DAHC-tree: An Effective Index for Approximate Search in High-Dimensional Metric Spaces

Jurandy Almeida, Eduardo Valle, Ricardo da S. Torres, Neucimar J. Leite

Institute of Computing, University of Campinas – UNICAMP
13083-852, Campinas, SP – Brazil
{jurandy.almeida, rtorres, neucimar}@ic.unicamp.br, mail@eduardovalle.com

Abstract. Similarity search in high-dimensional metric spaces is a key operation in many applications, such as multimedia databases, image retrieval, object recognition, and others. The high dimensionality of the data requires special index structures to facilitate the search. A problem regarding the creation of suitable index structures for high-dimensional data is the relationship between the geometry of the data and the organization of an index structure. In this paper, we study the performance of a new index structure, called Divisive-Agglomerative Hierarchical Clustering tree (DAHC-tree), which reduces the effects imposed by the above liability. DAHC-tree is constructed by dividing and grouping the data set into compact clusters. We perform a rigorous experimental design and analyze the trade-offs involved in building such an index structure. Additionally, we present extensive experiments comparing our method against state-of-the-art of exact and approximate solutions. The conducted analysis and the reported comparative test results demonstrate that our technique significantly improves the performance of similarity queries.

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design—*access methods*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*indexing methods*

Keywords: clustering methods, database indexing, metric access methods, metric spaces, similarity search

1. INTRODUCTION

Similarity search in high-dimensional metric spaces is a subject of interest for many research communities. For over two decades, significant research efforts have been spent trying to improve its performance. In spite of that, many issues are still considered open problems. A problem regarding indexes for search in high-dimensional spaces is the relationship between the data geometry and index organization.

Most existing indexes are constructed by partitioning a set of objects using distance-based criteria. In order to keep the balance of the structure, the data set is divided into even sized parts, ignoring the inherent grouping of data. In general, those techniques follow two basic paradigms. One type of methods produces disjoint partitions, but ignores the distribution properties of the data [Bustos and Navarro 2004; Xu et al. 2008; Chávez et al. 2008]. The other type of methods produces non-disjoint groups, which greatly affect the search performance [Zezula et al. 1998; Clarkson 1999; Goldstein and Ramakrishnan 2000; Chávez and Navarro 2003; Lin et al. 2005].

In this paper, we study the effectiveness of a new index structure, called Divisive-Agglomerative Hierarchical Clustering tree (DAHC-tree), which is constructed by dividing and grouping the data set into compact clusters. It combines the advantages of both disjoint and non-disjoint paradigms, improving the search results.

The performance claims on DAHC-tree are supported by a rigorous experimental design used both for exploring the parameter space of the method and for comparing it with state-of-the-art solutions.

The conducted analysis and the reported comparative test results demonstrate that our approach significantly improves the performance in processing similarity queries.

The major contributions of this paper are:

- First, we propose DAHC-tree, a new index structure which performs very well for approximate similarity search in high-dimensional metric spaces.
- Second, we study how the relationship between the geometry of the data and the organization of an index structure may greatly affect the search performance.
- Third, we analyze the performance of DAHC-tree using a rigorous experimental design. Our results show that DAHC-tree achieves a high recall rate.
- Fourth, we evaluate the effectiveness of DAHC-tree through a comparison with LSH (Locality Sensitive Hashing) [Gionis et al. 1999], a competing index structure which is currently considered the state-of-the-art in high-dimensional indexing.
- Finally, we assess the efficiency of DAHC-tree by comparing it to several state-of-the-art indexes for exact similarity search in metric spaces.

A preliminary version of this work was presented at ACIVS 2008 [Rocha et al. 2008]. Here, we introduce several innovations. First, we redesign the structure as an index, supporting the storage in disk pages and the execution of similarity queries. Additionally, we present new strategies for both steps (agglomerative and divisive) of our approach. Finally, we report new experiments both for the analysis of our method and for the comparison with other techniques.

The remainder of this paper is organized as follows. Section 2 introduces some basic concepts of the similarity search problems. Section 3 describes the related work. Section 4 presents DAHC-tree and shows how to apply it to approximate similarity search. Section 5 reports the results of our experiments and compares the performance of our approach with other methods. Finally, we offer our conclusions and discuss the limitations of DAHC-tree in Section 6.

2. BASIC CONCEPTS

Traditional database systems [Ramakrishnan and Gehkre 2003; Elmasri and Navathe 2005] are able to efficiently deal with structured records by using the *exact match* paradigm. However, complex data types, such as multimedia data (audio, image, and video), biological data (genomic and protein sequences), among others, cannot be represented effectively as structured records [Zezula et al. 2005].

In those cases, *similarity search* [Jagadish et al. 1995] has been established as a fundamental paradigm. Essentially, the problem is to find, in a set of objects, those which are the most similar to a given query object. The similarity between any pair of objects is computed by some distance function, being understood that low values of distance correspond to high degrees of similarity [Patella and Ciaccia 2009].

The commonest types of similarity queries include (1) *range* queries, where all the objects whose distance to the query does not exceed a threshold are requested; and (2) *k-nearest neighbors* (k-NN) queries, where a specified number k of objects, which are closest to the query are requested [Zezula et al. 2005].

Several index structures have been proposed to speed up similarity queries [Gaede and Günther 1998; Chávez et al. 2001; Almeida et al. 2010]. They can be broadly classified, depending on their field of applicability, as multi-dimensional (or spatial) and metric access methods, where the former only applies when the feature space is a vector space [Patella and Ciaccia 2008].

Empirical studies, however, have pointed out that those approaches are not efficient when applied to high-dimensional spaces [Weber et al. 1998; Hinneburg et al. 2000]. In such cases, almost the entire index structure is accessed by a single query.

In order to accelerate the search, it is usual to offer a quality/time trade-off: for saving search time, it is accepted a degradation in the quality of the result. This is the goal of *approximate similarity search* [Amato and Savino 2008].

Approaches to approximate similarity search can be broadly classified in methods that exploit space transformations and methods that reduce the amount of data to be accessed [Zezula et al. 2005]. In the first category, approximation is achieved by changing the object representation and/or distance function with the objective of reducing search cost [Gionis et al. 1999; Valle et al. 2008; Weber and Böhm 2000]. In the second category, strategies omit parts of the dataset that are not likely to contain qualified objects [Bennett et al. 1999; Li et al. 2002; Rocha et al. 2008].

Many transformation techniques need objects to be represented as vectors and cannot be directly applied to generic metric spaces. In those cases, a natural approach is to embed the feature space into a vector space, so that the distances of the embedded objects approximate the actual distances [Hjaltason and Samet 2003]. Recent studies, however, have shown that using such embedding over high-dimensional metric spaces is sometimes not very effective, incurring in a high approximation error, so that practically all distance information is lost [Khot and Naor 2005].

