Approximate Similarity Search in Metric Spaces using Inverted Files *

Giuseppe Amato
ISTI-CNR
Via G. Moruzzi, 1
56124, Pisa, Italy
giuseppe.amato@isti.cnr.it

Pasquale Savino
ISTI-CNR
Via G. Moruzzi, 1
56124, Pisa, Italy
pasquale.savino@isti.cnr.it

ABSTRACT

We propose a new approach to perform approximate similarity search in metric spaces. The idea at the basis of this technique is that when two objects are very close one to each other they 'see' the world around them in the same way. Accordingly, we can use a measure of dissimilarity between the view of the world, from the perspective of the two objects, in place of the distance function of the underlying metric space. To exploit this idea we represent each object of a dataset by the ordering of a number of reference objects of the metric space according to their distance from the object itself. In order to compare two objects of the dataset we compare the two corresponding orderings of the reference objects. We show that efficient and effective approximate similarity searching can be obtained by using inverted files, relying on this idea. We show that the proposed approach performs better than other approaches in literature.

Categories and Subject Descriptors: H.3 [Information Storage and Retrieval]: H.3.3 Information Search and Retrieval;
Keywords: Approximate Similarity Search, Access Methods

1. INTRODUCTION

Traditional database systems are able to efficiently deal with structured records by using the *exact match* paradigm. However, in some cases data cannot be represented effectively as structured records. Consider for instance, text data, multimedia data, or even biological data. In these cases, the *similarity search* paradigm [11] has to be used instead of exact match. Similarity searching consists in retrieving data that are similar, with respect to an application dependent similarity function, to a given query.

Techniques for improving performance of exact match search cannot be used for similarity search, and the efficiency problem of the similarity searching paradigm has been discussed since long time [18, 3, 20]. Techniques to efficiently deal with similarity search in very specific applications, as for instance full-text search [15, 19], were developed. However, efficient approaches that allow application to generic similarity search problems still need to be investigated. A promising direction to address this issue is the *approximate similarity search* [21, 4, 1, 9] paradigm. Approximate similarity search provides an improvement in similarity search performance at the price of some imprecision in the results.

We propose a technique for approximate similarity search when data are represented in generic metric spaces. The metric space approach to similarity search requires the similarity between objects of a database to be measured by means of a distance (dissimilarity) function, which satisfies the metric postulates: positivity, symmetry, identity, and triangle inequality. A comprehensive dissertation on similarity search in metric space can be found in [20].

The basic idea of our proposal is that when two objects o_1 and o_2 are very similar (which in metric spaces means that they are close one to each other), their view of the surrounding world (their perspective) is similar as well. This implies that, if we take a set of objects from the database and we order them according to their similarity to o_1 and o_2, the obtained orderings are also similar. The basic idea here is that we can approximatively judge the similarity between any two arbitrary objects o_1 and o_2, by comparing the ordering, according to their similarity to o_1 and o_2, of a group of reference objects, instead of using the actual distance function between the two objects.

Clearly, it is possible to find some special examples where very similar (or even identical) orderings correspond to very dissimilar objects. For instance, if reference points are all positioned on a line, two objects that are positioned on another line orthogonal to the first one will produce the same ordering of the reference points, independently of their actual position. However, we will see that in real cases, even with a random selection of the reference points, the accuracy of this approach is very good.

We will also see that this technique can be very efficiently implemented relying on the use of the inverted files, exploiting several years of investigation on scalable search techniques on this data structure. In fact, our experiments prove that this technique offers performance, in terms of efficiency and accuracy, much higher than other methods existing in literature.

The structure of the paper is as follows. Section 3 formalizes the idea of searching by using the perspective of the objects. Section 4 shows how this idea can be efficiently supported by the use of inverted files. Sections 5, 6, and 7 discuss some relevant optimizations to the basic idea and assess the performance of the approach trough extensive experimentation. Section 8 compares the

*This work was partially supported by the DELOS NoE and the Multimatch project, funded by the European Commission under FP6 (Sixth Framework Programme).

INFOSCALE 2008, June 4-6, Vico Equense, Italy

DOI 10.4108/ICST.INFOSCALE2008.3486

proposed approach to approaches from the literature. Section 9 concludes and discusses possible future research directions.

2. RELATED WORKS

Techniques for approximate similarity search can be broadly classified in techniques that exploit space transformations and techniques that reduce the amount of data to be accessed. Our approach can be considered an hybrid approach given that, as we will see, even if it it mainly based on a space transformation it also adopts techniques to reduces the amount of data accessed. Many approaches need objects to be represented as vectors and cannot be applied to generic metric spaces. A more complete survey can be found in [20].

Among space transformation techniques we mention dimensionality reduction techniques as, for instance, those proposed in [7, 12], where the authors propose techniques to approximatively and efficiently compute the inner product between two vectors by exploiting an ad-hoc dimensionality reduction. Space transformation is also used in approximate search techniques based on VA-Files [17] where dimensionality reduction is obtained by quantizing the original data objects. Other techniques that fall in the category of space transformation are FastMap [8], which can be mainly used in vector spaces, and MetricMap [16] suited to metric spaces.

Techniques that reduce the space to be examined basically aim at improving performance by accessing and analyzing less data that is technically needed to have a mathematically precise result. These strategies try to infer which are the most promising portions of the space to be examined or to decide when it might be useless to continue searching for qualifying objects. Many of these techniques were defined exploiting data structures that use an hierarchical decomposition of the space, as for instance the M-Trees [5]. In [13, 14] a technique that analyzes the angle formed between objects in a ball region, the center of this region, and a query object, to decide when a region has to be accessed is proposed. This technique can be applied on any data structure based on hierarchical decomposition of the space by means of ball regions when data are represented in a vector space. A technique employing a user-defined parameter as an upper bound on approximation error is presented in [21]. The error parameter is used as an upper bound to the error introduced if a region of the space is not accessed when searching. A technique that retrieves k approximate nearest neighbors of a query object by returning k objects that statistically belong to the set of l ($l \geq k$) actual nearest neighbors of the query object is also presented in [21]. The value l is specified by the user as a fraction of the whole dataset. A technique called *Probably Approximately Correct* (PAC) nearest neighbor search in metric spaces is proposed in [4]. The approach searches the approximate nearest neighbor to a query object guaranteeing that the introduced error is within a user-specified confidence interval. A technique that uses a proximity measure to decide which tree nodes can be pruned, even if their bounding regions overlap the query region, is proposed in [1]. When the proximity of a node's bounding region and the query region is small, the probability that qualifying objects will be found in their intersection is also small. A user-specified parameter is employed as a threshold to decide whether a node should be accessed or not. If the proximity value is below the specified threshold, the node is not promising from a search point of view, and thus not accessed.

