In Proc. 10th International Symposium on Spatial and Temporal Databases (SSTD'07), Boston, U.S.A., 2007.

Generalizing the Optimality of Multi-Step *k*-Nearest Neighbor Query Processing

Hans-Peter Kriegel, Peer Kröger, Peter Kunath, and Matthias Renz

Institute for Computer Science, Ludwig-Maximilians Universität München
{kriegel,kroegerp,kunath,renz}@dbs.ifi.lmu.de
WWW home page: http://www.dbs.ifi.lmu.de

Abstract. Similarity search algorithms that directly rely on index structures and require a lot of distance computations are usually not applicable to databases containing complex objects and defining costly distance functions on spatial, temporal and multimedia data. Rather, the use of an adequate multi-step query processing strategy is crucial for the performance of a similarity search routine that deals with complex distance functions. Reducing the number of candidates returned from the filter step which then have to be exactly evaluated in the refinement step is fundamental for the efficiency of the query process. The state-of-the-art multi-step k-nearest neighbor (kNN) search algorithms are designed to use only a lower bounding distance estimation for candidate pruning. However, in many applications, also an upper bounding distance approximation is available that can additionally be used for reducing the number of candidates. In this paper, we generalize the traditional concept of R-optimality and introduce the notion of R_I-optimality depending on the distance information I available in the filter step. We propose a new multi-step kNN search algorithm that utilizes lower- and upper bounding distance information (I_{lu}) in the filter step. Furthermore, we show that, in contrast to existing approaches, our proposed solution is $R_{I_{lu}}$-optimal. In an experimental evaluation, we demonstrate the significant performance gain over existing methods.

1 Introduction

In many database applications such as molecular biology, CAD systems, multimedia databases, medical imaging, location-based services, etc. the support of similarity search on complex objects is required. In general, the user wants to obtain as many true hits as soon as possible. Usually, in all these applications, similarity is measured by metric distance functions. The most popular query types are distance range (or ε-range) queries, k-nearest neighbor (kNN) queries, and – more recently – reverse kNN queries. Those queries can be supported by index structures such as the R-tree [1] or the R*-tree [2] and their variants for Euclidean data or by the M-tree [3] and its variants for general metric data. These index structures are designed for shrinking down the search space of tentative hits in order to scale well for very large databases.

However, index structures usually invoke a large number of distance computations and, thus, do neither account for the increasing complexity of the database objects nor for the costly distance functions used for measuring the similarity. To cope with complex data objects and costly distance functions, the paradigm of multi-step query processing has been defined for spatial queries such as point queries and region queries [4, 5]. This paradigm has been extended to similarity search in databases of complex objects performing distance range queries [6, 7] and kNN queries [8, 9]. The key idea of multi-step query processing is to apply a so-called *filter step* using a cheaper distance function, the so-called *filter distance*, in order to prune as many objects as possible (as true hits or true drops). For the remaining candidates, for which the query predicate cannot be decided using the filter distance, the exact (more costly) distance needs to be evaluated in the so-called *refinement step*.

In most applications, two different types of filter distances are commonly used for multi-step query processing. First, a lower bounding filter distance produces distances that are always lower or equal to the exact distance and can be used to discard true drops. Second, an upper bounding filter distance produces distances that are always greater or equal to the exact distance and can be used to identify true hits. While both types of filter distances have successfully been used for distance range queries, all existing multi-step kNN query processing algorithms only use the lower bounding filter distance. Thus, these approaches can only prune true drops, but cannot identify true hits in the filter step. Furthermore, these approaches cannot report true hits already after the filter step, but need to refine the candidates before reporting them as hits. However, in applications where the results are further processed and this processing is quite costly due to the complexity of the data objects, it is desirable to output true hits as soon as possible even if the result set is not yet complete. Obviously, using an upper bounding filter distance may allow to output a first set of true hits already after the filter step before refinement. In addition, using an upper bounding filter distance could significantly reduce the number of candidates that need to be refined and, thus, could clearly improve query execution times. As a consequence, the storage required to manage intermediate candidates during the entire filter-refinement procedure can also be reduced.

In general, using also upper bounding distance information in the filter step yields several advantages as long as no ranking of the kNNs is needed. However, in many applications, users only want the result of a given kNN query rather than a ranking. For example, a restaurant owner planning a public relations campaign by sending a fixed number k of flyers to potential costumers may choose the k customers with the smallest distance to the restaurant's location. In addition, many data mining algorithms that rely on kNN computation such as density-based clustering, kNN classification, or outlier detection only require the result of a kNN query, but not its ranking. Many of those methods use the result of kNN queries for further processing steps.

