cx-Sim: A Metric Access Method for Similarity Queries with Additional Conditions

Leandro C. Soares and Daniel S. Kaster

Department of Computer Science – University of Londrina
P.O.Box 10.011 – CEP 86057-970 – Londrina, Paraná, Brazil
leandro.cavalari.soares@gmail.com, dskaster@uel.br

Abstract. The fast growth of complex data repositories, such as images, videos and time series, in recent years is intensifying the importance of developing efficient search strategies over these data types. Applications that deal with complex data employ similarity queries to retrieve data, often combining similarity conditions with conditions over other associated attributes of traditional data types. There are several indexing structures for answering similarity queries, however most of them do not work when there are additional search conditions. The existing structures that answer queries combining conditions over complex and traditional attributes only support keyword-based conditions. This article presents a new metric access method to efficiently execute similarity queries with additional conditions over complex data. The proposed method, called the Condition-eXtended Similarity tree (cx-Sim tree), is a composite index that is able to answer similarity queries with general conditions (not only keyword-based), combining an ordered tree to store a traditional attribute with a forest of similarity trees to store a complex attribute. The article also presents results of experiments using three real datasets that show that our approach outperformed existing methods in a great extent.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Image databases; H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.3.3 [**Information Search and Retrieval**]: Information filtering

Keywords: condition-extended k-NN queries, metric access methods, multimedia databases, similarity queries

1. INTRODUCTION

The technological advance of eletronical devices in recent years, with the popularization of mobile devices (sensors, smartphones, tablets and notebooks), intensified the necessity of effective administration of massive databases of complex data. The concept of complex data in this article refers to data that is not representable by a traditional data type (characters, numerical types, etc.), such as multimedia and scientific data, time series and georeferenced data. Considering only the popular web social applications (Youtube, Facebook, Instagram, Flickr etc), the volume of this kind of data that flows worldwide by minute is very large. The medical area is another example that has been generating huge amounts of data of several types (images, videos, series, etc.), through the increasing availability of screening machines, cardiographs and other equipments.

Planning how to store and retrieve elements effectively and efficiently is still a research challenge. The efficiency achieved by the current Database Management Systems (DBMSs) to handle traditional data has stimulated interest in using them to manage complex data. Although DBMSs still lack support for retrieving complex data by content, they allow integrating complementary search conditions in a query. The extraction of relevant information from databases of complex data can be enhanced if other attributes associated to complex data are also considered in the similarity query predicate. Examples of such kind of queries in multimedia domain include:

Return the k ultrasonography images that are the most similar to a query image i_q regarding exams performed in patients between 35 and 50 years old

Return the k images that are the most similar to a query image i_q such that the resolution is greater than 7 MP or that were taken with a Canon camera

Besides images and videos, the family of complex data can be complemented by georeferencing the concerned object/place. An example of query in the geographical domain is:

Return the posts in a social network that were submitted using a smartphone not farther than 1 km from a given coordinate c_q in the last 24 hours

The relational operators ($<$, $\leq$, $\geq$, $>$) are not generally applicable to complex data, because it is impossible to sort images directly, unless the comparison operator is applied to an associated information – not complex – (for example: name, date, etc.) or a certain orderable feature extracted from the complex data [Pola 2010]. Searches for complex data are usually performed using similarity query operators [Barioni et al. 2009]. The applicability of these operators on a specific data domain requires the existence of a function that calculates the similarity between pairs of elements. This is usually a distance function, whose result is the inverse of the similarity between the compared elements, where 0 (zero) denotes identical elements.

Similarity search has been extensively explored in the literature. However, most of the proposals consider similarity queries over complex data as isolated operations, apart from other query operations regarding associated atributes. Statements composed by elaborated search expressions involving complex and traditional attributes allow representing queries that are useful for several applications. There are a few indexing structures proposed to execute queries combining similarity and traditional attributes, nevertheless the traditional search criteria is limited to keyword-based conditions.

This article presents a new metric access method that enables to operate complex and traditional data together, called the cx-Sim tree (*Condition-eXtended Similarity tree*). The cx-Sim tree allows the execution of similarity queries extended with conditions over other attributes not limited to keyword-based conditions in an efficient way. This hybrid approach allows testing directly in the index both the similarity and the additional criteria, usually involving attributes of traditional data types. The article also shows results of experiments performed with real datasets in which the proposed method outperformed existing methods, regarding total time, disk accesses, distance calculations and comparisons.

This article is organized as follows: section 2 presents the theoretical foundation about the execution of similarity queries; section 3 presents the main indexing structures for accelerating similarity query processing; section 4 presents the proposed metric access method; section 5 shows the experiments performed with three real datasets and the obtained results; finally, the section 6 concludes the article and brings extension proposals.

2. SIMILARITY QUERIES

As new data types keep emerging (such as images, videos, XML and others), traditional DBMSs are in constant update in order to efficiently processing them. Traditional data types, such as numbers, short texts and dates, can be queried through relational comparison operators ($<$, $\leq$, $\geq$, $>$), due to the existence of a total order relation in the data domain. On the other hand, new complex data types, such as multimedia and spatial data types, do not have a total order relation and the operators $=$ e $\neq$ are practically useless. This is evidenced, for example, recording the same situation under different angle and lighting: the images can be visually similar but the probability of being exactly equal is minimum.

Similarity queries have proved to be the most appropriate solution for retrieving these data types. These queries are based on relations among elements in a similarity space defined by a set the intrinsic properties of data and a (dis)similarity function that computes the (dis)similarity degree between pairs of elements. Similarity queries retrieve elements that meet a similarity criteria related to one or more reference elements, also called query elements.

There are several variations of similarity queries, including selections and joins by similarity and aggregated similarity searches. The most common variations are the *Range queries* and the *k-Nearest Neighbor queries* [Chávez et al. 2001].

—Range Query (Rq): given a dataset $S \in \mathbb{S}$, where $\mathbb{S}$ is a domain of complex data, a reference element $s_q \in \mathbb{S}$, a distance function δ defined over $\mathbb{S}$ and a coverage radius ξ, a Rq returns every element $s_i \in S$ whose distance from s_q is less than or equal to ξ. That is:

$$Rq(\delta, s_q, \xi) = \{s_i | \delta(s_i, s_q) \leq \xi\} \tag{1}$$

—k-Nearest Neighbor query (k-NNq): given a dataset S in the complex domain $\mathbb{S}$, a reference element $s_q \in \mathbb{S}$, a distance function δ defined over $\mathbb{S}$ and a positive integer number k, a k-NNq retrieves the k elements $s_j \in S$ that are the closest to s_q with regard to δ. That is:

$$kNNq(\delta, s_q, k) = K = \{s_i \in S | \forall s_j \in S - K, \delta(s_i, s_q) \leq d(s_j, s_q)\}, \tag{2}$$

2.1 Condition-Extended Similarity Queries

A similarity query can be extended with additional conditions to the search, defined over attributes associated to the complex data. The trivial type of condition is the keyword-based condition, which ckecks whether an element is tagged with a given (set of) keyword(s). However, there are many queries that demand general conditions (i.e. not only keyword-based). The queries presented in the introduction are examples of condition-extended similarity queries (e.g. "*Return the k ultrasonography images that are the most similar to a query image i_q regarding exams performed in patients between 35 and 50 years old*"). If the similarity operation is a Rq and the additional condition is a typical conditional expression of the relational algebra selection operator (σ), this query can be expressed by a conjunctive selection (or by cascaded selections) and optimized using the equivalence rules valid for the relational selection, as they also hold for range queries [Ferreira 2008]. On the other hand, if the similarity operation is a k-NNq, this assumption is not valid because the k-NNq is not commutative with the relational selection.