Comparison of our method is made against the proximity based method, presented above, which from other experiments seems to be one of the best performing techniques.

3. PERSPECTIVE BASED SPACE TRANSFORMATION

Let $\mathcal{D}$ be a domain of objects and $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$ be a metric distance function between objects of $\mathcal{D}$. Let $RO \subset \mathcal{D}$, be a set of reference objects chosen from $\mathcal{D}$.

Given an object $o \in \mathcal{D}$, we represent it as the ordering of the reference objects RO according to their distance d from o. More formally, an object $o \in \mathcal{D}$ is represented with $\bar{o} = O^{RO}_{d,o}$, where $O^{RO}_{d,o}$ is the ordered list containing all objects of RO, ordered according to their distance d from o.

We denote the position in $O^{RO}_{d,o}$ of a reference object $ro_i \in RO$ as $O^{RO}_{d,o}(ro_i)$. For example, if $O^{RO}_{d,o}(ro_i) = 3, ro_i$ is the 3rd nearest object to o among those in RO. We call $\bar{\mathcal{D}}$ the domain of the transformed objects. $\forall o \in \mathcal{D}, \bar{o} \in \bar{\mathcal{D}}$.

Figure 1 exemplifies the transformation process. Figure 1a) sketches a number of reference objects (black points), data objects (white points), and a query object (gray point). Figure 1b) shows the encoding of the data objects in the transformed space. We will use this as a running example throughout the reminder of the paper.

As we anticipated before, we assume that if two objects are very close one to each other, they have a similar view of the space. This means that also the orderings of the reference objects according to their distance from the two objects should be similar. In order to measure the similarity between two orderings we use the *Spearman Footrule Distance* (SFD) [6], which is a popular measure to compare ordered lists. Given two ordered lists S_1 and S_2, containing all objects of RO, the SFD between S_1 and S_2 is computed as the sum of the absolute differences between positions of objects in the two orderings. More formally

$$SFD(S_1, S_2) = \sum_{ro \in RO} |S_1(ro) - S_2(ro)|$$

We denote the distance between objects in the transformed domain as $\bar{d}(\bar{o_1}, \bar{o_2}) = SFD(O^{RO}_{d,o_1}, O^{RO}_{d,o_2})$.

The transformed domain $\bar{\mathcal{D}}$ and the distance $\bar{d}$ can be used to perform approximate similarity search in place of the domain $\mathcal{D}$ and the distance function d. Figure 1c) shows the distance, computed in the transformed space, of the data objects from the query object. It can easily be seen that it is consistent (it gives the same ordering) with the actual distance of data objects from the query.

In the following sections we will discuss how approximate similarity search can be executed very efficiently and precisely using this space transformation.

4. USING INVERTED FILES

Let us suppose that we have a dataset $X \subset \mathcal{D}$ and a query $q \in \mathcal{D}$. Suppose we want to search for the k objects of X nearest to q. An exhaustive approach is that of ordering the entire dataset X according to the distance from q and to select the first k objects. Let $\bar{X}$ be the dataset in the transformed space and $\bar{q} \in \bar{\mathcal{D}}$ the transformed query. The approximate ordering of X with respect to q can be obtained in $\bar{\mathcal{D}}$ by computing the distance $\bar{d}(\bar{q}, \bar{o}), \forall o \in X$. In the following we will show that this ordering can be obtained by representing (indexing) the transformed objects with inverted files and

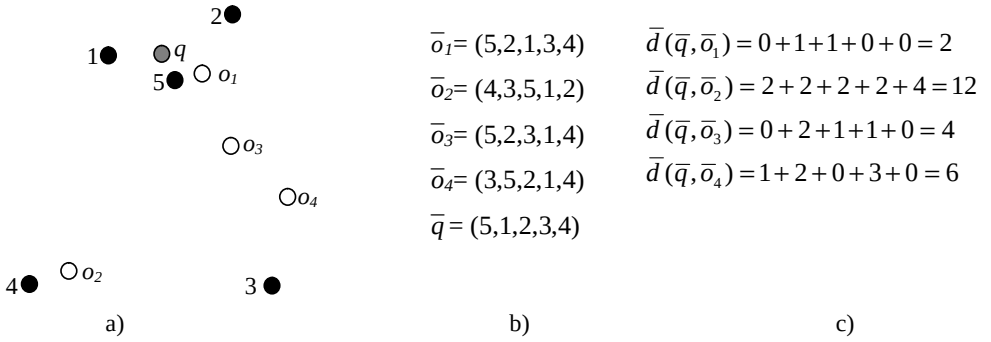

Figure 1: Example of perspective based space transformation. a) Black points are reference objects; white points are data objects; the gray point is a query. b) Encoding of the data objects in the transformed space. c) Distances in the transformed space

using search algorithms derived from the full text search area [15, 19]. Let us see this in details.

We can index transformed objects with inverted files as follows. Entries (the lexicon) of the inverted file are the objects of RO. The posting list associated with an entry $ro \in RO$ is a list of pairs $(o, O_o^{RO}(ro)), o \in X$, that is a list where each object o of the dataset X is associated with the position of the reference object ro in $\bar{o}$. In other words, each reference object is associated with a list of pairs each referring an object of the dataset and the position of the reference object in the transformed object's representation. For instance, an entry $(o, 7)$ in the posting list associated with reference object ro, indicates that ro is the 7th closest object to o among those in RO.

Therefore, the inverted file has the following overall structure:

$$ro_1 \rightarrow ((o_1, O_{o_1}^{RO}(ro_1)), \ldots, (o_n, O_{o_n}^{RO}(ro_1)))$$
$$\ldots$$
$$ro_m \rightarrow ((o_1, O_{o_1}^{RO}(ro_m)), \ldots, (o_n, O_{o_n}^{RO}(ro_m)))$$

where n is the size of the dataset X and m is the size of the set of reference objects RO.

Figure 2 shows the inverted file obtained for the running example in Figure 1.