In this paper, we propose a novel multi-step query processing algorithm for kNN search using both a lower and an upper bound in the filter step. We show that this algorithm is optimal, i.e. that it produces a minimum number of candidates which need to be refined. For that purpose, we generalize the notion of R-optimality taking the distance estimations available in the filter step into account. In a broad experimental evaluation, we show that when using our novel multi-step query algorithm, the application of an upper bound in addition to a lower bound in the filter step yields a significant performance gain over the traditional approach using only lower bounding distance approximations. In particular, we show that this performance gain is not only in terms of the number of candidates that need to be refined, implying a runtime improvement, but also in terms of space requirements.

The rest of the manuscript is organized as follows. In Section 2 we discuss existing multi-step kNN query processing algorithms. A generalized notion of the optimality for multi-step kNN algorithms is presented in Section 3. Section 4 presents a novel multi-step kNN algorithm that meets the requirements of our new generalized optimality. Section 5 presents our experimental evaluation and Section 6 concludes the paper.

2 Multi-step *k*NN Query Processing

Let $\mathcal{D}$ be a database of objects and dist be a distance function on these objects. For a given query object q and a given positive integer $k \in \mathbb{N}^+$, a k-nearest neighbor (kNN) query on a database $\mathcal{D}$ retrieves the objects in $\mathcal{D}$ that have the k smallest distances to q, formally

Definition 1 (kNN query, kNN-distance). *For a query object q and a query parameter $k \in \mathbb{N}$, a kNN query in $\mathcal{D}$ returns the smallest set $NN^{\mathcal{D}}(q,k) \subseteq \mathcal{D}$ that contains (at least) k objects from $\mathcal{D}$, for which the following condition holds:*

$$\forall o \in NN^{\mathcal{D}}(q,k), \forall o' \in \mathcal{D} - NN^{\mathcal{D}}(q,k) : \mathrm{dist}(q,o) < \mathrm{dist}(o',q).$$

With $\mathrm{nn}_k{-}\mathrm{dist}(q,\mathcal{D}) = \max\{\mathrm{dist}(q,o) | o \in NN^{\mathcal{D}}(q,k)\}$ we denote the k^{th} nearest neighbor distance (also called kNN-distance) of q (w.r.t. $\mathcal{D}$).

Since the database $\mathcal{D}$ is usually clear from context, we write $NN(q,k)$ and $\mathrm{nn}_k{-}\mathrm{dist}(q)$ instead of $NN^{\mathcal{D}}(q,k)$ and $\mathrm{nn}_k{-}\mathrm{dist}(q,\mathcal{D})$, respectively.

Let us note that in case of tie situations we include all ties into the result set. Thus, the cardinality of the result set may exceed k and the result of a kNN query is deterministic.

A naive solution for answering a given kNN query is to scan the entire database $\mathcal{D}$ and test for each object if it is currently among the k-nearest neighbors. This naive algorithm has a runtime complexity of $O(N \cdot QP)$, where $N = |\mathcal{D}|$ denotes the number of objects in $\mathcal{D}$ and QP denotes the cost of evaluating the query predicate for one single object, which is usually dominated by the complexity of the applied distance function dist. Obviously, using such a naive solution for kNN query processing is very expensive and not feasible for a very large set of complex objects. In fact, the problem is two-fold: On one hand, since

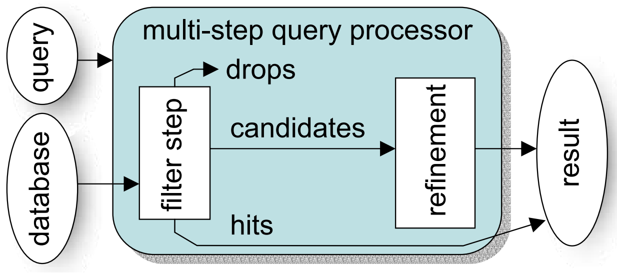


Fig. 1. Multi-step query processor.

the number of objects N in a database is usually very large, a sequential scan over all objects to evaluate the query predicate would produce very high I/O cost. On the other hand, due to the complexity of the distance function used in the above mentioned applications, the evaluation of the query predicate QP of one single object usually demands high CPU cost. In addition, many applications deal with very large objects such as audio or video sequences. As a consequence, the evaluation of the query predicate also invokes I/O cost and, thus, the cost for evaluating a query predicate become the bottleneck during query processing.