On the other hand, techniques that reduce the amount of data examined aim at improving performance by accessing and analyzing less data than is technically needed. State-of-the-art indexes are constructed based on partitioning a set of objects using distance-based criteria. In general, those techniques follow two basic paradigms: disjoint or non-disjoint. The former partitions the data set into disjoint clusters, but ignores the distribution properties of the data [Bustos and Navarro 2004; Xu et al. 2008; Chávez et al. 2008]. The latter produces non-disjoint groups, which greatly affect the search performance [Zezula et al. 1998; Clarkson 1999; Goldstein and Ramakrishnan 2000; Chávez and Navarro 2003; Lin et al. 2005]. In order to keep the balance of the structure, they divide the data set into even sized parts, ignoring the inherent grouping of data.

3. RELATED WORK

In this work, we are interested in approximate algorithms, which relax the condition of delivering the exact solution. A survey on techniques for approximate similarity search is presented in [Patella and Ciaccia 2008; 2009]. It proposes a classification scheme for existing approaches, considering the most relevant characteristics of them: type of data (metric or vector spaces), error metrics (changing space or reducing comparisons), quality guarantees (none, deterministic or probabilistic parametric/non-parametric), and user interaction (static or interactive).

Three different algorithms to solve approximate k-NN queries with the M-tree [Ciaccia et al. 1997] are presented in [Zezula et al. 1998]. The first one reduces the current searching radius of the k-NN query through relative distance errors. Another technique employs the distance distribution to stop the search when the probability of finding a better result does not exceed an user-specified threshold. The third technique simply interrupts the search when the improvement in the result set falls below a threshold.

A data structure called $M(S, Q)$ to answer nearest neighbor queries is proposed in [Clarkson 1999]. It requires a training data set Q of objects, taken to be representative of typical query objects. This data structure may fail to return a correct answer, but the failure probability can be arbitrarily small at the cost of increasing the query time and space requirements for the index.

The P-Sphere tree [Goldstein and Ramakrishnan 2000] is a two-level index structure for approximate 1-NN search. In order to find the nearest neighbor of the query object, the partition closest to the query object is accessed. The query is solved through a sequential scan of objects contained in such partition.

The approach described in [Chávez and Navarro 2003] is a probabilistic framework based on stretching the triangle inequality. The idea is general, but the authors applied it to pivot-based searching

algorithms. Their technique allows for the reduction of the search radius by using the (inverse of the) distance distribution so as to provide a probabilistic guarantee on the approximate result.

In [Lin et al. 2005], several techniques are presented to perform approximate searches in high-dimensional spaces using R-tree-like structures [Guttman 1984]. Basically, recognizing the fact that the difficult task in k-NN searching is to guarantee the correctness of the result, the authors propose several heuristics for aggressively pruning partitions. Two of such heuristics use Monte Carlo simulation to estimate the probability of finding a better result in a given partition, while the remaining three strategies estimate such probability by using distance bounds for the hyper-rectangular region associated to each partition. Partitions leading to a probability lower than an user-specified threshold are pruned from the search.

The method presented in [Chávez et al. 2008] performs similarity queries by searching for objects that can give enough positive proofs to include them in a list of candidates. This process is the reverse of that adopted in many existing indexes, which attempt to find a proof to reject objects from the candidate list. The algorithmic idea is quite simple and works as a modification of the basic pivot-based LAESA sequential algorithm [Micó et al. 1994]. For each object in the data set the pivots are sorted from nearest to farthest; the same is performed for the query. Then, objects in the data set are sorted for increasing similarity of their sorted lists of pivots to the query object. Finally, only an user-specified fraction of objects is examined.

Locality Sensitive Hashing (LSH) [Gionis et al. 1999] is currently the most popular high-dimensional indexing method. Unlike most other index structures, the algorithmic idea behind LSH is not based on a tree structure, but on hashing the data set into buckets. The chosen hash functions are constructed in order to guarantee that very close objects coincide in the same bucket with much higher likelihood than those far apart.

One major benefit of LSH is the simplicity of its algorithmic idea. In the LSH, the objects are represented as vectors. Each vector is projected onto a set of k random lines through the search space. The lines are partitioned into fixed-sized intervals (determined by a radius R) and each of the intervals is named by a symbol. Projecting to k lines gives k symbols, which are then concatenated to a word of length k. These words are built over an alphabet, whose cardinality is defined by the number of partition intervals, and form a kind of locality sensitive fingerprint. The smaller the radius R, the more intervals are created and, hence, the more symbols the alphabet contains.

In order to efficiently search for those vectors, they are hashed via a standard hash function into a hash table. Since the partitions do not overlap and the likelihood of separating two close neighbors also increases with fingerprint length k, it needs several such hash tables (parameter L in LSH notation) to guarantee a certain probability in recall.

During query processing with LSH, the query vector q needs to look up the appropriate buckets for all L hash tables. Therefore, q is projected to all k lines for each individual table and the result is concatenated to a k length fingerprint, which then references the bucket in the hash table that must be read from disk. At this point, a sequential scan is performed over all candidate vectors referenced in this bucket, and those qualifying for the query are returned. After all L hash tables have been looked up this way, all vectors in the result set are sorted according to their distances to q and returned.

Different from all of the previous techniques, DAHC-tree does not divide the data set into disjoint or non-disjoint groups. Instead, it is an index structure that combines the advantages of both those strategies. Moreover, DAHC-tree splits the data set into compact sets by respecting its distribution, not a predefined size for them.

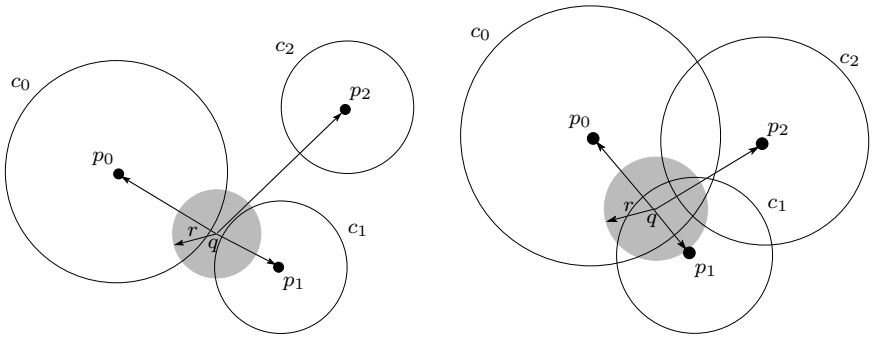
4. THE DAHC-TREE

Traditional index structures based on disjoint partitioning approaches use early termination strategies which stop the similarity search before its natural (exact) end. It is generally processed by first identifying the partition to which the query object belongs. This is usually obtained by traversing

the index structure using partitions whose reference object is closest to the query object. Once the above partition is identified, a sequential scan is performed, and objects qualifying for the query are returned.

