Solving approximate similarity queries

Tran Khanh Dang

School of Computing Science, Middlesex University, The Burroughs, Hendon, London NW4 4BT, UK. Email: k.dang@mdx.ac.uk

Supporting similarity search capabilities in data repositories helps satisfy user information needs rather than only user data needs like conventional DBMSs. This is desired for many modern database applications. However, as the data repository contains high-dimensional data, solutions to similarity search problem become cost-inefficient due to the so-called dimensionality curse. This phenomenon has been observed and shown that in high-dimensional data spaces the probability of overlaps between a query and data regions in a multidimensional access method (MAM) is very high. Hence, the execution of a similarity query may require accessing a vast number of the data regions and the performance of MAMs significantly decreases. Approximate similarity search has been introduced in order to lighten complexities of the problem. However, most research work done so far focuses mainly on approximate nearest neighbor (NN) and range queries in a single-feature data space. In practice, multiple-condition queries appear more frequently and get more complicated to deal with in whatever sense. In this article, we present efficient approaches to three types of approximate similarity queries: approximate multi-feature NN, approximate single-feature NN, and approximate range queries. Specially, we will use the Vague Query System, one among flexible query answering systems for conventional DBMSs, as a case study to illustrate and establish the practical value of our proposed solutions. Experimental results with both synthetic and real data sets will confirm the efficiency of these solutions.

Keywords: approximate nearest neighbor query, approximate range query, multidimensional access method, SH-tree, Vague Query System (VQS), intelligent information retrieval

1. A PRELIMINARY DECLARATION OF PROBLEMS

As we know, both nearest neighbor and range searching problems are among the most important and fundamental problems in computational geometry because of its numerous important application areas [1, 2]. Specially, in many modern database applications, high-dimensional searching problems arise when complex objects are represented by vectors of d numeric features. As the dimension d increases high enough, it is difficult to design an algorithm for the searching problem that achieves both nearly linear space and poly-logarithmic query time in the worst case. The reason is that processing these problems exactly may result in very cost-ineffective data accesses (IO-cost) and/or computations (CPU-cost), especially with high-dimensional data sets. In other words, this means that as the dimension d increases, the space and/or time complexities also increase dramatically. For this reason, in many cases, one is willing to allow approximately solving the high-dimensional searching problem in order to alleviate the costs while can still keep the acceptably high accuracy of the returned results. Formal definitions of the approximate problems with respect to the nearest neighbor searching problem and bounding-sphere\(^1\) range searching problem in d-dimensional spaces are given below:

\(^1\)Note that there are many other possible bounding ranges such as hyper-rectangles, half-spaces, etc.
**Definition 1** (approximate nearest neighbor searching problem): Given a set \( S \) of \( n \) data objects in a \( d \)-dimensional space, \( \mathbb{R}^d \). A data object \( P \in S \) is called a \((1+\varepsilon)\)-approximate nearest neighbor of a given query object \( Q \) with any real \( \varepsilon > 0 \) if for all other data objects \( P' \in S \):\
\[ \text{dist}(P, Q) \leq (1+\varepsilon) \cdot \text{dist}(P', Q) \]
where \( \text{dist}(X, Y) \) represents the distance between objects \( X \) and \( Y \).

**Definition 2** (approximate bounding-sphere range searching problem): Given a set \( S \) of \( n \) data objects in a \( d \)-dimensional space, \( \mathbb{R}^d \). For any real \( \varepsilon > 0 \), a given object \( Q \) and a real value \( r \geq 0 \), a data object \( P \in S \) is considered belonging to the result set of an approximate range query \((Q, r, \varepsilon)(center Q, radius r, and a tolerant error \varepsilon)\) if the following inequality holds:
\[ \text{dist}(P, Q) \leq (1+\varepsilon) \cdot r \]
where \( \text{dist}(X, Y) \) represents the distance between objects \( X \) and \( Y \).

Although the problem of answering approximate nearest neighbor and range queries has also extensively been studied as published in a rich literature such as [1–4], it is mostly satisfied for only low- or medium-dimensional data sets. In high-dimensional spaces, the key obstacle is so-called the curse of dimensionality (cf. section 3.1). In this paper, we shall introduce adaptations to the SH-tree [5] for answering such approximate queries in high-dimensional feature spaces.

On the other hand, the searching problem that we mentioned above is confined only to one single data set. A more frequently appeared query type in practice is multi-feature nearest neighbor queries, which are sometimes referred to as complex vague queries [6]. Answering such queries requires the system to search on some feature spaces individually and then combine the searching results to find the final answers returned to users. The feature spaces are commonly multidi- mensional spaces and may consist of a vast amount of data. Therefore searching costs, including IO-cost and CPU-cost, are prohibitively expensive for complex vague queries. In the case of a single-feature space, to alleviate the costs, the problem of answering nearest neighbor and approximate nearest neighbor queries has been proposed and quite well-added in the literature. In this paper, we shall introduce an approach, called e-ISA, for finding \((1+\varepsilon)\)-approximate nearest neighbor(s) of complex vague queries, which must deal with the problem over multiple feature spaces. This approach is based on a novel, efficient and general algorithm called ISA-Incremental hyper-Sphere Approach [6], which has been introduced for solving the nearest neighbor problem in the Vague Query System (VQS) [7]. To the best of our knowledge, the e-ISA is one of a few vanguard solutions for dealing with the problem of answering approximate multi-feature nearest neighbor queries. Our experimental results with both uniformly distributed and real data sets will prove the efficiency of the proposed approach.

Furthermore, besides the more frequent occurrence, complex vague queries are also more challenging and practical in modern information/data searching systems like semantic-based similarity search systems or flexible query answering systems (cf. chapter 3 of [5]). Hence, although in this paper we introduce solutions to dealing with all three typical kinds of approximate queries, including approximate single-feature nearest neighbor queries, approximate bounding-sphere range queries, and approximate complex vague queries, we shall mainly focus on addressing approximate complex vague queries (i.e., approximate multi-feature nearest neighbor queries). The two other types of approximate similarity queries are very promising effects resulting from the SH-tree (see section 2).

The rest of this paper is organized as follows: Section 2 briefly gives an introduction to the SH-tree, which will be used to manage multidimensional data sets for our later experiments. Section 3 is dedicated to intensively introducing the e-ISA, a new approach to approximate complex vague queries. Sections 4 and 5 approach approximate nearest neighbor queries and approximate bounding-sphere range queries, respectively, for high-dimensional data sets. Finally, section 6 gives conclusions and presents future work.

### 2 THE SH-TREE: A FLEXIBLE SUPER HYBRID INDEX STRUCTURE FOR MULTIDIMENSIONAL DATA

The SH-tree has been proven one of the most flexible and efficient multidimensional access methods (MAMs). In this section, we concisely introduce the SH-tree structure. More detailed discussions about the tree can be found in [5].

Because the SH-tree is planned to apply not only for point data objects, but also for extended data objects so it has a no overlap-free space partitioning strategy for directory nodes. The idea of this approach is to easily control objects that cross a selected split position and to solve the storage utilization problem. There are three node kinds in the SH-tree: internal, balanced and leaf nodes. Each internal node \( i \) has the structure like \( \langle d, lo, up, other\_info\rangle \), where \( d \) is the split dimension, \( lo \) represents the left (lower) boundary of the right (higher) partition, \( up \) represents the right (higher) boundary of the left (lower) partition, and \( other\_info \) consists of additional information as node type information, pointers to its left, right child and its parent node as well as \( meta\_data \) like the data object number of its left, right child. While \( up\_lo \) means no overlap between partitions of its two child nodes, \( up\_lo \) indicates that the partitions are overlapping. This structure is similar to ones introduced in the SKD-tree [8] and the Hybrid tree [9]. The supplemental information like the meta-data as mentioned above also gives hints to develop a cost model for nearest neighbor queries in high-dimensional space, or to estimate the selectivity of range queries, etc. Moreover, let \( B_{Ri} \) denote bounding rectangle of internal node \( i \). The bounding rectangle of its left child is defined as \( B_{R}(\cap(d_{sup})) \). Note that \( \cap \) denotes geometric intersection. Similarly, the bounding rectangle of its right child is defined as \( B_{R}(\cap(d_{sub})) \). This allows us to apply algorithms used in the DP (data partitioning)/R-tree based index techniques to the SH-tree.

