OMNI-HISTOGRAMS: A NOVEL PARTITIONING APPROACH FOR SIMILARITY SEARCH

Marcos V. N. Bedo
Fluminense Northwest Institute/Fluminense Federal University – (INFES/UFF)
Av. João Jasbick, S/N – Aeroporto – S. A. Pádua/RJ – Brazil
marcosbedo@id.uff.br

Rodolfo A. de Oliveira
Fluminense Northwest Institute/Fluminense Federal University – (INFES/UFF)
Av. João Jasbick, S/N – Aeroporto – S. A. Pádua/RJ – Brazil
rodolfooliveira@id.uff.br

Daniel S. Kaster
Dept. of Comp. Science/State University of Londrina – Brazil – (DC/UEL)
Celso Garcia Cid, S/N – C. Universitário – Londrina/PR – Brazil
dskaster@uel.br

Daniel C. M. de Oliveira
Institute of Computing/Fluminense Federal University – (IC/UFF)
Gal. Milton Tavares de Souza, S/N – S. Domingos – Niterói/RJ – Brazil
danielcmo@ic.uff.br

Caetano Traina Jr.
Institute of Math. and Comp. Sciences/University of São Paulo – (ICMC/USP)
Av. Trab. São Carlense, 400 – Centro – S. Carlos/SP – Brazil
caetano@icmc.usp.br

ABSTRACT

Similarity searching supports several computational tasks, such as classification and content-based retrieval. A plethora of indexes has been proposed aiming at enhancing similarity queries, being the Omni-family one of the most versatile. The main strength of Omni methods is they handle the data elements regarding a small set of carefully selected pivots. In this study, we improve the Omni-family and create a new class of indexes called *Omni-histograms*. Our approach summarizes distance distributions to Omni-pivots in such a way histograms' buckets are also employed for the partitioning of the search space into disjoint regions. The resulting structures boost query executions by using the frequency within each region for limiting both disk accesses and distance calculations. Experiments on real-world datasets showed our approach outperforms existing methods in up to 113%.

Keywords: Similarity search; access methods; histograms; k-NN queries; pivot selection.

Como Citar:
BEDO, M.; OLIVEIRA, R.; KASTER, D.; OLIVEIRA, D.; TRAINA JR., C.. Omni-Histograms: A novel partitioning approach for similarity search. In: SIMPÓSIO DE PESQUISA OPERACIONAL E LOGÍSTICA DA MARINHA, 19., 2019, Rio de Janeiro, RJ. **Anais** [...]. Rio de Janeiro: Centro de Análises de Sistemas Navais, 2019.

1. INTRODUCTION

Similarity searching is one of the most employed paradigms for the handling and querying of data that are "alike" but not "equal". The paradigm supports a broad variety of computational tasks, such as clustering, classification, and content-based retrieval [1, 2, 3]. In those tasks, a type of query that is often requested in practice is the k-nearest neighbor (k-NN) query. Examples of k-NN searches include: **(Q1)** Find the 3 closest beaches to 'Copacabana Beach', and **(Q2)** Find the 5 paintings of the 'Renaissance period' which are the most similar to 'Mona Lisa'. Notice **(Q2)** contains a filtering condition on the orthogonal attribute 'Art Period', which limits the candidates to the query answer [4, 5].

Different metric access methods have been proposed to speed up similarity-based queries [6, 7, 8]. Such methods accelerate similarity searches by targeting an optimization criterion, such as the number of disk accesses, the number of distance calculations, or the overhead caused by the querying algorithm [9]. One of the most versatile strategies for improving k-NN search is the Omni approach [8], which aims at reducing both distance calculations and disk accesses. It uses a few, but relevant, elements from the dataset as *pivots* to create new and multidimensional representation for each data element [10]. If an underlying access method is employed for the handling of these new representations, then distance calculations are avoided by following both the triangle inequality distances to the pivots and the pruning rules of the underlying structure. Therefore, the Omni approach can be implemented on top of disk-based indexes, such as B-Tree [11], R-Tree [12], or even Sequential Scan. Notice a new access method is generated whenever the Omni approach is coupled to an existing one, which creates the Omni-family of access methods [8].

Traditionally, statistics derived from distances between data elements are employed for enhancing the execution of similarity searches [13, 14, 15]. Moreover, recent studies indicate distance distributions characterize the effectiveness of metric indexing strategies [13, 16, 17]. Seizing the performance enhancement brought by the Omni approach, this paper employs pivot-based distance histograms for the creation of a new class of access methods, which we call *Omni-histograms*. They distinguish themselves by their *partition constraints*, which leads to distinct organizations of the search space and different performances in the execution of k-NN queries. In this paper, we focus on five Omni-histogram variants that cover most of the partition strategies, namely `Equi-Width`, `Equi-Depth`, `V-Optimal`, `Compact-distance Histogram` and `Curve-Fitting`. The idea behind our structures is the creation of disjoint and pivot-indexed regions, which are also the bucket boundaries for histograms of distance distributions collected from the pivots' perspective. Accordingly, the frequency within each region is available before the query execution and enables the bounding of both disk accesses and distance calculations.

We performed extensive experiments on real-world datasets for comparing the performance of Omni-histograms and previous Omni-family methods in the task of executing k-NN queries with and without orthogonal attributes. The results indicate Omni-histograms achieved significant gains for queries in arbitrary search spaces. Therefore, the contributions of this study are summarized as follows:

1. We introduce the Omni-histogram class of access methods. Omni-histograms organize the search space according to pivot-based distance distributions, and
2. We propose a bounded and incremental k-NN search algorithm over Omni-histograms whose parameters are automatically calculated.

The remainder of the paper is organized into four sections. Section 2 provides the background on similarity searching and discusses related work. Section 3 introduces the Omni-histograms, their settings, and algorithms. Section 4 provides an evaluation of Omni-histograms, while Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1. Similarity searching

Similarity searching is the information retrieval process in which the query includes an element of a domain, and the answer is composed of a set of elements of the same domain that are somehow similar to the query instance [18]. Among the several types of similarity queries, two are the most basic, namely the range query and the k-nearest neighbor (k-NN) query [1, 9]. Let $\mathbb{S}$ be a data domain, $S \subseteq \mathbb{S}$ be a set of elements, and δ be a metric that holds the properties of symmetry, non-negativity, and triangle inequality, then the pair $\langle \mathbb{S}, \delta \rangle$ is a *metric space* in which similarity searches are performed [19]. Given a query element s_q, a range query Rq retrieves all elements of S which are at most a given threshold $\xi \in \mathbb{R}_+$ from s_q such that $Rq\,(S, s_q, \xi) = \{s_i \in S \mid \delta(s_i,\ s_q) \leq \xi\}$. In contrast, k-NN queries return a quantity k of elements whose distance to the query element s_q are the smallest. Therefore, a k-NN query is a variation of the Rq query, *i.e.*, a Rq with a set radius ξ such that $|Rq| = k$ [14]. The pair $\langle s_q,\ \xi \rangle$ defines a *closed query ball* in the search space, which covers more or fewer elements according to ξ. Therefore, small values of ξ may lead to empty result sets, whereas if ξ is indiscriminately increased, all elements of S can be returned [2, 20]. Since k-NN queries enable to control the result cardinality, reducing their execution time is the focus of our investigation.