Kaster et al. [2011] and Kaster [2012] proposed that a k-nearest neighbor query can be extended including additional conditions that modify the search, introducing a new similarity query operation that extends the k-NN basic operator, called the condition-extended k-Nearest Neighbor query ($_ck$-NNq). A $_ck$-NNq is an exact search that retrieves the k elements nearest to the reference element which satisfy the additional condition c provided. This new operation allows answering similarity queries that were not representable before. The condition c can be of several types over attributes of complex and traditional data types, allowing to use relational comparison operators and keyword-based match. He also proposed algorithms to execute variations of $_ck$-NNq with sequential execution and using metric access methods to index complex data, discussed in the next section.

3. ACCESS METHODS AND ALGORITHMS FOR COMPLEX DATA SIMILARITY SEARCH

As several complex data are represented through multidimensional feature vectors extracted from their content, they can be indexed using Spatial/Multidimensional Access Methods, such as the R-tree. However, these structures degrade quickly with the increase of the number of dimensions. Analyzing

the research lines regarding similarity over complex data, it can be noticed that most proposals of access methods that work better with complex data fit in two families:

—the first one uses Locality-Sensitive Hashing (*LSH*) to measure the similarity between pairs of elements;

—the second one applies the similarity notion over a metric space.

Methods from the first family – based on *LSH* – usually provide a fast response time over an exact result. Specifically, they retrieve approximate results in sublinear time. For example, Slaney and Casey [2008] use the *LSH* in k-scalar projections to search quickly similar elements in very large databases. This technique was employed on identifying duplicated web pages, musics and objects inside images. Kulis at al. [2009] present a method that provides a balance between the image representation and the distance metric in the quality os results. Such balance is achieved through a fine tuning of the hashing parameters (number of *buckets* by *hash*, number of *hash functions*, notion of near and far, collision probability, etc.) and enable producing the result in sublinear time. The main limitation of this family is that the query result is approximate.

The second family of access methods is based on query executions with exact results using the concept of metric space. A metric space is defined by a data domain and a metric, which is a distance function defined over elements of the data domain that satisfies three properties: symmetry, non-negativity and triangle inequality. There are several Metric Access Methods (MAMs) to acelerate queries over complex data, specially, over multimedia data. The MAMs allow to minimize the number of comparisons (distance calculations) and the number of disk accesses during the query processing using properties of the metric spaces, particularly the triangle inequality [Traina and Traina Jr. 2003]. Such result is obtained subdividing the dataset in blocks or nodes and, for each of them, one element is elected as the representative of the node/block and it will be used to index the group.

There are several proposals of static and dynamic metric access methods. The first dynamic MAM (i.e. a MAM that does not degenerate with insertions and deletions, maintaining the search efficiency) was the M-tree [Ciaccia et al. 1997]. The M-tree is a tree with bottom-up growth and with two types of nodes (index and leaf nodes), which remains height balanced with insertions and deletions, without the necessity of periodic reorganizations. Other dynamic MAMs widely used are the Slim-tree [Traina and Traina Jr. 2003] and the DBM-tree [Vieira et al. 2010]. The Slim-tree is an evolution of the M-Tree, presenting as improvements the minimization of overlapping among nodes and a fast split algorithm based on the *Minimum Spanning Tree* (MST), while the DBM-tree explores the varying density of elements in the dataset allowing creating, in a controlled way, unbalanced trees. Traina Jr. et al. [2007] also proposed a family of MAMs called the Omni family, which elects a set of global representatives (the *Omni-Foci base*) for the whole dataset and performs the pruning by using the distances of each element to each focus in the omni-foci base. Such approach allows reducing the number of distance calculations and of disk accesses, achieving high performance in similarity searches. The study of Yousri et al. [2007] minimizes the comparisons regarding high dimensionality data using variable-size leaf nodes, therewith achieving performance 55% better than traditional search. Finally, McFee and Lanckriet [2011] used variations of the KD-tree to identify similarity in musical content. The latter approach leads to a tradeoff between accuracy and complexity in the execution of a k-NN query.

3.1 Access Methods and Algorithms for Similarity Queries with Conditions

The aforecited articles treat the similarity question considering only complex data. That is, the similarity searches are considered as isolated operations. This section presents works that rely on the combination of traditional and complex data in the indexing structure to execute similarity searches. These works can be divided into two classes according to the kind of condition associated to the

similarity query: works that support only keyword-based conditions and works that support general conditions.

Works that support only keyword-based conditions employ either signature files or inverted files. A signature file is a bitmap associated to a tree node where each bit correponds to a keyword. A bit 1 indicates that there is at least one element that has the respective keyword in the node or in the subtree rooted at the node and a bit 0 indicates that there is no such element. The signature file of an index node is given by the *bitwise or* among the signature files of its child nodes, allowing to prune subtrees or nodes that do not have elements with desired keyword(s) during the search. Examples of structures that employ signature files are the RS-tree [Park and Kim 2003], the SPY-TEC+ [Park and Lee 2011] and the $IR^2 - tree$ (*Information Retrieval R-tree*) [De Felipe et al. 2008]. An inverted file is an efficient structure for text information retrieval that support information retrieval ranking based on keywords. Examples of structures that employ inverted files to accelerate spatial keyword queries are the DIR-tree (Document similarity enhanced Inverted file R-tree) [Cong et al. 2009] and the S2I (Spatial Inverted Index) [Rocha-Junior et al. 2011]. Another structure that employs inverted files is the IQ-tree [Chen et al. 2013], which focus on continuous spatial keyword queries over a data stream.

With regard to works that support similarity queries with general conditions, to the best of our knowledge, there is no access method specifically designed to answer such queries. The work of Kaster et al. [2011] proposed new algorithms over an existing access method that improve the execution performance of similarity queries with general conditions. These algorithms answer condition-extended similarity queries (section 2.1) and they were implemented over the Slim-tree (although they can be implemented over other structures such as the R-tree or the M-tree) using two variations: Table-Slim and Covering-Slim.