Balanced nodes are just above leaf nodes and they are not hierarchical (see Figure 1). Each of them has a similar structure to that of an internal node of the SR-tree [10]. This is a specific characteristic of the SH-trees. It conserves the data cluster, in part, and makes the height of the SH-tree smaller as well as employing the SR-tree’s superior aspects during the querying phase. Moreover, it also shows that the SH-
trees are not simple in binary shape as in typical KD-tree based index techniques. They are also multi-way trees as R-tree based index techniques:

BN: <B₁, B₂, ...Bₙ> (minBNₑ ≤ n ≤ maxBNₑ)
Bi: <BS, MBR, num, child_pointer>

In general, a balanced node consists of n entries B₁, B₂, ...Bₙ with minBNₑ ≤ n ≤ maxBNₑ, where minBNₑ and maxBNₑ are the minimum and maximum number of entries in the node, respectively. Each entry Bi keeps information of a leaf node including four components: a bounding sphere BS, a minimum bounding rectangle MBR, the object number of leaf node num and a pointer to it pointer.

Each leaf node of the SH-tree has the same structure as that of the SS-tree [11] (because the SR-tree is just designed for points but the SH-tree is planned for both points and extended objects):

LN: <L₁, L₂, ...Lₘ> (minOₑ ≤ m ≤ maxOₑ)
Lᵢ: <obj, info>

As we see above, each leaf node of the SH-tree consists of m entries L₁, L₂, ...Lₘ, with minOₑ ≤ m ≤ maxOₑ, where minOₑ and maxOₑ are minimum and maximum number of entries in a leaf, respectively. Each entry Lᵢ consists of a data object obj and information in the structure info as coordinate values of the data object’s feature vector, a radius that bounds the data object’s extent in the feature space, the data object’s MBR, etc. If data objects in the database are complex, obj is only an identifier instead of a real data object. In addition, in case the SH-tree is only applied to point data objects, each Lᵢ is similar to that of the SR-tree: Lᵢ: <obj, feature_info>. In this case, the other information of the objects is no longer needed. For example, the parameter radius is always equal to zero and MBR is the point itself. Figures 1a and 1b show a possible partition of an example feature space and its corresponding mapping to the SH-tree, individually. Assume we have a two-dimensional feature space D with the size of (0, 0, 10, 10). With the split information (d, lo, up) = (1, 6, 6), the BRs of left and right children of the internal node 1 are BR₂ = D ∩ (d 6) = (0, 0, 6, 10) and BR₃ = D ∩ (d 6) = (6, 0, 10, 10), individually. For the internal node 2, assume that (d, lo, up) = (2, 3, 4), we have BR₄ = BR₂ ∩ (d 4) = (0, 0, 6, 4), BR₅ = BR₂ ∩ (d 3) = (0, 3, 6, 10). Similarly, for the internal node 3, with (d, lo, up) = (2, 5, 6) we obtain BR₆ = BR₃ ∩ (d 6) = (6, 0, 10, 6), BR₇ = BR₃ ∩ (d 5) = (6, 5, 10, 10), and so on. Note that, the BRs coordinate information is not stored in the SH-tree explicitly, but it is dynamically computed when necessary.

Furthermore, the storage utilization constraints of the SH-tree must ensure that each balanced node is always filled with at least minBNₑ entries and each data page must contains at least minOₑ data objects. Therefore, each subspace according to a balanced node will hold DON data objects (DON denotes the data object number of each balanced node) and DON satisfies the following condition: minOₑ × minBNₑ ≤ DON ≤ maxOₑ × maxBNₑ. We will see that the SH-tree is also very efficient for approximate nearest neighbor (NN) and range queries in high-dimensional spaces.

2. Except for the root node where information about coordinates of the whole data space, in the running example it is D = (0, 0, 10, 10), is kept track of and incrementally updated during the tree creation process. Using this information and information about the splits in internal nodes we can calculate BRs of the subspaces during traversing the SH-tree.
3. SOLVING APPROXIMATE COMPLEX MULTI-FEATURE NEAREST NEIGHBOR QUERIES

As discussed in [6], addressing complex multi-feature nearest neighbor (M-FNN) queries can be accomplished by the Incremental hyper-Sphere Approach (ISA). Although the ISA is more efficient than a previous proposed approach, the Incremental hyper-Cube approach (ICA) [12], in all terms of IO- and CPU-cost, in this section we shall present important modifications to further improve the ISA so that it can also work well in high-dimensional feature spaces and when the query condition number increases.

3.1 Introduction and related research

Similarity search has emerged and become a fundamental paradigm for a variety of application areas as image/multimedia processing [13, 14], data mining [15], CAD database systems [16], time-series databases [17, 18], information retrieval (IR) systems [19], geographical information systems (GISs) and tourist information systems (TISs) [20], digital libraries [21], etc. In these contexts, an important problem is how to find the data object(s) that is most similar to a given query object. The standard approach is to perform search in feature vector repositories of the data objects and use specific computational functions, e.g., Euclidean metric, to measure the distance between feature vectors. Similarity search therefore becomes a nearest neighbor query over the feature vector spaces.

Notably, there are some proposals to extend and facilitate the conventional relational database management systems (RDBMSs) with similarity search capabilities as [7, 22–25] (see [5] for detailed discussions). Specially, in [7], Kueng and Palkoska proposed an approach to solving the similarity search problem in the conventional RDBMS context but it has aspects similar to approaches to the NN problem in feature vector spaces of modern database applications just mentioned above. More concretely, the VQS, which is introduced in [7], is an extension to the conventional RDBMSs and can operate on the top of them to retrieve tuples semantically close to user’s queries. The VQS key feature is the concept of Numeric-Coordinate-Representation-Tables (NCR-Tables) that store semantic meta-information of attributes. In fact, attributes of arbitrary types in a query relation/view are mapped to Euclidean spaces and kept by NCR-Tables. When the RDBMS fails to find an exact match for a given query Q, the VQS will search on some NCR-Tables corresponding to the query conditions of Q and return the best match for Q. Intuitively, NCR-Tables in the VQS are equivalent to the feature vector spaces mentioned above.

Solving NN queries has been researched for a long time and has attracted attention of lots of researches, e.g., [26–30]. The de facto standard solution is to employ MAMs [31, 32] to index feature values, which are often multidimensional vectors. Searching is later done with the support of MAMs to speed-up the performance and to decrease the costs, including IO-cost and CPU-cost. Nevertheless, when the number of dimensions of the feature spaces increases, MAMs typically suffer from the so-called dimensionality curse. This phenomenon has been observed and shown that in high-dimensional feature spaces, e.g., greater than 16, the probability of overlaps between a query and data regions is very high. The execution of a similarity query may therefore require accessing many of the data regions, and thus the performance of MAMs significantly decreases. Specially, in such cases, a linear scan over the whole data set would perform better than MAMs [33, 34].

In practice, the mapping of attributes of objects to coordinates of vectors is heuristic in many application domains [35]. Therefore, an approximate similarity search is in many cases as good as an exact search or a “true” similarity search. As presented in Definition 1, given a data set S and a query object Q, a data object P ∈ S is called a \((1+\varepsilon)\)-approximate nearest neighbor of Q with \(\varepsilon > 0\) if for all other data objects \(P' \in S\):

\[
dist(Q, P) \leq (1+\varepsilon) \cdot \dist(Q, P')
\]

There is a great number of previous researches that have concentrated on addressing the approximate NN queries over a single feature space such as [1, 3, 35–38]. Here we call such queries approximate single-feature NN (S-FNN) queries. In section 4 we shall elaborate more on this query type.

