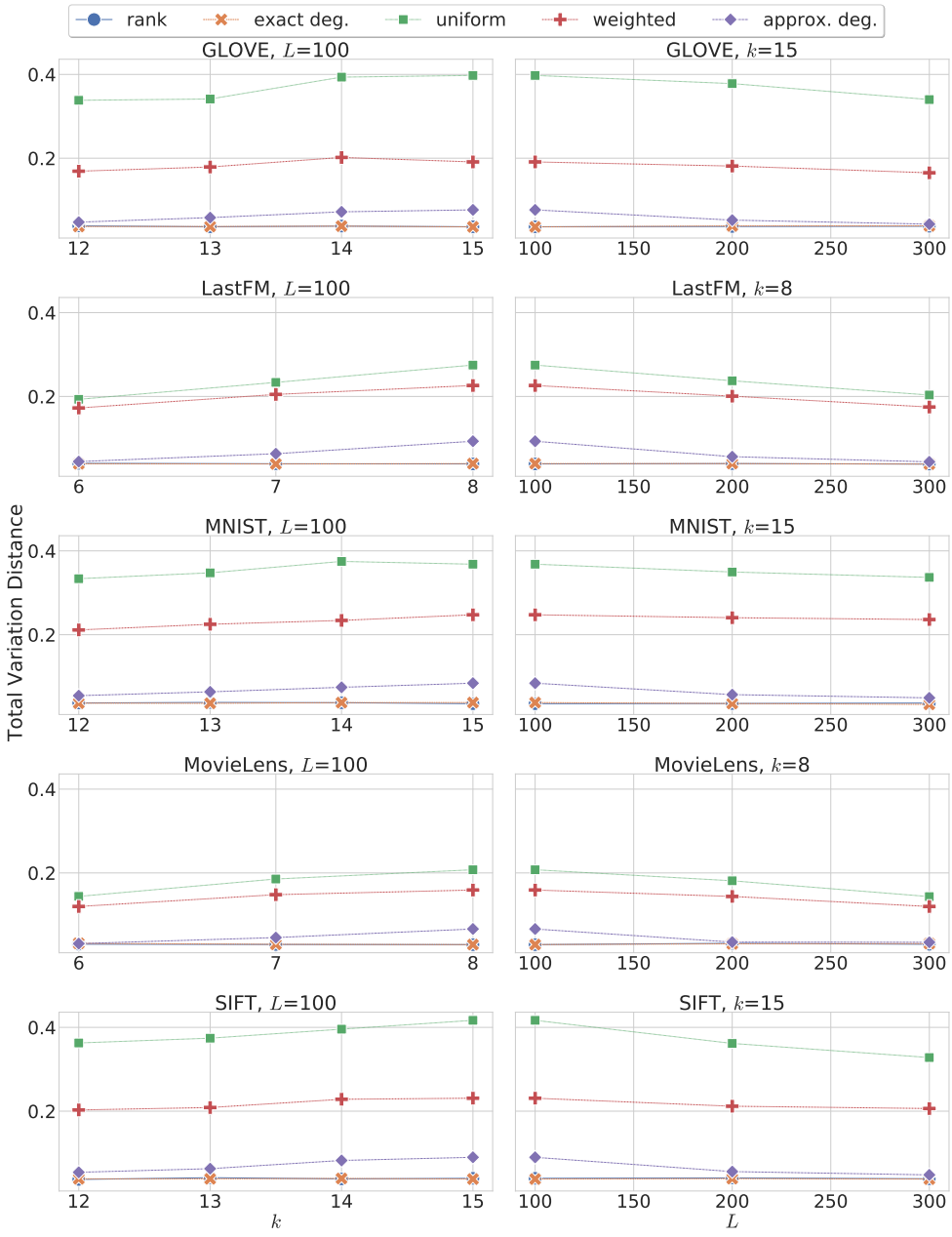

Fig. 1. Total Variation Distance of considered methods on the different datasets. The x-axis represents k (concatenation length, left column) and L (# repetitions, right column); y-axis represents TVD. k and L values are varied to introduce more collisions and evaluate the impact of these collisions.

[The figure has ten plots.]The plots represent the output distribution on the five methods on five datasets. The x axis gives the k value; the y axis gives the total variation distance.

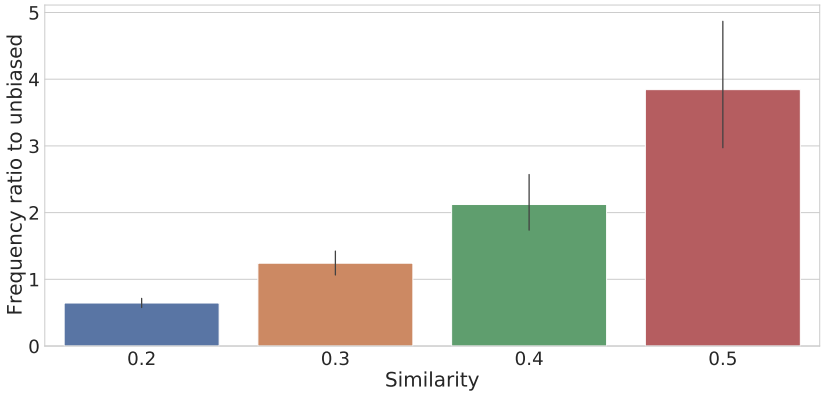

Fig. 2. Bias introduced by uniform sampling from LSH buckets on the Last.FM dataset. The task is to (repeatedly) retrieve a uniform user among all users with similarity at least 0.2 to a fixed user. The result is split up into 4 buckets by rounding down the similarity to the first decimal. Error bars show the standard deviation. Compared to an unbiased sample, user vectors with small similarity are underrepresented, and users with high similarity are, by a factor of around 4 on average, overrepresented.
[The figure has one plot.]The plot represents the bias introduced by uniform sampling. The x axis gives the similarity; the y axis gives the frequency ratio to unbiased. The frequency increases with similarity.