The variation **Table-Slim** uses an unmodified Slim-tree storing the feature vector and the row identifier (*rowId*) of each indexed tuple. The execution of a similarity query recursively traverses the structure pruning the nodes/elements that do not satisfy the similarity criterion. If the algorithm reaches an index entry that satisfies the similarity criterion, it has to check the additional search condition. To do so, the algorithm accesses the data table to retrieve the tuple pointed by the *rowId* stored in the entry, as the attribute referenced in the condition is not inside the index. If the element satisfies the additional condition it is included in the result set and the search continues. The main weakness of the variation Table-Slim is that it demand a high number of data disk accesses, however, it is faster than the sequential scan for moderate selectivities of the additional condition. Moreover, the Table-Slim only requires the index on the complex attribute, being able to answer queries whose condition involve any other attribute of the table storing the complex data.

The variation **Covering-Slim** modifies the Slim-tree to be a covering index for a ${}_ck$-NN query. A covering index is one that can answer a query without performing further lookups on the table. This is achieved in the Covering-Slim by storing the attribute(s) involved in the additional condition along with the feature vector and the *rowId* in the leaf index entries of the Slim-tree. The execution of a similarity query is identical to the execution of the Table-Slim variation, however the additional condition is checked directly on the index, reducing drastically the number of disk accesses. The Covering-Slim variation is always faster than the Table-Slim and it is much fast than the sequential scan for moderate-to-high selectivities of the additional condition. However, it can only be employed to answer a condition-extended query if a proper covering index is available and it performs worse than the sequential scan for high selectivities of the additional condition.

The method proposed in this article aims at accelerating the execution of similarity queries with general conditions, being efficient even for high selectivities of the additional search condition. This work is innovative with regard to the Table-Slim and Covering-Slim algorithms as it presents a new indexing structure combining a complex attribute and a traditional attribute. The proposed structure is designed to improve the pruning regarding both the additional condition and the similarity criterion.

Indeed, it does not require to encapsulate the value of the traditional attribute for each entry at the index, thus the index size is reduced. These characteristics makes the proposed structure a suitable alternative for condition-extended similarity queries over large complex databases. The next section presents the proposed method.

4. THE PROPOSED METHOD: THE CX-SIM TREE

This section presents the proposed metric access method, called the *cx-Sim tree* (*Condition-eXtended Similarity tree*). The cx-Sim tree aims at speeding up the execution of similarity queries with general additional conditions involving attributes associated to the complex data. It is defined as a composite index that stores one traditional and one complex attribute, being structured in two levels, as showed in Figure 1:

—the first level (l_1) stores the traditional attribute values and is organized with a B^+-tree, therefore, benefitting from operations in logarithmic time;

—the second level (l_2) stores the complex attributes in a forest of dynamic metric trees.

Methods without this layer of traditional data are able to generate lots of false positives analysing only the complex data and in an expensive way.

The main advantage of combining a B^+-tree and a forest of dynamic metric trees together is the space partitioning, which is unfeasible using only one structure. Each distinct value indexed by the B^+-tree points to a metric tree in l_2 that stores the complex data elements that have this value for the traditional attribute. Therefore, the metric trees in l_2 are disjoint regarding the traditional attribute value. This allows reaching the exact subsets of the indexed elements (partitions) that satisfy the additional query condition by searching the B^+-tree. After having identified the partitions that contain the elements qualified regarding the condition on the traditional attribute, the search over the respective metric tree(s) is started. Moreover, the index is compact as only one distinct value of the traditional attribute is stored, regardless of the quantity of elements that have this value for the attribute.

The organization of the *cx-Sim tree* relies on the existence of the total order relation at the traditional data domain. Thus, the most expensive operations during the execution of a similarity search – disk accesses and distance calculations between complex attibutes – occur only after the traditional attribute filter and in an assertive way (i.e. only over data that really have the potential to compose the result set).

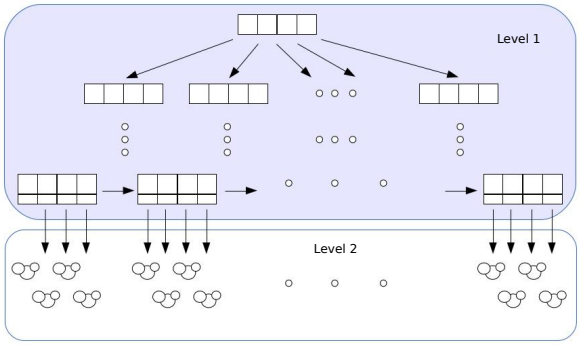

Fig. 1. Representation of the *cx-Sim tree* access method.

The *cx-Sim tree* is dynamic (i.e. it does not degrade with updates) as it relies on dynamic trees in both levels of the structure. It is worth noticing that it is not height balanced, as the height of each tree in the level 2 depends on the amount of elements that share the same value for the traditional attribute indexed in the level 1. Nevertheless, this fact does not degrade the performance of the operations over the structure. If every element has the same value for the traditional attribute, there will be only one tree in level 2. Considering that the dynamic metric tree employed at level 2 is balanced, the height of the *cx-Sim tree* is logarithmic in the number of indexed elements as it is given the height of the metric tree plus one, which is the height of the B^+-tree. On the other extreme, if every element has a distinct value for the traditional attribute, the height of the *cx-Sim tree* is also logarithmic as it is given the height of the B^+-tree plus one, which is the height of the metric trees.

The following subsections describe the main operations of the *cx-Sim tree*.

4.1 Insertion

The insertion of elements in the *cx-Sim tree* is shown in the Algorithm 1. Each distinct value or range in l_1 has a reference to the region of l_2 where the complex attribute must be added. Then, the complex attribute is inserted in the respective Slim-tree at l_2, identifying, through a recursive process, the leaf node whose coverage area will suffer lower increase after the insertion of the new element. The smaller the increase of the node's coverage area, the lower the overlapping rate and, consequently, the better the performance of the structure during search operations.

Algorithm 1: cx–Sim tree Insert Operation

```
   Data: featureVector, traditionalValue, rowId
   Result: True if the insertion is successful or false otherwise
 1 begin
 2   if !level1.search(traditionalValue) then
 3     level1.insert(traditionalValue);
 4     level2 = new SlimTree(traditionalValue);
 5   end
 6   level2 = getSlimTree(traditionalValue);
 7   if level2.insert(featureVector, rowId) then
 8     return true;
 9   end
10   return false;
11 end
```

The proposed method has a characteristic that affects the index size: the first level was projected to store only distinct values. Thus, the sharing of traditional attributes by distinct complex attributes does not imply data duplication in l_1, yielding a smaller index.