Although there are also numerous researches in M-FNN query answering as [25, 39, 40–43], we cannot apply any solution among them to the approximate M-FNN queries in the VQS (see [12] for more details about M-FNN queries in the VQS). Of late, we have introduced an approach named Incremental hyper-Sphere Approach (ISA) [6] that has been

![Figure 2](image-url) Random access is impossible in the VQS
shown to be efficient and general for supporting similarity search capabilities in semantic based similarity search systems. The ISA is an improvement and a generalization of an approach proposed for the VQS in [12]. Specially, the ISA can be applied to searching on multiple feature spaces that do not satisfy three conditions so that Fagin’s algorithm (the A₀ algorithm) [39] and all its improvements like [41, 42] become applicable. Three conditions of the feature spaces in order to employ the Fagin’s algorithm are as follows [44]:

1. There is at least a key for each feature space to be searched.
2. There is a mapping between the keys.
3. We must ensure that the mapping is one-to-one.

Intuitively, condition (1) is always satisfied in the VQS, however, condition (3) does not hold. In the VQS, each Fuzzy Field is also the key for the corresponding NCR-Table but there is no mapping one-to-one between Fuzzy Fields of the NCR-Tables. Figure 2 depicts an illustrative example to make this clearer.

In Figure 2 we have a query relation with two attributes Attr_1 and Attr_2 whose attribute values are being mapped to the NCR-Tables NCR_1 and NCR_2, respectively. Moreover, let’s assume that attribute Attr_1 (resp. Attr_2) is mapped to attribute Domain_1 (resp. Domain_2) in the table NCR_1 (resp., NCR_2) and it becomes the primary key of NCR_1 (resp. NCR_2). To make the NCR-Tables NCR_1 and NCR_2 satisfy the three conditions as mentioned above we must ensure that for each value of the key attribute Domain_1 in the table NCR_1 there is only one value of the key attribute Domain_2 in the table NCR_2 (and vice versa) and both of them belong to a certain tuple in the query relation. If this is satisfied then the random accesses to the NCR-

---

**Figure 3** Advantages of the ISA over the ICA

---

**Figure 4** The Incremental hyper-Sphere Approach (ISA) for answering Complex M-FNN Queries

---

3. See [5] for our definition and discussions about semantic based similarity search systems.
Tables are possible. Nevertheless, this does not hold in the context of our problem: Figure 2 shows an example situation in which value $x_1$ in NCR_1 can be combined with at least two other values $y_1, y_2$ in NCR_2 to form two different tuples ($x_1, y_1$) and ($x_1, y_2$) in the running query relation (another example showing in Figure 2 is value $y_1$ in table NCR_2 and values $x_1, x_2$ in table NCR_1 with respect to two tuples ($x_1, y_1$) and ($x_2, y_1$) in the query relation).

Recently, there are two algorithms proposed for dealing with the approximate NN queries over multiple feature spaces published in [45, 46]. We call these queries approximate M-FNN queries. Nevertheless, the algorithm for dealing with approximate M-FNN queries introduced in [46] has solved the problem in a different context from that the VQS is facing. In details, the authors presented an algorithm to find multimedia objects that approximately similar to a given object and then proved that this approximate algorithm is instance optimal. This algorithm, however, makes an assumption that random access [39] is possible. This assumption just holds as three other conditions as presented above are satisfied.

Moreover, in [45] the authors introduced an algorithm that is a generalization of the A0 algorithm [39] to deal with M-FNN queries called the J* algorithm which looks like the ISA very much. Then, they presented a modified version of the J* algorithm to address approximate M-FNN queries. However, here we must note that although the ISA looks very similar to the J* algorithm, they are intrinsically different:

1. The J* algorithm assumes the availability of ranked values on each subsystem involving the query conditions. However, the ISA does not assume this and it, instead, points out how to make this availability using an algorithm, named the incremental algorithm adapted for range queries, which is modified from one of the state-of-the-art algorithms for k-nearest neighbor problem [47] (see [5, 6] for the details of the adapted algorithm).

2. The J* algorithm firstly reduces the processed states by inheriting all properties of an A* algorithm [48]: the database access cost is alleviated by an iterative deepening technique. With the ISA, the database access cost and the processed states are reduced by taking into account the incremental hyper-sphere range queries over each index and by computing the maximum searching radii dynamically (see the optimal ISA version in chapter 6 of [5]).

Also, the ISA was inspired by the Incremental hyper-Cube Approach introduced by Kueng et al. [12], which had been introduced earlier and had the same overall goals as that of the J* algorithm. In this paper we will present an approach based on the ISA to efficiently and generally solving approximate M-FNN queries, which are also called approximate complex vague queries (CVQs).

The rest of this section is organized as follows: Section 3.2 concisely summarizes and discusses the ISA, which is the base of our new approach presented in the next section. Section 3.3 is dedicated to introducing e-ISA: an incremental lower bound approach for efficiently finding approximate nearest neighbors of CVQs. Section 3.4 presents experimental results that prove the efficiency of our proposed approach, the e-ISA.

### 3.2 The ISA: An efficient approach for solving complex vague queries

As we have already known, query processing in the conventional RDBMSSs is not flexible enough to support similarity search capabilities directly. That means when the available data in a relational database does not match a user’s query precisely, the system will only return an empty result set to the user. This limits their applicability to domains where only crisp answers are meaningful. In many other application domains, however, the users expect not only the crisp results returned but also some other results close to the query in a certain sense. Systems can solve this problem are called flexible query answering systems (FQASs) [5]. Among FQASs proposed, the VQS [7] is very notable because it can be easily built, extensible and can work on top of the existing

![Figure 5 Overview of problem solved by the ISA](image)

![Figure 6 Incremental algorithm adapted for range queries](image)
SOLVING APPROXIMATE SIMILARITY QUERIES

RDBMSs. Specially, the VQS employs NCR-Tables to store semantic background information of attributes in a query relation/view and the similarity search is then carried out with their support. This aspect is similar to approaches to the modern information retrieval applications [19]. In [12], to enhance the VQS, the authors presented an Incremental hyper-Cube Approach (ICA) for finding the best match for M-FNN queries. Unfortunately, the ICA is not general and has weaknesses lead to degenerate the search performance of the VQS. Recently, in [6] we introduced an Incremental hyper-Sphere Approach (ISA) to improve and generalize the ICA for solving M-FNN queries. One of the ISA key features is to use hyper-sphere range queries instead of hyper-cube ones as the ICA to reduce I/O- and CPU-cost during the search process. Figure 3 illustrates the scenario for this idea.

Assume that the searched feature space as illustrated in Figure 3 is involving a query condition qi of a running CVQ and the first appropriate tuple of the query relation is found after finishing hyper-cube range query hi0 (see [12] for details of the ICA). In the next step, to verify whether or not this tuple is the best match for the query, the ICA must extend the searching radius using formula 3 (see Figure 4) and create a new hyper-cube query hi1. Intuitively, disk pages intersected with and objects located in the gray area s = hi1 – ci1 are unnecessary to be accessed and verified but the ICA always does this while executing query hi1. For the ISA, only objects within hyper-sphere ci1 and disk pages intersected with this hyper-sphere are accessed and further examined.

Moreover, for each NCR-Table (i.e., feature space) involving the query, the ISA employs an incremental algorithm for range queries, which is modified from one of the state-of-the-art algorithms for k-NN queries originally presented in [27]. Briefly, the main purpose of this algorithm is to overcome the unnecessary disk access problem for hyper-sphere range queries in which the results of a previous range query during the search on each feature space can be reused for the extended range query after that. This modified algorithm is also proven to be optimal in terms of disk access number as the original one (see [5] for details). Figure 4 briefly summarizes high-level abstract steps of the ISA final version.

With reference to Figure 4, the input information of the ISA can be viewed under a different form as shown in Figure 5 in which finding the best match for user’s query needs to perform search on several NCR-Tables, which keep semantic background meta-information of attributes of the query relation/view and are being managed by MAMs.

After carrying out step 1 there are several returned NCR-Values for each query condition qi. As noted above, here the incremental algorithm adapted for range queries is used to avoid repetitive disk accesses when processing the range queries. This is illustrated in Figure 6: During the execution of range queries qi1, qi2, the algorithm does not have to access again disk pages accessed by range queries qi0, qi, respectively. This is one of the advantages of the ISA over the ICA: In Figure 3, for instance, to process range query hi1, the ICA must access again all disk pages accessed during the execution of range query hi0.

