

Optimizing Fair Approximate Nearest Neighbor Searches using Threaded B+-Trees

Omid Jafari ^[0000–0003–3422–2755], Preeti Maurya, Khandker Mushfiqul Islam,
and Parth Nagarkar^[0000–0001–6284–9251]

New Mexico State University, Las Cruces, New Mexico, USA
{ojafari, preema, mushfiq, nagarkar}@nmsu.edu

Abstract. Similarity search in high-dimensional spaces is an important primitive operation in many diverse application domains. *Locality Sensitive Hashing* (LSH) is a popular technique for solving the Approximate Nearest Neighbor (ANN) problem in high-dimensional spaces. Along with creating fair machine learning models, there is also a need for creating data structures that target different types of fairness. In this paper, we propose a fair variant of the ANN problem that targets *Equal opportunity* in group fairness in the ANN domain. We formally introduce the notion of fair ANN for *Equal opportunity* in group fairness. Additionally, we present an efficient disk-based index structure for finding Fair approximate nearest neighbors using Locality Sensitive Hashing (*FairLSH*). Moreover, we present an advanced version of *FairLSH* that uses cost models to further balance the trade-off between I/O cost and processing time. Finally, we experimentally show that *FairLSH* returns fair results with a very low I/O cost and processing time when compared with the state-of-the-art LSH techniques.

Keywords: Approximate Nearest Neighbor Search · Similarity Search · Locality Sensitive Hashing · Fairness · Equal Opportunity

1 Introduction

In recent years, many real-world applications use machine learning algorithms for their decision making systems (e.g. job interviews, credit card offers, etc.). Often, these algorithms make discriminative and biased decisions towards specific individuals or group of individuals. There have been several works (such as [14,7,22,17]) that have studied different types of fairness and biases in these decision making systems. Moreover, even if algorithms may not be biased, they could amplify the latent bias that exists in the data. As a result, researchers have proposed new methods to deal with the algorithmic and data biases in classification [1], clustering [4,18], optimization [6], risk management [9], resource allocation [10], and many other domains.

Bias in the data used for training machine learning algorithms is a major challenge in developing fair algorithms. Here, in a rather different problem, we are interested in handling the bias imposed by the data structures used by such

algorithms. In particular, data structures, regardless of how the data is handled and how it is collected, involve bias in the way they respond to searches.

In general, fairness can be divided into two categories: 1) individual fairness and 2) group fairness. The goal of individual fairness is to treat similar individuals similarly and the goal of group fairness is to treat similar groups of individuals similarly. Both categories can be further divided into sub-categories [21]. One such sub-category of group fairness is *Equal Opportunity*, which states that if individuals in multiple groups of people qualify for an outcome, those groups of people should receive the outcome at equal rates [21].

Finding nearest neighbors of a given query is an important problem in many domains. For high-dimensional datasets, the traditional index structures suffer from the well-known problem of *Curse of Dimensionality* [8]. It is shown that even linear searches are faster than using these traditional index structures for high-dimensional datasets [5]. A solution to this problem is to search for approximate nearest neighbors instead of exact neighbors that results in much better running times. Locality Sensitive Hashing (LSH) [12] is a popular technique for solving the Approximate Nearest Neighbor (ANN) problem in high-dimensional spaces that takes a sub-linear time (with respect to the dataset size) to find the approximate nearest neighbors of a given query. LSH maps points in the high-dimensional space to a lower-dimensional space by using random projections. The intuition behind LSH is that close points in the high-dimensional space will map to the same hash buckets in the lower-dimensional space with a high probability and vice-versa.

1.1 Motivation

An important benefit of LSH is that it provides theoretical guarantees on the accuracy of the results. Moreover, LSH is a data-independent method (i.e. the index structure is not affected by data properties such as data distribution). Therefore, when the distribution of data changes, data-dependent methods (such as deep hashing approaches) need to re-generate the indexes. Additionally, LSH is known for its ease of disk-based implementations, making it very scalable as the dataset size grows [19]. Often, in various applications, there is a need to run approximate nearest neighbor searches in order to find fair neighbors of a given query. In these applications, there is a growing need to remove discrimination and bias towards specific individuals or a group of individuals. Here, the goal of fairness is to remove arbitrariness of the search strategy and base it upon pre-defined conditions such that neighbors of a given query that belong to different groups would have the same probability of being chosen in the final results.

There is no existing work that studies the *equal opportunity of group fairness* in the domain of ANN search. Existing state-of-the-art LSH approaches lead to wasted I/O while tackling the *equal opportunity* notion in group fairness (because they are not designed to efficiently search for fair nearest neighbors). Therefore, in this paper, our goal is to design a fair, yet efficient, disk-based LSH index structure, called *FairLSH*, that can reduce the I/O costs and processing times for finding the fair nearest neighbors.