2.2. The Omni-family of access methods

Pivot-table strategies [9, 10] rely on precomputing and storing distances $\delta(s_i, p)$ of the elements $s_i \in S$ to a set of pivots $p \in \mathcal{P}$. Therefore, given a range query $Rq\,(S, s_q, \xi)$, the triangle inequality property of metric distance functions ensures elements $s_j \in S$ outside the query ball $\langle s_q,\ \xi \rangle$ comply with pruning rule $|\delta(s_j, p) - \delta(p, s_q)| > \xi$ for at least one pivot $p \in \mathcal{P}$ [18]. The Omni approach [8] combines such a pruning rule with the clustering of precomputed distances to reduce both distance

calculations and random disk accesses by means of the querying algorithm. The approach clusters the precomputed distances by means of a broad set of underlying access methods so that incremental k-NN searches [21] can be executed. Accordingly, both the triangle inequality distance to the pivots and the pruning rules of the underlying access method are used for avoiding distance calculations. Omni-pivots $p \in \mathcal{P} \subseteq S$ are fetched in linear time by the Omni Hull-Foci algorithm, and the number of pivots is calculated as the dataset fractal dimension ($\lceil \mathcal{D} \rceil$) [8]. Every element $s_i \in S$ is mapped into an *Omni-coordinate* through $\mathcal{P}$, as in Definition 1. Figure 1 provides an example of such mapping for (a) one, and (b) two pivots.

Definition 1 (Omni-coordinate)**.** *Given an ordered set of pivots $\mathcal{P}$ and an element $s_i \in \mathbb{S}$, the Omni-coordinate $O(s_i)$ of s_i is the set of distances from s_i to each pivot $p \in \mathcal{P}$ such that $O(s_i) = \{\delta(s_i, p_1), \ldots, \delta(s_i, p_{|\mathcal{P}|})\}$. The set of Omni-coordinates of all elements s_i in a dataset S is denoted by O_S.*

As for Omni implementation, the approach relies on an abstraction layer that includes S, $\mathcal{P}$, O_S and a map between S and O_S. Any existing access method can become part of the *Omni-family* by extending the abstraction layer and providing a partition to O_S.

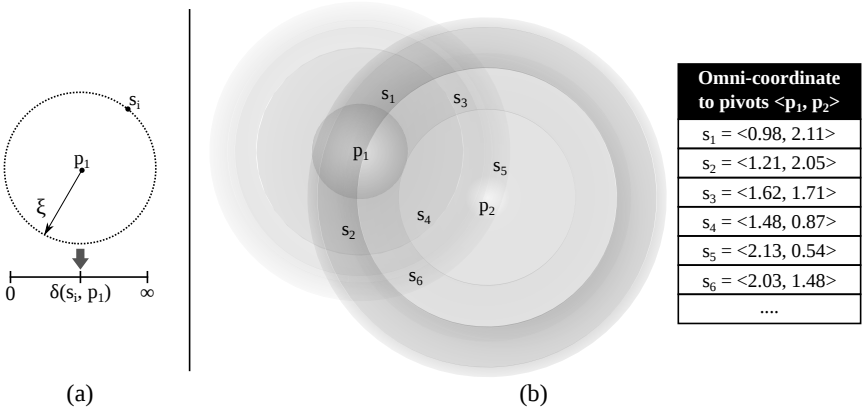


Figure 1: Mapping of a metric space into a multidimensional space according to the L_2 distance. (a) Map of distances for a single pivot p_1 to a one-dimensional space. (b) Map and distance distribution for 100 random points and two pivots.

2.3. Solving of k-NN Queries by Best-First Search

Unlike range queries, k-NN queries do not have a radius defined beforehand and, therefore, the access method cannot draw a query ball for limiting the search space [14, 15]. Alternatively, clustered access methods' algorithms employ a branch-and-bound strategy for the pruning of regions during the query [6]. Such a strategy initially sets the radius to a maximum ($\xi = \infty$) and dynamically reduces it until the k-nearest neighbors have been found. For a faster radius reduction, regions are evaluated in order of proximity to the query element, which results in a procedure known as *best-first* search (bf-kNN).

An optimization of bf-kNN is the estimation of the initial radius ξ', which delimits a search region, and reduces the number of disk accesses and distance

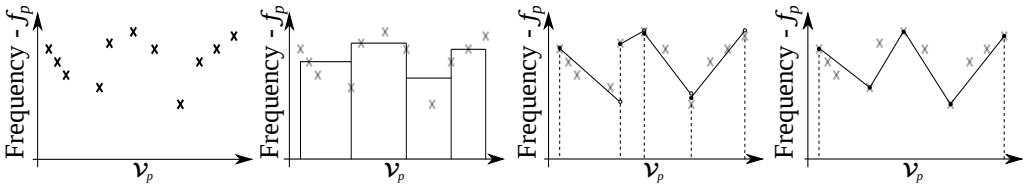


Figure 2: Histograms of $\mathcal{T}_p$ with a fixed number of buckets and distinct partition constraints. (a) Original pivot-based distribution. (b) `V-Optimal` histogram. (c) `Curve-Fitting` histogram. (d) `Compact-distance` histogram.

calculations. In this case, estimated radius ξ' and s_q define the query ball, and the access method executes a combination of range and bf-kNN to return the k closest elements [14] – A procedure we call *limited bf-kNN*. Radius estimation depends on metric statistics and requires the gathering of distance distributions [14, 19]. Such statistics can be calculated following *viewpoints*, in the form of pivot-based distance distributions, which are formally given by Definition 2.