In contrast, the techniques based on non-disjoint groups usually employ relaxed branching strategies in order to avoid accessing data that are not likely to contain qualified objects for the query. For this, the approximate similarity search algorithm is performed by visiting the groups in increasing order of distance from their reference objects to the query object. The algorithm terminates when the distance from the reference object of the current group to the query object satisfies a given stop condition.

If the objects in the data set are uniformly distributed, there exists a high probability of the query region intersects several clusters and, hence, a lot of qualified objects cannot be returned. In order to clarify the above situation, look at Fig. 1, in which a range query (gray region) using the query object q and a range r is posed. The circumferences bound three clusters c_0, c_1, and c_2. The reference object of each cluster is denoted by the points p_0, p_1, and p_2, respectively.



(a) Disjoint partitions. (b) Non-disjoint partitions.

Fig. 1. Effects of the partitioning paradigms in a data set with an uniform distribution.

Fig. 1(a) illustrates the effects of the use of disjoint partitions. In this case, the query object q is closest to the reference object p_1, thus only objects of the partition c_1 that are covered by the region of the query response are returned. Note that objects belonging to the partition c_0 and which also qualify for the query cannot be returned. However, if the query range r is small enough, the whole query can be found in a single partition.

We show the effects of non-disjoint partitioning methods in the Fig. 1(b). In this figure, the query object q is closest to the reference object p_1, thus the group c_1 is the first to be visited. Next, the algorithm may analyze the groups c_0 and c_2, respectively. If the stopping criterion is rather relaxed, all the three groups c_0, c_1, and c_2 can be accessed. On the other hand, when a more relaxed stopping condition is used, many qualified objects can be rejected. Therefore, it is difficult to define a suitable stopping criteria in order to guarantee a good quality/time trade-off.

When the objects in the data set are sparsely distributed, both disjoint and non-disjoint paradigms may produce tight clusters. Nevertheless, the use of fixed-sized partitions in order to maintain the balance of those structures may divide inherent grouping of data, as illustrated in Fig. 2.

In the example, there are two natural groups in the data, bounded by the dashed circumferences. Due to the balance constraints, each group may be divided into smaller clusters, delimited by solid circumferences. When it happens, the shortcomings previously discussed may arise and, consequently, the search results may be greatly affected.

The novelty of DAHC-tree is to combine the advantages of both disjoint and non-disjoint approaches. In DAHC-tree, the data set is divided into compact clusters by respecting its distribution. This strategy overcomes the above disadvantages.

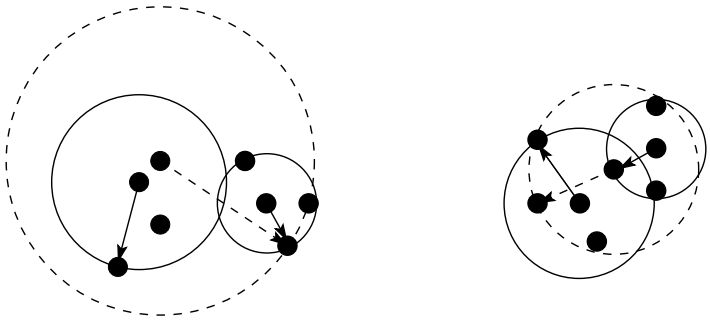


Fig. 2. Effects of fixed-sized partitions in a data set with a sparse distribution.

4.1 Overview of DAHC-tree

Consider an initial set of objects, we first divide the objects into groups based on their global distribution. We can refine this partitioning further by dividing each existing group based on the local distribution of a subset of objects. This process may be repeated by taking a smaller subset at each time until no further improvements are possible. Finally, we have a hierarchical set of groups. This is roughly the basic idea of DAHC-tree, where such a model is adapted to disk. In other words, given a query object, we can reduce the search space by gradually considering a subset of objects with a more relevant distribution.

At each level, we cluster the data around reference objects. Each cluster is a partition in the sense that objects in the same cluster have similar distances to its reference object. Next, we create subsets of data by combining adjacent clusters. This strategy captures the data distribution in the sense that far away objects are separated into non-adjacent clusters. After that, we apply this process recursively until each subset of data can no longer be divided. Finally, we have a set of clusters, which is a hierarchical partitioning of the data.

In order to explicitly highlight the novelty of DAHC-tree, we elaborate further on how DAHC-tree benefits and combines the advantages of both disjoint and non-disjoint partitioning approaches. On the one hand, DAHC-tree partitions the data set into clusters based on distances to reference objects. There exists a full order on distances to a same reference object and, hence, the clusters are disjoint. On the other hand, DAHC-tree generates different subsets of data for each level by combining objects of adjacent clusters. The same object may appear into several subsets of data, thus the clusters may overlap.

At the end, it is noteworthy that DAHC-tree may be unbalanced. However, for similarity search in high-dimensional spaces, unbalanced trees may provide better performance than balanced trees, as stated by Chávez and Navarro [Chávez and Navarro 2005]. They have shown that, for similarity search in high-dimensional spaces, the search cost is determined by the pruning rate of the search space, not by the height of the tree. The pruning rate of the search space is directly related to how the data set is separated. The balanced tree partitions the data set into even sized parts, ignoring the data distribution. DAHC-tree partitions the data set by the data distribution, thus it may separate the data set better than balanced partitioning. For a more detailed discussion of the benefits of unbalanced trees in similarity search, refer to [Chávez and Navarro 2005].

4.2 DAHC-Tree Creation

Overall, DAHC-tree is an unbalanced tree index generated by the hierarchical partitioning of the data set. Like other metric trees, the objects of the data set are stored into fixed size disk pages. Each page holds a predefined maximum number of objects K. Table I summarizes the symbols used in this paper.

Table I. Summary of symbols and definitions

Symbols	Definitions
$d(x, y)$	distance function between objects x and y
f	number of partitions selected for creating a subset of objects
k	the number of partitions spanned by a set
K	capacity of a disk page
N	number of objects in a set
O	a set of objects
C	a set of partitions

DAHC-tree has two kinds of nodes: leaf nodes and index nodes. Each index node corresponds to a single disk-page and contains a partitioning information. In contrast, each leaf node consists of a list of disk pages and, hence, may have an unlimited capacity. The objects of the data set are stored in both index and leaf nodes.