1.2 Contributions

In this paper, we propose an efficient, disk-based index structure for finding Fair approximate nearest neighbors using Locality Sensitive Hashing, called *FairLSH*. The following are the primary contributions of this paper:

- We formally introduce the notion of fair approximate near neighbors for equal opportunity in group fairness.
- We present a tree-based and disk-based index structure, called *FairLSH-Basic*, that reduces disk I/O costs and processing times for finding fair approximate nearest neighbors.
- We further improve the efficiency by proposing a cost model-based variant of our index structure, called *FairLSH-Advanced*, that uses a user-input threshold to tune the trade-off between I/O costs and processing times, and hence further improve performance.
- Lastly, we experimentally evaluate the both variants of *FairLSH* on several datasets for different fairness scenarios and show that *FairLSH* outperforms the state-of-the-art techniques in terms of performance efficiency.

To the best of our knowledge, we are the first work that tackles the group fairness notion of *equal opportunity* in the ANN domain.

2 Related Work

2.1 LSH and its variants

Locality Sensitive Hashing (LSH) is one the most popular techniques for solving the Approximate Nearest Neighbor (ANN) problem in high-dimensional spaces. LSH was first proposed in [12] for the Hamming distance and was later extended for the Euclidean distance in [8]. Then, the concepts of *Collision Counting* and *Virtual Rehashing* were introduced in [11] that solved the two main drawbacks of E2LSH [8], which were large index sizes and a large search radius. The idea of using query-aware hash functions where the indexes are created such that the query is an anchor of a bucket was proposed in QALSH [15] to solve the issue when close points to the query were mapped to different buckets. Moreover, QALSH uses B+-trees as its index structure for efficient lookups and range queries on the hash functions.

I-LSH [19] was recently proposed to improve the I/O cost of QALSH by incrementally increasing the search radius in the projected space instead of using exponential radius increases. However, as shown in [16], I-LSH achieves this I/O cost optimization at the expense of a costly processing time spent on finding closest points in the projected space. Recently, PM-LSH [24] was proposed to utilize a confidence interval value and estimate the Euclidean distance with the goal of reducing the overall query processing time. Moreover, a method, called R2LSH [20], was proposed that uses two-dimensional projected spaces (instead of one-dimensional spaces) to improve the I/O cost of the query processing.

2.2 Fairness in ANN Search

Definitions of fairness are commonly categorized as 1) group fairness, in which the aim is to treat different groups equally, and 2) individual fairness, in which the purpose is to treat individual people of the same profile similarly [21]. So far, only two works have studied the idea of fairness in the ANN domain.

[13] proposes to remove the bias from exact neighborhood and approximate neighborhood (E2LSH) searches by using sampling techniques with the goal of providing individual fairness. Additionally, another work [3] has been proposed that considers equal opportunity in individual fairness in the sense that all points near a query should have the same probability to be returned. In [3], authors first use uniform sampling techniques and then build a data structure for fair similarity search under inner product. Very recently, a joint work [2] containing [13] and [3] has been proposed that connects ideas from both works (i.e. equal opportunity and independent range sampling). Our proposed work is different from the prior works [13,3,2] in two main aspects: 1) Unlike these prior works that focus on individual fairness, our work focuses on equal opportunity in group fairness, and 2) these prior works are designed specifically for the original LSH design (E2LSH). Particularly, their proposed data structures are applicable only for LSH designs that use the multiple hash functions in multiple hash tables (Compound Hash Keys). State-of-the-art LSH designs, such as [11,15,19], use advanced techniques such as *Collision Counting* that makes having multiple hash tables unnecessary (thus saving on space and time). Our proposed work is designed specifically for these state-of-the-art LSH designs.

3 Background and Key Concepts

In this section, we describe the key concepts behind LSH. Given a dataset $\mathcal{D}$ with n points in a d -dimensional Euclidean space $\mathcal{R}^d$ and a query point q in the same space, the goal of c -ANN search (for an approximation ratio $c > 1$) is to return points $o \in \mathcal{D}$ such that $\|o - q\| \leq c \times \|o^* - q\|$, where o^* is the true nearest neighbor of q in $\mathcal{D}$ and $\|\cdot\|$ is the Euclidean distance between two points. Similarly, c - k -ANN search aims at returning top- k points such that $\|o_i - q\| \leq c \times \|o_i^* - q\|$ where $1 \leq i \leq k$.

Definition 1 (LSH Family). A hash function family $\mathcal{H}$ is called (r, c, p_1, p_2) -sensitive if it satisfies the following conditions for any two points x and y in a d -dimensional dataset $\mathcal{D} \subset \mathcal{R}^d$:

- if $\|x - y\| \leq R$, then $\Pr[h(x) = h(y)] \geq p_1$, and
- if $\|x - y\| > cR$, then $\Pr[h(x) = h(y)] \leq p_2$

Here, p_1 and p_2 are probabilities, R is the distance between two points (commonly referred to as the radius), and c is an approximation ratio. LSH requires $c > 1$ and $p_1 > p_2$. The conditions show that the probability of mapping two points to a same hash value decreases as their distance increases.