Definition 2 (Pivot-based distance distribution – $\mathcal{T}_p$)**.** *Given a dataset S, a metric δ, and a pivot $p \in \mathcal{P}$, $\mathcal{T}_p$ captures the distance from each $s_i \in S$ to p. Distance value set $\mathcal{V}_p$ contains the distinct and sorted values of $\delta(s_i,\ p)$, i.e., $\mathcal{V}_p = \{v_p(j) : 1 \leq j < m_p,\ m_p \leq |S|\}$, where $v_p(m_p)$ is the largest distance between any s_i to p. Frequency $f_p(j)$ is the number of elements of S whose distance $\delta(s_i,\ p) = v_p(j)$. Therefore, $\mathcal{T}_p$ is the ordered set of pairs $\mathcal{T}_p = \{\langle v_p(1), f_p(1)\rangle, \ldots, \langle v_p(m_p), f_p(m_p)\rangle\}$, and $\mathcal{T}_p^+$ is the extension of $\mathcal{T}_p$ to $\mathbb{R}$ by setting 0 as the frequency of any $v_p \in \mathbb{R}_+ \setminus \mathcal{V}_p$.*

Histograms [22] can be employed for the summarization of pivot-based distance distributions. In this case, the histogram sort and source parameters are $\mathcal{V}_p$ and the set of frequencies, respectively. Distances within each bucket are uniformly distributed, but frequencies are approximate according to a user-posed *histogram partition constraint*. Hence, a histogram $H_p = \{b_1, \cdots, b_\beta\}$ partitions either $\mathcal{T}_p$ or $\mathcal{T}_p^+$ into β mutually disjoint *buckets*. Each bucket b_i covers a range of the value set such that $0 \leq low(b_i) < up(b_i) \leq low(b_{i+1}) < up(b_{i+1}) < \infty$, where $low(b_i),\ up(b_i) \in \mathcal{V}_p$. For the partitioning of $\mathcal{T}_p^+$ an extra bucket $b_{\beta+1}$ is employed in the interval $\big(up(b_\beta), \infty\big)$ with frequency 0. Examples of classical partition constraints include the `Equi-Width` and `Equi-Depth` histograms [13, 22, 23] constructed with uniformly spaced buckets regarding v_p (distance value) and f_p (frequency), respectively. More complex constraints involve the optimization of an objective function in the partitioning of the source parameter. For instance, `V-Optimal` histograms partition $\mathcal{T}_p$ by minimizing the variance of the frequencies within each bucket. `Curve-Fitting` histograms [23] improve `V-Optimal` by approximating the frequencies through a set of polynomial functions, while `Compact-distance` histograms [13] simplify `Curve-Fitting` histograms by approximating $\mathcal{T}_p$ as a continuous piecewise linear function. Figure 2 shows examples of histograms for $\mathcal{T}_p$ following the former three partition constraints, which generate different approximation errors.

2.4. Limited Best-First Search *vs.* Incremental Search

Another approach for the execution of k-NN queries is the incremental search [21], a procedure known as *inc-kNN*. Such a strategy limits the number of

distance calculations based on an optimality principle. The idea is to incrementally retrieve the closest element to the query instance by using two priority queues. The first queue sorts the partitions to be evaluated, while the second one sorts the elements of the already examined regions, which include the next potential nearest neighbor. Elements in the second priority queue are sorted by their distances to the query instance, while the partitions in the first queue are sorted by the minimum and maximum distances between their boundaries and the query element. The decision to select the next nearest neighbor is made upon the evaluation of both queues. If the top of the second queue is closer to the query instance than the minimum distance of the region on top of the first queue, then the first element of the second queue is the next nearest neighbor. Otherwise, the partition of the first queue must be loaded from disk and its elements be inserted into the second priority queue.

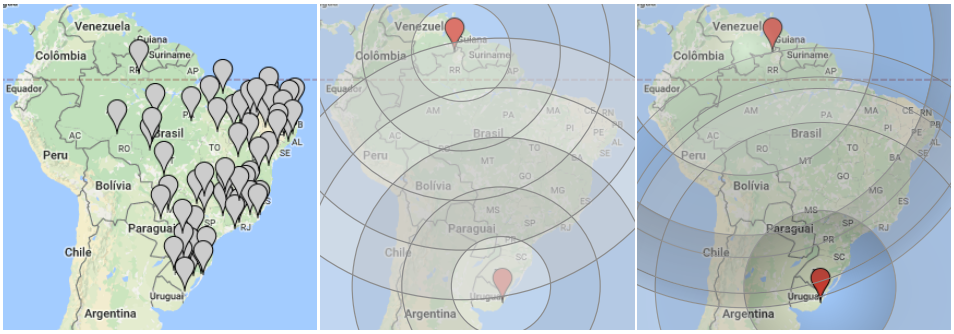
One advantage of inc-kNN is it enables solving queries with orthogonal attributes, as in the query example **Q2** about *the nearest Renaissance paintings most similar to 'Mona Lisa'*, a feat that *limited* bf-kNN is unable to accomplish. However, inc-kNN may require an execution time greater than *limited* bf-kNN in other cases [14, 15]. It happens because inc-kNN must handle two expensive priority queues (especially the second one) for the solving of k-NN queries, which generates an overhead in both memory and processing. Moreover, inc-kNN does not define a query ball and, consequently, it may not benefit from elevator-based disk scheduling that could improve the k-NN search performance.

In this study, we combine the best of both inc-kNN and *limited* bf-kNN approaches in the form of a bounded and incremental k-NN algorithm on top of Omni-histograms. Our approach enables the definition of a query ball and the sorting of partitions and elements into priority queues, which limits on-demand disk accesses and leads to a reduction in the number of distance calculations.

3. THE OMNI-HISTOGRAMS

This section proposes the Omni-histograms, a new and robust class of metric access methods that extends the abstraction layer of the Omni-family. The addition of histograms into the Omni approach enables both the partitioning of Omni-coordinates and k-NN search optimization. In particular, Omni-histograms divide the search space into disjoint regions by using the Omni-pivots, whereas the final number of regions depends on the number of buckets for each pivot-based histogram. The following parameters define a specific instance of an Omni-histogram: *(i)* the number of pivots $|\mathcal{P}|$, *(ii)* the maximum number of buckets β, and *(iii)* a histogram partition constraint $\mathcal{R}$. The overall idea is to take advantage of the Omni abstraction layer for the gathering of extended pivot-based distributions so that they can be clustered according to the histogram partition constraint.

Therefore, an Omni-histogram can be seen as the set of non-independent pivot-based distance histograms derived from the set of Omni pivots. Formally, an Omni-histogram $\mathcal{H}$ is defined as a set of pairs $\mathcal{H} = \{\langle p,\ H_p\rangle,\ p \in \mathcal{P}\}$, where H_p is the histogram partitioning of $\mathcal{T}_p^+$. Figure 3 shows an example of a 2D dataset with geographical coordinates of Brazilian cities and its partitioning by two distinct



(a) (b) (c)

Figure 3: A dataset with geographical coordinates where most cities are closer to the coast. Histograms are constructed for two pivots, $\beta = 5$ buckets, and L_2 distance. The darker regions are the denser ones. (a) A sample of 60 *medoids*. (b) `Equi-Width Omni-histogram` regions. (c) `Compact-distance Omni-histogram` regions.

Omni-histograms. Figure 3(a) presents a sample of the spatial distribution of 60 medoids found by the k-medoids clustering algorithm, while Figures 3(b) and (c) show the `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` built for the same dataset with regards to L_2 distance.