The structure of a leaf node is

$$leafnode\ [\ \text{array of} < oid(o_i), d(o_i, o_{ref}), o_i >\ ],$$

where $oid(o_i)$ is the identifier of the object o_i and $d(o_i, o_{ref})$ is the distance from the object o_i to the reference object o_{ref} of this leaf node. The structure of an index node is

$$indexnode\ [\ \text{array of} < o_i, r(o_i), d(o_i, o_{ref}), ptr(T(o_i)) >\ ],$$

where o_i keeps the reference object of the subtree $T(o_i)$ pointed by $ptr(T(o_i))$ and $r(o_i)$ is the covering radius of that region. The distance between o_i and the reference object of this node o_{ref} is kept in $d(o_i, o_{ref})$. The pointer $ptr(T(o_i))$ points to the root node of the subtree $T(o_i)$ rooted by o_i.

The tree construction is performed in a top-down fashion. In order to clarify this approach, look at Fig. 3. At the beginning, the set of objects $O = \{o_1, o_2, \ldots, o_N\}$ is considered to be part of a single partition. Objects in this set are first divided into $k \leq K$ disjoint subpartitions $c_1, c_2, \ldots, c_k$. Information about all those subpartitions form the index node of the first level of the tree. For each partition c_i, a subset O_{c_i} is created by grouping the objects of c_i and the objects of f adjacent partitions. To build subsequent levels of the tree, this process of dividing and grouping is repeated for all of the new subset of objects at each level, creating the hierarchy of index nodes. The process stops when the number of objects in a subset is less than or equals to K or the number of partitions spanned by a subset is less than the double of f. Then, the objects in the subset are written to a leaf node on disk.

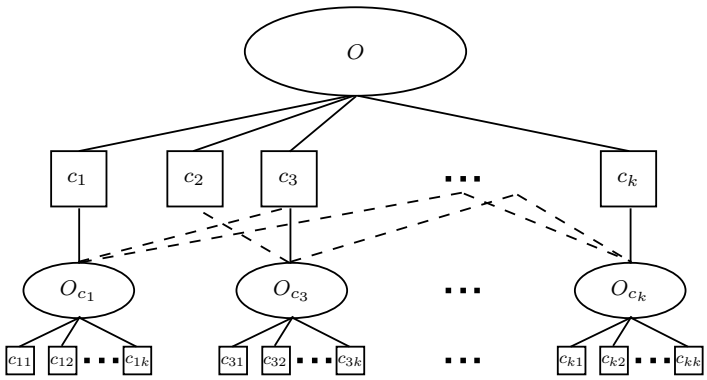


Fig. 3. A representation of DAHC-tree.

Algorithm 1 formalizes the above procedure. It starts by checking the cardinality of the set of

objects O (line 2). If it can fit into a disk page, the function CREATE-LEAFNODE is used to create a leaf node (line 3). Otherwise, we call the function SPLIT in order to divide the set into $k \leq K$ partitions (line 5). The function SPLIT can use any partitional clustering method, such as k-medoids [Bishop 2006]. The partitional algorithm is responsible for finding the reference objects of each level. Next, we check if the set can be divided (line 7). If so, we call the function CREATE-INDEXNODE, that creates an index node using those reference objects (line 10). Thereafter, for each partition, the function COMBINE is used to create a subset of objects (line 12). This function is responsible for grouping objects of adjacent partitions. After that, we repeat this process for all of the new subset of objects (line 14). Finally, the function UPDATE-INDEXNODE is used to update the information in each entry of the index node (line 15).

Algorithm 1 DAHC-tree construction.

```
 1: function TREE-CONSTRUCT(f, d, K, N, O)
 2:     if N ≤ K then
 3:         return CREATE-LEAFNODE(N, O);
 4:     else
 5:         C ← SPLIT(d, K, N, O);                          ▷ Divisive step
 6:         k ← cardinality(C);
 7:         if k < 2 f then
 8:             return CREATE-LEAFNODE(N, O);
 9:         else
10:             Parent ← INITIALIZE-INDEXNODE(k, C);
11:             for each c ∈ C do
12:                 O_c ← COMBINE(f, d, c, k, C);           ▷ Agglomerative step
13:                 N_c ← cardinality(O_c);
14:                 Child ← TREE-CONSTRUCT(f, d, K, N_c, O_c);   ▷ Deepening
15:                 UPDATE-INDEXNODE(c, Parent, Child);
16:             end for each
17:             return Parent
18:         end if
19:     end if
20: end function
```

In our implementation, the algorithms for choosing reference objects in the function SPLIT are: **kmedoids**, which uses the well-known PAM (Partitioning Around Medoids) algorithm [Bishop 2006] to partition data; and **random**, which partitions the data using reference objects selected at random (i.e., the initialization step of the PAM algorithm, which is the most common realization of the k-medoid clustering). The latter is the default method due to its better performance.

DAHC-tree provides two options for the function COMBINE: **mindistance**, in which the partitions are combined using the minimum distance between their reference objects; and **maxoverlap**, in which the partitions are combined using the maximum overlap between their cover regions. The default method for the COMBINE algorithm is "*mindistance*".

4.3 Similarity Queries

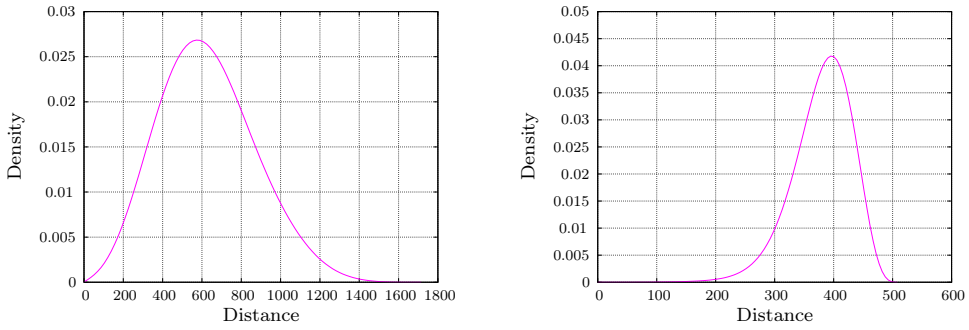
DAHC-tree uses early termination strategies to answer similarity queries. During query processing, the query object first traverses the hierarchy of index nodes of the DAHC-tree. At each level, we compute the distances from the query object to the reference objects. The search is then directed to the partition whose reference object is closest to the query object. This process is repeated until the search reaches a leaf node. At this point, a sequential scan is performed, and objects qualifying for the query are returned.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate and compare the performance of our technique in different scenarios. We implemented DAHC-tree from scratch in C++. The experiments were performed on a machine equipped with a processor Intel Xeon QuadCore X3320 2.5 GHz and 8 Gbytes of DDR2-memory. The machine run Gentoo Linux (2.6.31 kernel) and the ext3 file system.