Definition 2 (Collision Counting). In [11], it is theoretically shown that only those points that collide (are mapped to the same bucket) with the query in at least l projections (out of m) are chosen as candidates. Here, l is the collision count threshold and calculated as $l = \lceil \alpha \times m \rceil$, where α is the collision threshold percentage calculated using $\alpha = \frac{zp_1+p_2}{1+z}$ and m is the total number of projections calculated as $m = \lceil \frac{\ln(\frac{1}{\delta})}{2(p_1-p_2)^2} (1+z)^2 \rceil$. Here, $z = \sqrt{\ln(\frac{2}{\beta})/\ln(\frac{1}{\delta})}$, where β is the allowed false positive percentage (i.e. the allowed number of points whose distance with a query point is greater than cR). [11] sets $\beta = \frac{100}{n}$, where n is the cardinality of the dataset.

Definition 3 (Virtual Rehashing). [11] starts query processing with a very small radius, and then, exponentially increases the radius in the following sequence: $R = 1, c, c^2, c^3, \dots$, where c is an approximation ratio. If at level- R , enough candidates are not found, the radius is increased until found.

4 Problem Specification

Definition 4 (Fair ANN). The definition of Fair ANN in this paper is focused on the equal opportunity problem in group fairness in the ANN domain. Given a dataset $\mathcal{D}$ with n points in a d -dimensional Euclidean space $\mathcal{R}^d$, a query point q , and two groups of points in $\mathcal{D}$ labeled o^A and o^B , the goal of Fair ANN is to find $\text{top-}\lfloor k/2 \rfloor$ points $o^A \in \mathcal{D}$ and $\text{top-}\lfloor k/2 \rfloor$ points $o^B \in \mathcal{D}$, such that $\|o_i^A - q\| \leq c \times \|o_i^{A,*} - q\|$ and $\|o_i^B - q\| \leq c \times \|o_i^{B,*} - q\|$, where $1 \leq i \leq \lfloor k/2 \rfloor$, and $o_i^{A,*}$ and $o_i^{B,*}$ are the true nearest neighbors of q in $\mathcal{D}$ from each group.

In this paper, our goal is to return Fair ANN for a given query q while reducing the overall I/O costs and processing times while maintaining the accuracy of the result. Note that, in this work, we only focus on two distinct groups in the dataset. We leave the problem of Fair ANN for multiple groups as future work. In section 5, we present the design of our index structure, *FairLSH*.

5 FairLSH

In this section, we first describe the naive approaches for solving the Fair ANN problem using the existing LSH methods. We then present the design of our proposed index structure, *FairLSH*, which consists of two variants: *FairLSH-Basic* and *FairLSH-Advanced*. Given a query point, our goal is to efficiently return $\text{top-}\lfloor k/2 \rfloor$ NN from each of the two groups of points in the dataset.

5.1 Naive Approaches

In Section 3, we explained how LSH families are used to map high-dimensional points into a lower-dimensional space while preserving locality. In order to retrieve fair results, we make the following changes to existing LSH methods:

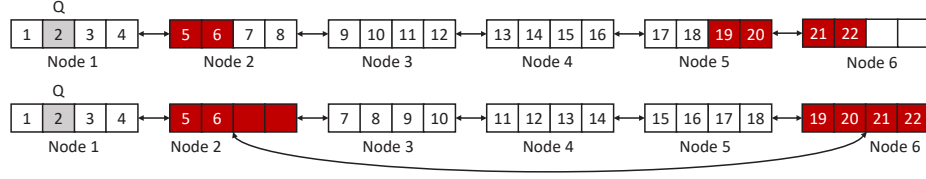


Fig. 1: (a) QALSH leaf nodes (top) compared to (b) FairLSH leaf nodes (bottom)

Naive Strategy 1: A naive strategy to find $\text{top-}[k/2]$ nearest points from each group is to simply divide the dataset into two separate datasets (based on their labels), and run LSH individually on the two separate datasets. The drawbacks of this strategy are: 1) it is not space efficient since two sets of indexes need to be maintained, and 2) redundant processing needs to be performed on the two sets of indexes which results in an increase in the overall query processing time.

Naive Strategy 2: All LSH-based methods have several stopping conditions that make the search algorithm stop once enough points are found. In the second naive strategy, we change these stopping conditions to continue the search algorithm until enough points are found *from each group of points*.

For example, Figure 1(a) shows the leaf nodes of a B+-tree from QALSH [15]. In this example, the query is point ID 2 which is located in *Node 1*, white points are from group A, and red points are from group B. Assuming that $k = 8$, then our goal is to find four nearest points from each of the groups A and B.