An *Omni-bucket* b^* covers a hyper region in the search space defined by a set of buckets from distinct pivot-based histograms in $\mathcal{H}$. The boundaries of an *Omni-bucket* b^* are the limits of buckets $b_i \in H_p$ related to b^* and $\mathcal{H}$ by pivots $p \in \mathcal{P}$. Each element of S falls into only one Omni-bucket calculated by the distance between the element and $\mathcal{P}$. The frequency of each Omni-bucket is the number of elements which lie inside the corresponding hyper region. We call *Omni bucket-coordinate* the mapping between an element $s_j \in \mathbb{S}$ and its Omni-bucket as in Definition 3.

Definition 3 (Omni bucket-coordinate)**.** *Given an Omni-histogram $\mathcal{H}$ and an element $s_j \in \mathbb{S}$, the Omni bucket-coordinate $B(s_j)$ of s_j addresses the Omni-bucket b^* of $\mathcal{H}$ whose limits include s_j. Therefore, $B(s_j) = \{\langle p, b_i\rangle \mid \forall\ \langle p, H_p\rangle \in \mathcal{H}, b_i \in H_p;\ low(b_i) \leq \delta(s_j, p) < up(b_i)\}$. If the order for pivot set $\mathcal{P}$ is fixed, then $B(s_j)$ can be written as $B(s_j) = \{b_{i1}, \ldots, b_{i|\mathcal{P}|}\}$. The set of Omni bucket-coordinates regarding all elements of S is denoted B_S.*

Omni-histograms organize the search space into three levels. The first level includes the Omni-buckets, B_S and a map between O_S and B_S ($\mathcal{M}(O_S, B_S)$, for short). The second level includes the Omni-coordinates O_S and a map between S and O_S ($\mathcal{M}(S, O_S)$, for short). Finally, the third level includes all the elements of S and their mapping to O_S and B_S ($\mathcal{M}(O_S, B_S)$, for short). Figure 4 shows an example of the relationship between Omni-bucket coordinates and Omni-coordinates.

3.1. Generalized Omni-Histograms

The frequency of an Omni-bucket b^* can be expressed as the distribution of elements within b^* regarding any orthogonal attribute A. In this case, the frequency of b^* itself is a histogram on A regarding the data distribution of elements inside the hyper region delimited by b^*. We generalize Omni-histograms by expressing the

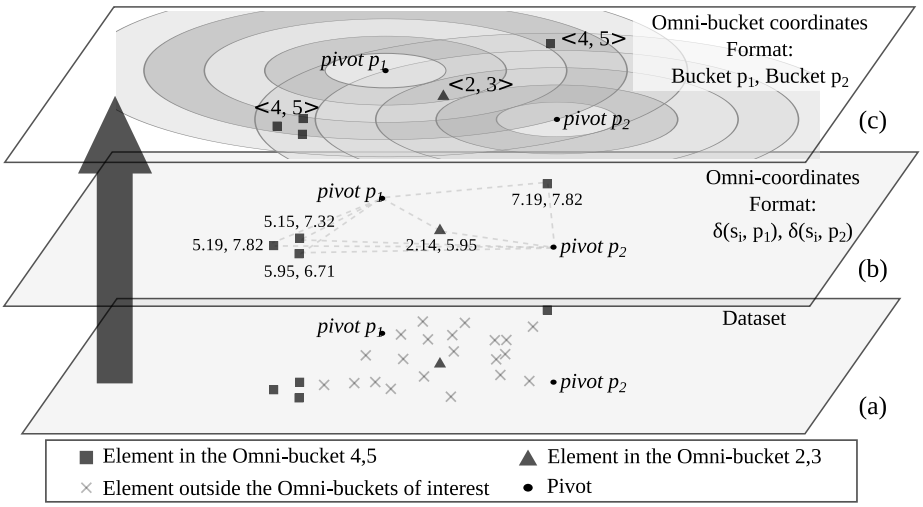


Figure 4: Omni-histogram partitions for Omni-buckets $\langle 2, 3\rangle$ and $\langle 4, 5\rangle$ regarding pivots p_1 and p_2. (a) Data elements. (b) Instances are mapped according to its Omni-coordinates that define two distributions: from p_1, and p_2. (c) Distance distributions are partitioned into Omni-buckets that reference their elements.

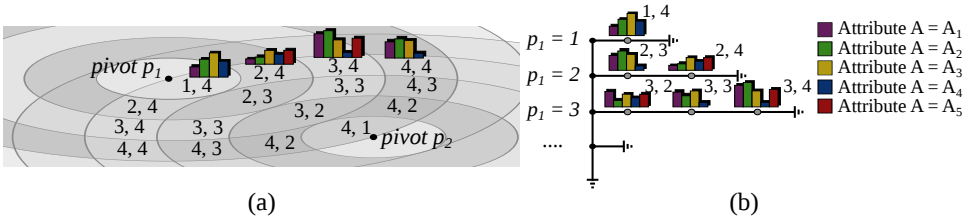


Figure 5: Structure of a generalized Omni-histogram. (a) `Equi-Width Omni-histogram`. (b) Adjacency list for `Equi-Width` histograms on a orthogonal attribute, which are the frequencies of Omni-buckets.

frequency of Omni-buckets as separated histograms, whereas the histogram build for the orthogonal attribute may comply with a partition constraint distinct from that of $\mathcal{H}$. Figure 5 shows an example of a generalized Omni-histogram in which the hyper regions are enumerated by their Omni bucket-coordinates, and the frequency of each Omni-bucket is an Equi-Width histogram on an orthogonal attribute A.

Algorithm 1 constructs a generalized Omni-histogram by using pivot-based distance distributions on attribute S and data distributions on attribute A. The algorithm maps the elements of S into the Omni-buckets by calculating their coordinates. Next, it builds a separated histogram H_A for each Omni-bucket b^* of $\mathcal{H}$ regarding the attribute A by taking into account the elements covered by b^*. Finally, histograms H_A are set as the Omni-buckets' frequencies.

3.2. Solving k-NN Queries with Omni-Histograms

The k-NN search in Omni-histograms is optimized regarding two aspects: *(i)* disk accesses are calculated beforehand as in range queries, and *(ii)* distance calculations are minimized through incremental processing. Omni-histograms estimate

Algorithm 1: `CreateGeneralizedOmniHistogram`$(S, A, \beta_S, \beta_A, \mathcal{R}_1, \mathcal{R}_2)$.