A basic algorithm that quickly computes the distance $\bar{d}$ of all objects of the dataset X from q using an inverted file data structure is given in Figure 3. The Algorithm uses an accumulator a_o, associated with each object o found, to incrementally compute the distance $\bar{d}(\bar{o}, \bar{q})$. The set of accumulators A is initially empty (line 1.). The posting lists of the various entries are accessed (lines 2. and 3.) and for each entry in a posting list (line 4.) if the object is seen for the first time a new accumulator is added to the list of accumulators (lines 5.–7.). Then the value of the accumulator is updated by adding the difference in position of the current reference object ro, in $O_{d,q}^{RO}$ and $O_{d,o}^{RO}$ (line 8.). At the end of the algorithm execution all objects are associated with an accumulator that contains their distance $\bar{d}$ from the query object. It is easy to maintain the accumulators ordered during the algorithm execution so that at the end we have the entire dataset ordered. This algorithm is very similar to algorithms that incrementally compute the dot product in

$1 \rightarrow ((o_1,3),(o_2,4),(o_3,4),(o_4,4))$
$2 \rightarrow ((o_1,2),(o_2,5),(o_3,2),(o_4,3))$
$3 \rightarrow ((o_1,4),(o_2,2),(o_3,3),(o_4,1))$
$4 \rightarrow ((o_1,5),(o_2,1),(o_3,5),(o_4,5))$
$5 \rightarrow ((o_1,1),(o_2,3),(o_3,1),(o_4,2))$

Figure 2: Inverted file obtained for the running example of Figure 1

IN: query: q,
reference objects: RO,
posting lists associated with reference objects;
OUT: The set of accumulators: A
1. Set $A \leftarrow \{\}$.
2. For each $ro \in RO$
3. Let pl be the posting list associated with ro
4. For each $(o, O_{d,o}^{RO}(ro)) \in pl$
5. If $a_o \notin A$
6. Set $a_o = 0$
7. Set $A \leftarrow A \cup \{a_o\}$
8. Set $a_o = a_o + |O_{d,q}^{RO}(ro) - O_{d,o}^{RO}(ro)|$

Figure 3: Basic searching algorithm using inverted files.

text retrieval systems. The difference is that here we compute the Spearman Footrule Distance.

In the following we will discuss how we can estimate the number of reference objects to be used with a dataset. In addition we will also discuss the cost of this solution in terms of storage space and searching complexity. We will see that this preliminary solutions has some performance drawbacks. However, in the remaining sections will suggest some relevant modifications to this basic strategy that guarantee much higher efficiency still offering a high approximation accuracy. We will use the above solution as a baseline to evaluate the proposed refinements.

4.1 How to chose the size of RO

A basic question that should be answered is how to choose the number of reference objects to be used to map $\mathcal{D}$ into $\bar{\mathcal{D}}$.

Given a set of reference objects RO, the maximum number of different objects that can be represented in the transformed space is given by the factorial of the size of RO: $\#(RO)!$. This follows by the fact that objects in $\bar{\mathcal{D}}$ are represented by permutations of all elements of RO. However, we are not actually interested in the number of objects that we can represent in $\bar{\mathcal{D}}$. Rather, we are more interested in the capability of the distance $\bar{d}$ to order objects with respect to a query object $\bar{q}$. It is easy to see that the maximum distance measured by the SFD is $(\#RO)^2/2$. In addition, note that SFD always computes even values. Thus, the maximum number of different distances that can be computed is $(\#RO)^2/4$. This means that if the size of RO is too small, many objects will have the same distance from any query object. In order to be able to distinguish the distance of each object of $\bar{X}$ from any query object, the size of RO should be *at least*

$$\#RO \geq 2 \cdot \sqrt{\#X} \tag{1}$$

This is just a theoretical and quite rough lower bound. It clearly does not guarantee that all objects will have a different distance from any query object.

According to Equation 1, in order to index a dataset containing 50,000 objects we need about 448 reference objects. In order to index 1,000,000 objects we need about 2000 reference objects.

4.2 Storage and search cost

Let's estimate the cost of storing and searching the inverted file, using the proposed structure, with the lower bound given by Equation 1, and the search algorithm of Figure 3.

Each posting list contains a pair for every object of the dataset X. Therefore the size of each posting list is proportional to the size of the dataset X. The storage space needed to store the inverted file is consequently proportional to $\#RO \cdot \#X \approx 2 \cdot \sqrt{\#X} \cdot \#X = 2 \cdot (\#X)^{3/2}$, which is higher than linear even if smaller than quadratic with respect to the size of the dataset X.

Therefore, suppose we need 6 bytes to encode a pair in the posting list, the amount of space needed to store an inverted file for a dataset of size 50,000 is about 128MB. In case of a dataset containing 1,000,000 objects we need 11GB.

The algorithm given in Figure 3 has to read all posting lists in order to process a query. Therefore, suppose a disks with average transfer rate of 50MB/s, we need about 2.5 seconds for the database with 50,000 objects and about 3.5 minutes for the dataset containing 1,000,000 objects. This is clearly outperformed by most traditional access methods for similarity search [10, 5] even without approximation, provided that objects can be represented by vectors or can be represented in metric spaces.

In the reminder of the article we will modify this basic idea to gain orders of magnitude in performance. We will proceed according to these guidelines: 1) we will reduce the search cost by reducing the number of posting lists accessed during search (the approach proposed above accesses all posting lists); 2) we will reduce the size of the inverted file by reducing the number of posting lists where each object is referred (at the moment each object is referred by every posting list); 3) we will further reduce the search cost by reducing the amount of entries read in every posting list (at the moment all entries of a posting list are read).