In this example, the original QALSH method will start reading *Node 1* and fetching the first four points and it will continue reading *Node 2*. After that, since the stopping conditions are met, the algorithm will stop. However, we only find two nearest points (5 and 6) and the results are not fair. Naive strategy 2 changes stopping conditions such that the algorithm continues reading *Node 3* and *Node 4* as well. By doing this, the algorithm finds enough points from each group and if more than enough points are found, they can be pruned at another step using Euclidean distance calculations.

The main drawback of this naive strategy (as it can also be seen in the given example) is that extra and unnecessary nodes are read which results in an increase in the I/O cost. As a result, we present two variants of a novel index structure in the next section that use threaded B+-trees and cost models to optimize the I/O cost and processing time.

5.2 Design of FairLSH-Basic

The main intuition behind *FairLSH-Basic* is that when enough nearest neighbors of a group are found, we should avoid reading the points of that group from the disk. Hence, our goal is to skip those points that will lead to unnecessary I/O in order to improve processing time.

Skipping the points in the current LSH index structures has several challenges that include: 1) Since hashed points are ordered without considering their groups,

Algorithm 1: Query phase of FairLSH-Basic and FairLSH-Advanced

Input: $\mathcal{D}$ is the dataset; $\vec{q}$ is the query point; $\mathcal{L}$ is the label of dataset points; $\mathcal{I}$ is m index structures created in indexing phase; k is the number of nearest neighbors to find; l is the collision threshold, $\vec{a}$ is the random vector generated in the indexing phase.

Output: $\lfloor k/2 \rfloor$ nearest points to $\vec{q}$ in $\mathcal{D}$ from each label

Variable: cc is the collision count of points; $\mathcal{C}$ is the candidates list; R is the number of nodes to read

```

1  $h(\vec{q}) := \vec{a} \cdot \vec{q}$ ;
2 Find query leaf node in  $m$  index structures;
3 Let  $R = 1$ ;
4 while TRUE do
5   for  $i = 1$  to  $m$  do
6     Read  $R$  nodes around the query node;
7     if  $\lfloor k/2 \rfloor$  label  $A$  nearest neighbors are found then
8       Set next round nodes to only label B nodes;
9     else
10      Set next round nodes to adjacent nodes;
11     foreach point  $\vec{o}$  in leaf node do
12        $cc[\vec{o}] := cc[\vec{o}] + 1$  if  $cc[\vec{o}] \geq l$  then
13         Add  $\vec{o}$  to  $\mathcal{C}$ ;
14   if  $(|\{o \in \mathcal{C} \wedge \mathcal{L}[o] = 1 \wedge \|o - q\| \leq c \times R\}| \geq \lfloor k/2 \rfloor)$  and
       $(|\{o \in \mathcal{C} \wedge \mathcal{L}[o] = 2 \wedge \|o - q\| \leq c \times R\}| \geq \lfloor k/2 \rfloor)$  then
15     break;
16    $R := R \times c$ ;
17 return  $\{o \in \mathcal{C} \wedge \mathcal{L}[o] = 1 \wedge \|o - q\| \leq c \times R\}$  and
       $\{o \in \mathcal{C} \wedge \mathcal{L}[o] = 2 \wedge \|o - q\| \leq c \times R\}$ 

```

when reading a “page-size” of data from the disk, we might get points from different groups (e.g. node 2 of Figure 1(a)), and 2) The current index structures only have pointers to the sibling nodes and there is no possibility to avoid certain nodes of hash functions (that contain unnecessary data) in the index structure. *FairLSH-Basic* uses a *group-aware* strategy when creating the leaf nodes to only allow points belonging to the same group to be added to a single leaf node and to create a new leaf node when a new point belongs to a different group. Furthermore, *FairLSH-Basic* uses threaded B+-tree structures that allow arbitrary pointers between different nodes of the tree. In the current version of *FairLSH-Basic*, these arbitrary pointers are created between leaf nodes belonging to group B and the idea of using smart pointers where the algorithm can detect which nodes require pointers between them is left for future work.

In the query phase of *FairLSH-Basic*, the goal is to find $\lfloor k/2 \rfloor$ nearest points from each group to a given query point. Similar to other tree-based LSH methods, the leaf nodes are searched in an exponential search radius manner (starting from the query node) and the collision counting process (explained in section 3) is

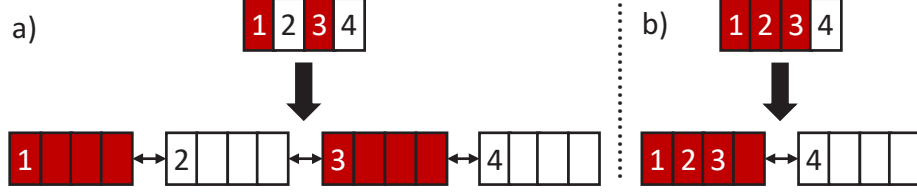


Fig. 2: Breakdown of node containing (a) sparse and (b) dense groups

carried away. However, when enough nearest neighbors are found from a group and we only need to get nearest neighbors from the other group, *FairLSH* uses its arbitrary pointers to skip unnecessary nodes instead of doing a range query (line 8 of Algorithm 1).