4.2 Search

The Algorithm 2 represents the query execution and is initiated searching for the values, or ranges of values, that satisfy the condition ($<$, $\leq$, $\geq$, $>$, $=$, $\neq$) in the l_1 benefitting from operations in $O(\log n)$ time, where n is the number of distinct values, or ranges of values, stored in this level. If the values or ranges of values are found, the algorithm obtains the reference to the l_2 regions where the complex attributes (that meet the above condition) will be analised according to the similarity criteria specified in the query.

If the similarity search is a range query, the search algorithm traverses the underlying Slim-trees using the *branch-and-bound* strategy, performing pruning of subtrees such that their coverage areas do not intersect the area bounded by the query radius. If the similarity search is a k-NN query, the initial radius is set to infinity, being gradually reduced during the execution of the operation. The

structure is traversed using the *best-first* strategy, which orders the subtrees to be visited according to the minimum distance between their coverage area and the query element.

A query whose condition defined over the traditional attribute is satisfied by more than an entry in l_1 requires to search more than one Slim-tree at l_2. In these cases, the final query result is given by merging the results of all searches. This can be done sequentially or in parallel. If the execution is sequential, one search feeds the next one with the partial result set produced and if the execution is in parallel, the searches must share a global result set. This allows to produce the final (merged) result set as well as to improve the pruning capacity when executing k-NN queries by reducing the coverage area faster than it would be if each search was not aware of the other search results.

Algorithm 2: cx–Sim tree Search Operation

```
   Data: featureVector, traditionalValue, condition, k
   Result: The query result set
1  begin
2  |  level2Trees = level1.search(traditionalValue, condition);
3  |  resultSet = null;
4  |  foreach l2Tree in level2Trees do
5  |  |  l2Tree.search(featureVector, k, resultSet);
6  |  end
7  |  return resultSet;
8  end
```

4.3 Deletion

The deletion of elements of the *cx-Sim tree* uses the search operation. Finding the combined key to be removed at l_1, the algorithm fires the process to remove the element at the respective Slim-tree as shown in the Algorithm 3. The search operation in the Slim-tree is a *Point Query* because it is desired to remove the particular element that results of a pontual search. Furthermore, it can generate reorganizations in the Slim-tree, such as distribution of elements among sibling nodes, exchange of representatives and/or removal of nodes and reinsertion of elements.

If the deletion in the Slim-tree leaves the tree empty (removal of the root node), the algorithm propagates this change to the element at l_1 that references this tree. The value of a traditional attribute that references a complex data is removed from l_1 when the last complex data referenced by it is removed. This operation fires reorganization operations at B^+-tree to complete the removal of this key. The reorganization operations at level l_1 (*unsplit*, distribution of elements among sibling nodes etc.) can be propagated to the tree's root.

Algorithm 3: cx–Sim tree Delete Operation

```
   Data: featureVector, traditionalValue
   Result: True if the element was deleted and false otherwise
1  begin
2  |  if level1.search(traditionalValue) then
3  |  |  level2 = getSlimTree(traditionalValue);
4  |  |  if level2.delete(featureVector) then
5  |  |  |  if level2.isEmpty() then
6  |  |  |  |  level1.delete(traditionalValue);
7  |  |  |  end
8  |  |  |  return true;
9  |  |  end
10 |  end
11 |  return false;
12 end
```

The next section presents experiments executed over the *cx-Sim tree*.

5. EXPERIMENTS

The objective of this experiment is present the improvement in efficiency of similarity queries with additional conditions over complex data, comparing the proposed method with other methods that answer similarity queries with general conditions (not only keyword-based). Under this assumption, only *Covering-Slim* and *Table-Slim* (described in section 3.1) compose a reliable and comparable baseline for the experiments. To enable a fair comparison, we implemented the *cx-Sim tree* using Slim-trees in the second level of the structure. This choice is also corroborated by the Slim-tree be height-balanced and fast for high-dimensional and metric data. While the Slim-tree based methods accept all the range of values from the traditional data types, the keywords-based solutions limit the set of possible values and are based on low-dimentional trees.

This section presents the results obtained by applying *cx-Sim*, *Covering-Slim* and *Table-Slim* over three real datasets: one containing georeferenced data from US cities census, other containing medical images and the third one containing georeferenced data from vehicles. These are called, respectively: *USCities*, *HCImages* and *BRPositions*.

The main differences between these methods will be analyzed considering the most expensive and relevant operations of a similarity search as metrics: disk accesses, distance calculations, comparisons and total time.

5.1 Description of the Data Sets and of the Experiments

The dataset *USCities* has as complex attribute the geographic coordinates of cities of the United States of America and has about 100 numerical attributes containing demographics from the American census of 2000[1]. The dataset *HCImages* is composed by color histograms (256 gray levels) and 13 attributes from traditional types (numeric and textual), extracted from the headers of medical images in the DICOM format of radiological exams performed by Clinical Hospital of the Faculty of Medicine of Ribeirão Preto at the University of São Paulo (HCFMRP-USP). The last one, *BRPositions*, refers to the historical positions of vehicles that makes use of logistical solutions of the company Veltec Soluções Tecnológicas S/A[2].

The datasets *USCities*, *HCImages* and *BRPositions* have, respectively, 25,374, 501,100 and 2,715,873 elements, thus permit evaluate the behavior of the proposed method with distinct volume of data. For natural reasons of confidentiality, these bases do not identify particularly the US habitants, the patients or the vehicles and/or theirs owners. They have only attributes that permit identify individual data (complex and traditional).

The feature of defining queries and interpreting the results considering geographic data was decisive to choose the first and third datasets because the similarity is more intuitive. In geographic applications the similarity is commonly measured by the distance between elements in the 2D space. Due to this, the function adopted for the USCities and the BRPositions datasets was the Euclidean distance (L_2). The function adopted for the *HCImages* was the Manhattan distance (L_1). It is worth noticing that the proposed method is naturally applicable to other complex data (videos, scientific or time-series data), being sufficient to define a feature vector to represent complex data and an appropriate distance function.

The *cx-Sim tree* was developed in C++ using the Arboretum library. The Arboretum is an open

[1]In 2000, the U.S. Census Bureau carried out a census in the United States of America, whose aggregated data are available at `http://www.census.gov/main/www/cen2000.html`
[2]Available at `http://www.veltec.com.br`

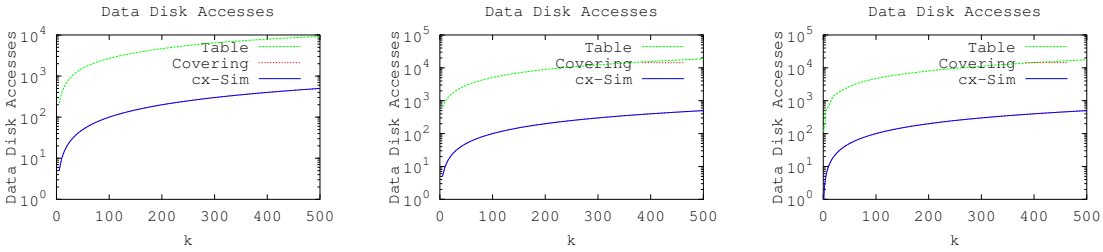

Fig. 2. Number of accesses to the data file, varying the number of neighbors (k).

source library of indexing methods for complex data, developed in C++ by the Databases and Images Group of the Institute of Mathematics and Computer Science of the University of São Paulo (GBDI-ICMC-USP)[3]. The experiments were executed in a computer equipped with an Intel CORE i5 processor, 4GB of RAM, 500GB of SATA2 HDD and the Ubuntu Linux operational system (version 12.04 / 64 bits). Besides *cx-Sim*, such data was also indexed to execute algorithms of similarity queries extended with conditions using the variations *Table-Slim* e *Covering-Slim* (see section 3.1).