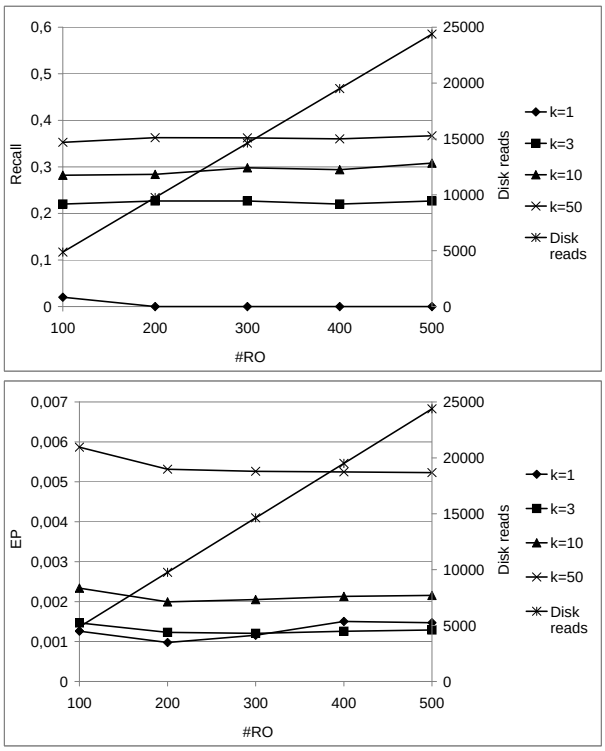

Figure 4: Performance varying the number of reference objects.

As a baseline, in order to assess the tradeoff between efficiency and approximation accuracy, we will use the basic approach presented above, whose performance is empirically evaluated in the following section.

4.3 Performance evaluation

In this section, we report results of an experimental evaluation of the above presented idea. As we said before we use this as a baseline to compare enhancements proposed in the next sections.

We conducted our experiments using a dataset of color histograms represented in 32 dimensions. This data set was obtained from the UCI Knowledge Discovery in Databases Archive [2]. The color histograms were extracted from the Corel image collection as follows. The HSV space is divided into 32 subspaces (colors), using 8 ranges of hue and 4 ranges of saturation. The value for each dimension of a vector is the density of each color in the entire image. The distance function used to compare two feature vectors is the histogram intersection implemented as L_1. The dataset contains 50,000 objects. 50 random objects were used as queries and were not inserted in the index.

As suggested in [20], we used the measure of the recall and position error to assess the accuracy of the method. Specifically, given a query object q, recall is defined as

$$R = \frac{\#(S \cap S^A)}{\#S} \tag{2}$$

and the position error is defined as

$$EP = \frac{\sum_{o^A \in S^A} |OX(O^A) - S^A(O^A)|}{\#S^A \cdot \#X}, \quad (3)$$

where S and S^A are the ordering of the k closest objects to q found respectively by an exact similarity search algorithm and by the proposed method. OX is the ordering of the entire dataset X with respect to the distance from q.

In the experiments we set the disk block size to 4 KBytes and each entry of the posting lists is encoded with 4 bytes[1].

Results are shown in Figure 4. The graphs show the recall and the position error varying the number of reference objects RO for various options of the k closest objects to the query considered. The graphs also show the number of disk block reads varying the number of reference objects. In the experiment we varied the size of RO from 100 up to 500 objects. Note that according to the observations made in Section 4.1 the size of RO should be about 448 objects. We measured performance using k equal to 1, 3, 10, and 50. Reference objects RO were chosen randomly from the dataset.

As expected, the search cost increases linearly with the size of RO. Note also that, given that the algorithm orders the entire dataset, the number of disk reads is clearly independent from the number of considered objects k. Surprisingly, we note that the accuracy of the method is not heavily affected by size of RO. In fact we note that accuracy slightly improves (recall increases and error decreases) when increasing size of RO from 100 to 200. However, it remains basically constant for the remaining values. This suggests that the size obtained according to guidelines in Section 4.1 is in fact overestimated. This effect is exploited, as we will see, in enhancements of this technique presented in next sections.

Looking more specifically at the results we can observe that recall improves with larger values of k. This is intuitively explained by the fact that when the result set is large, the probability to have elements from the exact result is higher. For instance, when we retrieve 50 objects (k is 50), on average we obtained a recall of 0.37. On the other hand, the position error is smaller for small values of k. This implies that generally, even if the true nearest neighbors are missed, the real ranks of the retrieved objects are not far from to the nearest neighbors. For instance, when k is 1, we obtained a position error of about 0.001. This means that the 50-th nearest neighbor was found on average instead of the first nearest neighbor, on a database of 50.000 objects. With large values of k, the probability that result objects with high rank are missed and replaced by objects with much higher real rank is higher, so errors are also larger.

5. SEARCHING WITH THE K_S CLOSEST REFERENCE POINTS

In previous section we have seen that no additional performance improvement is obtained when the size of RO increases above certain values. Encouraged by the observation that an higher number of reference points does not necessarily imply higher effectiveness, we can modify the search algorithm in such a way that not all reference points are used.

[1]This can be obtained by maintaining the entries of the posting list ordered according to the position and by using a small index for each posting lists that gives the offset corresponding to the possible positions. This is further discussed in Section 7.

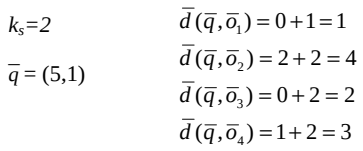

Figure 5: Encoding the query with the k_s closest reference objects and computing the distance in the transformed space using this reduced encoding.

More specifically, when we want to process a query object q, instead of representing it as the ordering $O^{RO}_{d,q}$ of all the objects in RO, we rather represent it as the ordering of the k_s ($k_s \leq \#RO$) reference objects of RO closest to q. We denote this ordering as $\bar{q} = O^{RO}_{k_s,d,q}$. The distance $\bar{d}$ is now computed by using a variant of the Spearman Footrule Distance, the *Induced Footrule Distance* (IFD), defined as:

$$IFD(S_1, S_2) = \sum_{ro \in S_1} |S_1(ro) - S_2(ro)|$$

The IFD considers just the elements occurring in the first ordered list and discards the others. Differently than SFD, it produces both even and odd values so the maximum distance also corresponds to the maximum number of different distances.

Figure 5 shows the distance, computed in the transformed space according to the IFD, of data objects from the query objects when the query is represented just by using the k_s closest reference objects.

In order to support this optimization, the search algorithm given in Figure 3 needs only to be modified in line 2. In fact, it should not access the posting lists associated with all reference objects. Rather it only accesses the posting lists associated with the k_s closest reference objects to q. Line 2. is therefore changed to "For each $ro \in O^{RO}_{k_s,d,q}$".

As we will empirically prove in the following, using the k_s reference objects for representing queries has both the effect of reducing the search costs, given that just k_s posting lists will be accessed, and increasing the effectiveness, given that the reference objects closest to q are the ones that more effectively represent the neighboring of q.