An example of how the search process of *FairLSH-Basic* works is shown in Figure 1 (b). In this example, the query resides in Node 1, white points are from group A, and red points are from group B. With an assumption of $k = 8$, our goal is to find four nearest neighbors from each group. *FairLSH-Basic* starts by reading Node 1 and fetching the first four points, and now, we have enough neighbors from group one (assuming that the query is also a point from group one). After that, *FairLSH-Basic* reads Node 2 and fetches two points from group two. From this moment, since we only need to read group two points, the algorithm jumps to Node 6 and reads the remaining points from that node. This way, *FairLSH-Basic* has saved the I/O cost of reading three unnecessary nodes (3, 4, and 5). Note that the indexing phase prevents points from multiple groups to be in the same node (e.g. points 17, 18, 19, and 20 in one node); therefore, *FairLSH-Basic* can always skip unnecessary nodes.

It is worth mentioning that Figure 1 is only showcasing one projection as a simple example; in the real scenario, we have several hash projections with more complex distributions. In Section 6, we show that *FairLSH-Basic* performs much better than state-of-the-art techniques.

5.3 Drawbacks of FairLSH-Basic

The main benefit of *FairLSH-Basic* is that it is effective in reducing disk I/O costs when points from different groups are not sparsely distributed in the nodes. Although the indexes are created offline, *FairLSH-Basic* has a processing overhead in the query phase which is related to the extra pointers between the nodes (compared to only performing a range query search). However, this processing overhead is negligible compared to the savings in disk I/O cost, especially when we have a dense distribution of group points. In this scenario, as shown in Figure 2 (a), *FairLSH-Basic* breaks down the nodes containing a mixed group of points (i.e. points from group A and group B) and eliminates the need to read unnecessary points from the indexes.

On the other hand, the processing overhead of *FairLSH-Basic* increases when points from different groups are sparsely distributed in the nodes. Figure 2(b)

shows an example of this scenario. In this scenario, *FairLSH-Basic* breaks down the nodes which results in multiple nodes containing only one point. As a result, the indexes will contain more pointers and more “book-keeping” is required during the query processing phase. We also observed that in reality, both of the mentioned scenarios (i.e. sparse and dense distribution) happen in the indexes. In other words, different hash functions in the same index structure can have nodes containing a sparse distribution of group points, and also nodes containing a dense distribution of group points.

5.4 Design of FairLSH-Advanced

To remedy the drawbacks of *FairLSH-Basic*, we present a cost-based strategy, *FairLSH-Advanced*, that can smartly detect if breaking down a node is going to positively or negatively affect the overall performance.

There are three costs associated with reading a node into the main memory: 1) C_s : cost of disk seeks, 2) C_h : cost of reading the header (i.e. pointers), and 3) C_p : cost of reading the payload (i.e. points). C_s is defined as $IO_{calls} \times seek_{speed}$, where $seek_{speed}$ is the time it takes to perform a disk seek and can be obtained from disk manufacturer or by benchmarking the disk. C_h is defined as $node_{count} \times node_{headersize} \times IO_{speed}$, where $node_{count}$ is the number of nodes we are going to read, $node_{headersize}$ is the header size of each node, and IO_{speed} is the time it takes to read data from the disk and can be obtained from disk manufacturer. Finally, C_p is defined as $points_{count} \times IO_{speed}$, where $points_{count}$ is the number of points that we want to read from the disk.

Given a node in the indexing phase, the goal of *FairLSH-Advanced* is to decide whether breaking down a node is going to be beneficial or not. Therefore, we introduce a cost to represent each scenario (i.e. C_{Before} and C_{After} for the cost before and after breaking down the node respectively) and the difference of these two costs can be used to make the decision. Thus, we have:

$$C_{Before} = C_{s,B} + C_{h,B} + C_{p,B} \quad (1)$$

$$C_{After} = C_{s,A} + C_{h,A} + C_{p,A} \quad (2)$$

Note that $C_{s,B}$, $C_{h,B}$, and $C_{p,B}$ are the cost of disk seeks, cost of reading the header, and cost of reading the payload respectively before we break down a node, and $C_{s,A}$, $C_{h,A}$, and $C_{p,A}$ are the costs after we break down a node.