$\mathcal{H} \leftarrow \emptyset$; $\mathcal{P} \leftarrow$ `omni_pivots` (S);
for $p \in \mathcal{P}$ **do**
  Create $H_p($`partition_constraint`$(\mathcal{R}_1), S, \beta_S)$ for $\mathcal{T}_p^+$;
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{\langle p,\ H_p \rangle\}$;
for $s_i \in S$ **do**
  Add $\mathcal{M}(s_i, O(s_i))$; $\mathcal{M}(O(s_i), B(s_i))$; $\mathcal{M}(s_i, B(s_i))$;
for $b^* \in \mathcal{H}$ **do**
  Create $H_A($`partition_constraint`$(\mathcal{R}_2), A, \beta_A)$ regarding b^*;
  Set frequency of b^* as H_A;
return $\mathcal{H}$;

the minimum radius that defines a query ball in which it is ensured at least k elements can be found for the k-NN query, whereas only Omni-buckets that intercept the query ball are load into main memory for evaluation. The minimum number of elements within an Omni-bucket is calculated by using the frequency within the partition and a possible query-imposed filter on the orthogonal attribute. Routine `numElements()` is implemented for such a calculation and returns the accumulated area within the histogram on the orthogonal attribute whose data values comply with query criteria.

Omni-histograms minimize the number of distance calculations for the non-discarded buckets by using two baseline functions: *maxdist*() and *mindist*(). Func-

Algorithm 2: `omni_hist-kNN`(s_q, k, A_c)

$pq_1 \leftarrow \{b^*\}$;`/* Omni-buckets sorted by maxdist() and mindist() */`
$pq_2 \leftarrow \emptyset$; `/* Omni-Buckets sorted by mindist() and maxdist() */`
$\xi' \leftarrow \infty$; $k' \leftarrow k$; RS $\leftarrow \emptyset$; `/* Resulting` s_i `sorted by` $\delta(s_i, s_q)$ `*/`
while *not* pq_1*.empty()* **do**
  $b^* \leftarrow pq_1.pop()$;
  if $k' > 0$ **then**
    $pq_2.push(b^*)$; $k' \leftarrow k' - b^*$.`numElements`(A, A_c); $\xi' \leftarrow b^*.maxdist()$;
  else
    if $b^*.mindist() \leq \xi'$ **then** $pq_2.push(b^*)$;
while *(not* pq_2*.empty()) and (*pq_2*.top().mindist() ¡* ξ'*)* **do**
  $b^* \leftarrow pq_2.pop()$;
  for $s_i \in b^*$ **do**
    if (check(s_i, A, A_c)) *and* $(O(s_i) - O(s_q) \leq \xi')$ *and* $(\delta(s_i, s_q)$ *¡* $\xi')$ **then**
      RS.push(s_i);
      if RS*.size() ¿* k **then**
        RS.removeLastElement(); $\xi' \leftarrow$RS.maxdist();
return RS;

tion $mindist()$ of the query element s_q to an Omni-bucket $b^* \in \mathcal{H}$ is the minimum distance between s_q and a boundary of b^*. It is calculated as $min(|low(b_i) - \delta(s_q, p)|, |up(b_i) - \delta(s_q, p)|)\ \forall\ p \in \mathcal{P}$, where b_i is the bucket of H_p that defines b^*. On the other hand, $maxdist()$ of $b^* \in \mathcal{H}$ is calculated as $max\big(up(b_i) + \delta(s_q, p)\big)\ \forall\ p \in \mathcal{P}$. Therefore, the combination of functions $mindist()$ and $maxdist()$ with routine `numElements()` enables the cutting of partitions that do not include any candidate element to the query answer. In particular, `numElements()` defines the query ball, whereas $mindist()$ enables the incremental evaluations of Omni-buckets covered by the query ball that include the k-nearest elements to s_q.

Algorithm 2 describes a k-NN search on Omni-histograms for a query element s_q and a filter A_c on the orthogonal attribute A. First, it limits the number of disk accesses by selecting and sorting the candidate Omni-buckets that intercept the query ball and proceeds with the goal of delaying the distance calculations as much as possible. Next, Algorithm 2 starts a loop on pq_2 that runs until k-nearest neighbors are found or pq_2 becomes empty. The closest bucket b^* within pq_2 is picked for evaluation, and their elements s_i are verified through the filtering criteria on the orthogonal attribute by *boolean* routine `check`(s_i, A, A_c). If the criteria are satisfied, the algorithm applies the triangle inequality rule $O(s_i) - O(s_q) \leq \xi'$, which verifies if at least one of the precomputed distances satisfies $|\delta(s_i, p) - \delta(p, s_q)| \leq \xi'$.

Therefore, the distance between s_i and s_q is calculated only when the criteria on the orthogonal attribute are satisfied, and no pruning is performed in the Omni-coordinates. In such a case, s_i is inserted into the result set RS, a priority queue sorted by distance $\delta(s_i, s_q)$. If s_i is selected for insertion into RS and the result set has already k elements, the algorithm pushes s_i into RS, removes the last element of the priority queue, and updates the pruning radius ξ'. Figure 6 illustrates the running of Algorithm 2 for query example **(Q2)** that retrieves the five *Renaissance paintings* which are the most similar to 'Mona Lisa'.

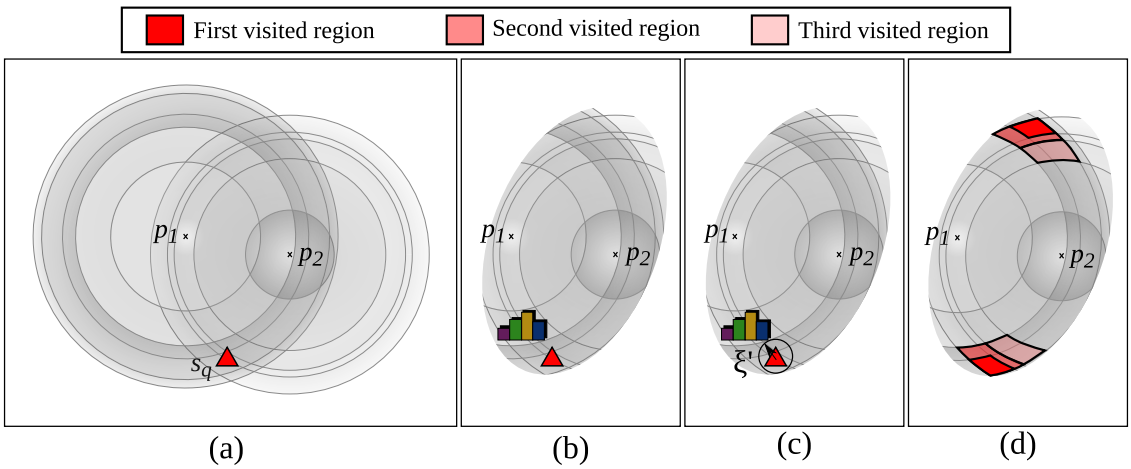


Figure 6: k-NN search example for query element s_q, filter A_c, and distance L_2. (a) Location of s_q Omni-bucket coordinate. (b) Histograms on the orthogonal attribute define the query radius ξ'. (c) Buckets intercepting the query ball are marked for evaluation. (d) Closest regions are visited first.