Figure 1 provides an overview over the resulting output distribution of the five different approaches on the five datasets. Each data point is the *total variation distance*[4] (or *statistical distance*) between the output distribution and the uniform distribution. Let us first concentrate on the standard choices $k = 15, L = 100$ for Euclidean distance and $k = 8, L = 100$ for Jaccard similarity. We make the following observations. Firstly, there is a clear difference in the output distribution between the (approximately) fair approaches and the two standard choices *uniform* and *weighted (uniform)*. On all datasets, there is a large deviation from the uniform distribution with large differences between the two approaches on the three Euclidean distance datasets, and a marginal differences on the two Jaccard similarity datasets. The degree approximation is much closer to a uniform distribution, but for standard parameter choices the TVD is around .04 to .08. The rank approach described in Section 3.1 is nearly indistinguishable from the exact degree computation. This means that on the used datasets there is little overlap between the neighborhood of the query points, and the dependence considerations discussed in Section 3.1 do not materialize.

As visible from the plot, we vary the parameters to introduce more collisions between the query and dataset points in the following two ways: We shorten the number of concatenated hash functions for a fixed number of repetitions, and we increase the number of repetitions for a fixed concatenation length. In both cases, the output distributions are closer to the uniform distribution.

Since it is difficult to argue about the unfairness of a distribution that is around .4 away in total variation distance from the uniform distribution, we provide a "zoomed in" visualization in Figure 2. In this plot, we show the difference in reported frequencies between the uniform/uniform approach and fair sampling for a fixed query on the Last.FM dataset. We carry out repeated queries such that each point should be reported 100 times. This is true for the fair sampling approach (with frequencies between 92 and 108, omitted in the graph), but we can clearly see a large difference to the standard sampling approach. This approach heavily under-reports points at lower similarity above the threshold .2

[4]For two discrete distributions μ and ν on a finite set X, the total variation distance is $\frac{1}{2}\sum_{x\in X}|\mu(x) - \nu(x)|$.

Dataset	r_1	r_2	r_3
Last.FM	2.1/3.7/3.7	2.7/5.0/6.5	3.4/7.4/12.5
MovieLens	2.5/4.1/4.7	6.3/15.8/19.3	22.2/104.4/140.3
SIFT	8.0/132.9/334.75	6.6/34.4/61.2	4.0/15.4/15.4
Glove	17.0/82.7/84.7	8.5/19.4/19.5	4.5/6.7/6.7
MNIST	2.6/51.0/92.5	3.2/39.4/42.2	4.7/19.8/19.9

Table 2. Ratio $n(\mathbf{q}, cr)/n(\mathbf{q}, r)$ of number of points with similarity at least cr and number of points with similarity at least r for different values of c and r. Each cell reports three values $c_1 r_i/c_2 r_i/c_3 r_i$. For Last.FM and MovieLens we use similarities $(r_1, r_2, r_3) = (0.15, 0.2, 0.25)$ and $(c_1, c_2, c_3) = (\frac{2}{3}, \frac{1}{3}, \frac{1}{5})$. For the remaining three datasets, we use $(c_1, c_2, c_3) = (1.25, 2, 3)$. For distances, we use r_1 values of 250 (SIFT), 4.7 (GloVe), 1250 (MNIST) based on average nearest neighbor distances, and increment twice in steps of 50, 0.5, and 250, respectively.

(only 60 reports on average), and heavily over-reports points at high similarity (more than 400 reports in the maximum). This means that standard approaches will indeed yield biased neighborhoods in real-world situations. Using algorithms that solve the independent sampling version of the r-NN problem eliminate such bias.

7.2 Additional cost factor for solving the exact neighborhood variant (Q2)

The running time bounds of the algorithms in Section 5 and Section 6 have an additional additive or multiplicative running time factor $\tilde{O}(n(\mathbf{q}, cr)/n(\mathbf{q}, r))$, putting in relation the number of points at distance at most $cr, c \geq 1$, (or similarity at least $cr, c \leq 1$,) to those at threshold r. The values r and cr are the distance/similarity thresholds picked when building the data structure. In general, a larger gap between r and cr makes the n^ρ term in the running times smaller. However, the additive or multiplicative cost $\tilde{O}(n(\mathbf{q}, cr)/n(\mathbf{q}, r))$ can potentially be prohibitively large for worst-case datasets. One can use a variant of LSH as a density estimator to approximate this ratio quickly, see [38, Section 5]; naturally, this estimation deteriorates to the worst case query time, when the ratio is large.