The evaluation has tested the three data structures with condition-extended similarity searches, such as: i) Return k cities nearest to the city X and population_median_age between Y and Z years *(USCities)*; ii) Return k exams most similar to the exam E and patient_age between F and G years *(HCImages)*; iii) Return k positions nearest to the reference point P and velocity $\leq$ Q km/h *(BRPositions)*.

These queries suffered two controlled and identical variations to the three methods: i) *Number of nearest neighbors (k):* variation between 1 and 500; ii) *Condition selectivity involving the traditional attribute:* variation between 50% and 100%.

For each distinct value of these variations the searches were executed with query elements randomly selected (the cities X, the exams E and reference points P), and the results presented following refer to the average of all executions for each setup (average of 500 queries for the *USCities* and *BRPositions* datasets and 100 queries for the *HCImages* dataset).

5.2 Results and Discussion

Our experimental evaluation starts by varying the number of nearest neighbors *(k)* and the Figure 2 presents the number of accesses to the data file. Each option shows that the methods *Covering-Slim* and *cx-Sim* present quantities significantly smaller than the third method – *Table-Slim*. The first two showed only k disk accesses by execution while the third one reached the order of 18, 35 and 37 times more accesses to the bases *USCities*, *HCImages* and *BRPositions*, respectively. Such difference is acceptable: to analyze whether or not a tuple should be added to the result set, the *Table-Slim* needs one disk access to get the complex data and another to access the traditional one. It is noted that in some graphs the vertical axis is shown on a log scale to facilitate the method comparisons.

Related to the index access, the *cx-Sim* got the requested result with much less effort than other methods. According to Figure 3, the amount of accesses was smaller at order of 5, 8 and 6 times to the datasets *USCities*, *HCImages* and *BRPositions*, respectively. Due this huge difference related to index accesses, the *cx-Sim* also showed to be more efficient regarding the total amount of disk accesses, as shown in Figure 4.

[3]Available at `http://gbdi.icmc.usp.br/arboretum`

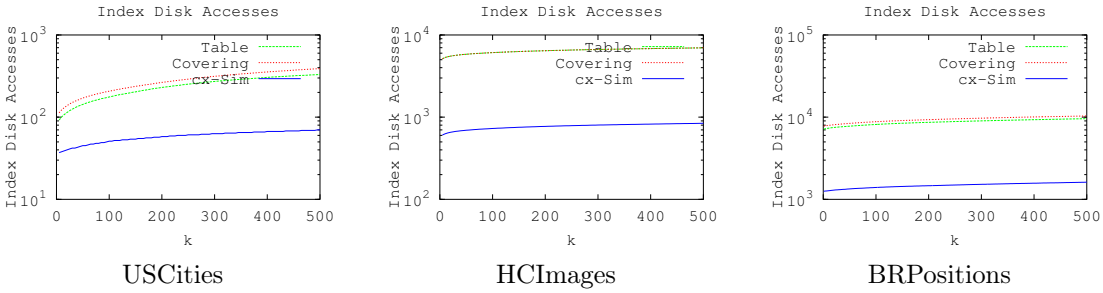

Fig. 3. Number of accesses to index blocks, varying the number of neighbors (k).

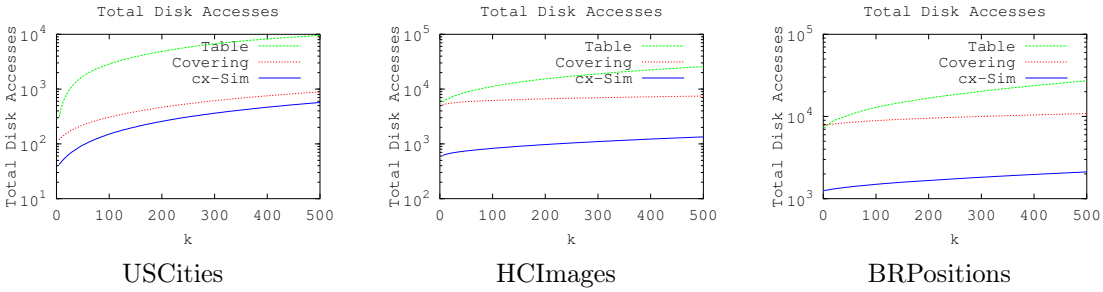

Fig. 4. Total amount of disk accesses, varying the number of neighbors (k).

One issue where the outcome obtained by *cx-Sim* was considerably better and deserves can be noted through the Figure 5 where are presented the comparison number between traditional data. Due to this two level architecture, once fixed the condition (traditional data), this MAM presents constant amount of comparisons independently of the selected k, because the effort to search at the B$^+$-Tree is the same. Such result is not possible to the other MAMs and these got results in order of 390, 1876 and 120 times more to the bases *USCities*, *HCImages* and *BRPositions*, respectively.

In addition to the number of comparisons, we have the amount of distance calculations between pairs of complex data. According to informations presented in Figure 6, the results got through the *cx-Sim* are considerably smaller than those obtained by the other two, *Covering-Slim* and *Table-Slim*. Such difference was in order of up to 7,5, 8,5 and 6,6 times more to the bases *USCities*, *HCImages* and *BRPositions*, respectively.

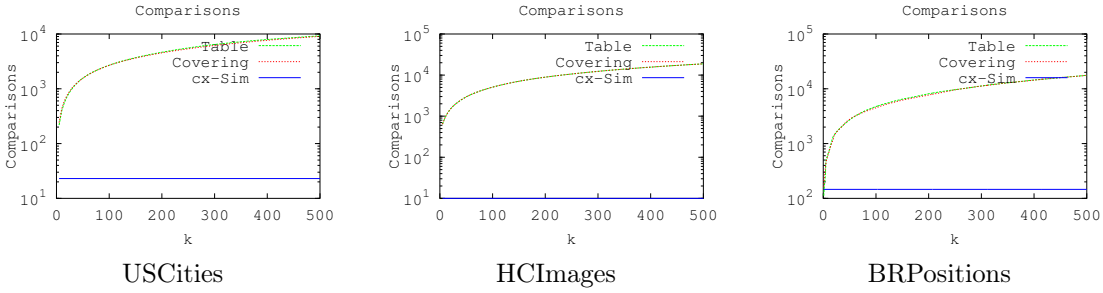

Fig. 5. Number of comparisons, varying the number of neighbors (k).