DAHC-tree was tested using two sets of images described in literature and extensively used by the computer vision and image processing communities. The first set contains 72,000 images from Amsterdam Library of Object Images (ALOI)[1] [Geusebroek et al. 2005]. We converted each image to a 256-dimensional feature vector by computing a Color Correlogram [Huang et al. 1997]. Each color correlogram is a table indexed by color pairs, where the k-th entry for a pair $< i, j >$ specifies the probability of finding a pixel of color j at a distance k from a pixel of color i in the image. The distance function used to compare the feature vectors is the Manhattan (l_1) distance. The other data set was obtained by extracting local features from the ETH-80 Image Set[2] [Leibe and Schiele 2003], which is a set of 3,280 images. In this study, we use the well-known SIFT method [Lowe 2004], which is the most popular approach for extracting local features from images. The resulting collection contains a total of 134,173 SIFT descriptors. Each SIFT descriptor consists of a 128-dimensional feature vector. The distance function used to compare the feature vectors is the Euclidean (l_2) distance.

Fig. 4 shows the distance density functions of both databases. Observe the differences in densities of the individual data collections. It is worth noting that those databases are characterized by different distance distributions.



(a) ALOI Dataset (b) ETH-80 Dataset

Fig. 4. Distance density functions of the data set used in the experiments.

We randomly selected about one percent of each collection to be used as queries (700 images for the ALOI dataset and 1,300 SIFT descriptors for the ETH-80 dataset). Five replications were performed for each corpora to ensure statistically sound results. For both the datasets, we performed range queries, with a search radius that retrieved, on average, 0.05 percent of the database (i.e., 36 images for the ALOI dataset and 67 SIFT descriptors for the ETH-80 dataset). The ground-truth were obtained by an exhaustive sequential scan over those collections.

The measurement taken at each experiment were the recall (i.e, the ratio between the number of qualifying objects retrieved and the total number of qualifying objects) and the average number of distance calculations. We performed five replications for each test in order to guarantee statistically significant results.

Our experiments are intended to answer the following questions:

[1] `http://staff.science.uva.nl/~aloi/`
[2] `http://tahiti.mis.informatik.tu-darmstadt.de/oldmis/Research/Projects/categorization/eth80-db.html`

- How do fixed-sized partitions affect the recall of an approximate search?
- How do the parameters of DAHC-tree affect its performance?
- How does the performance of DAHC-tree compare to LSH?
- How does the performance of DAHC-tree compare to exact search algorithms?

In the following, we report and discuss the results obtained for each question above.

5.1 Fixed-Sized versus Variable-Sized Partitions

In this section, we study how the relationship between the geometry of the data and the organization of an index structure may greatly affect the search performance. For this purpose, we compare two different approaches. The first is the algorithm presented in Section 4.2, that divides the data according to their distribution, allowing for partitions with different sizes. The other is a simple modification of the previous technique, in which the data set is divided into even sized partitions, ignoring the inherent grouping of data. This is obtained by eliminating the stop condition implemented in the lines 7-9 of the Algorithm 1.

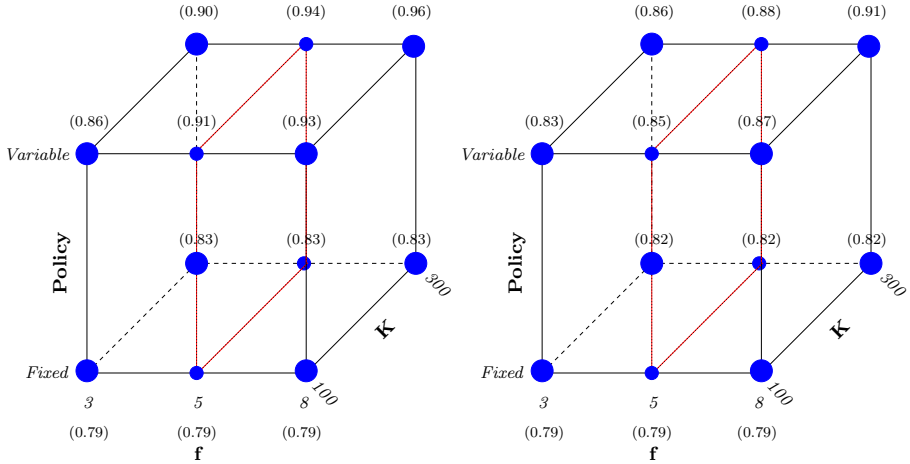
In those experiments we have evaluated the interplay between three parameters:

- The number f of partitions selected for creating a new level: the larger this parameter, the more the intensity of replicated objects on the DAHC-tree. Thus, it tends to improve the quality of the results at the cost of creating larger trees. We tried the factors 3, 5, and 8.
- The page capacity K: larger disk pages tend to improve the quality of the results at the cost of more processing time (since more data gets to be examined). It also tends to lessen the space overhead (since the trees tend to be shallower). We tried disk pages with a capacity to store 100 and 300 objects.
- The leaf size policy: with fixed-sized partitions, the leaves are forcibly divided in order to fit in the specified size, even when all data belongs to tight cluster. With variable-sized partitions we authorize the creation of "big leaves" in order to avoid breaking up inherent grouping of data.

The experiment consisted in testing exhaustively the combinations of all selected parameter levels. The results are shown in Figure 5, where each axis of the cube represents variation in a single parameter. The value parentheses is the recall achieved by each combination of those parameters. The analysis of the experiment is interesting, for it shows that when the leaf is fixed (bottom face), the combination of adjacent partitions does not help to improve the results. This happens because data which will be grouped for geometrical reasons will be later separated because of the excessively restraining impositions of the data structure. When the leaf is variable, however, the results improve as more partitions are combined, as we had expected. In both cases, larger leaves tend to give better results.

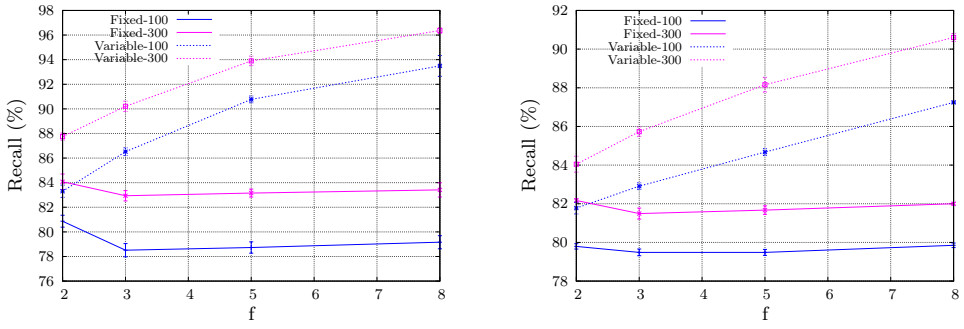
Those results demonstrate the strong interaction between the geometric and data structure constraints, showing that when those are not compatible, the index as a whole suffers. Fig. 6 shows a different view of the results including an additional level for the parameter f (2, 3, 5, 8). If the parameter f is equals to 2, DAHC-tree plays a special case, since the distances between objects are symmetric in metric spaces. Otherwise, the results confirm that the variable strategy consistently outperforms the fixed one, with a high statistical significance (confidence higher than 0.99).