Step 2 of the algorithm tries to find an appropriate record/tuple, which belongs to the query relation and contains the returned NCR-Values in step 1 for each query condition (cf. Figure 5). Step 3 is intuitive: among appropriate records found in step 2, the ISA chooses one, i.e. Tmin, having minimum total distance TDmin, which is calculated by formula 2. Depending on TDcur the ISA computes the maximum search radius for each query condition qi using formula 3. The search is continued and to reduce the searched space for each qi, these maximum searched radii are dynamically updated whenever a new record Tmin is found and its total distance TDnew<TDcur, which is calculated by formula 2. Depending on TDcur the ISA computes the maximum search radius for each query condition qi using formula 3. The search is continued and to reduce the searched space for each qi, these maximum searched radii are dynamically updated whenever a new record Tmin is found and its total distance TDnew<TDcur, i.e. condition (b) in step 4 holds. In this case, Tmin replaces Tnew and the search is then kept on until condition (a) in step 4 satisfies. At this time the current Tmin is returned in step 5. Note that when condition (a) in step 4 does not hold for a certain qi then the search is still continued for that qi. This also means the ISA is not a uniform algorithm [41]. As proven in [5], after finishing step 5, there is no any other record in the entire query relation/view S having a smaller total distance to Q than that of the returned record Tmin. Therefore, the ISA correctly returns the best match for the query Q and totally overcomes all shortcomings of the ICA. Furthermore, we also showed in [5] that the ISA increasingly improves the ICA when the dimension d of data sets (NCR-Tables) involved in the query increases.

Although experimental results in [6] have shown that the ISA is significantly outperforms the ICA originally introduced in [12] by orders of magnitude in both terms of I/O- and CPU-cost, its performance is still decreased when the dimension number of the feature spaces or the query
condition number increases. For high dimensional feature spaces, the problem is caused by the dimensionality curse, by which probability of overlaps between a query and data regions is very high, hence query processing requires more I/O and CPU costs. For the second reason, i.e. the increase of the query condition number, Figure 7 shows our experimental results of 100 randomly selected three-condition queries in only two-dimensional spaces for both ICA and ISA (see section 3.4 for more information on common test conditions). We can observe that the affected object number in the ISA, which is directly proportional to CPU-cost, is not much less than that of the ICA and thus the ISA performance is decreased. To face these problems we propose a variant of the ISA called \( \varepsilon \)-ISA to approximately answer CVQs (or M-FNN queries). Section 3.3 will detail the \( \varepsilon \)-ISA.

### 3.3 The \( \varepsilon \)-ISA: towards efficiently solving approximate M-FNN problem

Inequality as shown in equation 4 looks like one as shown in equation 1. However, differences are inside them as a CVQ may consist of several S-FNN queries. Each query condition of a CVQ is separately addressed and the distance metrics can be different among each subsystem dealing with these query conditions. Nevertheless, the returned distances of objects corresponding to each query condition are then normalized into the interval \([0, 1]\), i.e. the factor \((d_i/D_i)\) in formula 2. The total distance \(TD\) of each record to the query \(Q\), which is calculated using formula 2, is the weighted sum of the normalized element distances just mentioned above. In accordance with the description of the ISA as shown in Figure 4, an appropriate record \(T_{\text{min}}\) having a minimum \(TD\) among the found records must be verified if it is the nearest neighbor of \(Q\). As with the approximate problem, however, if its total distance \(TD_{\text{cur}}\) is satisfied inequality as shown in equation 4, it will be returned immediately as a \((1+\varepsilon)\)-approximate nearest neighbor of \(Q\) without any more searches. Section 3.3.1 presents a variant of the ISA using lower bound total distance (LBTD) of the “true” nearest neighbor of \(Q\) to solve this problem.

#### 3.3.1 Finding Approximate Nearest Neighbor

Processing steps of the algorithm \( \varepsilon \)-ISA described in Figure 8 below are quite similar to those of the ISA described in Figure 4 except for some differences as follows. First, the input information includes a real \(\varepsilon>0\) that is served as a tolerant error to judge whether or not a found appropriate record is a \((1+\varepsilon)\)-approximate NN of the query. In practice, this value is usually supplied by the user. Second, the output of the \( \varepsilon \)-ISA will be an approximate NN instead of the “true” NN of the query. As the ISA, from step 1 to step 3 the \( \varepsilon \)-ISA tries to find an appropriate record in the query relation/view \(S\).
The record $T_{app}$ chosen in step 3 has the minimum TD to the query among the found appropriate records. At this point we still do not know if $T_{app}$ is a $(1+\varepsilon)$-approximate NN of $Q$ even though it is a candidate for the best match of the query. Then, however, in step 4 the $\varepsilon$-ISA computes a LBTD for the “true” NN of $Q$ as follows:

$$LBTD = \min \{TD_{cur}, d_i\}, i = 1, 2 \ldots n$$

where the value of $n$ represents query condition number of the complex vague query being processed, i.e., the query $Q$.

In formula 5, each $d_i$ is the weighted normalized distance from the query condition $q_i$ to the farthest object (i.e., NCR-Value in the VQS) returned so far in the corresponding feature space being managed by a multidimensional index structure $F_i$. Because the incremental algorithm adapted for range queries employs a priority queue [6] thus $d_i$ is also the weighted normalized distance from $q_i$ to the last returned object so far in the corresponding feature space. Particularly, each $d_i$ will be calculated as follows:

$$d_i = w_i \cdot t_i / (D_i \cdot w_{sum}), i = 1, n$$

where:
- $w_i$: weight of the corresponding query condition $q_i$
- $w_{sum}$: sum of weights over all query conditions
- $D_i$: diameter of the corresponding ith feature space
- $t_i$: raw (not normalized) distance from $q_i$ to the last returned object so far in the ith feature space

Basically, the main idea here is to find a value of LBTD that is ensured to be always less than or equal to the distance from the true NN to the query $Q$. For clearer, Figure 9 depicts an example to illustrate this idea with a two-condition M-FNN query.

In Figure 9a, assume that the query $Q$ of two conditions $q_1$ and $q_2$ is executing and the sequential range queries for $q_1$ and $q_2$ are $q_{10}$, $q_{11}$, $q_{12}$ and $q_{20}$, $q_{21}$, $q_{22}$, individually. These range queries are carried out in step 1 of the algorithm. Figure 9b shows a part of the possible example query relation/view with respect to Figure 9a. Assume that an appropriate record $T_{app}(A, B, \ldots)$ with the total distance $TD_{cur}$ is found after step 3 of the $\varepsilon$-ISA as sketched by a solid line segment connecting two objects: $A$ of $q_{11}$ and $B$ of $q_{20}$. Moreover, we also suppose that the $\varepsilon$-ISA is now doing range queries $q_{12}$ and $q_{22}$.

Intuitively, if $T_{app}$ is not the “true” NN of $Q$ then the total distance TD of this “true” NN cannot be less than the weighted normalized searching radii $d_1$ and $d_2$ of range queries $q_{12}$ and $q_{22}$, which are computed as follows: $d_1 = r_{q_{12}} \cdot w_1 / (D_1 \cdot w_{sum})$ and $d_2 = r_{q_{22}} \cdot w_2 / (D_2 \cdot w_{sum})$, where $r_{q_{12}}$, $r_{q_{22}}$ are radii of $q_{12}$ and $q_{22}$, respectively; $w_1, w_2, D_1, D_2$ are defined as described in formula 6. This total distance TD is equal to the weighted normalized radius of $q_{12}$, i.e. equals $d_1$, for example, when the appropriate record found is an exact composition of $q_2$ and a returned NCR-Value $C$ for the range query $q_{12}$ (see the first dashed line $T_1$ as illustrated in Figure 9a that is being represented for an appropriate record $T_1(C, q_2, \ldots)$ as illustrated in Figure 9b). A similar explanation is for dashed line $T_2$. Therefore, in accordance with Figure 9, we have: $LBTD = \min \{TD_{cur}, \text{weighted normalized radius of } q_{12}, \text{weighted normalized radius of } q_{22}\} = \min \{TD_{T_{app}}, d_1, d_2\}$. Note that here we do not exclude a case in which $app$ is itself the “true” NN of $Q$ and, practically, we can also observe that LBTD = $\min \{TD_{T_{app}}, TD_{T_1}, TD_{T_2}\}$ where $TD_{T_1}$, $TD_{T_2}$ are total distances of records $T_1$, $T_2$ to the query, individually. Therefore, we call $d_i$ terms as shown in formula 5 partial total distances.