As an example, we consider the nodes in Figure 2 and assume that $seek_{speed} = 6$, $node_{headersize} = 3$, and $IO_{speed} = 13$. In Figure 2(a), before breakdown, we have $IO_{calls} = 1$, $node_{count} = 1$, and $points_{count} = 4$. When we break the node down, we have $IO_{calls} = 4$, $node_{count} = 4$, and $points_{count} = 4$. Therefore, we have $C_{Before} = (1 \times 6) + (1 \times 3 \times 13) + (4 \times 13) = 97$, $C_{After} = (4 \times 6) + (4 \times 3 \times 13) + (4 \times 13) = 232$, and $C_{After} - C_{Before} = 232 - 97 = 135$. Similarly, in Figure 2(b), we have $C_{Before} = (1 \times 6) + (1 \times 3 \times 13) + (4 \times 13) = 97$, $C_{After} = (2 \times 6) + (2 \times 3 \times 13) + (4 \times 13) = 142$, and $C_{After} - C_{Before} = 142 - 97 = 45$.

$$\begin{cases} \text{break down,} & \text{if } C_{After} - C_{Before} \leq \theta \\ \text{do not break down,} & \text{if } C_{After} - C_{Before} > \theta \end{cases} \quad (3)$$

FairLSH-Advanced utilizes a user-input parameter, called θ , in the indexing phase. As shown in Equation 3, if the $C_{After} - C_{Before}$ of a node is lower than θ , the node will be broken down and vice versa. Since nodes have different costs, it is crucial to find a good θ value such that the index will be efficient enough in the query processing phase. Note that the query processing phase of *FairLSH-Advanced* is similar to *FairLSH-Basic* (Algorithm 1). In Section 6, we show how using the cost-based strategy and the user-input parameter can improve the performance of *FairLSH-Basic*.

6 Experimental Evaluation

In this section, we evaluate the effectiveness and fairness of our two proposed methods, *FairLSH-Basic* and *FairLSH-Advanced*. All experiments were run on a machine with the following specifications: Intel Core i7-6700, 16GB RAM, 2TB HDD, and Ubuntu 20.04 OS. All codes were written in C++ and compiled with gcc v9.3.0 with the -O3 optimization flag. Since the code of PM-LSH was not released when writing this paper, we compare our two strategies with the following state-of-the-art disk-based alternatives:

- **C2LSH**: Fair top-k results are found using C2LSH [11].
- **QALSH**: Fair top-k results are found using QALSH [15].

We modified existing state-of-the-art algorithms (C2LSH and QALSH) to output fair nearest neighbors by using the naive strategy explained in Section 5.1.

6.1 Datasets

Due to the lack of availability of high-dimensional real-world datasets that target fairness, we create synthetic datasets for our experimental setup. In order to cover different scenarios that might happen in different applications, we construct seven synthetic datasets. There are two groups in each dataset and the goal of this paper is to give both of these groups the same opportunity (i.e. equal opportunity) to appear in the final results. We assign a binary label (A or B in our explanation) to each dataset point to represent these groups. Each one of the groups contain 50% of the dataset. In this work, we experiment with scenarios where A and B have different distributions. Label A data points are generated using a Beta distribution with $\alpha = 2$ and $\beta = 8$, and label B data points are generated using a Beta distribution with $\alpha = 8$ and $\beta = 2$. For future work, we will experiment with different types of distributions and different values of α and β . Table 1, summarizes the characteristics of our synthetic datasets. We choose 100 random points as our queries and report the average as the final result. We also experimented on datasets with 100,000 points with 100 and 1,000 dimensions, and we observed similar results. Due to space limitations, we do not include those results.

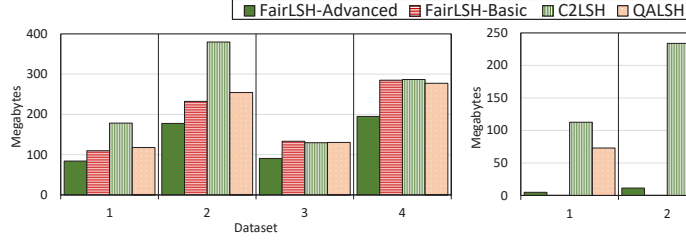


Fig. 3: Amount of Data Read

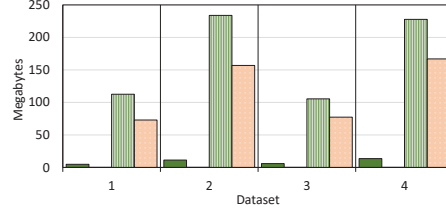


Fig. 4: Amount of Wasted IO

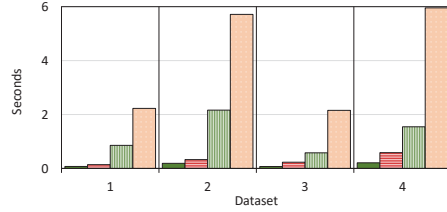


Fig. 5: Algorithm Time

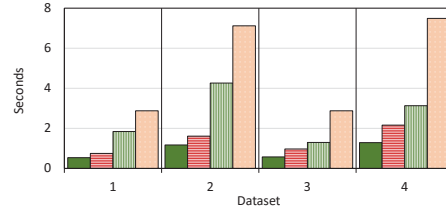


Fig. 6: Query Processing Time

Name	# of Points	# of Dim.
D1	500,000	100
D2	1,000,000	100
D3	500,000	1,000
D4	1,000,000	1,000