Table 2 depicts the ratio of points with similarity at least cr and at least r. We see that one has to be careful and choose a rather small approximation factor, since the ratio $n(\mathbf{q}, cr)/n(\mathbf{q}, r)$ can easily be the dominating factor. For example, let us consider SIFT with $n = 10\,000$ sampled points and $r = 250$. Using Euclidean LSH, we have to set $\mathsf{L} \sim n^{1/c}$ ([28]). For $c = 1.25$, the ratio of points at distance r and $1.25r$ is only 8, but L has to be set to roughly 1585. With $c = 2$ we set $\mathsf{L} = 100$ and the ratio of near and cr-near points cost is 132.9, roughly balancing these terms. For $c = 3$, L can be set to 21, but the ratio $n(\mathbf{q}, cr)/n(\mathbf{q}, r)$ approaches 335. This shows that a careful inspection of data/query set characteristics might be necessary to find good parameter choices that balance cost terms.

7.3 Running time observations (Q3)

We implemented the LSH for Euclidean distance in C ++ to compare running times between the different approaches. We carry out the same queries as before, but repeat each experiment ten times. To study the influence of the degree of points, we carry out each experiment with $\mathsf{L} = 100$ and $\mathsf{L} = 300$ repetitions, respectively. Experiments were run on an Intel i7-4790 CPU clocked at 3.60GHz with 4 cores and 32 GB of RAM running CentOS 7. All experiments only used a single thread. The code was compiled with gcc 8.3 with the flags `-O3 -march=native -ffast-math`.

To simplify the code structure and the use of additional data structures, we made the following implementation choices: (i) In the rank data structure, we store each bucket as a list of points sorted by their rank. In the perturbation step, all buckets are sorted again after exchanging a rank. (ii) For the other data structures, we carry out a linear scan on the bucket sizes to select a bucket (by weight). In addition to the methods discussed in the beginning of this section,

we also implemented a naive approach using LSH for Fair NN, named **Collect**. Given a query point, it collects all the colliding points in the L buckets in a set, inspects the points in a random order, and returns the first near point found that way. Exact timings and more statistics on the individual algorithms can be found in Appendix B.

Figure 3 reports on the average running time needed by the different methods to answer the $100M(\mathbf{q})$ queries. Let us focus on $\mathsf{L} = 100$ repetitions. The running times range from 10 to 100ms for the non-fair sampling methods. They are about a magnitude larger for the (approximately) fair approaches discussed in our paper, which are themselves a factor of 100 faster than the naive approach of collecting all points. With respect to exact and approximate degree computation, we see that the former is around 2 times slower than the approximate approach. This is as good as what we can expect since the avg. degree of a near point is around 2 for all three datasets. Rank perturbation, which leads to similar distributions as the optimal exact degree computation in the considered workloads, is roughly as fast as the optimal variant on GLOVE and MNIST, but 2 times faster on SIFT. Next, let us consider $\mathsf{L} = 300$. `Collect` scales roughly with a factor 3, as expected. `Uniform/Uniform` and `Weighted Uniform` scale by a factor of 1.5 to 2 because more far points are retrieved and have to be discarded. `Exact degree` takes roughly 10 times longer when increasing the repetition count by 3. This is because a degree computation takes three times longer, the success probability in a single round drops by a factor 3 on average, and there are more non-near points colliding with the query. On the other hand, `approx degree` scales with a factor 4-7 depending on the dataset. This is because the approximate degree computation inspects the same number of repetitions as before; the degree of a near point is roughly three times higher on average. On the other hand, the success probability drops by a factor of three, and there are more non-near points that have to be discarded. Rank pertubation provides the best scaling with a factor 3-4 depending on the dataset. It inspects three times as many tables and has to discard slightly more points.

7.4 Fairness in the approximate version (Q4)

We turn our focus to the approximate neighborhood sampling problem, cf. Definition 4. Recall that an algorithm solving this problem may return points in $B(\mathbf{q}, cr) \setminus B(\mathbf{q}, r)$ as well, which speeds up the query since it can avoid additional filtering steps as discussed in Theorem 2. In the following, we will provide a concrete example that the output of an algorithm that solves this problem might yield unexpected sampling results.

Let us define the following dataset over the universe $\mathcal{U} = \{1, \ldots, 30\}$: We let $\mathbf{X} = \{16, \ldots, 30\}$, $\mathbf{Y} = \{1, \ldots, 18\}$, and $\mathbf{Z} = \{1, \ldots, 27\}$. Furthermore, we let $\mathcal{M}$ contain all subsets of $\mathbf{Y}$ having at least 15 elements (excluding $\mathbf{Y}$ itself). The dataset is the collection of $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ and all sets in $\mathcal{M}$. Let the query $\mathbf{Q}$ be the set $\{1, \ldots, 30\}$. To build the data structure, we set $r = 0.9$ and $cr = 0.5$. It is easy to see that $\mathbf{Z}$ is the nearest neighbor with similarity 0.9. $\mathbf{Y}$ is the second-nearest point with similarity 0.6, but $\mathbf{X}$ is among the points with the lowest similarity of 0.5. Finally, we note that each $\mathbf{x} \in \mathcal{M}$ has similarity ranging from 0.5 to $0.5\overline{6}$. Under the approximate neighborhood sampling problem, all points can be returned for the given query.