Fig. 7 shows the space occupation (in terms of the number of disk pages) by the index structure for different combinations of those parameters. We use log scale in order to highlight the behavior of each choice. Notice that the space requirements are reduced by changing the leaf size policy from fixed to variable. One of the reasons is the better occupation of the nodes. It is noteworthy, therefore, that, only by organizing the index structure in a different way, we significantly improved the search results and the storage utilization.



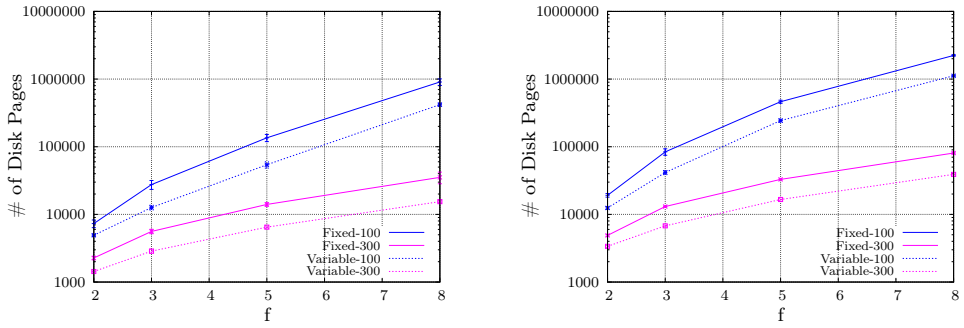
(a) ALOI Dataset (b) ETH-80 Dataset

Fig. 5. The interplay between the parameter f, the page capacity K, and the leaf size policy. Each axis of the cube represents variation in a single parameter. The value parentheses is the recall achieved by each combination of those parameters.



(a) ALOI Dataset (b) ETH-80 Dataset

Fig. 6. The recall achieved by different combinations of the levels selected for the parameter f, the page capacity K, and the leaf size policy.



(a) ALOI Dataset (b) ETH-80 Dataset

Fig. 7. The space occupation by the index structure for different combinations of the levels selected for the parameter f, the page capacity K, and the leaf size policy.

5.2 Exploration of the Parameter Space

In this section, we explore the parameter space of DAHC-tree by performing a full factorial design. The factorial design reveals the relative importance of each parameter of DAHC-tree, including cross-

effects. We have covered the four most important parameters, the page capacity K, the number f of partitions selected for creating a new level, the strategies for choosing reference objects in the SPLIT algorithm, and the strategies for the COMBINE algorithm.

Table II summarizes the results of the factorial analysis of variance (ANOVA) in the ALOI dataset. We separately analyzed the influence of the parameters of DAHC-tree for both the effectiveness (quality of search results) and the efficiency (number of distance calculations). The percentages indicate the relative contribution of each parameter to the observed variation in each analyzed response. For instance, a simple change of strategy in the SPLIT algorithm explains 4.73% of the variation observed in the effectiveness of DAHC-tree. The models p-values indicate high statistic significance. Notice that both the effectiveness and the efficiency of DAHC-tree is higher dependent to the choice of the parameters K and f.

Table II. ANOVA results for each parameter of DAHC-tree in the ALOI dataset.

Factors			**Explanatory Power**	
Parameter	*Levels*		*Effectiveness*	*Efficiency*
K	200	400	53.24%	88.39%
f	3	5	38.91%	9.90%
SPLIT	kmedoids	random	4.73%	0.21%
COMBINE	mindistance	maxoverlap	0.90%	0.04%
		Model p-values:	< 0.0001	< 0.0001

The summary results obtained by the analysis of DAHC-tree in the ETH-80 dataset are presented in the Table III. As in the ALOI dataset, the parameters K and f were the most responsible for the observed variations. A few cross-effects were found statistically significant, but their individual contributions were always less than 3%.

Table III. ANOVA results for each parameter of DAHC-tree in the ETH-80 dataset.

Factors			**Explanatory Power**	
Parameter	*Levels*		*Effectiveness*	*Efficiency*
K	200	400	47.51%	78.04%
f	3	5	45.29%	19.00%
SPLIT	kmedoids	random	5.12%	0.00%
COMBINE	mindistance	maxoverlap	0.21%	0.01%
		Model p-values:	< 0.0001	< 0.0001

One of the drawbacks of DAHC-tree relies on the fact that its parameters K and f must be tuned to obtain quality results. Although the proper values of K and f are data set dependent, we empirically found the following rules of thumb to be useful for finding good values:

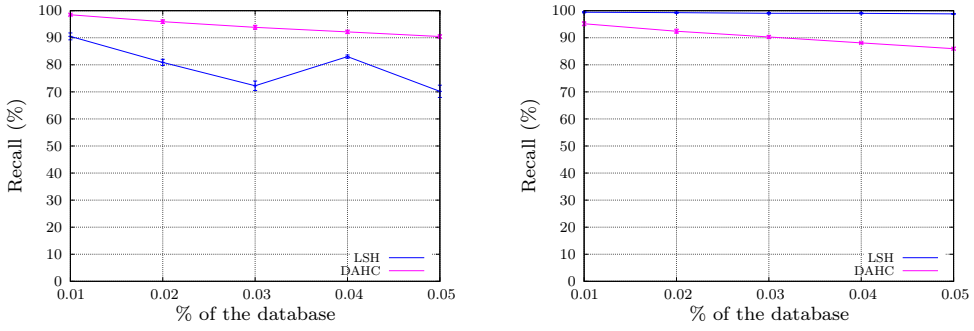
- Choose a reasonable page capacity K. Once a leaf node may hold several disk pages, their size must be large enough to avoid breaking up inherent grouping of data. It must also contain enough objects so that if a query object is near to a partition, then the probability that a significant number of qualified objects are in the partition is high. On the other hand, a partition should not be so large as to prolong query time unnecessarily.
- Determine the number f of partitions selected for creating a new level. The f value must be at least three, so that objects may be combined. Once a page capacity K is chosen, one can compute how many partitions the data set will be divided. If the number of partitions is too small, we may reduce the page capacity K in order to increase the number of partitions.