Depending on the LBTD computed by step 4, step 5 of the algorithm decides whether record $T_{app}$ is a $(1+\varepsilon)$-approximate NN of $Q$ or not. Step 6 is quite similar to step 4 of the ISA to ensure that the $\varepsilon$-ISA will not access unnecessary objects and unneeded disk pages during the search process. In the case if the current searching radii of every $q_i$ are
greater than or equal to their corresponding maximum searching radii then searching on every $F_i$ is stopped and, actually, the returned record is the “true” NN of the query $Q$ (because at this time the $\varepsilon$-ISA simply becomes the ISA). The correctness of the $\varepsilon$-ISA is the evident truth that can easily be perceived from the above discussions as well as the correctness of the ISA as mathematically proven in [5].

3.3.2 A Generalized Algorithm for Finding Approximate k-Nearest Neighbors

The $\varepsilon$-ISA version described in section 3.3.1 just returns a $(1+\varepsilon)$-approximate NN of a CVQ. In practice, almost users need some approximate top-k NNs to the CVQ. We define a record/tuple is a $(1+\varepsilon)$-approximate kth NN to a complex M-FNN query as follows:

**Definition 4** (the approximate kth nearest neighbor of a complex M-FNN query): Given a M-FNN query $Q$ of $n$ query conditions, a query relation/view $S$ and feature spaces (e.g. the NCR-Tables) corresponding to the mapped attributes of $S$. Assume the total distance (TD) from each record of $S$ to $Q$ is calculated by formula 2. A record $T_{app} \in S$ is defined as a $(1+\varepsilon)$-approximate kth NN of $Q$ with $\varepsilon > 0$ if the ratio of the TD between $Q$ and $T_{app}$ and the TD between $Q$ and its true kth NN is at most $(1+\varepsilon)$.

In fact, in Definition 4, a $(1+\varepsilon)$-approximate kth NN of $Q$ can be closer to $Q$ than the true kth NN (cf. [1]). On the other hand, note that the incremental algorithm adapted for range queries [5], which is being used for the $\varepsilon$-ISA as described in Figure 8, is originated from an algorithm for finding k-nearest neighbors of a S-FNN query in which k may be unknown in advance [27, 47]. Hence, the problem of dealing with approximate k-nearest neighbors for M-FNN queries and k is not necessary to be known beforehand is easily solved by making some modifications to the $\varepsilon$-ISA as presented below:

1. A priority queue $PQ$ (or a sorted buffer) is needed to keep all appropriate records to $Q$ (as well as their other necessary information like the total distance to the query) that have been found in step 3 of the $\varepsilon$-ISA as introduced in Figure 8.
2. The next approximate NN of the query $Q$ is verified similarly to that of the $\varepsilon$-ISA version, but here the record at the top of $PQ$ takes part as $T_{app}$ in step 3 of the algorithm (i.e. the $\varepsilon$-ISA).
3. When a record at the top of $PQ$ is ensured to be the next approximate NN to the query $Q$, it will be added to the result set and be removed from $PQ$.
4. In the case $PQ$ is empty but the user still needs more results, the search is continued normally on all the involved feature spaces. Due to the aspects of the incremental algorithm adapted for range queries, the search will continue from where it left when computing the previous approximate NN.

The correctness of this $\varepsilon$-ISA version is easily inferred from that of the $\varepsilon$-ISA presented in the previous section. Besides, to address the approximate k-NN problem for complex M-FNN queries in the case the value of $k$ is known in advance, we do not need to maintain all appropriate records to $Q$ in a PQ as discussed above, but only $k$ closest appropriate records to $Q$. The other modifications in this case are still the same as mentioned above. In section 3.4 below we show experimental results to prove the efficiency of the $\varepsilon$-ISA.

3.4 Experimental results for approximate M-FNN queries

In order to establish the practical value of our algorithm, the $\varepsilon$-ISA, we implemented it based on the ISA and carried out a number of experiments to compare the $\varepsilon$-ISA with the ISA.

![Figure 10](image.png)

**Figure 10** Two-condition (4-d and 8-d) NN queries, different $\varepsilon$ values
Our experiments used the following data sets:

1. Uniformly distributed (also called synthetic) data sets of 2, 4 and 8 dimensions. Each synthetic data set has 100,000 objects of the form \(<id, values \text{ list}>\) and takes role as a feature space (i.e., a NCR-Table in the VQS).

2. Real data sets of 9 and 16 dimensions: Each of them has more than 64,000 feature vectors of images of the form \(<id, values \text{ list}>\) and they are based on color moments and color histogram of the images, respectively (cf. http://kdd.ics.uci.edu/).

The experiments were carried out for CVQs of 2 and 3 query conditions. For each feature space of all experiments, 100 query points were randomly selected from the corresponding data set. Furthermore, we used the SH-tree to manage the storage and the retrieval of multidimensional data objects in each feature space. We selected the SH-tree because it is one of the most flexible and efficient multidimensional index structures [5]. All the tests were tackled on an Ultra-SPARC-II 360MHz with 512 Mbytes of main memory and some Gigabytes of hard disk capacity. All programs were implemented in C++. The page size was 8Kb for the data sets so as to meet with the disk block size of the operating system.

For all experiments, we simulated the database access performance to the query relation by comparing field id of the affected objects. For example, if a 4-dimensional object \(<id_4, values \text{ list}>\) and an 8-dimensional object \(<id_8, values \text{ list}>\) were retrieved during the search and \(id_4=id_8\), then they were considered belonging to a record in the query relation. This record is called an appropriate record in this paper. Intuitively, for a particular query optimization processor in the database system, the affected object number in each feature space is directly proportional to the cost of database access. Therefore, we reported the disk access number of the searches on the SH-trees and the affected object number for all experiments. The affected objects here are objects located in the last hypersphere range query for each feature space.

Our first experiment was carried out with the data sets of 4 and 8 dimensions to find \((1+\varepsilon)\)-approximate NN for 2-condition M-FNN queries. The \(\varepsilon\) values selected for the tests were 0.2, 0.4, 0.6, 0.8 and 1. Figures 10a, 10b and 10c show the experimental results. Figure 10a shows the percentage of disk access savings for each feature space against \(\varepsilon\) values; Figure 10b shows the percentage of affected object savings against \(\varepsilon\) values; and Figure 10c depicts the accuracy of the approximate results, compared to the true NNs, with respect to the \(\varepsilon\) values.

We observe in the first experiment that the accuracy is very high while all costs are significantly saved. For example, for \(\varepsilon=0.6\) (that means the tolerant error is allowed up to 60%), the \(\varepsilon\)-ISA saved over 55% and nearly 7% disk accesses, nearly 80% and over 35% affected object number with respect to 4- and 8-dimensional feature space, individually, while it still preserved the accuracy of the returned results at 90%. The experimental results were also promising for the small \(\varepsilon\) values, e.g. for \(\varepsilon=0.2\) as depicted in Figure 10c, the accuracy was up to 98% and the cost savings were still quite high, e.g. over 27% of disk accesses and over 44% of affected objects for the 4-dimensional feature space were saved during the search.