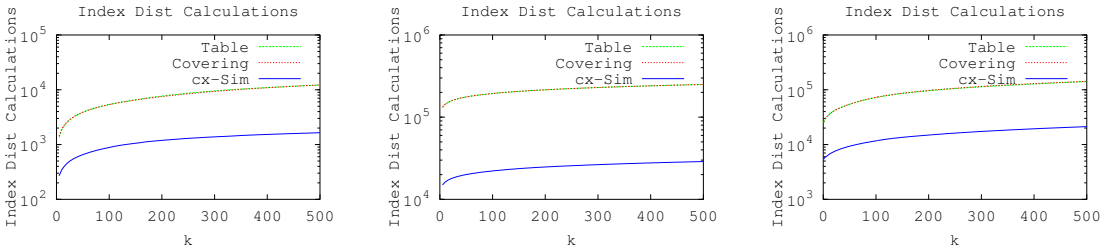

Fig. 6. Number of distance calculations in the index, varying the number of neighbors (k).

There is one issue where the *cx-Sim* presented values greater than *Table-Slim* and *Covering-Slim*, particularly at the experiments fed with the georeferenced bases. It is the amount of data added to the result set during the search and can be noted at Figure 7. This difference was the smallest obtained from the executed methods – it was in order of up to 27%, 3% and 11% greater then the bases *USCities*, *HCImages* and *BRPositions*, respectively. Such result was not significant related to the performance of condition-extended similarity searches, because according to Figure 8, the average wall clock time of the queries executed with the *cx-Sim* presented execution times considerably smaller. Related to the base *USCities*, these were in order of up to 30% and 84% smaller than the methods *Covering-Slim* and *Table-Slim*, respectively. Considering the base *HCImages*, the differences were in order of up to 84% and 88% smaller than the methods *Covering-Slim* and *Table-Slim*, respectively. *BRPositions*, in turn, the differences were in order of up to 79% and 84% smaller than the methods *Covering-Slim* and *Table-Slim*, respectively.

The second part of the experiment considers the selectivity variation of the search condition. At this experiment phase, the k – variable of previous phase – was fixed to 100.

With regard to the number of accesses to the data file (Figure 9) remained fixed to methods *cx-Sim* and *Covering-Slim*, i.e., k. For the same reasons mentioned above, such value remained increasing to method *Table-Slim*, mainly at the base *USCities*.

Related to the amount of index accesses (Figure 10), the behavior of this phase was different from the previous: only to the MAM *cx-Sim* the amount of accesses decrease as the selectivity increase. Such behavior is expected because its architecture ensures that will be analyzed only tuples of the complex index with the traditional condition already validated. The other methods do not have this restrictive feature.

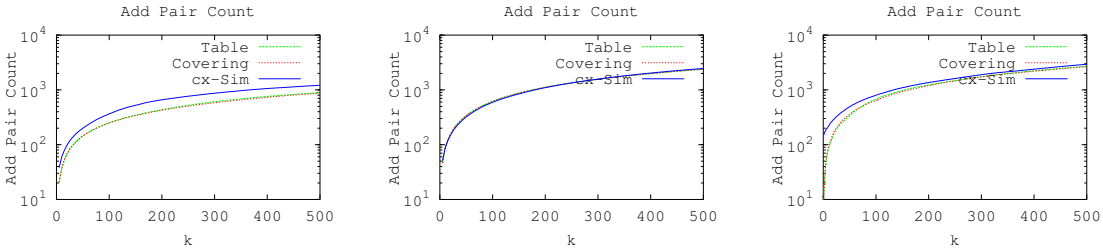

Fig. 7. Number of queue operations in the result set (number of insertions at the auxiliar data structure), varying the number of neighbors (k).

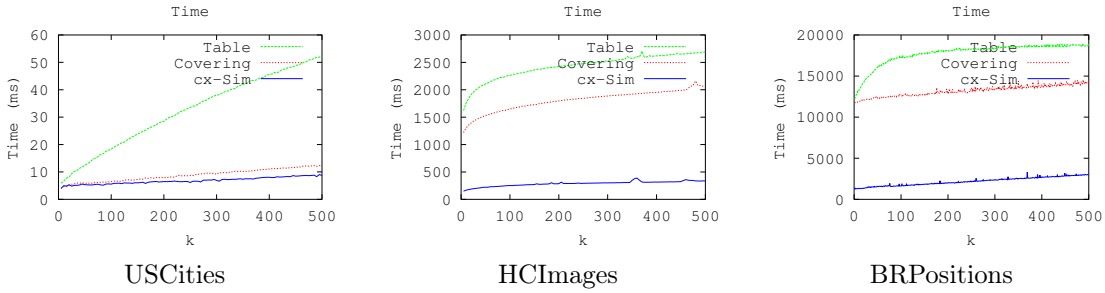

Fig. 8. Wall clock time, varying the number of neighbors (k).

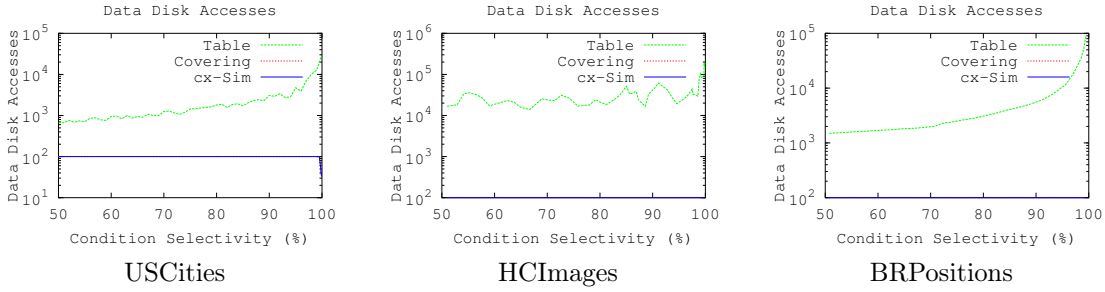

Fig. 9. Number of accesses to the data file, varying the selectivity of the search condition.

The same behavior also is seen related to: the total amount of disk accesses (Figure 11), the amount of comparisons between traditional data (Figure 12) and the amount of distance calculations between pairs of complex data (Figure 13).

As in the first phase of the experiment, the *cx-Sim* also presents greater values than *Table-Slim* and *Covering-Slim* related to the amount of data added to the result set during the searches, particularly in the georeferenced bases. It can be seen at Figure 14.