Table 1: Characteristics of the datasets

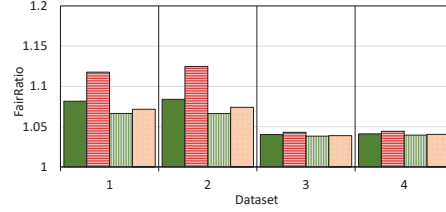


Fig. 7: Fairness Accuracy Ratio

6.2 Evaluation Criteria

The performance and fairness of the compared techniques are evaluated using the following criteria:

- **Index IO Size:** The amount of total data read from index files.
- **Wasted IO Size:** The amount of unnecessary data read from index files (e.g. reading label A points while we have enough label A candidates and should only look for label B candidates).
- **Algorithm Time:** The processing time of index files once they are read into the main memory. The algorithm time consists of operations such as Collision Counting which are explained in section 3.
- **Query Processing Time:** The overall time of finding fair approximate nearest neighbors. We observed that the wall-clock times were not consistent (i.e. running the same query multiple times on the same indexes would return drastically different results, mainly because of disk cache and instruction cache issues). Therefore, following [23], for a Seagate 1TB HDD

with 7200 RPM, we consider a random seek to cost 8.5 ms on average, and the average time to read data to be 0.156 MB/ms. Thus, we have $QPT = \#DiskSeeks \times 8.5 + dataRead \times 0.156 + AlgTime + FPRemTime$, where $FPRemTime$ is the cost of reading the candidate data points and computing their exact Euclidean distance for removing false positives. We do not report individual $FPRemTime$ results since they are similar for different methods and negligible (less than one millisecond).

- **Accuracy:** In order to define an accuracy metric, we consider the Euclidean distance (between the candidate and query) and fairness. The ground truth of our problem is the closest points of each label to the given query. For example, for $k = 100$, if the dataset is split into 50% label A points and 50% label B points, the goal is to find the 50 closest label A points and the 50 closest label B points to the given query. We define our accuracy metric, called FairRatio, as following:

$$FairRatio = \frac{1}{k} \left(\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{\|o_i^A - q\|}{\|o_i^{A,*} - q\|} + \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{\|o_i^B - q\|}{\|o_i^{B,*} - q\|} \right) \quad (4)$$

where o_i^A and o_i^B are the i th label A and label B points respectively that are returned by the LSH technique, and $o_i^{A,*}$ and $o_i^{B,*}$ are the i th label A and label B points from the query in the ground truth. FairRatio of 1 means that the returned results are fair and have the same distance from the query as the ground truth. The closer this value is to 1, the higher the accuracy.

6.3 Parameter Settings

For the state-of-the-art methods, we used the same parameters suggested in their papers. For QALSH, *FairLSH-Basic*, and *FairLSH-Advanced* (since we use the same hashing formula as QALSH as explained in section 5), we used $w = 2.781$, $\delta = 0.1$, and $c = 2$. For C2LSH, we used $w = 2.184$, $\delta = 0.1$, and $c = 2$.

In this work, we focus on only two data point labels (A and B) in the dataset, $k = 100$, and the goal of finding 50 nearest points from label A and 50 nearest points from label B. We leave experimenting on other parameter settings for future work. For *FairLSH-Advanced*, we tried different values of θ and observed improvements over compared methods for all experimented values. Due to space limitations, we only show results for $\theta = 100$.

6.4 Discussion of the Results

In this section, we compare the performance and accuracy of *FairLSH-Basic* and *FairLSH-Advanced* using the criteria explained in section 6.2 against the state-of-the-art methods.

- **Index IO Size:** Figure 3 shows the total amount of data read from the index files. Since C2LSH has more index files compared to other methods, it always

has a higher I/O cost compared to the other methods. *FairLSH-Basic* has a lower amount of data read compared to C2LSH and QALSH since its index structure is designed to not read unnecessary points from the disk. *FairLSH-Advanced* has the lowest amount of index I/O since its index structure is further optimized to skip reading unnecessary node headers.