The second experiment was done with two data sets of 4 dimensions. The \(\varepsilon\) value was set to 0.2. Here we did the experiment to show the efficiency of the \(\varepsilon\)-ISA in dealing with approximate k-nearest neighbor problem. The values of \(k\) were set to 5, 10, 15, 20, 25 and 30. Figures 11a, 11b and 11c show the experimental results with notations similar to those of the previous experiment. The results obtained were quite good for small values of \(k\) and better for larger values of \(k\). Specially, for \(k=25\) or 30, disk access savings were over 10% and affected object savings were over 23% for each 4-dimensional feature space, while the accuracy was maintained at nearly 90%
against the true k-nearest neighbors of the selected queries.

Next experiment was performed on the data sets of 2 dimensions. For this experiment the $\epsilon$-ISA was carried out to find $(1+\epsilon)$-approximate NN for 3-condition queries. The main purpose of this test was to show effect of the query condition number on the efficiency of the $\epsilon$-ISA. The $\epsilon$ values chosen were 0.25, 0.5, 0.75 and 1. The results are shown in Figures 12a and 12b. Figure 12a depicts all costs savings and Figure 12b shows the accuracy, compared to the true NN of the selected queries, with respect to the $\epsilon$ values. The experimental results are particularly considerable. For $\epsilon = 0.5$ (that means the tolerant error is permitted up to 50%) the $\epsilon$-ISA saved nearly 18% of disk accesses and over 17%, 22% and 24% of the affected objects for each 2-dimensional feature space, individually, while still preserving the accuracy of the approximate results at 91%.

The last experiment was carried out with the two real data sets of image feature vectors. Here the $\epsilon$ value was set to 1, i.e. the tolerant error was permitted up to 100%. 100 two-condition M-FNN queries were performed and the results were quite good: the $\epsilon$-ISA saved nearly 18% disk accesses and over 17%, 22% and 24% of the affected objects for each 2-dimensional feature space, individually, while still preserving the accuracy of the approximate results at 91%.

4. SOLVING APPROXIMATE SINGLE-FEATURE NEAREST NEIGHBOR QUERIES

Although dimension-reduction techniques have been introduced to lighten complexities of the similarity search problem in high-dimensional feature spaces, e.g. Karhunen-Loeve transform, [18], etc. and even there are also a various number of approaches to this problem, the research results obtained so far are still unsatisfactory, and thus much more research is necessary. Of late, there has been an increasing interest in avoiding or alleviating the curse of dimensionality by resorting to approximate nearest neighbor searching. As declared in [36], and with reference also to Definition 1, only a small value of $\epsilon$ could suffice for most practical purposes. Below we present some typical results resulting from previous researches for the problem of $(1 + \epsilon)$-approximate S-FNN query answering and related work.

There are some notable surveys of data structures for nearest neighbors as [31, 32, 50]. In such literature, one can find valuable information about variants of k-d-trees, R-trees, hashing tables, and structures based on space-filling curves. The problem is that the proposed structures almost only perform well in low- and medium-dimensional spaces. In high-dimensional spaces, they all exhibit poor behavior in the worst case as well as typical cases [1, 51].

For approximate nearest neighbor, the situation is slightly better. The problem was firstly studied by Bern [52]. He proposed a data structure based on quad trees [53], which uses linear space and provides logarithmic query time. However, the approximation error factor for his algorithm is a fixed function of the dimension. In [4] Arya et al. introduced an algorithm with query time $O((1/\epsilon)\log n)$ and using a nearly linear data structure. The dependence on $\epsilon$ was reduced by Clarkson afterwards [54]. In [1] Arya et al. strengthened results in their previous work and obtained optimal $O(n)$ space pre-processing requirement cost, but with query time.
growing as $O(d^4)$. Recently, in [3] the author gave an algorithm with $O(n \log d)$ pre-processing cost and query time polynomial in $d$, $e$ and $n$. In the same paper, the author also gave another algorithm with pre-processing polynomial in $d$, $e$ and $n$ but with query time $O(n + d \log 3n)$. This second algorithm improves the $O(dn)$ time bound of the brute-force algorithm. Later, in [35, 36], the authors independently proposed new algorithms to further improve the space and time complexities. Concretely, their algorithms can eliminate all exponential dependencies in dimension, yielding a query time $O(d \log^{O(1)}(dn))$ and space $(dn)^{O(1)}$. Here, the $O$-notation hides constant factors exponentially depending on tolerant error parameter $e$, but not on dimension $d$.

In the context of semantic based similarity search systems or FQASs like the VQS, sometimes users want answers to queries as “Find employees living near the company” or “Find managers whose salary is approximately at 100 000$ annually”, etc. In such cases (and many other cases) we can observe that the query specification is even itself a “guess”. Therefore, approximate answers to such queries are of course suitable. Furthermore, in many cases the difference between the NN and a “good” approximation is indistinguishable from a practical point of view [37].

Intrinsically, answering approximate queries requires satisfying two conflicting requirements as follows: (1) low processing costs (IO-cost, CPU-cost, memory cost), and (2) high accuracy of the results (i.e., low errors, or more exactly, low effective tolerant errors — cf. section 3.4). In section 3, we discussed and introduced an approximate k-NN algorithm for solving complex multi-feature NN queries. The proposed algorithm has a very promising efficiency satisfying both requirements (1) and (2) above. It has been proven through the theoretical analyses as well as the experiments. In this section, we present an adaptation to the SH-tree for answering approximate S-FNN queries. This adapted algorithm is briefly described as in Figure 13.

This approximate k-NN algorithm is a variant of the k-NN algorithm, which is itself in turn an adaptation to the incremental NN algorithm [47], for finding k-nearest neighbors with the SH-tree. The algorithm also uses a priority queue $PQ$ to realize the best-first search method. It means that the next node or object will be accessed is always put on top of the queue. The comparison between elements in the queue is done with respect to their distances to the query object $Obj$.

The algorithm is easy to understand as described in Figure 13. Note that the SH-trees have three kinds of nodes: internal, balance, and leaf nodes (cf. section 2). Depending on which kind of node at the top of $PQ$ the algorithm will process it accordingly. In line 9, the algorithm check if no other data objects that have not been seen so far can have a
distance to Obj less than \( Rset[k].dist/(1+\epsilon) \), in which \( Rset[k].dist \) represents the distance of the current k-th nearest neighbors to Obj. If this is true then current k nearest neighbors (k-NNs) in the sorted buffer \( Rset \) are returned as approximate k-NNs of Obj. Here we should note that the distance from Obj to a bounding of any node in the tree is always a lower bound with respect to its distance to any data object (or any child node) contained in this node. If the condition in line 9 does not hold, then the algorithm will check the type of object seen at the top of PQ. In case this is an internal node, the algorithm must determine bounding regions for its left and right child nodes, \( \text{MINDIST} \) from Obj to them and then put them into PQ if necessary. Conditions in lines 16 and 18 are to make sure that the left and right child nodes, individually, perhaps result in a data object that has a smaller distance to Obj than the k-th current NN in \( Rset \). These conditions prune nodes that should put into the queue. The famous lower bound distance metric \( \text{MINDIST} \) was introduced [26]. In case the object at the top of \( PQ \) is a balanced node, the algorithm computes the distance from Obj to each child of the balanced node (i.e., to a leaf node), \( \text{Distance}(Obj, Leaf) \), and puts the leaf into \( PQ \) if the condition in line 23 holds. Note that, the distance \( \text{Distance}(Obj, Leaf) \) is defined as the longer one between the minimum distances to the leaf’s MBR (Minimum Bounding Rectangle) and BS (Bounding Sphere) [5]. In case a leaf node has been seen at the top of \( PQ \) then the algorithm will process similarly to the above case. It will firstly access objects in the leaf and then computes the distance from Obj to each of them, \( \text{ObjectDistance}(Obj, Object) \). If this distance satisfies the condition in line 28, the corresponding data object will be put into the result buffer \( Rset \) and then this sorted result buffer will be updated.