5.3 Comparison with LSH

In this section, we compare DAHC-tree with LSH, the most popular approach for conducting approximate similarity search in high-dimensional spaces. One major drawback of LSH is its necessity of the data be represented as vectors and, hence, it cannot be directly applied to generic metric spaces. In those cases, a natural approach is to embed the feature space into a vector space, so that the distances of the embedded objects approximate the actual distances [Hjaltason and Samet 2003]. Recent studies, however, have shown that using such embedding over high-dimensional metric spaces is sometimes not very effective, incurring in a high approximation error, so that practically all distance information is lost [Khot and Naor 2005].

As we have explained in Section 3, LSH relies on three parameters: the fingerprint length k, the number of hash tables L, and the search radius R. For our experimental evaluation, we adopted the original LSH implementation[3]. It is optimized to estimate the best choice for both parameters k and L based on the data distribution and a given radius R. In order to guarantee a fair comparison, we created a new index structure for each radius R. The parameters used to build a DAHC-tree were: "mindistance" for the COMBINE algorithm, the "kmedoids" strategy for reference objects, disk pages with a page capacity K equals to 300, and f set to 3.

Fig. 8 presents a comparison of the recall of LSH and DAHC-tree for different query radii (in terms of the average percentage of the database retrieved by the search radius). Note that, for the ALOI dataset, by increasing the search radius, DAHC-tree performs better than LSH, with a high statistical significance (confidence higher than 0.99). For instance, considering a search radius that retrieves, on average, 0.03 percent of the database (i.e., 21 images), DAHC-tree is ≈30% (22 percentual points) better than LSH. One of the reasons is a consequence of the definition of LSH: the larger the radius R, the smaller the gap between the probabilities of collision for close points and far points [Andoni and Indyk 2008]. On the other hand, for the ETH-80 dataset, LSH outperforms DAHC-tree. This happens because the hash functions for the Euclidean space (l_2) are more stable than for Manhanttan distances (l_1).



(a) ALOI Dataset (b) ALOI Dataset

Fig. 8. Recall for LSH and DAHC-tree as a function of the query radius (in terms of the average percentage of the database retrieved by the search radius).

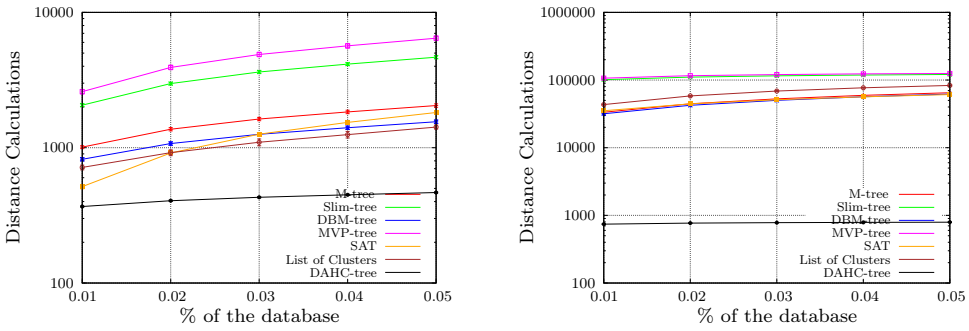
5.4 Comparison with exact searching algorithms

In this section, we compare DAHC-tree with MVP-tree [Bozkaya and Özsoyoglu 1999], SAT [Navarro 2002], List of Clusters [Chávez and Navarro 2005], M-tree [Ciaccia et al. 1997], Slim-tree [Traina Jr. et al. 2002], and DBM-tree [Vieira et al. 2006], which are the most popular approaches for exact similarity search in generic metric spaces.

[3] `http://www.mit.edu/~andoni/LSH/`

For our experimental evaluation, we adopted the implementation of MVP-tree, SAT, and List of Clusters from the Metric Space Library[4] and the implementation of M-tree, Slim-tree, and DBM-tree from the GBDI Arboretum Library[5]. In order to guarantee a fair comparison, all the compared methods were configured using their best recommended setup. DAHC-tree was built using the same parameters reported in Section 5.3.

Fig. 9 presents a comparison of the efficiency (number of distance calculations) of DAHC-tree and the exact techniques for different query radii (in terms of the average percentage of the database retrieved by the search radius). The results are plotted in log scale to minimize the large difference resulting from queries with small and large radii, which makes the comparison easier.



(a) ALOI Dataset (b) ETH-80 Dataset

Fig. 9. The query performance (given by the average number of distance calculations) for DAHC-tree and the exact methods as a function of the query radius (in terms of the average percentage of the database retrieved by the search radius).

Clearly, DAHC-tree is more efficient than the exact methods for performing similarity queries, with a high statistical significance (confidence higher than 0.99). This effect is visible for both databases. Notice that DAHC-tree saves at least 50% of distance calculations when compared to the best exact technique.

It can be seen from the plots in Fig. 8 and Fig. 9 that DAHC-tree improves the efficiency on similarity queries by orders of magnitude while incurring in small loss of effectiveness (typically 5-15%) regardless the database. For instance, considering the ETH-80 dataset and a search radius that retrieves, on average, 0.03 percent of the database (i.e., 40 SIFT descriptors), DAHC-tree achieves ≈90% of recall (i.e., 36 SIFT descriptors). However, the effectiveness loss is small when compared to its efficiency gain. For the same settings, DAHC-tree saves more than 98% of distance calculations when compared to the best exact algorithm.

6. CONCLUSIONS

In this paper, we have shown how the relationship between the geometry of the data and the organization of an index structure may greatly affect the search performance of both disjoint and non-disjoint techniques described in the literature.

Furthermore, we have presented DAHC-tree, a new approach for performing approximate similarity search in high-dimensional metric spaces. It is an index structure that combines the advantages of both disjoint and non-disjoint strategies. DAHC-tree is constructed by dividing and grouping the data set into compact clusters by respecting its distribution. This strategy reduces the effects imposed by the above liability.

[4] http://www.sisap.org/Metric_Space_Library.html
[5] http://www.gbdi.icmc.usp.br/arboretum/

Our experiments have demonstrated the strong interaction between the geometric and data structure constraints, showing that when those are not compatible, the index as a whole suffers. Through experiments we also have learned that DAHC-tree is higher dependent to the choice of the parameters K and f. In spite of that, the conducted analysis and the reported comparative test results have shown that DAHC-tree significantly improves the performance in processing similarity queries.

Finally, we summarize the limitations of DAHC-tree and our future research plan.