Considering the wall clock time and the base *USCities*, both methods – *cx-Sim* and *Covering-Slim* – alternate each other as holders of the best results until the selectivity reaches about 90%. Thereafter, the *cx-Sim* starts decreasing the execution time while the *Covering-Slim* do not, according to Figure 15. Related to the base *HCImages*, the average search time of *cx-Sim* was up to 98% and 95% smaller than *Table-Slim* and *Covering-Slim*, respectively. The *BRPositions* base, in turn, the average search time of *cx-Sim* was up to 84% and 79% smaller than *Table-Slim* and *Covering-Slim*,

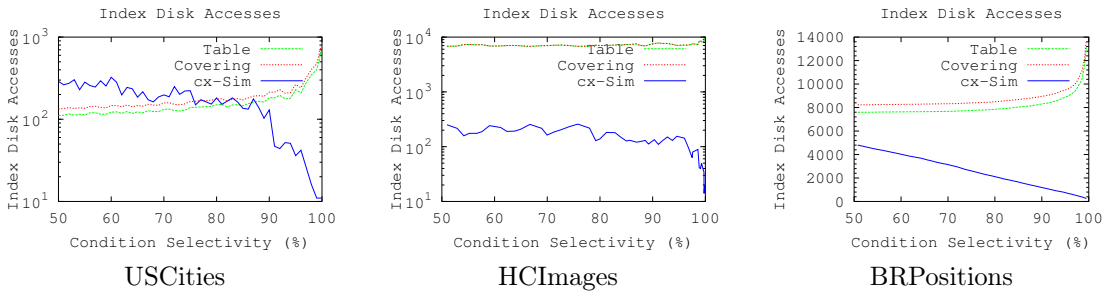

Fig. 10. Number of accesses to index blocks, varying the selectivity of the search condition.

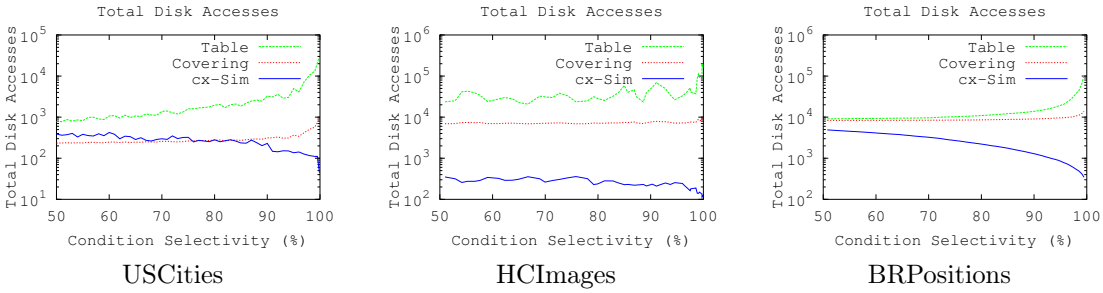

Fig. 11. Total amount of disk accesses, varying the selectivity of the search condition.

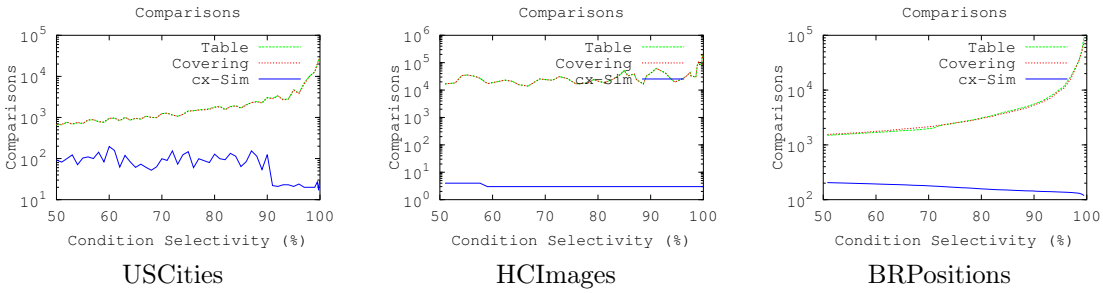

Fig. 12. Number of comparisons, varying the selectivity of the search condition.

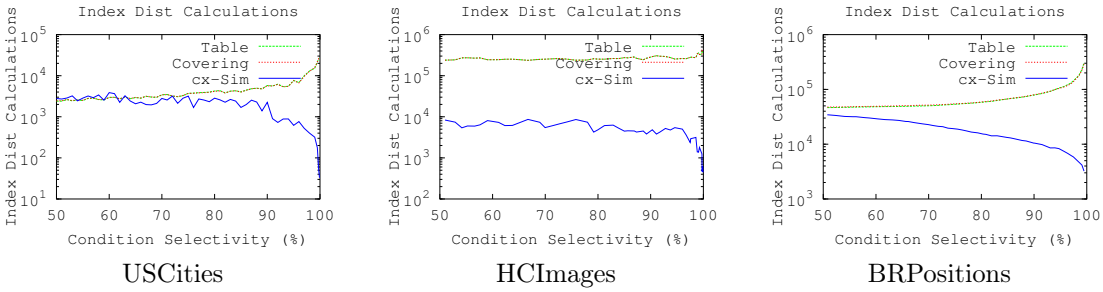

Fig. 13. Number of distance calculations in the index, varying the selectivity of the search condition.

respectively.

As a general result of the two parts of the experiment using the three datasets, we can realize that the algorithms of the *cx-Sim* MAM showed to be considerably more efficient that its competitors. These results were proved by: i) reduced amount of accesses to the data file; ii) reduced amount of total disk accesses; iii) reduced amount of comparisons between traditional data; iv) reduced amount of distance calculations between pairs of complex data; v) reduced average wall clock time.

Comparing directly the *cx-Sim tree* with the *Covering-Slim* – the method with the second better performance in the experiments – the *cx-Sim* also presents a better strategy to store the data inside the index. This can be seen through the follow comparison: while the *Covering-Slim* stores one entry with a traditional data internally to each complex data, the *cx-Sim* stores only the complex one. The traditional data are stored in a separate structure and without duplication. This optimization related to data store is also a result of the chosen architecture.

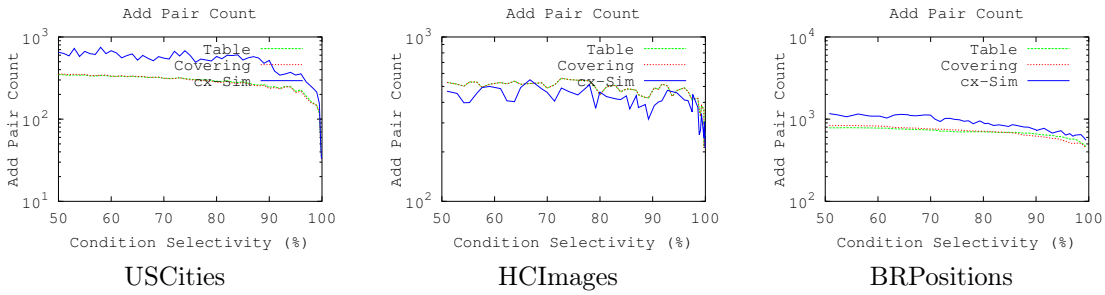

Fig. 14. Number of queue operations in the result set (number of insertions in the auxiliar data structure), varying the selectivity of the search condition.