The correctness of this algorithm\(^\text{10} \) is obvious. Note that, as any approximate algorithms proposed in the literature, performance of this algorithm depends on the choice of \( \epsilon \) values. Intuitively, the larger the \( \epsilon \) value is, the faster the algorithm terminates. However, this may negatively affect the quality of the result (more exactly, the effective \( \epsilon \) values may be smaller). We would also remark here that our algorithm as presented in Figure 13 is \( \epsilon \)-independent, implying that once the algorithm has been built, any \( \epsilon \) value can also be used. Obviously, setting \( \epsilon = 0 \) will cause our algorithm to compute the true NN. We have implemented the algorithm and done tests with a real data set of 9-dimensional image feature vectors based on their color moments (cf. section 3.4). We performed 100 approximate 10-NN queries with different \( \epsilon \) values. All the tests were carried out on a SUN ULTRA 30, CPU 300 MHz, 384 Mbytes of the main memory and some Gigabytes of the hard disk capacity. Experimental results are presented in Table 1.

In Table 1, the first two columns show experimental results of two adapted k-NN algorithms for the SH-tree as introduced in [5]: algorithm 1 adapted from [26] and algorithm 2 from [27]. Other columns show experimental results of approximate \( k \)-NN algorithm as described in Figure 13 with different values of the tolerant error \( \epsilon \). The first row shows IO-cost (i.e., the disk access number averaged over 100 test queries), the second row shows averaged CPU time that needs to complete each query, and the third row presents the accuracy of k-NN results returned. Evidently, for algorithms 1 and 2, i.e. correct k-NN (C-kNN) algorithms, the accuracy of results obtained equals 100%. This is also correct for approximate k-NN algorithm as \( \epsilon = 0 \). From results showing in Table 1, we observe that the approximate k-NN algorithm performs very well: It can keep interestingly high accuracy of the returned results while significantly reducing costs of the query processing. For example, with \( \epsilon = 0.25 \), this approximate algorithm can save nearly 44% (resp. ~24%) of IO-cost, and nearly 36.4% (resp. ~74.6%) of CPU-cost compared to adapted algorithms 1 and 2, respectively, meanwhile it still keeps the accuracy at 100%. One more interesting result is that the approximate algorithm with \( \epsilon = 0 \) performs better than both adapted algorithms. It can be explained by the more efficient pruning strategy when we know the \( \epsilon \) value in advance (see lines 16, 18, and 23 of the algorithm), which leads to a smaller priority queue in size. This also confirms a conclusion presented in [47] that the size of the priority queue affects the performance of queue operations during the algorithm’s execution (and thus affects the performance of the algorithm). More importantly, this result leads us to conclude that in the case of k-NN queries with \( \epsilon \) is determined beforehand, the algorithm as shown in Figure 13 could be a suitable substitute for all previously adapted algorithms for the SH-tree (note that \( \epsilon \) value is set to zero in this case).

To conclude this section, we present here an example to more intuitively show the effect of \( \epsilon \) value on the quality of results returned to an approximate query. Figure 14 sketches the scenario.

In Figure 14 we assume that \( q \) is the query object and \( p \) is the current NN with \( \text{ObjectDistance}(q, p) = r \). Assume the bounding sphere (q, r) is now overlapping a leaf node LN1 and an internal node IN1 and they will be visited in this order (according to their distances to \( q \)). Note that, leaf nodes in the SH-tree have both minimum bounding rectangle (MBR) and bounding sphere (BS), while internal nodes have only bounding rectangle (BR, even not MBR). With NN algorithms, they have to first access the leaf node LN1, and find out that \( \text{ObjectDistance}(q, x) < \text{MINDIST}(q, \text{IN1}) \), and \( \text{ObjectDistance}(q, x) < r \) (in fact, in this case we have \( \text{ObjectDistance}(q, x) < \text{MINDIST}(q, \text{IN1}) < r \)). Therefore, object \( x \) will be returned as the “true” NN without accessing the internal node IN1. Nevertheless, with the approximate algorithm as presented in Figure 13, when \( p \) has already been determined to be the current approximate NN and when the leaf node LN1 comes to the top of the priority queue, the algorithm will find out that \( \text{Distance}(q, \text{LN1}) > r/(1+\epsilon) \), and thus \( p \) is immediately returned as the approximate NN without more accesses like the NN algorithms. But, if \( \epsilon \) value is chosen small enough so that inequality \( \text{Distance}(q, \text{LN1}) \geq r/(1+\epsilon) \) still does not hold, then more accesses to disk pages and objects should be the case.

5. Solving Approximate Range Queries

Range searching is one of the most crucial important problems in computational geometry because it arises in many application areas and a wide variety of geometric problems can be formulated as a range searching problem [55]. There

\(^{10}\) Also note here that, in [5] we introduced two different adaptations of the incremental k-NN algorithm originally proposed in [27] to the SH-tree.
is the sheer volume of literature on this problem over the last decades such as [2, 56, etc.]. There are also some recent surveys that have summarized the development history of the problem as [55, 57]. With reference to [2, 55], there are typical kinds of ranges like hyper-rectangles, spheres, half-spaces, or simplices. The final goal is to preprocess the given data set S into a data structure so that for a range query R, the data objects in R∩S can be reported or counted efficiently. As the nearest neighbor problem, almost researches and approaches in range searching problem have focused on solving the problem exactly. However, lower bounds show that if linear space is assumed, the problem cannot be solved in polylogarithmic time, except for the case of orthogonal ranges [2]. This suggests that it is worth considering variants of the problem. One of the vanguard approaches to this was originally introduced in [2].

Firstly, the algorithm calculates outer range \( bs^+ \) and inner range \( bs^- \) (line 7) for parameter \( bs \) (cf. Figure 16) then it examines type of node. If this is an internal node, the algorithm checks if \( br \) is contained in \( bs^+ \) (line 10). If this is the case, all data objects contained in node are added to the result set buffer \( Rset \). On the contrary, the algorithm checks if \( br \) does not intersect with \( bs^- \) (line 12). If this is correct then nothing will be done more with this node (line 13). Otherwise, the algorithm computes bounding rectangles for left and right child nodes of the internal node (lines 14, 15) and then makes recursive calls to procedure \( AppBSRangeQuery \) for these left and right child nodes (lines 16, 17). In case node is not an internal node, but a balanced node (line 18), processing steps are similar to those of the internal node above except that parameter \( br \) in the recursively procedure calls for the leaf nodes (child nodes of the balanced node) is \( NULL \) (line 24). The reason is that we do not need such information any more in processing steps with the leaf nodes (lines 25-29).

Although the returned results of algorithm presented in Figure 15 comply with Definition 2, the processing in this algorithm is in fact a little different: the algorithm treats the range as a fuzzy object, meaning that objects lying within distance \( \varepsilon*(bs.R) \) of the boundary of the bounding sphere range query \( bs \) either may or may not be returned. This is the same as algorithm proposed in [2]. Also, note that the algorithm presented in [2] computes the accumulated weight of the points satisfying the approximate query, whereas our algorithm here will retrieve all objects satisfying the approximate query. The proof of the correctness of our algorithm remains the same as that presented in [2]. Figure 16 depicts a graphical view of our problem:

From Figure 16 we can easily observe that objects belonging to the area \( (bs^+ − bs^-) \) may be returned and may also not be returned, whereas objects belonging to \( bs^- \) will always be returned, whereas objects belonging to \( bs^+ \) may be satisfied with an answer that is just relatively accurate. Towards supporting and satisfying the user information need and integrating the SH-tree into such systems, we develop an approximate algorithm, as described in Figure 15, for answering bounding sphere12 range queries in the SH-trees.

It is not difficult to follow the algorithm. Among parameters of the algorithm, parameter \( br \) (bounding rectangle of node) may be questioned. However, we shall call procedure \( AppBSRangeQuery \) with parameter node as the root node of the SH-tree so \( br \) can easily be determined because the root of the SH-tree always keeps and updates this value whenever a new object is inserted to the tree.

Table 1 Experimental results of approximate k-NN queries

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>I/O-cost</th>
<th>CPU-cost</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg1 (C-kNN)</td>
<td>76.22</td>
<td>0.099</td>
<td>100%</td>
</tr>
<tr>
<td>Alg2 (C-kNN)</td>
<td>56.12</td>
<td>0.248</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 1: Experimental results of approximate k-NN queries.