The search cost in this case is proportional to $k_s \cdot \#X$ that, when $k_s \ll \#RO$, is significantly smaller than before. Next section discusses the effectiveness of this approach for various values of k_s.

5.1 Performance evaluation

Here we discuss the results obtained by applying the above idea. The settings of the experiments are exactly as before. The only difference is that in this case we maintain fixed the size of RO and we evaluate the performance by varying the number k_s of objects from RO used to represent the query q.

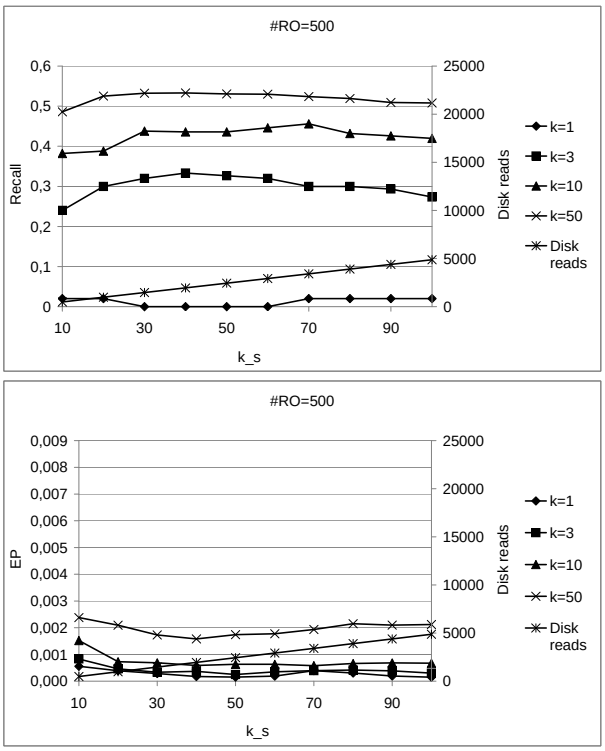

Figure 6: Performance searching with a subset of reference objects. We maintain fixed the total number of reference objects.

Results are reported in Figure 6. As before, the graphs show the recall, the position error and the number of disk block reads. We used 500 as size for RO and we varied the value of k_s from 10 to 100. Clearly, the cost increases linearly with k_s and (when k_s is smaller than $\#RO$) it is lower than the cost obtained before, given that less posting lists are accessed.

Looking at the accuracy measures, on the other hand, we observe a non intuitive behavior. Surprisingly, accuracy is always better than that obtained using all objects of RO to represent the query (compare Figure 6 with Figure 4). On average we obtain the best results when k_s is 50. In this case, when we retrieve 50 objects (k is 50), the recall is about 0.54 and the error is about 0.0019. In previous experiments, using all 500 objects of RO, to represent the query, we had a recall of about 0.38 and an error of about 0.0052. This, added to the fact that search cost is one order of magnitude lower (about 2500 disk reads compared to 25000 of the previous approach) demonstrates that the method is very promising: we have higher accuracy at much lower cost. This strange behavior can be explained by observing that the closest reference objects are the most reliable to represent an object to be indexed, while reference objects that are far from the indexed object introduce noise. In other words nearby objects have the same ordering of the nearest reference objects, while ordering of reference objects that are far from them might be completely different. This effect can be exploited to also encode the objects to be indexed with fewer reference objects, in addition to the query. This is discussed more in details in next section.

6. INDEXING WITH THE K_I CLOSEST REFERENCE POINTS

The idea of taking just the closest reference objects can also be used to represent any object that has to be indexed, rather than just the query.

Let $k_i \leq \#RO$ be the number of reference objects used for indexing. In this case every object can be represented as $\bar{o} = O^{RO}_{k_i,d,o}$, using a smaller amount of reference objects. Note that, in this case, different objects will be typically represented by different reference objects, given that different objects will have different neighbor reference objects.

This representation of an object will be clearly smaller than using all reference objects. In addition, this has also the effect of reducing the size of the inverted file. In fact every object will be just inserted into k_i posting lists, by reducing their size and by also reducing the search cost. The size of each posting list will depend on the number of objects that have the associated reference objects in the first k_i positions. However, given that every object will be inserted in k_i posting lists, the overall size of the inverted file will be proportional to $k_i * \cdot \#X$. Let $k_s \leq k_i$ be the number of reference objects used for searching. On average, we can estimate that the searching cost will be proportional to $k_s \cdot (k_i * \cdot \#X)/\#RO$.

Figure 7 shows the encoding of data objects in the transformed space, when just the k_i closest reference objects to an object is used (Figure 7a), the corresponding inverted file (Figure 7b), and the distances from the query computed in the transformed space according to this modification (Figure 7c). Note that the IFD computes the distance by summing the position differences of reference objects in the query and data object encoding. However, now it can happen that a reference object occurs in the query encoding and it does not occur in an object's encoding. When this happens we suppose that the position difference of the missing reference objects is greater than the maximum possible. That is, we suppose that the difference is $k_i + 1$.

Let us now discuss how the search algorithm has to be used with this modification to the indexing strategy. The basic algorithm that we gave previously in Figure 3, computes the distance of objects from the query incrementally using the accumulators. This techniques relies on the fact that every object appears exactly once in every posting list. Therefore every posting list contributes to compute the distance of every object. However, as discussed above, some reference objects might not occur in some object's encoding and consequently some objects might be missing from some posting lists. In fact, when an object is indexed with its k_i closest reference objects, it will appear just in k_i posting lists. This means that when processing a query, an object might be encountered in some posting lists and it might not be seen in others. This leads to miscalculation of the distance. In fact, when an object is not seen in a posting list, it is as if 0 is erroneously summed to its accumulator, in correspondence of the reference object associated with the posting list. However, note that 0 means that the position of the reference object in an object is exactly the same than its position in the query, while the actual position difference (so the distance) should be higher.