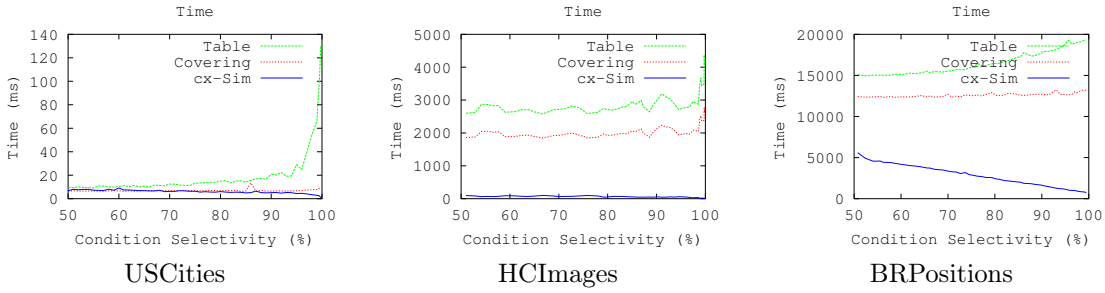

Fig. 15. Wall clock time, varying the selectivity of the search condition.

6. CONCLUSION

This article presented the *cx-Sim tree*, a new metric access method developed to execute condition-extended similarity queries. The benefits provided by the proposed method result of its architecture:

—traditional and complex data are stored in heigh-balanced trees at two different levels;
—traditional data values stored without duplication.

Such features make possible to optimize the data storage and to reduce the index size, as well as they provide a significant performance improvement.

As it could be seen in the presented experiments, the proposed method outperformed considerably its predecessors *Covering-Slim* and *Table-Slim*. Among the evolutions presented, the *cx-Sim tree* achieved:

—reduced amount of disk accesses;
—reduced amount of distance calculations as only elements that certainly have the possibility to compose the result set were considered;
—reduced average search time, which is a result of the improvement on the disk access and distance calculation requirements.

The increased efficiency and the effectiveness of methods like the proposed one bring important alternatives to the area of complex data retrieval by similarity, allowing fast retrieval of relevant data to the users. Considering also traditional data in the search predicates, the filter possibilities over the dataset are enriched, providing support to more elaborated queries in new applications.

Future work includes to optimize the management of the query result set to improve data pruning and achieve a smaller average search time and to evolve the *cx-Sim tree* to answer other variations of condition-extended similarity queries.

REFERENCES

Barioni, M. C. N., Razente, H. L., Traina, A. J. M., and Traina Jr., C. Seamlessly Integrating Similarity Queries in SQL. *Software Practice and Experience* 39 (4): 355–384, 2009.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. Searching in Metric Spaces. *ACM Computing Surveys* 33 (3): 273–321, 2001.

Chen, L., Cong, G., and Cao, X. An Efficient Query Indexing Mechanism for Filtering Geo-Textual Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA, pp. 749–760, 2013.

Ciaccia, P., Patella, M., and Zezula, P. M-Tree: an efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Databases*. Athens, Greece, pp. 426–435, 1997.

Cong, G., Jensen, C. S., and Wu, D. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *Proceedings of the VLDB Endowment* 2 (1): 337–348, 2009.

De Felipe, I., Hristidis, V., and Rishe, N. Keyword Search on Spatial Databases. In *Proceedings of the IEEE International Conference on Data Engineering*. Cancun, Mexico, pp. 656–665, 2008.

Ferreira, M. R. P. *Suporte a Consultas por Similaridade Unárias em SQL*. M.S. thesis, University of São Paulo, Brazil, 2008.

Kaster, D. S. *Tratamento de Condições Especiais para Busca por Similaridade em Bancos de Dados Complexos*. Ph.D. thesis, University of São Paulo, Brazil, 2012.

Kaster, D. S., Oliveira, W. D., Bueno, R., Traina, A. J. M., and Traina Jr., C. Nearest Neighbor Queries with Counting Aggregate-based Conditions. *Journal of Information and Data Management* 2 (3): 401–416, 2011.

Kulis, B., Jain, P., and Grauman, K. Fast Similarity Search for Learned Metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (12): 2143–2157, 2009.

McFee, B. and Lanckriet, G. R. G. Large-Scale Music Similarity Search With Spatial Trees. In *Proceedings of the International Society for Music Information Retrieval Conference*. Miami,Florida, pp. 55–60, 2011.

Park, D.-J. and Kim, H.-J. An Enhanced Technique for k-Nearest Neighbor Queries with Non-Spatial Selection Predicates. *Multimedia Tools and Applications* 19 (1): 79–19, 2003.

Park, D.-J. and Lee, D.-H. Spy-Tec+: an integrated index structure for k-nearest neighbor queries with semantic predicates in multimedia database. *International Journal of Software Engineering and Knowledge Engineering* 21 (7): 989–1011, 2011.

Pola, I. R. V. *Explorando Conceitos da Teoria de Espacos Métricos em Consultas por Similaridade Sobre Dados Complexos*. Ph.D. thesis, University of São Paulo, Brazil, 2010.

Rocha-Junior, J. B., Gkorgkas, O., Jonassen, S., and Nørvåg, K. Efficient Processing of Top-k Spatial Keyword Queries. In *Proceedings of the International Conference on Advances in Spatial and Temporal Databases*. Minneapolis, MN, pp. 205–222, 2011.

Slaney, M. and Casey, M. Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes]. *IEEE Signal Processing Magazine* 25 (2): 128–131, 2008.

Traina, A. J. M. and Traina Jr., C. Similarity Search in Multimedia Databases. In *Handbook of Video Databases: design and applications*, 1 ed. Boca Raton: CRC Press, pp. 711–738, 2003.

Traina Jr., C., Santos Filho, R. F., Traina, A. J. M., Vieira, M. R., and Faloutsos, C. The Omni-Family of All-Purpose Access Methods: a simple and effective way to make similarity search more efficient. *VLDB Journal* 16 (4): 483–505, 2007.

Vieira, M. R., Chino, F. J. T., Traina, A. J. M., and Traina Jr., C. Revisiting the DBM-Tree. *Journal of Information and Data Management* 1 (1): 129–132, 2010.

Yousri, N., Ismail, M., and Kamel, M. Adaptive Similarity Search in Metric Trees. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Montreal, Canada, pp. 419–424, 2007.