As before, we build an LSH data structure. We used $\mathsf{L} = 3$ repetitions and $k = 5$. To solve Approximately Fair ANN, we let the algorithm collect all the points found in the L buckets and return a point picked uniformly at random among those points having similarity at least 0.5. This experiment is repeated 10,000 times with new hash functions. The empirical sampling probabilities of the sets $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are 1.5%, 0.06%, and 2.6%, respectively. This observation shows that the notion of approximate neighborhood does not provide a sensible guarantee on the individual fairness of users in this example. Not only are sampling probabilities very differently, the set $\mathbf{X}$ is more than 25 times more likely than $\mathbf{Y}$ to be returned, even though $\mathbf{Y}$ is more similar to the query. This is due to the clustered neighborhood of $\mathbf{Y}$, making many

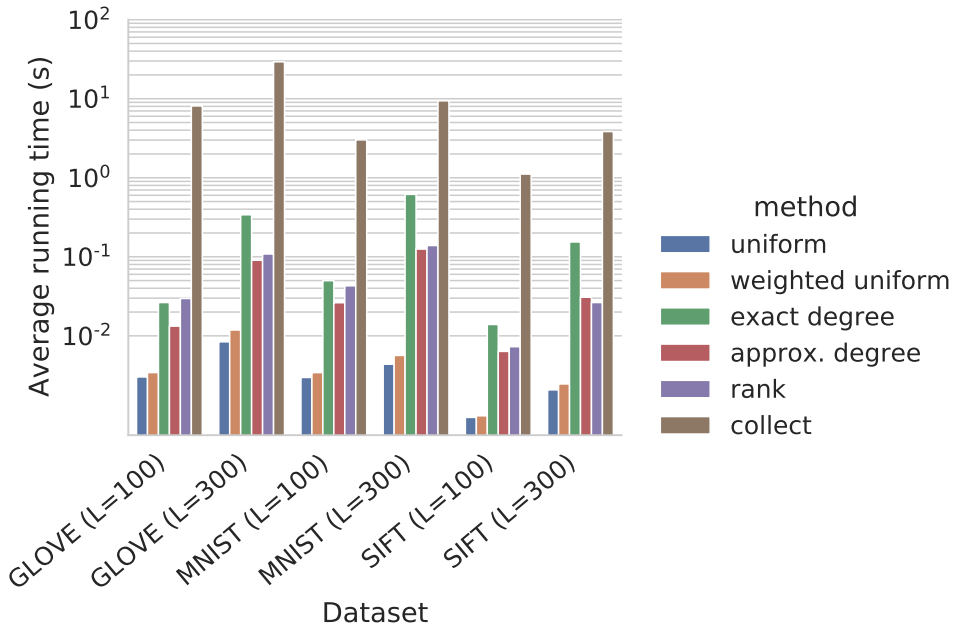

Fig. 3. Comparison of average running times among the six methods on GLOVE, MNIST, and SIFT with 100 and 300 repetitions. [The figure has six plots.]Each plot provides on the y axis the running times of the six methods on one of the six datasets.

other points appear at the same time in the buckets. On the other hand, $\mathbf{X}$ has no close points in its neighborhood (except $\mathbf{Z}$ and $\mathbf{Q}$).

We remark that we did not observe this influence of clustered neighborhoods on the real-world datasets. However, it is important to notice that approximate neighborhood *could* introduce unintentional bias and, furthermore, can be exploited by an adversary to discriminate a given user (e.g., an adversary can create a set of objects $\mathcal{M}$ that obfuscate a given entry $\mathbf{Y}$.)

Based on this example, one could argue that the observed sampling behavior is intentional. If the goal is to find good representatives in the neighborhood of the query, then it certainly seems preferable that $\mathbf{X}$ is reported with such high probability (which is roughly the same as all points in $\mathcal{M}$ and $\mathbf{Y}$ combined). Our notion of fairness would make the output less diverse, since $\mathbf{X}$ clearly stands out from many other points in the neighborhood, but it is supposed to be treated in the very same way. Such a trade-off between diversity and fairness was also observed, for example, by Leonhardt *et al.* [54].

8 CONCLUSIONS

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 [56], diversity in recommender systems [3], privacy preserving similarity search [65], and estimation of kernel density [22]. Moreover, it would be interesting to investigate

techniques for providing incentives (i.e., reverse discrimination [56]) to prevent discrimination: an idea could be to merge the data structures in this paper with distance-sensitive hashing functions in [13], which allow to implement hashing schemes where the collision probability is an (almost) arbitrary function of the distance. Further, 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 [7] in another setting. Finally, for some of the data structures presented here, the query time is also a function of the *local density* around the query point (e.g. $n(q, cr)/n(q, r)$), it would be ideal to find the optimal dependence on this local density.

ACKNOWLEDGEMENTS

S. Har-Peled was partially supported by a NSF AF award CCF-1907400. F. Silvestri was partially supported by UniPD SID18 grant and PRIN Project n. 20174LF3T8 AHeAD.

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. 2017. Research Directions for Principles of Data Management (Abridged). *SIGMOD Rec.* 45, 4 (2017), 5–17.

[2] Eytan Adar. 2007. User 4xxxxx9: Anonymizing query logs. (01 2007). http://www2007.org/workshops/paper_52.pdf Appeared in the workshop *Query Log Analysis: Social and Technological Challenges*, in association with WWW 2007.

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

[4] Peyman Afshani and Jeff M. Phillips. 2019. Independent Range Sampling, Revisited Again. In *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA (LIPIcs, Vol. 129)*, Gill Barequet and Yusu Wang (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Wadern, Germany, 4:1–4:13. https://doi.org/10.4230/LIPIcs.SoCG.2019.4

[5] Peyman Afshani and Zhewei Wei. 2017. Independent Range Sampling, Revisited. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria (LIPIcs, Vol. 87)*, Kirk Pruhs and Christian Sohler (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Wadern, Germany, 3:1–3:14. https://doi.org/10.4230/LIPIcs.ESA.2017.3

[6] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna M. Wallach. 2018. A Reductions Approach to Fair Classification. In *Proc. 35th Int. Conf. Mach. Learning* (ICML) *(Proc. of Mach. Learn. Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, , 60–69. http://proceedings.mlr.press/v80/agarwal18a.html

[7] Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh. 2017. Parameter-Free Locality Sensitive Hashing for Spherical Range Reporting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 239–256.