- *Control parameter tuning.* As we discussed in Section 5.2, the effectiveness and the efficiency of DAHC-tree is higher dependent to the choice of the parameters K and f. We have provided some parameter-tuning guidelines in the paper. We plan to investigate a mathematical model which allows directly to determine the parameters.
- *Incremental partitioning.* In addition, most top-down approaches are offline algorithms and the partitions can be sensitive to insertions and deletions. We plan to extend DAHC-tree to perform regional repartitioning for supporting insertions and deletions after the initial creation of the index structure.
- *Measuring performance using other metrics.* In this study, we use only the recall to measure the performance of similarity queries. We plan to employ other metrics (e.g., [Zezula et al. 2005]) to compare the performance between different indexing schemes.

ACKNOWLEDGMENT

We would like to thank Anderson Rocha, Mario A. Nascimento, and Siome Goldenstein, for their valuable suggestions. This research was supported by Brazilian agencies FAPESP (Grant 07/52015-0, 08/50837-6, 09/18438-7, and 09/05951-8), CNPq (Grant 311309/2006-2, 472402/2007-2, and 306587/2009-2), and CAPES (Grant 01P-05866/2007).

REFERENCES

Almeida, J., Torres, R. S., and Leite, N. J. Bp-tree: An efficient index for similarity search in high-dimensional metric spaces. In *Proceedings of the ACM International Conference on Information and Knowledge Management*. Toronto, ON, Canada, 2010.

Amato, G. and Savino, P. Approximate similarity search from another "perspective". In *Proceedings of the Italian Symposium on Advanced Database Systems*. Mondello, Italy, pp. 247–254, 2008.

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM* 51 (1): 117–122, 2008.

Bennett, K. P., Fayyad, U. M., and Geiger, D. Density-based indexing for approximate nearest-neighbor queries. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Diego, CA, USA, pp. 233–243, 1999.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Inc., 2006.

Bozkaya, T. and Özsoyoglu, Z. M. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems* 24 (3): 361–404, 1999.

Bustos, B. and Navarro, G. Probabilistic proximity searching algorithms based on compact partitions. *Journal of Discrete Algorithms* 2 (1): 115–134, 2004.

Chávez, E., Figueroa, K., and Navarro, G. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (9): 1647–1658, 2008.

Chávez, E. and Navarro, G. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters* 85 (1): 39–46, 2003.

Chávez, E. and Navarro, G. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters* 26 (9): 1363–1376, 2005.

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

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 Data Bases*. Athens, Greece, pp. 426–435, 1997.

Clarkson, K. L. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry* 22 (1): 63–93, 1999.

Elmasri, R. A. and Navathe, S. B. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 2005.

Gaede, V. and Günther, O. Multidimensional access methods. *ACM Computing Surveys* 30 (2): 170–231, 1998.

Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. M. The amsterdam library of object images. *International Journal of Computer Vision* 61 (1): 103–112, 2005.

Gionis, A., Indyk, P., and Motwani, R. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*. Edinburgh, Scotland, pp. 518–529, 1999.

Goldstein, J. and Ramakrishnan, R. Contrast plots and p-sphere trees: Space vs. time in nearest neighbour searches. In *Proceedings of the International Conference on Very Large Data Bases*. Cairo, Egypt, pp. 429–440, 2000.

Guttman, A. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Boston, MA, USA, pp. 47–57, 1984.

Hinneburg, A., Aggarwal, C. C., and Keim, D. A. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the International Conference on Very Large Data Bases*. Cairo, Egypt, pp. 506–515, 2000.

Hjaltason, G. R. and Samet, H. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (5): 530–549, 2003.

Huang, J., Kumar, R., Mitra, M., Zhu, W.-J., and Zabih, R. Image indexing using color correlograms. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. San Juan, Puerto Rico, pp. 762–768, 1997.

Jagadish, H. V., Mendelzon, A. O., and Milo, T. Similarity-based queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*. San Jose, CA, USA, pp. 36–45, 1995.

Khot, S. and Naor, A. Nonembeddability theorems via fourier analysis. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*. Pittsburgh, PA, USA, pp. 101–112, 2005.

Leibe, B. and Schiele, B. Analyzing appearance and contour based methods for object categorization. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. Madison, WI, USA, pp. 409–415, 2003.

Li, C., Chang, E. Y., Garcia-Molina, H., and Wiederhold, G. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering* 14 (4): 792–808, 2002.

Lin, K.-I., Nolen, M., and Kommeneni, K. Utilizing indexes for approximate and on-line nearest neighbor queries. In *Proceedings of the IEEE International Symposium on Database Engineering and Applications*. Montreal, Canada, pp. 83–88, 2005.

Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60 (2): 91–110, 2004.

Micó, L., Oncina, J., and Vidal, E. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15 (1): 9–17, 1994.

Navarro, G. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (1): 28–46, 2002.

Patella, M. and Ciaccia, P. The many facets of approximate similarity search. In *Proceedings of the IEEE International Workshop on Similarity Search and Applications*. Cancun, Mexico, pp. 10–21, 2008.

Patella, M. and Ciaccia, P. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms* 7 (1): 36–48, 2009.

Ramakrishnan, R. and Gehkre, J. *Database Management Systems*. McGraw-Hill Co., Inc., 2003.

Rocha, A., Almeida, J., Nascimento, M. A., Torres, R., and Goldenstein, S. Efficient and flexible cluster-and-search approach for cbir. In *Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems*. Juan-les-Pins, France, pp. 77–88, 2008.

Traina Jr., C., Traina, A. J. M., Faloutsos, C., and Seeger, B. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering* 14 (2): 244–260, 2002.

Valle, E., Cord, M., and Philipp-Foliguet, S. High-dimensional descriptor indexing for large multimedia databases. In *Proceedings of the ACM International Conference on Information and Knowledge Management*. Napa Valley, CA, USA, pp. 739–748, 2008.

Vieira, M. R., Traina Jr., C., Chino, F. J. T., and Traina, A. J. M. DBM-tree: Trading height-balancing for performance in metric access methods. *Journal of the Brazilian Computer Society* 11 (3): 37–52, 2006.

Weber, R. and Böhm, K. Trading quality for time with nearest neighbor search. In *Proceedings of the International Conference on Extending Database Technology*. Konstanz, Germany, pp. 21–35, 2000.

Weber, R., Schek, H.-J., and Blott, S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Data Bases*. New York, NY, USA, pp. 194–205, 1998.

Xu, W., Miranker, D. P., Mao, R., and Ramakrishnan, S. R. Anytime k-nearest neighbor search for database applications. In *Proceedings of the IEEE International Workshop on Similarity Search and Applications*. Cancun, Mexico, pp. 139–148, 2008.

Zezula, P., Amato, G., Dohnal, V., and Batko, M. *Similarity Search: The Metric Space Approach*. Springer-Verlag, Inc., 2005.

Zezula, P., Savino, P., Amato, G., and Rabitti, F. Approximate similarity retrieval with m-trees. *The VLDB Journal* 7 (4): 275–293, 1998.