To solve this, we change the way in which the accumulators are used. Figure 8 shows the modified algorithm. When objects are seen for the first time the associated accumulator is initialized with the maximum possible distance for an object, which is $(k_i + 1) \cdot k_s$ (line 6.). In fact an object can be seen at most in k_s posting lists and we assume an unseen reference object having position $(k_i + 1)$, as

k_i=3

$\overline{o}_1$= (5,2,1)
$\overline{o}_2$= (4,3,5)
$\overline{o}_3$= (5,2,3)
$\overline{o}_4$= (3,5,2)

a)

$1 \rightarrow ((o_1,3))$
$2 \rightarrow ((o_1,2),(o_3,2),(o_4,3))$
$3 \rightarrow ((o_2,2),(o_3,3),(o_4,1))$
$4 \rightarrow ((o_2,1))$
$5 \rightarrow ((o_1,1),(o_2,3),(o_3,1),(o_4,2))$

b)

k_s=2

$\overline{q}$= (5,1)

$\overline{d}(\overline{q},\overline{o}_1) = 0+1 = 1$
$\overline{d}(\overline{q},\overline{o}_2) = 2+(k_i+1) = 6$
$\overline{d}(\overline{q},\overline{o}_3) = 0+(k_i+1) = 4$
$\overline{d}(\overline{q},\overline{o}_4) = 1+(k_i+1) = 5$

c)

Figure 7: Indexing with the k_i closest reference objects to data objects and searching with the k_s closest reference objects to the query. a) Encoding of data objects. b) Inverted file. c) Distance from the query in the transformed space.

IN: query: q,
reference objects: RO,
posting lists associated with reference objects,
number of reference objects used for indexing: k_i,
number of reference objects used for searching: k_s;
OUT: The set of accumulators: A
1. Set $A \leftarrow \{\}$.
2. For each $ro \in O^{RO}_{k_s,d,q}$
3. Let pl be the posting list associated with ro
4. For each $(o, O^{RO}_{d,o}(ro)) \in pl$
5. If $a_o \notin A$
6. Set $a_o = (k_i+1) * k_s$
7. Set $A \leftarrow A \cup \{a_o\}$
8. Set $a_o = a_o - (k_i+1) + |O^{RO}_{d,q}(ro) - O^{RO}_{d,o}(ro)|$

Figure 8: Searching algorithm that uses an inverted file where the closest k_i reference objects were used to index objects.

discussed before. Every time an object is encountered its current distance is reduced by replacing (subtracting) the maximum possible difference, which is (k_i+1), with the actual position difference from the query (line 8.).

6.1 Performance evaluation

Let us see what is the performance when just the k_i closest objects of RO are used to represent objects. The experiments that we performed are similar to those presented before. However in this case we fix both the size of RO and the number k_s of objects used to represent the query, and we vary the settings for k_i. Specifically, we choose 500 as size for RO, we set k_s to 50, and we vary k_i from 50 to 500.

Results are reported in Figure 9. Also in this case, we see that the search cost linearly increases with values of k_i. When all objects of RO are used for searching ($k_i = 500$), clearly, the cost is the same than that obtained in previous experiment in correspondence of k_s set to 50.

For what concerns the accuracy, we can see that after a small improvement obtained moving setting for k_i from 50 to 100, the performance remains substantially constant. Differences in performance, when settings of k_i range from 100 to 500, are unnoticeable. The clear advantage is that we obtain again the same accuracy with much lower cost. In fact, when k_i is set to 100 we have a search cost of about 600 disk blocks read, rather than 2500 (with k_s=50 and all reference objects used to index) or 25,000 (when all reference objects were used to index and search), as discussed in previous experiments. This confirms previous intuition that the closest reference objects are the most reliable to represent an object to be indexed.

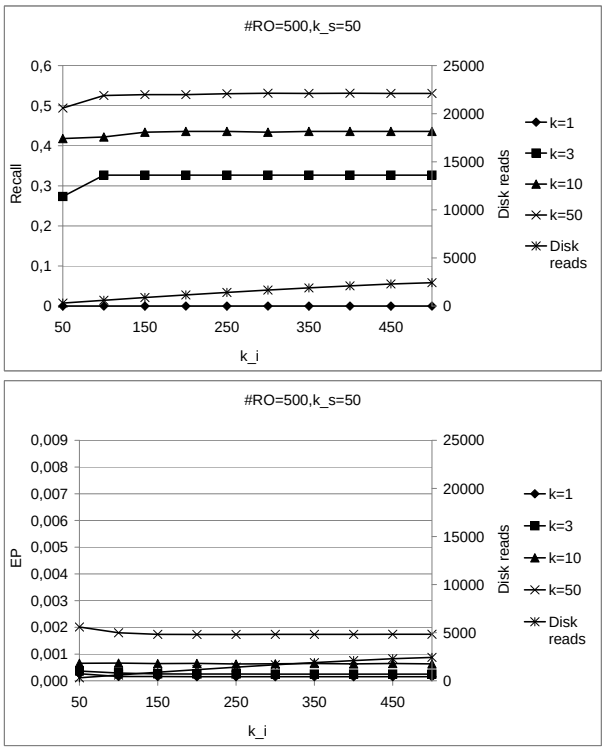

Figure 9: Performance indexing with a subset of reference objects. We maintain fixed the total number of reference objects and the subset of reference objects used for searching.

7. ACCESSING A FRACTION OF THE POSTING LISTS

Previous algorithms read the entire content of the accessed posting lists. However, also here we can find some opportunity for reducing the search cost. We can observe that in the posting lists there

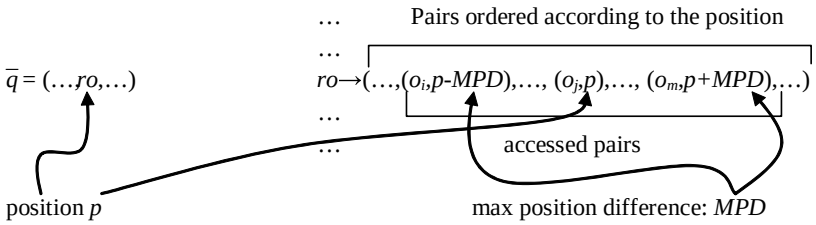

Figure 10: Accessing a portion of the posting lists: just the consecutive entries in the interval $[p - MPD, p + MPD]$ are accessed.

are some pairs that are more promising that others. Specifically, the most promising pairs are those whose position (the position of the reference object) is closer to that of the query. In fact, the IFD is computed by summing the position difference of a reference object in the query and in the searched objects. Given that we are interested in the k closest objects to the query, it is likely that objects whose position difference is high will fall outside the first k. Therefore, we can use a threshold parameter MPD which indicates the Maximum Position Difference and we can access just the pairs whose position difference is below the threshold.