[8] Piotr Indyk Alexandr Andoni. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122. https://doi.org/10.1145/1327452.1327494

[9] Josh Alman and Ryan Williams. 2015. Probabilistic Polynomials and Hamming Nearest Neighbors. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, USA, 136–150. https://doi.org/10.1109/FOCS.2015.18

[10] Alexandr Andoni. 2005. E2LSH 0.1 User manual. (2005). https://www.mit.edu/~andoni/LSH/manual.pdf

[11] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. 2017. Optimal Hashing-Based Time-Space Trade-Offs for Approximate near Neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 47–66.

[12] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.* 87 (2020), 17. https://doi.org/10.1016/j.is.2019.02.006

[13] Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Francesco Silvestri. 2018. Distance-Sensitive Hashing. In *Proc. 37th ACM Symposium on Principles of Database Systems (PODS)*. Association for Computing Machinery, New York, NY, USA, 89–104.

[14] Martin Aumüller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. 2021. Fair near neighbor search via sampling. *SIGMOD Rec.* 50, 1 (2021), 42–49. https://doi.org/10.1145/3471485.3471496

[15] Martin Aumüller, Rasmus Pagh, and Francesco Silvestri. 2020. Fair Near Neighbor Search: Independent Range Sampling in High Dimensions. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020,*

Dan Suciu, Yufei Tao, and Zhewei Wei (Eds.). ACM, New York, NY, USA, 191–204. https://doi.org/10.1145/3375395.3387648

[16] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. 2019. Scalable Fair Clustering. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, , 405–413. http://proceedings.mlr.press/v97/backurs19a.html

[17] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM '02)*. Springer-Verlag, Berlin, Heidelberg, 1–10.

[18] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. 2017. Edge Estimation with Independent Set Oracles. *CoRR* abs/1711.07567 (2017), 29. arXiv:1711.07567 http://arxiv.org/abs/1711.07567

[19] Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. 2019. Fair Algorithms for Clustering. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). Curran Associates, Inc., Red Hook, NY, USA, 4955–4966. http://papers.nips.cc/paper/8741-fair-algorithms-for-clustering

[20] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Proc. Compression and Complexity of Sequences*. IEEE Computer Society, USA, 21–29.

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

[22] Moses Charikar and Paris Siminelakis. 2017. Hashing-Based-Estimators for Kernel Density in High Dimensions. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, Chris Umans (Ed.). IEEE Computer Society, USA, 1032–1043. https://doi.org/10.1109/FOCS.2017.99

[23] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. 2017. Fair Clustering through Fairlets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5036–5044.

[24] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvtiskii. 2019. Matroids, Matchings, and Fairness. In *Proceedings of Machine Learning Research (Proceedings of Machine Learning Research, Vol. 89)*, Kamalika Chaudhuri and Masashi Sugiyama (Eds.). PMLR, , 2212–2220. http://proceedings.mlr.press/v89/chierichetti19a.html

[25] Alexandra Chouldechova. 2017. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data* 5, 2 (2017), 153–163.

[26] Tobias Christiani. 2017. A Framework for Similarity Search with Space-Time Tradeoffs using Locality-Sensitive Filtering. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, USA, 31–46.

[27] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Annu. Sympos. Comput. Geom.* (SoCG) (Brooklyn, New York, USA). ACM, New York, NY, USA, 253–262. https://doi.org/10.1145/997817.997857

[28] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) *(SCG '04)*. Association for Computing Machinery, New York, NY, USA, 253–262. https://doi.org/10.1145/997817.997857

[29] Herbert Aron David and Haikady Navada Nagaraja. 2004. *Order statistics* (3rd ed.). John Wiley & Sons, New York, NY.

[30] Michele Donini, Luca Oneto, Shai Ben-David, John Shawe-Taylor, and Massimiliano Pontil. 2018. Empirical Risk Minimization under Fairness Constraints. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 2796–2806.

[31] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 214–226. https://doi.org/10.1145/2090236.2090255

[32] Hadi Elzayn, Shahin Jabbari, Christopher Jung, Michael Kearns, Seth Neel, Aaron Roth, and Zachary Schutzman. 2019. Fair Algorithms for Learning in Allocation Problems. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM, Association for Computing Machinery, New York, NY, USA, 170–179. https://doi.org/10.1145/3287560.3287571

[33] Kave Eshghi and Shyamsundar Rajaram. 2008. Locality Sensitive Hash Functions Based on Concomitant Rank Order Statistics. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Las Vegas, Nevada, USA) *(KDD '08)*. Association for Computing Machinery, New York, NY, USA, 221–229. https://doi.org/10.1145/1401890.1401921

[34] Brian S. Everitt, Sabine Landau, and Morven Leese. 2009. *Cluster Analysis* (4th ed.). Wiley Publishing, Chicago, IL.

[35] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* 31, 2 (1985), 182–209.

[36] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. 2021. The (Im)possibility of fairness: different value systems require different mechanisms for fair decision making. *Commun. ACM* 64, 4 (2021), 136–143.