We assume an `Compact-distance Omni-histogram` was constructed for partitioning *paintings*, whereas the frequencies of each Omni-bucket were described by `Equi-Width` histograms on orthogonal attribute *Art Period*. Accordingly, the first step of Algorithm 2 is locating the Omni-bucket corresponding to the 'Mona Lisa' painting so that all remaining Omni-buckets are sorted to the query element by functions *mindist*() and, then, *maxdist*() (Figure 6(a)). Next, `Equi-Width` histograms on the orthogonal attribute are used for calculating the number of elements within every Omni-bucket that satisfy filtering condition `Art Period = 'Renaissance'` (Figure 6(b)). Accordingly, the sorted list of Omni-buckets is traversed until the accumulated number k' of *Renaissance* elements of visited Omni-buckets become equals or greater than five. At this point, a query ball is defined by using 'Mona Lisa' and a radius ξ' corresponding to the *maxdist*() of the last examined Omni-bucket, and the inspection of the sorted list of partitions stops (Figure 6(c)).

The query ball performs the first pruning of Algorithm 2 so that only the Omni-buckets whose *mindist*() are not greater than ξ' are orderly loaded into main memory. One region is evaluated at a time according to its position in the priority queue (Figure 6(d)). For each inspection, the elements within the Omni-bucket are first evaluated by their Omni-coordinates and the filtering condition, which avoids unnecessary distance calculations. When the first candidate set of k-nearest neighbors sorted by distance to the query element is built, the query radius ξ' is adjusted, and the Omni-buckets of the priority queue in main memory becomes prunable by *mindist*() once again. If the distance of the instance on top of the candidate result set to the query element is lower than *mindist*() of the Omni-bucket on top of the priority queue, then the instance on top of candidate set can be safely returned as the next nearest neighbor. Accordingly, Algorithm 2 may either *incrementally* return the nearest neighbor or retrieve the entire set of k-nearest neighbors as a single final result.

4. EXPERIMENTS

This section reports on three experiments over six real-world datasets, namely `CITIES`[1], `BIKE`[2], `COLORS`[3], `CANVAS`[4], `BANK`[7], and `YEAST`[7]. The first experiment compares distinct histogram partition constraints for the identification of the most suitable settings of Omni-histograms. The second experiment aims at comparing Omni-histograms to access methods Omni-Sequential, Omni R-Tree and Sequential Scan regarding the execution of k-NN queries without orthogonal attributes. Finally, the last evaluation compares the same access methods in the solving of k-NN queries with filtering criteria. Omni-Sequential, Omni R-Tree, and Sequential Scan run inc-kNN algorithm, while Omni-histograms run `omni_hist-kNN` routine.

Table 1 summarizes the datasets characteristics and parameters employed in the trials. All comparisons were performed according to a 10-fold cross validation procedure (90% of data for indexing and 10% of data for querying, cycling) regarding

[1]Available at: `ibge.gov.br`
[2]Available at: `archive.ics.uci.edu/ml/datasets.html`
[3]Available at: `sisap.org/metricspaceslibrary.html`
[4]Available at: `commons.wikimedia.org/wiki/Category:Paintings`

accumulated query execution time. The experiments were executed in a computer with Intel® Core™ *i*7 2.67 GHz, 6 GB of RAM and HDD SATA III 7200 RPM.

Table 1: Datasets and parameters. A is the orthogonal attribute. $|S|$, Dim., $\lceil \mathcal{D} \rceil$ are the set cardinality, dimensionality, and fractal dimension, respectively.

Name	A	$\lvert S \rvert$	**Dim.**	$\lceil \mathcal{D} \rceil$	δ	**Description**
`CITIES`	n/a	$5,507$	2	2	L_2	Geographical coordinates of 5507 Brazilian cities.
`BIKE`	n/a	$17,379$	7	2	L_∞	UCI dataset of bike roads.
`COLORS`	n/a	$112,682$	112	3	L_1	SISAP dataset of features extracted from color images.
`BANK`	Type	$1,372$	4	2	L_∞	UCI dataset of features from bank notes.
`YEAST`	Location	$1,484$	8	4	L_2	UCI dataset of cellular location of proteins.
`CANVAS`	Art Period	$3,879$	16	4	L_1	Features extracted from Wikimedia photos of paintings.

4.1. Comparison of Omni-Histograms Settings

We selected the Omni-pivots according to the Omni Hull–Foci algorithm by setting $|\mathcal{P}| = \lceil \mathcal{D} \rceil$. The constraint number of buckets β was chosen following $\mathcal{P}$ so that all Omni-histograms fit in less than 0.0001% of available memory. In particular, we set five buckets for datasets `CITIES`, `BIKE` and `BANK`, three buckets for dataset `YEAST`, and two buckets for datasets `COLORS` and `CANVAS`. We experimented on five distinct partition constraints that generated five different Omni-histograms,

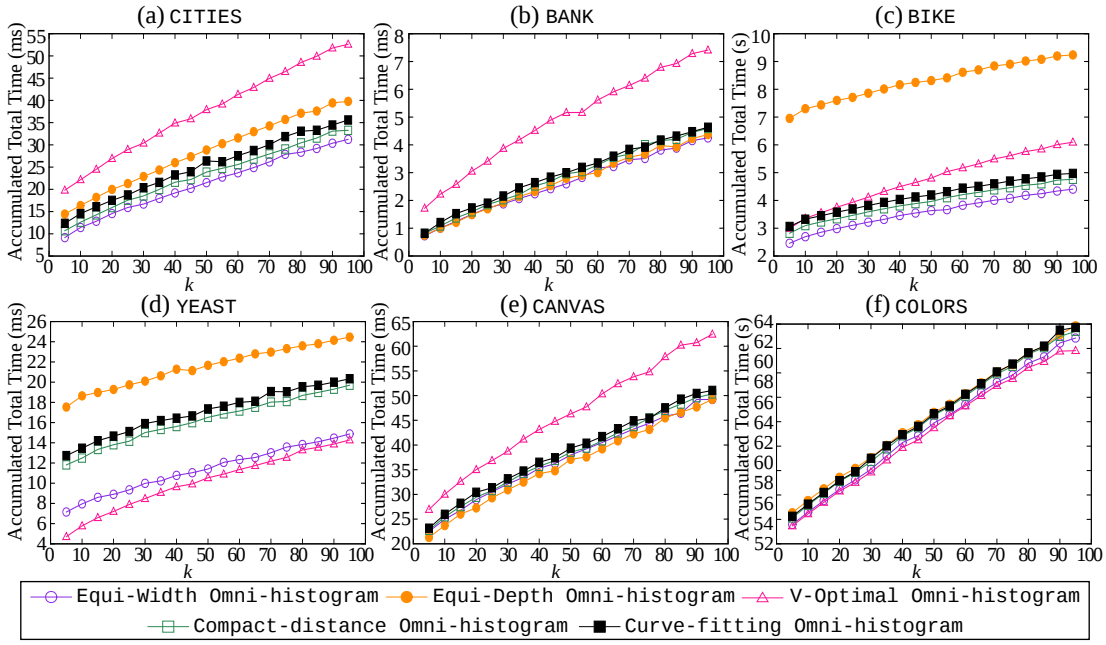


Figure 7: Comparison of distinct Omni-histograms in the execution of k-NN queries.