In order to do that efficiently we can maintain the pairs of the posting lists ordered according to their position and, rather than accessing the entire posting lists, we can sequentially scan just the portion that contain pairs whose position difference is below the threshold. For instance, suppose that reference object ro has position p in the query q, that is $O^{RO}_{d,q}(ro) = p$. Then, the search algorithm will just access the consecutive pairs of the posting list, associated with ro, whose position is in the interval $[p - MPD, p + MPD]$. This process is depicted in Figure 10.

Note that the number of possible positions is much smaller than the number of pairs in a posting list. Therefore, sorting of the entries of the posting lists, according to the position, can be efficiently performed in linear time using a count-sort algorithm, after bulk insertion.

The revised ranking algorithm, which accesses a subset of the posting lists content, is given in Figure 11. It first computes the minimum (mp) and the maximum (Mp) position that read pairs should have (lines 3. and 4.). Then, it accesses just the pairs in that interval (line 5. 6.). Note that direct access to the initial position of the interval can be obtained by maintaining an index (an array) for each posting list that indicates the offset corresponding to possible positions. The index for a posting list has size equal to the maximum possible position, which is k_i, and can be stored at the beginning of the posting list itself. Note that the use of an index for the positions in the posting list makes it possible to store only the reference to the object, rather than the pair containing the object and its position. In fact the position can be inferred from the index, knowing the absolute position of the object in the posting list.

7.1 Performance evaluation

In order to test the effect of accessing just a portion of the posting lists, we maintain fixed the size of RO, k_s, and k_i. We performed various tests varying the value of MPD. Specifically, the size of RO was set to 500, k_s to 50, and k_i to 100. Values for MPD ranged from 10 up to 100.

Results are reported in Figure 12. As intuition suggests, the search cost increases linearly with MPD, given that larger values of MPD

IN: query: q,
reference objects: RO,
posting lists associated with reference objects,
number of reference objects used for indexing: k_i,
number of reference objects used for searching: k_s,
maximum allowed position difference: MPD;
OUT: The set of accumulators: A

1. Set $A \leftarrow \{\}$.
2. For each $ro \in O^{RO}_{k_s,d,q}$
3. Let $mp = max(0, O^{RO}_{k_s,d,q}(ro) - MPD)$,
   $Mp = min(k_i, O^{RO}_{k_s,d,q}(ro) + MPD)$
4. Let pl be the posting list associated with ro
5. Let pl^{Mp}_{mp} the subset of pl containing pairs with positions between mp and Mp
6. For each $(o, O^{RO}_{d,o}(ro)) \in pl^{Mp}_{mP}$
7. If $a_o \notin A$
8. Set $a_o = k_i * k_s$
9. Set $A \leftarrow A \cup \{a_o\}$
10. Set $a_o = a_o - k_i + |O^{RO}_{d,q}(ro) - O^{RO}_{d,o}(ro)|$

Figure 11: Search algorithm that accesses a fraction of the posting lists.

imply more entries to be read in the posting lists. However, similarly to what we have observed before, reading many entries is unnecessary from the accuracy point of view. In fact, from the graph we can see that the performance increases when MPD ranges between 10 and 40, afterwards it practically remains constant. In a few words, we have again the same accuracy than before while accessing less information from the disk. Specifically, when we set MPD to 40, the search cost is about 390 disk block reads with an accuracy that is practically the same (in terms of recall and position error) than that obtained by reading the full posting lists, as can be seen comparing Figures 12 ,6, and 9.

8. COMPARISONS

In this section we assess the performance of the proposed approach in contrast with other methods of approximate similarity search in metric spaces, proposed in literature. Comparisons among some relevant methods [21, 1, 4] were already presented in other works [1, 20]. Thus, for brevity we just compare the proposed techniques with the method of approximate similarity search based on region proximity [1], which was previously recognized as one offering very high performance.

Approximate similarity search based on region proximity uses the M-Tree [5] access method, and relies on a strategy of pruning non promising subtrees, during the search phase, measuring the proba-

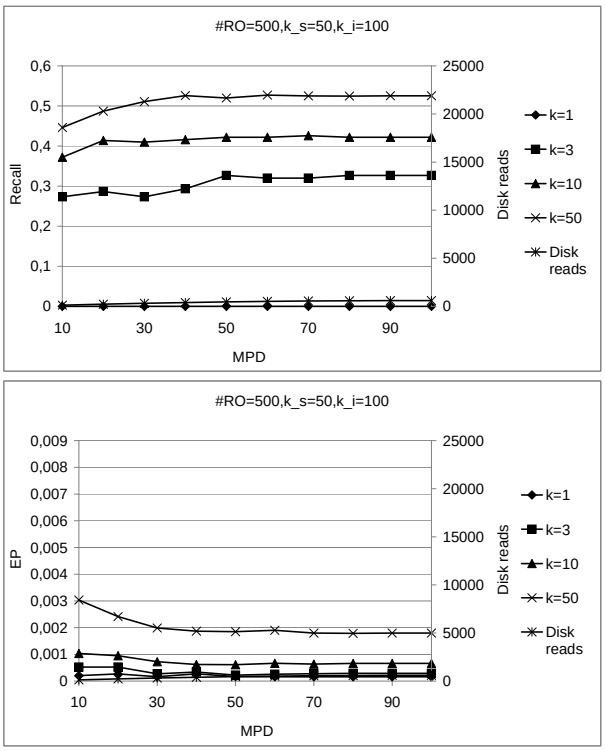

Figure 12: Performance accessing a portion of the posting lists We maintain fixed the total number of reference objects, the subset of reference objects used for searching, and the subset of reference objects used for indexing.