[37] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Theory Comput.* 8, Article 14 (2012), 30 pages. https://doi.org/10.4086/toc.2012.v008a014 Special issue in honor of Rajeev Motwani.

[38] Sariel Har-Peled and Sepideh Mahabadi. 2017. Proximity in the Age of Distraction: Robust Approximate Nearest Neighbor Search. In *Proc. 28th ACM-SIAM Sympos. Discrete Algs.* (SODA), Philip N. Klein (Ed.). SIAM, 1–15. https://doi.org/10.1137/1.9781611974782.1

[39] Sariel Har-Peled and Sepideh Mahabadi. 2019. Near Neighbor: Who is the Fairest of Them All?. In *Proc. 32th Neural Info. Proc. Sys.* (NeurIPS), Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). Curran Associates, Inc., Red

Hook, NY, USA, 13176–13187. http://papers.nips.cc/paper/9476-near-neighbor-who-is-the-fairest-of-them-all

[40] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Neural Info. Proc. Sys.* (NIPS), Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). Curran Associates Inc., Red Hook, NY, USA, 3315–3323. http://papers.nips.cc/paper/6374-equality-of-opportunity-in-supervised-learning

[41] David Harel and Yehuda Koren. 2001. On Clustering Using Random Walks. In *FSTTCS 2001: Foundations of Software Technology and Theoretical Computer Science, 21st Conference, Bangalore, India, December 13-15, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2245)*, Ramesh Hariharan, Madhavan Mukund, and V. Vinay (Eds.). Springer, NY, 18–41. https://doi.org/10.1007/3-540-45294-X_3

[42] Ahmad Basheer Hassanat, Mohammad Ali Abbadi, Ghada Awad Altarawneh, and Ahmad Ali Alhasanat. 2014. Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach. *CoRR* abs/1409.0919 (2014), 33–39. arXiv:1409.0919 http://arxiv.org/abs/1409.0919

[43] Xiaocheng Hu, Miao Qiao, and Yufei Tao. 2014. Independent range sampling. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, Richard Hull and Martin Grohe (Eds.). ACM, New York, NY, USA, 246–255. https://doi.org/10.1145/2594538.2594545

[44] Jiayuan Huang, Alexander J. Smola, Arthur Gretton, Karsten M. Borgwardt, and Bernhard Scholkopf. 2006. Correcting Sample Selection Bias by Unlabeled Data. In *Proceedings of the 19th International Conference on Neural Information Processing Systems* (Canada) *(NIPS'06)*. MIT Press, Cambridge, MA, USA, 601–608.

[45] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 604–613. https://doi.org/10.1145/276698.276876

[46] Sampath Kannan, Aaron Roth, and Juba Ziani. 2019. Downstream Effects of Affirmative Action. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (Atlanta, GA, USA) *(FAT* '19)*. Association for Computing Machinery, New York, NY, USA, 240–248. https://doi.org/10.1145/3287560.3287578

[47] Matt J Keeling and Ken T.D Eames. 2005. Networks and epidemic models. *Journal of The Royal Society Interface* 2, 4 (Sept. 2005), 295–307. https://doi.org/10.1098/rsif.2005.0051

[48] Jon Kleinberg, Himabindu Lakkaraju, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. 2017. Human decisions and machine predictions. *The quarterly journal of economics* 133, 1 (2017), 237–293.

[49] Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. 2019. Guarantees for Spectral Clustering with Fairness Constraints. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 3458–3467. http://proceedings.mlr.press/v97/kleindessner19b.html

[50] Donald Ervin Knuth. 1998. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, Boston, MA. https://www.worldcat.org/oclc/312898417

[51] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.

[52] Yi-Hung Kung, Pei-Sheng Lin, and Cheng-Hsiung Kao. 2012. An optimal k-nearest neighbor for density estimation. *Statistics & Probability Letters* 82, 10 (2012), 1786 – 1791. https://doi.org/10.1016/j.spl.2012.05.017

[53] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[54] Jurek Leonhardt, Avishek Anand, and Megha Khosla. 2018. User Fairness in Recommender Systems. In *Companion Proceedings of the The Web Conference 2018* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 101–102. https://doi.org/10.1145/3184558.3186949

[55] Ping Li and Christian König. 2010. B-Bit Minwise Hashing. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 671–680. https://doi.org/10.1145/1772690.1772759

[56] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. 2011. K-NN as an Implementation of Situation Testing for Discrimination Discovery and Prevention. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '11)*. Association for Computing Machinery, New York, NY, USA, 502–510. https://doi.org/10.1145/2020408.2020488

[57] Cecilia Munoz, Megan Smith, and DJ Patil. 2016. *Big Data: A Report on Algorithmic Systems, Opportunity, and Civil Rights*. Executive Office of the President and Penny Hill Press, Damascus, MD.

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

[59] Matt Olfat and Anil Aswani. 2019. Convex Formulations for Fair Principal Component Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 663–670. https://doi.org/10.1609/aaai.v33i01.3301663

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

[61] Frank Olken and Doron Rotem. 1995. Sampling from spatial databases. *Statistics and Computing* 5, 1 (01 Mar 1995), 43–57. https://doi.org/10.1007/BF00140665

[62] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natu ral Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

[63] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q. Weinberger. 2017. On Fairness and Calibration. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5684–5693.