namely `Equi-Width Omni-histogram`, `Equi-Depth Omni-histogram`, `V-Optimal Omni-histogram`, `Compact-distance Omni-histogram`, and `Curve-Fitting Omni-histogram`. Figure 7 shows the overall comparison of the average time required by each Omni-histogram to execute k-NN queries without orthogonal attributes.

Although `V-Optimal Omni-histogram` was the fastest method regarding datasets `YEAST` and `COLORS`, it was also one of the slowest on the remaining of experimented datasets. `Equi-Depth Omni-histogram` followed a similar behavior, *i.e.*, it was the fastest at solving k-NN queries on `CANVAS`, but showed poor performance on other datasets. On the other hand, `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` showed the most stable behavior, as they consistently achieved one of the top-3 performance regardless of the evaluated dataset. `Equi-Width Omni-histogram`, specifically, achieved the highest performance on datasets `CITIES` and `BIKE`. Therefore, we selected both `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` for the comparison of Omni-histograms to other Omni-family access methods.

4.2. Omni-Histograms *vs.* Other Members of Omni-Family

We compared Omni-histograms to access methods Omni-Sequential, Omni R-Tree, and Sequential Scan. Omni-Sequential is the general purpose method of the Omni-family, *i.e.*, it is implemented on top of Sequential Scan, whereas Omni R-Tree employs the R-Tree for the indexing of Omni-coordinates. Figure 8 shows the comparison between the Omni-histograms and previous Omni methods in the execution of k-NN queries without orthogonal attributes. `Equi-Width Omni-histogram` outperformed Omni-Sequential in up to 113% and 41% on datasets `CITIES` and `BIKE`, respectively. `Compact-distance Omni-histogram` outperformed Omni-Sequential in up to 83% and 30% in the same scenario.

Moreover, `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` outperformed Omni-Sequential for $k \leq 45$ in dataset `COLORS` (112 dimensions). In particular, `Equi-Width Omni-histogram` was up to 5.4% faster than Omni-Sequential on dataset `COLORS`, while `Compact-distance Omni-histogram` was

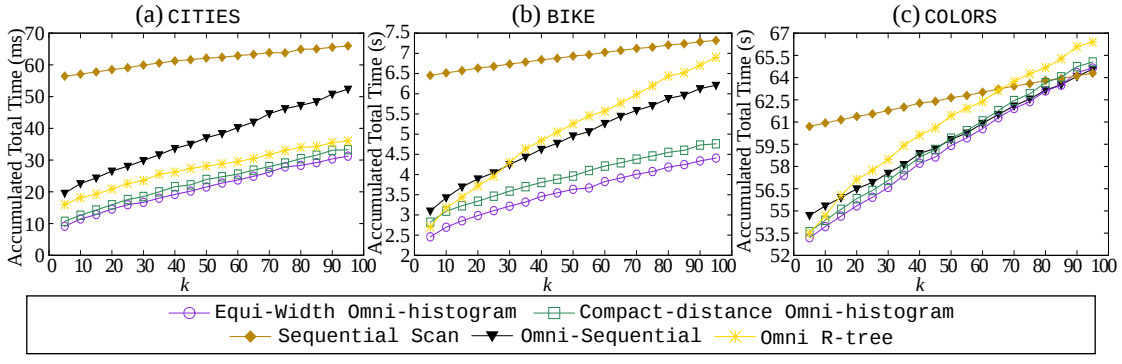


Figure 8: Comparison of Omni-histograms to Omni R-Tree, Omni-Sequential, and Sequential Scan regarding k-NN query execution time.

up to 3.7% faster in the same dataset. Both Omni-histograms outperformed Omni R-Tree in the majority of k values.

We highlight `Equi-Width Omni-histogram` was up to 500%, 160% and 16.1% faster than baseline Sequential Scan on datasets `CITIES`, `BIKE`, and `COLORS`, respectively. Likewise, `Compact-distance Omni-histogram` also outperformed Sequential Scan in up to 443%, 131% and 12.4% for the same datasets.

4.3. k-NN searching with Filtering Criteria

The last experiment provides a comparison between Omni-histograms and their competitors in the task of answering k-NN queries with orthogonal attributes. Each query was in the form of *"Find the k-nearest elements to s_q, where attribute A equals to A_c"*, being the orthogonal attribute A of each dataset described in Table 1. Accordingly, we set $A = A_c$ as in `Type='Class 0'` (55.5% of elements), `Location='mitochondrial'` (16.4% of elements), and `Art Period='Renaissance'` (20% of elements) for datasets `BANK`, `YEAST`, and `CANVAS`, respectively.

Figure 9 shows the comparison between the access methods regarding their execution time. `Equi-Width Omni-histogram` outperformed Omni-Sequential by 30%, on average, for every dataset, while `Compact-distance Omni-histogram` also outperformed Omni-Sequential by 24.3%, on average. Both `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` outperformed Omni R-Tree for every value of k. `Equi-Width Omni-histogram` also outperformed baseline Sequential Scan in up to 456%, 186% and 147% for datasets `BANK`, `YEAST` and `CANVAS`. Likewise, `Compact-distance Omni-histogram` was up to 455%, 164% and 125% faster than Sequential Scan when executing a k-NN query in the same datasets.

Such results indicate Omni-histograms consistently delivered faster executions of k-NN queries, with and without orthogonal attributes, in comparison to the competitors. In particular, `Equi-Width Omni-histogram` and `Compact-distance Omni-histogram` outperformed the former members of the Omni-family, Omni-Sequential and Omni R-Tree by 37.4% and 32.3%, on average, regarding all evaluated scenarios (datasets and values of k).