Figure 14 Approximate NN results and correct NN results

Figure 15 Approximate NN results and correct NN results

11 Algorithm as described in Figure 13 is a variant of the first one, which leads to higher CPU-cost during the query processing among the two. In table 1, this adapted algorithm is named “Alg2”.

12 In fact, our algorithm here can also be applied to other range types with some minor modifications.
Approximate algorithm for bounding sphere range queries:

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AppBSRangeQuery(bs, e, Rset, node, br)</td>
</tr>
<tr>
<td>2</td>
<td>BOUNDINGSphere bs //bounding sphere range query</td>
</tr>
<tr>
<td>3</td>
<td>FLOAT e //Epsilon - tolerant error</td>
</tr>
<tr>
<td>4</td>
<td>RESULTBUFFER Rset //result set buffer</td>
</tr>
<tr>
<td>5</td>
<td>NODEPOINTERS node //pointer to a node of the tree</td>
</tr>
<tr>
<td>6</td>
<td>BOUNDINGRECT br //bounding rectangle of node</td>
</tr>
<tr>
<td>7</td>
<td>Compute outer range bs+ and inner range bs-, which are the dilation and erosion, respectively, of bs by distance ε*(bs, R)</td>
</tr>
<tr>
<td>8</td>
<td>Case (node.Type) of</td>
</tr>
<tr>
<td>9</td>
<td>INTERNAL_NODE:</td>
</tr>
<tr>
<td>10</td>
<td>If br ⊆ bs- then</td>
</tr>
<tr>
<td>11</td>
<td>Descend the tree from this internal node and add all data objects in the leaf nodes to the result set Rset, and then return.</td>
</tr>
<tr>
<td>12</td>
<td>If br ∩ bs = ⊘ then</td>
</tr>
<tr>
<td>13</td>
<td>return</td>
</tr>
<tr>
<td>14</td>
<td>Otherwise, compute bounding rectangles for left and right children (note that d is the split dimension), and then recursively call AppBSRangeQuery for each of them.</td>
</tr>
<tr>
<td>15</td>
<td>BOUNDINGRECT left_rec = br ∩ (d ≤ node.up)</td>
</tr>
<tr>
<td>16</td>
<td>BOUNDINGRECT right_rec = br ∩ (d = node.lo)</td>
</tr>
<tr>
<td>17</td>
<td>AppBSRangeQuery(bs, e, Rset, node.Left, left_rec)</td>
</tr>
<tr>
<td>18</td>
<td>AppBSRangeQuery(bs, e, Rset, node.Right, right_rec)</td>
</tr>
<tr>
<td>19</td>
<td>BALANCED_NODE:</td>
</tr>
<tr>
<td>20</td>
<td>If br ⊆ bs- then</td>
</tr>
<tr>
<td>21</td>
<td>Descend the tree from this balanced node and add all data objects in the leaf nodes to the result set Rset, and then return.</td>
</tr>
<tr>
<td>22</td>
<td>If br ∩ bs = ⊘ then</td>
</tr>
<tr>
<td>23</td>
<td>return</td>
</tr>
<tr>
<td>24</td>
<td>For each entry Leaf in the balanced node do</td>
</tr>
<tr>
<td>25</td>
<td>AppBSRangeQuery(bs, e, Rset, Leaf, NULL)</td>
</tr>
<tr>
<td>26</td>
<td>LEAF_NODE:</td>
</tr>
<tr>
<td>27</td>
<td>For each object Object in the leaf node do</td>
</tr>
<tr>
<td>28</td>
<td>//compute distance between center of bs and Object</td>
</tr>
<tr>
<td>29</td>
<td>FLOAT dist = ObjectDistance(bs,C, Object)</td>
</tr>
<tr>
<td>30</td>
<td>//if Object falls into/overlaps the range query</td>
</tr>
<tr>
<td>31</td>
<td>If (dist ≤ bs, R) then</td>
</tr>
<tr>
<td>32</td>
<td>Add Object to the result set Rset</td>
</tr>
</tbody>
</table>

Figure 15 Pseudo-code of the adapted approximate range query algorithm

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced three types of approximate queries of great interest and efficient methods to address them: (1) approximate single-feature nearest neighbor (S-FNN) queries, (2) approximate multi-feature nearest neighbor (M-FNN) queries, and (3) approximate range queries. Our discussions were mainly focused on the second query type, i.e., approximate M-FNN queries. The other two are well-known and quite well-addressed with a great number of research publications.

In particular, we introduced an approach called ε-ISA (ISA stands for Incremental hyper-Sphere Approach) to efficiently solving the approximate nearest neighbor problem for M-FNN queries, which are also called complex vague queries (CVQs). To the best of our knowledge, the ε-ISA is one of the vanguard solutions for dealing with the approximate M-FNN queries. Although the ε-ISA is presented in the context of the Vague Query System (VQS) [7], it can also be applied to a variety of application domains such as multimedia databases (e.g., “find images that their shape, texture, and color are approximately similar to cat, cloud and red, individually”), GISs and TISs (e.g., “find a hotel located near the city centre, and priced at about 100 EUR/Night”), digital libraries, and so on. In general the ε-ISA is very useful for included in the result set. We also observe that objects returned but belonging to the area(bs+ – bs-) are false hits. In [2] the authors introduced a concept called the actual average error committed by the approximate algorithm as follows: Consider a range of radius r and a point at distance r'. If r'<r but the point was not included in the result set (in our case, it may be some objects in the area (bs - bs-)), the associated misclassification error is defined as the relative error, (r-r')/r. Otherwise, if r'>r but the point was included in the result set, the associated misclassification error is (r'-r)/r. This relative error is measured for every misclassified point and then we averaged them over all points lying in the area (bs+ – bs-) to establish the average error of the query. With a number of experiments, they observed that the average error was much lower than the allowed error factor ε.
application domains that the returned results need not to be exact but similar or approximately similar (with a certain tolerance error) to a given query. Our experimental results with both synthetic and real data sets have proven this. With a suitable $\varepsilon$ value, the $\varepsilon$-ISA can save a very high percentage of the costs, including both IO-cost and CPU-cost, while it still preserves the accuracy of the returned results at a particularly high value. Specially, the $\varepsilon$-ISA is also applicable to general feature spaces including ones that do not satisfy three conditions, which are pointed out in [44], to make Fagin’s algorithm [39] and all its variants such as [41, 42] become utilizable. A fruitful discussion for these approaches and considerable related information can be found in [46, 60, 61].

Besides, we also discussed the first and third query types in two separated sections with notable considerations. For each of them, we proposed efficient and effective adaptations to the SH-tree for dealing with the concerned problem. This one more time confirms the powerfulness as well as the flexibility of the SH-tree, a super hybrid index structure for multidimensional data. Moreover, from our experimental results in section 4, we have also realized that in the case of k-NN queries with $k$ is known in advance, our proposed algorithm as shown in Figure 13 is a suitable substitute for all adapted algorithms previously introduced for the SH-tree.

Our future work will concentrate on deploying the $\varepsilon$-ISA as well as the SH-tree for the real world applications such as GISs and TISs (see, e.g., [20]), modern IR systems, etc. Specially, other types of approximate similarity queries as approximate complex vague joins, approximate reverse queries (given a query object $Q$, find all data objects of the database to which $Q$ is the approximate nearest neighbor), etc. are also of interest and still need more research efforts.

REFERENCES


37 Ciaccia, P., Patella, M. (2000) PAC nearest neighbor queries: approximate and controlled search in high-dimensional and metric spaces, Proc. 16th Int. Conf. on Data Engineering, California, USA, 244–255


40 Ortega, M., Rui, Y., Chakrabarti, K., Mehrotra, S., Huang, T.S. (1997) Supporting similarity queries in MARS, Proc. 5th ACM Int. Conf. on Multimedia, Seattle, USA, 403–413

41 Nepal, S., Ramakrishna, M.V. (1999) Query processing issues in image (multimedia) databases, Proc. 15th Int. Conf. on Data Engineering, Sydney, Australia, 22–29


queries over web-accessible databases, *Proc. 18th Int. Conf. on Data Engineering, San Jose, California, USA, 369-382*