- **Wasted IO Size:** Figure 4 shows the amount of unnecessary data read from the index files. Wasted I/O happens when, in the query processing phase, enough nearest neighbors from group A are already found; but the algorithm keeps reading data related to group A. The wasted I/O size of *FairLSH-Basic* and *FairLSH-Advanced* are several orders of magnitude smaller compared to C2LSH and QALSH. *FairLSH-Advanced* has a slightly higher wasted I/O size than *FairLSH-Basic* since it is sacrificing wasted I/O over algorithm time.
- **Algorithm Time:** Figure 5 shows the time needed by the algorithms to find the candidates (excluding the time taken to read index files). QALSH has the highest algorithm time for all datasets since it uses non-optimized B+-trees that become significantly larger as the dataset size grows. It is interesting to note that although *FairLSH-Basic* and *FairLSH-Advanced* use more complex tree structures (due to more number of pointers), their algorithm time is lower than C2LSH and QALSH. This is due to avoiding processing of unnecessary nodes which offsets the overhead of using additional tree pointers. *FairLSH-Advanced* has a lower algorithm time than *FairLSH-Basic* since it is using cost models to optimize the index structures and balance the trade-off between wasted I/O cost and algorithm time.
- **Query Processing Time:** Figure 6 shows the overall time required to solve a given k-NN query and retrieve fair neighbors. *FairLSH-Basic* is always faster than C2LSH and QALSH because of savings in wasted I/O and algorithm time. *FairLSH-Advanced* is the fastest method compared to the others since its index structures are optimized to significantly reduce algorithm time while increasing wasted I/O cost slightly.
- **Accuracy:** Figure 7 shows the accuracy of all techniques. C2LSH and QALSH have a similar accuracy for all datasets, and *FairLSH-Basic* and *FairLSH-Advanced* have a slightly lower accuracy (i.e. higher FairRatio). The reason of this difference in accuracy is because when *FairLSH-Basic* and *FairLSH-Advanced* find enough nearest neighbors of a group, they will stop reading and processing points belonging to that group. However, C2LSH and QALSH continue this process and get more nearest neighbors and return the closest k neighbors at the end. We should mention that a difference of 0.05 in FairRatio is very small compared to the savings in I/O costs and algorithm time.

7 Conclusion and Future Work

In this paper, we define the group fairness notion of *Equal Opportunity* in the context of Approximate Nearest Neighbor domain. We proposed a novel index structure for efficiently finding fair top-k approximate nearest neighbors using Locality Sensitive Hashing, called *FairLSH*. Existing LSH-based techniques are

not capable of efficiently finding fair nearest neighbors to a given query. We proposed two novel strategies, *FairLSH-Basic* and *FairLSH-Advanced*, which uses threaded B+-trees and advanced cost models to optimize the overall query processing cost. Experimental results show the benefit of our proposed structures over state-of-the-art techniques. In the future, we plan to introduce a user-defined parameter to adjust the trade-off between fairness, accuracy, and processing time. We also plan to provide theoretical guarantees for the results of *FairLSH*.

References

1. Agarwal, A., et al., “A reductions approach to fair classification,” *arXiv* 2018.
2. Aumüller, M., et al., “Fair near neighbor search via sampling,” *SIGMOD Record* 2021.
3. Aumüller, M., et al., “Fair near neighbor search: Independent range sampling in high dimensions,” *SIGMOD* 2020.
4. Bera, S., et al., “Fair algorithms for clustering,” *NIPS* 2019.
5. Chávez, E., et al., “Searching in metric spaces,” *CSUR* 2001.
6. Chierichetti, F., et al., “Matroids, matchings, and fairness,” *AISTATS* 2019.
7. Chouldechova, A., “Fair prediction with disparate impact: A study of bias in recidivism prediction instruments,” *Big data* 2017.
8. Datar, M., et al., “Locality-sensitive hashing scheme based on p-stable distributions,” *SOCG* 2004.
9. Donini, M., et al., “Empirical risk minimization under fairness constraints,” *NIPS* 2018.
10. Elzayn, H., et al., “Fair algorithms for learning in allocation problems,” *FAccT* 2019.
11. Gan, J., et al., “Locality-sensitive hashing scheme based on dynamic collision counting,” *SIGMOD* 2012.
12. Gionis, A., et al., “Similarity search in high dimensions via hashing,” *VLDB* 1999.
13. Har-Peled, S., et al., “Near neighbor: Who is the fairest of them all?,” *NIPS* 2019.
14. Hardt, M., et al., “Equality of opportunity in supervised learning,” *NIPS* 2016.
15. Huang, Q., et al., “Query-aware locality-sensitive hashing for approximate nearest neighbor search,” *VLDB* 2015.
16. Jafari, O., et al., “Experimental Analysis of Locality Sensitive Hashing Techniques for High-Dimensional Approximate Nearest Neighbor Searches,” *ADC* 2021.
17. Kleinberg, J., et al., “Human decisions and machine predictions,” *QJE* 2018.
18. Kleindessner, M., et al., “Guarantees for spectral clustering with fairness constraints,” *arXiv* 2019.
19. Liu, W., et al., “I-lsh: I/o efficient c-approximate nearest neighbor search in high-dimensional space,” *ICDE* 2019.
20. Lu, K., Kudo, M., “R2lsh: A nearest neighbor search scheme based on two-dimensional projected spaces,” *ICDE* 2020.
21. Mehrabi, N., et al., “A survey on bias and fairness in machine learning,” *arXiv* 2019.
22. E.O., Munoz, et al., “Big data: A report on algorithmic systems, opportunity, and civil rights,” *Executive Office of the President* 2016.
23. Seagate ST2000DM001 Manual.: <https://www.seagate.com/files/staticfiles/docs/pdf/datasheet/disc/barracuda-ds1737-1-1111us.pdf>
24. Zheng, B., et al., “Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search,” *VLDB* 2020.