[64] Yinian Qi and Mikhail J. Atallah. 2008. Efficient Privacy-Preserving k-Nearest Neighbor Search. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems (ICDCS '08)*. IEEE Computer Society, USA, 311–319. https://doi.org/10.1109/ICDCS.2008.79

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

[66] Andrew D. Selbst, Danah Boyd, Sorelle A. Friedler, Suresh Venkatasubramanian, and Janet Vertesi. 2019. Fairness and Abstraction in Sociotechnical Systems. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (Atlanta, GA, USA) *(FAT* '19)*. Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/3287560.3287598

[67] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. 2006. *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT Press, Cambridge, MA.

[68] Stanislaw J. Szarek and Elisabeth Werner. 1999. A Nonsymmetric Correlation Inequality for Gaussian Measure. *Journal of Multivariate Analysis* 68, 2 (1999), 193 – 211.

[69] A. Torralba and A. A. Efros. 2011. Unbiased Look at Dataset Bias. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11)*. IEEE Computer Society, USA, 1521–1528. https://doi.org/10.1109/CVPR.2011.5995347

[70] Ryan Williams. 2005. A New Algorithm for Optimal 2-constraint Satisfaction and Its Implications. *Theor. Comput. Sci.* 348, 2 (2005), 357–365.

[71] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. 2017. Fairness Beyond Disparate Treatment & Disparate Impact: Learning Classification without Disparate Mistreatment. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1171–1180. https://doi.org/10.1145/3038912.3052660

A A LINEAR SPACE NEAR-NEIGHBOR DATA STRUCTURE

We will split up the analysis of the data structure from Section 6 into two parts. First, we describe and analyze 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 6.

A.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 $m \geq 1$ 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 $\mathbf{a}$.

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 Appendix A.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.

A.2 Analysis of Close Points

Lemma 22. *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 [26], we may assume that $\mathbf{x} = (1, 0, \ldots, 0)$ and $\mathbf{q} = (\alpha, \sqrt{1-\alpha^2}, 0, \ldots, 0)$. 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 $\langle \mathbf{a}, \mathbf{x}\rangle = \Delta_{\mathbf{x}}$ and the vector $\mathbf{a}'$ with $\langle \mathbf{a}', \mathbf{q}\rangle = \Delta_{\mathbf{q}}$, we have $\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon)$. We note that $\Pr[\max\{\langle \mathbf{a}, \mathbf{q}\rangle\} = \Delta] = 1 - \Pr[\forall i : \langle \mathbf{a}_i, \mathbf{q}\rangle < \Delta]$.

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

$$\begin{aligned}\Pr[\text{find } \mathbf{x}] \geq \Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha\Delta_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = \Delta_{\mathbf{x}} \text{ and } \langle \mathbf{a}', \mathbf{q}\rangle = \Delta_{\mathbf{q}}\right] \\ - \Pr[\forall i : \langle \mathbf{a}_i, \mathbf{x}\rangle < \Delta_{\mathbf{x}}] - \Pr\left[\forall i : \langle \mathbf{a}_i, \mathbf{q}\rangle < \Delta_{\mathbf{q}}\right]. \end{aligned} \tag{3}$$

Here, we used that $\Pr[A \cap B \cap C] = 1 - \Pr\left[\overline{A} \cup \overline{B} \cup \overline{C}\right] \geq \Pr[A] - \Pr\left[\overline{B}\right] - \Pr\left[\overline{C}\right]$. We will now obtain bounds for the three terms on the right-hand side of Eq. (3) separately, but we first recall the following lemma from [68]:

Lemma 23 ([68]). *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 $\mathbf{q} = (\alpha, \sqrt{1-\alpha^2}, 0, \ldots, 0)$ and $\mathbf{x} = (1, 0, \ldots, 0)$, 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 23, we may compute

$$\begin{aligned}\Pr\left[Z \geq -\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right] &= 1 - \Pr\left[Z < -\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right] \\ &= 1 - \Pr\left[Z \geq \frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}\right] \\ &\geq 1 - \frac{\operatorname{Exp}\left(-\frac{(f(\alpha,\varepsilon))^2}{2(1-\alpha^2)}\right)}{\sqrt{2\pi}\left(\frac{f(\alpha,\varepsilon)}{\sqrt{1-\alpha^2}}+1\right)} \geq 1-\varepsilon. \end{aligned} \tag{4}$$

Bounding the second term and third term. We first observe that

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

Setting $\Delta_{\mathbf{x}} = \sqrt{2\log m - \log(4\kappa\pi\log(m))}$ upper bounds this term by $\mathrm{Exp}[-\sqrt{\kappa}]$. 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 Eq. (3), 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 $\varepsilon' \geq \varepsilon + 2\delta$. □

A.3 Analysis of Far Points

Lemma 24. *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 $\Delta_{\mathbf{x}} \leq t_{\mathbf{x}}$. From the proof of Lemma 22, we know that

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

Setting $t_{\mathbf{q}} = \sqrt{2\log(m/\log(n)) - \log(4\pi\log(m/\log n))}$ shows that with high probability we have $\Delta_{\mathbf{q}} \geq t_{\mathbf{q}}$. Similarly, 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:

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

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 23, we have

$$\begin{aligned}\Pr\left[\langle \mathbf{a}, \mathbf{q}\rangle \geq \alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) \mid \langle \mathbf{a}, \mathbf{x}\rangle = t_{\mathbf{x}}\right] &\leq \frac{\mathrm{Exp}\left(-\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})^2}{2(1-\beta^2)}\right)}{\sqrt{\pi}\left(\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})}{\sqrt{1-\beta^2}} + 1\right)} \\ &\leq \mathrm{Exp}\left(-\frac{(\alpha t_{\mathbf{q}} - f(\alpha,\varepsilon) - \beta t_{\mathbf{x}})^2}{2(1-\beta^2)}\right) \\ &\overset{(1)}{=} \mathrm{Exp}\left(-\frac{(\alpha-\beta)^2 t_{\mathbf{x}}^2}{2(1-\beta^2)}\left(1 + O(1/\log\log n)\right)\right) \\ &= \left(\frac{1}{m}\right)^{\frac{(\alpha-\beta)^2}{1-\beta^2} + o(1)}, \end{aligned} \tag{5}$$

where step (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, the expected number of lists to be checked is not more than

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

which is $m^{1-\alpha^2+o(1)}$ using the value of $t_{\mathbf{q}}$ set above. This motivates to set Eq. (5) 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 $m^{1-\alpha^2+o(1)} = n^{\rho+o(1)}$. □

A.4 Efficient Evaluation

The previous subsections assumed that m filters can be evaluated and stored 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 [26].

Construction. Let $t = \lceil 1/(1-\alpha^2) \rceil$ and assume that $m^{1/t}$ is an integer. Consider t independent data structures $\mathcal{D}_1, \ldots, \mathcal{D}_t$, each using $m^{1/t}$ random vectors $\mathbf{a}_{i,j}$, for $i \in \{1, \ldots, t\}$, $j \in [m^{1/t}]$. Each $\mathcal{D}_i$ 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 $\mathcal{D}_1, \ldots, \mathcal{D}_t$, 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 is $O(n + tm^{1/t})$.

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 $\mathcal{D}_i$. 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 24, 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 $\mathcal{D}_i$, 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 $\mathcal{D}_i$. 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 22.

Let $\mathbf{y}$ be a point with $\langle \mathbf{q}, \mathbf{y}\rangle < \beta$. Using the same approach in the proof of Lemma 24, we observe that the probability of finding $\mathbf{y}$ in an individual $\mathcal{D}_i$ 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. □

B RUNNING TIMES

Table 3 contains the running time measurements that formed the basis for Figure 3.

Method	Dataset	Avg. Time (s)	Avg. Degree	Avg. Max. Degree	Avg. Non-near	Avg. Rejections
approx. degree	GLOVE (L=100)	0.013	1.800	5.600	4.460	0.326
approx. degree	GLOVE (L=300)	0.090	5.480	14.060	11.700	4.144
approx. degree	MNIST (L=100)	0.026	2.540	12.280	0.120	1.810
approx. degree	MNIST (L=300)	0.125	7.760	35.560	1.060	6.704
approx. degree	SIFT (L=100)	0.006	2.180	8.280	0.540	1.360
approx. degree	SIFT (L=300)	0.031	6.220	20.940	1.840	5.156
collect	GLOVE (L=100)	8.104	1.800	5.600	—	—
collect	GLOVE (L=300)	29.307	5.480	14.060	—	—
collect	MNIST (L=100)	3.016	2.540	12.280	—	—
collect	MNIST (L=300)	9.402	7.760	35.560	—	—
collect	SIFT (L=100)	1.114	2.180	8.280	—	—
collect	SIFT (L=300)	3.842	6.220	20.940	—	—
exact degree	GLOVE (L=100)	0.026	1.800	5.600	4.460	0.764
exact degree	GLOVE (L=300)	0.340	5.480	14.060	11.700	4.434
exact degree	MNIST (L=100)	0.050	2.540	12.280	0.120	1.546
exact degree	MNIST (L=300)	0.616	7.760	35.560	1.060	6.716
exact degree	SIFT (L=100)	0.014	2.180	8.280	0.540	1.138
exact degree	SIFT (L=300)	0.154	6.220	20.940	1.840	5.180
rank	GLOVE (L=100)	0.030	1.800	5.600	3.680	—
rank	GLOVE (L=300)	0.108	5.480	14.060	7.160	—
rank	MNIST (L=100)	0.043	2.540	12.280	0.100	—
rank	MNIST (L=300)	0.139	7.760	35.560	0.740	—
rank	SIFT (L=100)	0.007	2.180	8.280	0.380	—
rank	SIFT (L=300)	0.026	6.220	20.940	1.120	—
uniform	GLOVE (L=100)	0.003	1.800	5.600	3.584	—
uniform	GLOVE (L=300)	0.008	5.480	14.060	6.670	—
uniform	MNIST (L=100)	0.003	2.540	12.280	0.104	—
uniform	MNIST (L=300)	0.004	7.760	35.560	0.542	—
uniform	SIFT (L=100)	0.001	2.180	8.280	0.442	—
uniform	SIFT (L=300)	0.002	6.220	20.940	1.230	—
weighted uniform	GLOVE (L=100)	0.003	1.800	5.600	4.460	—
weighted uniform	GLOVE (L=300)	0.012	5.480	14.060	11.680	—
weighted uniform	MNIST (L=100)	0.003	2.540	12.280	0.120	—
weighted uniform	MNIST (L=300)	0.006	7.760	35.560	1.044	—
weighted uniform	SIFT (L=100)	0.001	2.180	8.280	0.540	—
weighted uniform	SIFT (L=300)	0.002	6.220	20.940	1.840	—

Table 3. Detailed measurements for the average running time, the average degree of a near point, the average maximum degree per query, the average number of non-near points inspected per query, and the number of rejections carried out by `exact degree` and `approximate degree`.