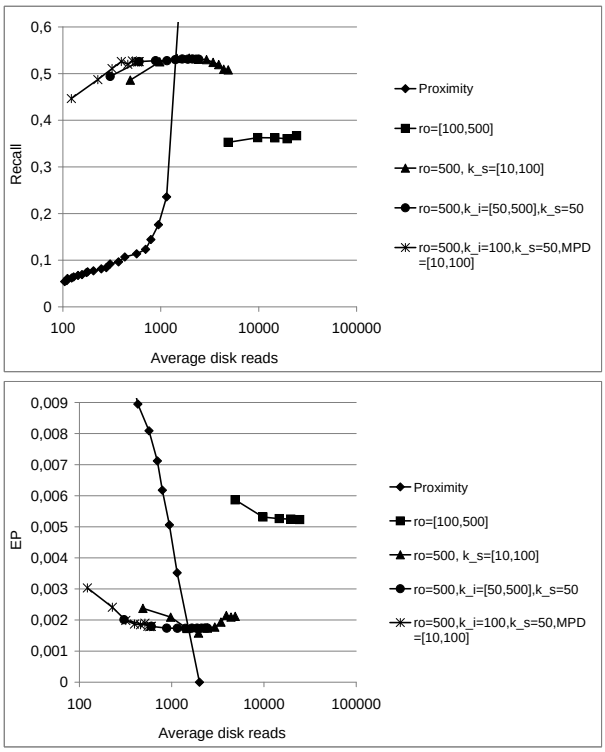

Figure 13: Comparisons of various setting of the proposed approach against a state-of-the-art method. We consider the case where the number k of retrieved objects is 50.

bility of overlap of the underlying ball regions with the query region. Tradeoff between accuracy and search cost can be tuned using a threshold on this probability of overlap. In order to have an objective comparison we used the same dataset described above [2] and the same queries. We also used the same size of the disk blocks in both cases (as we said before it was set to 4 KBytes).

Results are reported in Figure 13. The graphs show the performance of the four variants of our proposal and the proximity based method when the number k of searched objects is 50. For each method we plotted the search cost, in terms of disk block reads, against the recall and the precision error. A logarithmic scale is used for the search cost axis.

Leaving out of consideration our baseline method, where all reference objects are used for indexing and searching, the remaining variants typically offer much higher accuracy at the same search cost with respect to the method based on proximity. Specifically, when the search cost is between 100 and 1000 disk accesses, our methods offers a recall around 0.5 and a position error of about 0.02. On the other hand, the proximity based method offers a recall around 0,1-0,2 and a position error above 0.03. Note that the point with highest accuracy (recall 1 and error 0), offered by the proximity based method, was obtained with extreme settings of the approximation threshold corresponding to execute in fact exact similarity search.

We should also point out that the number of disk reads is an objective measurement of cost that, on the other hand, does not take into consideration the effect of data placement in the disk and disk access sequences. In fact, the proximity based method relies on a tree based data structure where search algorithm cannot sequentially access blocks in the disk. Rather, disk blocks are mostly accessed in a random fashion sequence. In contrast, a major advantage of inverted files is the possibility to perform searches by maximizing sequential disk accesses, whose cost is much lower than random accesses. Thus, due to this effect, the real performance advantage, of the proposed method over the proximity method, is in practice even higher than that shown in Figure 13.

From the graphs it is also evident how the best variant of our method is when just a portion of the posting lists are accessed.

9. CONCLUSIONS AND FUTURE WORK

In this paper we presented an approach to approximate similarity search in metric spaces based on a space transformation that relies on the idea of perspective from a data point. We proved through extensive experimentation that the proposed approach has clear advantages over other methods existing in literature. A major characteristics of the proposed technique is that it can be implemented by using inverted files, thus capitalizing on many years of investigation on efficient and scalable searching algorithms on this data structure. The proposed approach belongs to the category of general purpose access methods for similarity search. It can be applied to any application where the similarity search paradigm can be modelled using metric spaces.

There are still some issues that are worth of investigations to further improve this technique.

We did not discuss and investigate how to optimally chose the reference points. In the experiments we chose reference objects randomly from the dataset. However, performance can be higher if reference objects are selected carefully. In fact, in the introduction we gave an example of a bad choice of the reference points that returned misleading estimation of the similarity between objects. We believe that it is also possible to find strategies for choosing reference objects that offer an optimal accuracy of the similarity estimation.

Our method relies on the settings of some parameters. In order to obtain good performance, the size of RO, and the settings for k_s, k_i, and MPD have to be chosen. In this paper we chose these parameters empirically. However, they depends on specific characteristics of the dataset like its intrinsic dimensionality, the distance distributions, the data distribution, in addition to its size. A serious investigation on this will offer the possibility to optimally set the parameters beforehand by knowing the statistics of the database to be managed.

10. REFERENCES

[1] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, 21(2):192–227, 2003.

[2] S. D. Bay. The uci kdd archive. Irvine, CA: University of California, Department of Information and Computer Science. http://kdd.ics.uci.edu.

[3] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

[4] P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *ICDE*, pages 244–255, 2000.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 426–435. Morgan Kaufmann, 1997.

[6] P. Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *IMS Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, Hawyard Ca, 1988.

[7] Ö. Egecioglu and H. Ferhatosmanoglu. Dimensionality reduction and similarity computation by inner product approximations. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM 2000), McLean, Virginia, USA, November 6-11, 2000*, pages 219–226. ACM Press, 2000.

[8] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 18th ACM International Conference on Management of Data (SIGMOD 1995), San Jose, California, USA, May 22-25, 1995*, pages 163–174. ACM Press, 1995.

[9] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 503–511. IEEE Computer Society, 2001.

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

[11] H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California*, pages 36–45. ACM Press, 1995.

[12] Ü. Y. Ogras and H. Ferhatosmanoglu. Dimensionality reduction using magnitude and shape approximations. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM 2003), New Orleans, Louisiana, USA, November 3-8, 2003*, pages 99–107. ACM Press, 2003.

[13] S. Pramanik, S. Alexander, and J. Li. An efficient searching algorithm for approximate nearest neighbor queries in high dimensions. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS 1999), Florence, Italy, June 7-11, 1999*, volume 1. IEEE Computer Society, 1999.

[14] S. Pramanik, J. Li, J. Ruan, and S. K. Bhattacharjee. Efficient search scheme for very large image databases. In G. B. Beretta and R. Schettini, editors, *Proceedings of the International Society for Optical Engineering (SPIE) on Internet Imaging, San Jose, California, USA, January 26, 2000*, volume 3964, pages 79–90. The International Society for Optical Engineering, December 1999.

[15] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1983.

[16] X. Wang, J. T.-L. Wang, K.-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. An index structure for data mining and clustering. In *Knowledge and Information Systems*, volume 2, pages 161–184. Springer, 2000.

[17] R. Weber and K. Böhm. Trading quality for time with nearest neighbor search. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000), Konstanz, Germany, March 27-31, 2000*, volume 1777 of *Lecture Notes in Computer Science*. Springer, 2000.

[18] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998.

[19] I. H. Witten, A. Moffat, and T. C. Bell. *Bell: Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[20] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.

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