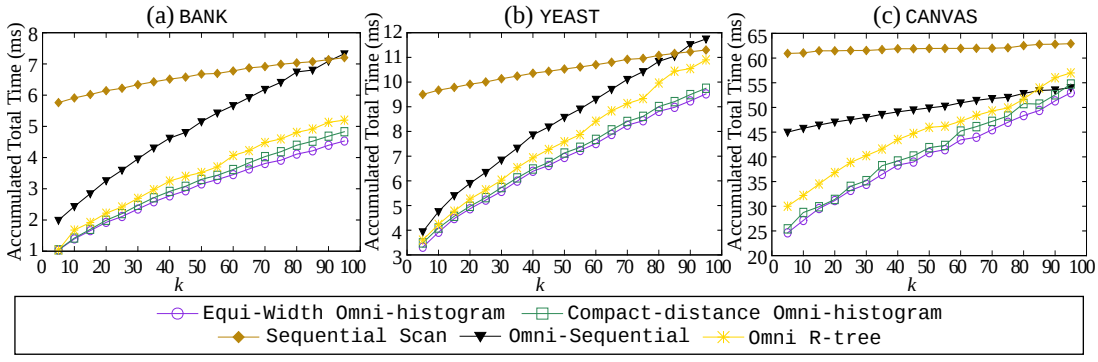


Figure 9: Comparison of Omni-histograms to Omni R-Tree, Omni-Sequential, and Sequential Scan regarding k-NN queries involving an orthogonal attribute.

5. CONCLUSION

In this study, we extended the Omni-family by creating a new class of metric access methods, the Omni-histograms. Such methods distinguish themselves by their partition constraints, which split data elements into disjoint buckets by following an optimization function. The frequency within Omni-histogram regions is represented as either the number of elements or the data distribution of an orthogonal attribute. By using these stored frequencies, our approach enables the use of a bounded and incremental k-NN search, which limits the number of visited regions (bounding disk accesses) and reduces the number of distance calculations. Experiments on real datasets showed k-NN queries (with and without orthogonal attributes) are faster executed by Omni-histograms in comparison to previous Omni methods. In particular, Omni-histograms outperformed Omni-Sequential in up to 113% and Sequential Scan in up to 500%. Future work includes the evaluation of other pivot selection strategies and algorithms in the setting of Omni-histograms.

Acknowledgments. The authors thank FAPERJ (G. E-26/010.101237/2018), CNPq, and CAPES for their financial support.

6. REFERENCES

[1] SILVA, Y. N. et al. Similarity queries: their conceptual evaluation, transformations, and processing. *VLDB*, Springer, v. 22, n. 3, p. 395–420, 2013. 2, 3

[2] YAO, B.; LI, F.; KUMAR, P. k-Nearest Neighbor queries and kNN-Joins in large relational databases (almost) for free. In: *International Conference on Data Engineering*. [S.l.]: IEEE, 2010. p. 4–15. 2, 3

[3] ZEZULA, P. Future trends in similarity searching. In: *International Conference on Similarity Search and Applications*. [S.l.]: Springer, 2012. p. 8–24. 2

[4] ALY, A. M.; AREF, W. G.; OUZZANI, M. Spatial Queries with kNN and Relational Predicates. In: *International Conference on Advances in Geographic Information Systems*. [S.l.]: ACM, 2015. p. 28:1–28:10. 2

[5] WU, D.; CONG, G.; JENSEN, C. S. A framework for efficient spatial web object retrieval. *The VLDB Journal*, Springer, v. 21, n. 6, p. 797–822, 2012. 2

[6] CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: *International Conference on Very Large Data Bases*. [S.l.]: VLDB Endowment, 1997. p. 426–435. 2, 4

[7] SKOPAL, T.; POKORNÝ, J.; SNÁŠEL, V. Nearest neighbours search using the PM-tree. In: *International Conference on Database Systems for Advanced Applications*. [S.l.]: Springer, 2005. p. 803–815. 2

[8] TRAINA JR., C. et al. The Omni-family of all-purpose access methods: A simple and effective way to make similarity search more efficient. *The VLDB Journal*, Springer, v. 16, n. 4, p. 483–505, 2007. 2, 3, 4

[9] CHÁVEZ, E. et al. Searching in metric spaces. *ACM Computing Surveys*, ACM, v. 33, n. 3, p. 273–321, 2001. 2, 3

[10] MOVSKO, J.; JAKUB, L.; SKOPAL, T. Clustered Pivot Tables for I/O-optimized Similarity Search. In: *International Conference on Similarity Search and Applications*. [S.l.]: ACM, 2011. p. 17–24. 2, 3

[11] GRAEFE, G. Modern B-Tree Techniques. *Foundations and Trends in Databases*, Now Publishers, Inc., v. 3, n. 4, p. 203–402, 2011. 2

[12] GUTTMAN, A. R-trees: A Dynamic Index Structure for Spatial Searching. *ACM Special Interest Group on Management of Data*. 2

[13] BEDO, M. V. N. et al. Beyond Hit-or-Miss: A Comparative Study of Synopses for Similarity Searching. *Journal of Information and Data Management*, SBC, v. 9, n. 1, p. 36–51, 2018. 2, 5

[14] TASAN, M.; OZSOYOGLU, Z. M. Improvements in distance-based indexing. In: *Scientific and Statistical Database Management Conference*. [S.l.]: IEEE, 2004. p. 161–170. 2, 3, 4, 5, 6

[15] VIEIRA, M. R. et al. Boosting knn queries by estimating suitable query radii. In: *Scientific and Statistical Database Management Conference*. [S.l.]: IEEE, 2007. p. 10. 2, 4, 6

[16] BUSTOS, C. et al. An empirical evaluation of intrinsic dimension estimators. In: *International Conference on Similarity Search and Applications*. [S.l.]: Springer, 2015. p. 125–137. 2

[17] PESTOV, V. Indexability, Concentration, and VC Theory. In: *International Conference on Similarity Search and Applications*. [S.l.]: ACM, 2010. p. 3–12. 2

[18] HETLAND, M. L. The basic principles of metric indexing. In: *Swarm Intelligence for Multi-objective Problems in Data Mining*. [S.l.]: Springer, 2009. p. 199–232. 3

[19] ZEZULA, P. et al. *Similarity Search: The Metric Space Approach*. [S.l.]: Springer, 2010. v. 1. 3, 5

[20] BöHM, C.; KREBS, F. The k-Nearest Neighbour Join: Turbo Charging the KDD Process. *KIS*, Springer, v. 6, n. 6, p. 728–749, 2004. 3

[21] HJALTASON, G. R.; SAMET, H. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, ACM, v. 28, n. 4, p. 517–580, 2003. 4, 5

[22] IOANNIDIS, Y. The history of histograms. In: *International Conference on Very Large Data Bases*. [S.l.]: VLDB Endowment, 2003. p. 19–30. 5

[23] KÖNIG, A. C.; WEIKUM, G. A framework for the physical design problem for data synopses. In: *International Conference on Extending Database Technology*. [S.l.: s.n.], 2002. p. 627–645. 5