Indexing methods (i.e., single-step query processing solutions) that enable to prune large parts of the search space help to reduce the set of objects for which the query predicate has to be evaluated, i.e. address the first problem of high I/O cost due to a sequential scan. Theoretically, using an index, the runtime complexity is decreased to $O(\log N \cdot QP)$. However, index structures have two important drawbacks when dealing with complex objects and costly distance functions. First, indexes in general rely on the assumption that the distance function used is a metric. Otherwise, if the distance function defined on the database objects is not metric (in particular if the triangle inequality is not fulfilled), indexes cannot be applied. Second, and more severely, indexes are primarily designed to reduce the number of page accesses, but usually invoke the evaluation of the query predicate for many objects, e.g. during the index traversal and for the evaluation of the candidates reported from the indexing method. Obviously, when dealing with complex objects where QP is the bottleneck, a single-step query processing strategy is no longer feasible. Rather, a multi-step query processing approach is required reducing the set of result candidates in a filter step using an approximate evaluation of the query predicate which can be computed much faster than the exact evaluation (and optimally does not invoke extra I/O cost). This reduces the QP-part of the runtime complexity. In the filter step, as many hits and drops as possible (the amount obviously depends on the quality of the approximation) may already be identified. Finally, the remaining candidates have to be exactly evaluated in a refinement step in order to complete the result set. Since the filter step can also be supported by an index structure,

```
k-NearestNeighborSearch(q,k)
1 initialize ranking on index I
2 initialize result = sorted_list⟨key, object⟩
3 initialize $d_{max} = \infty$ // stop distance
4 while o= ranking.getnext() and $LB(q,o) \leq d_{max}$ do
5     if $\text{dist}(q,o) \leq d_{max}$ then result.insert(dist(q,o),o)
6     if result.length $\geq k$ then $d_{max}$ = result[k].key
7     remove all entries from result where key $> d_{max}$
8 endwhile
9 report all entries from result
```

Fig. 2. Multi-step kNN algorithm proposed in [9].

additionally the N part in the runtime complexity is decreased. A schematic description of the multi-step query processor is illustrated in Figure 1.

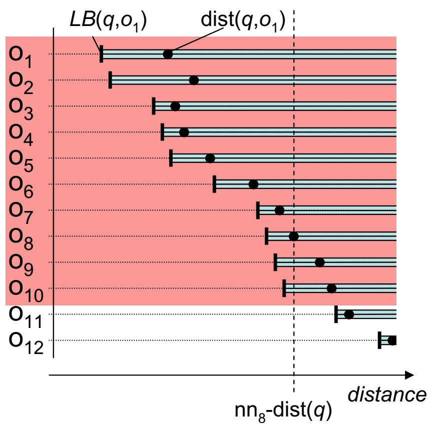
Obviously, a multi-step kNN algorithm is correct if the algorithm does neither produce false drops in the filter step, i.e. all drops do not fulfill the query predicate, nor produce false hits, i.e. all hits reported from the filter step really fulfill the query predicate.

The state-of-the-art multi-step kNN search method is the algorithm proposed in [9]. It uses a lower-bounding distance estimation LB in the filter step which is always lower or equal to the exact distance, i.e. for any query object q the *lower bounding property*

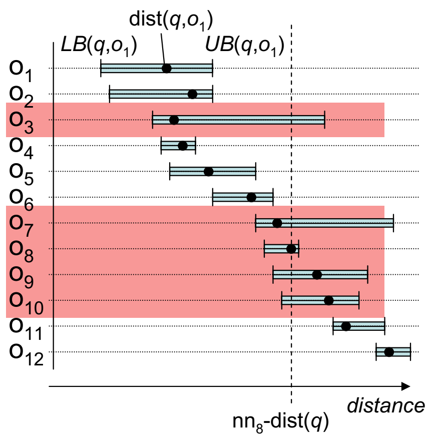
$$\forall o \in \mathcal{D} : LB(q,o) \leq \text{dist}(q,o)$$

holds. A lower bounding filter can be used to prune true drops. The basic idea of the proposed method in [9] is to iteratively generate candidates sorted by ascending lower bounding filter distances to the query object q. For that purpose, a ranking [10] of the database objects w.r.t. their filter distances to q is used. The multistep kNN query processing proposed in [9] is initialized with the first k objects from the ranking sequence having the k smallest filter distances. These objects are refined, i.e. their exact distances to q are computed, and are inserted into the current result set (sorted by ascending exact distances to q), representing the kNN of q w.r.t. the already refined objects. A so-called stop distance d_{max} is initialized as the distance of q to the k^{th} object in the current result set representing the kNN-distance of q w.r.t. the already refined objects. Now, an iteration starts that, in each step, performs the following: First, the next object c from the ranking sequence is fetched. If this object has a lower bounding distance estimation to q larger than the stop distance, i.e. $LB(q,c) > d_{max}$, the iteration stops. Otherwise, c is refined, i.e. the exact distance $\text{dist}(q,c)$ is computed, and, if necessary, c is added to the current result set and the stop distance d_{max} is adjusted. When the iteration stops, the current result set contains the kNN of q. The pseudo code of this algorithm is depicted in Figure 2.

In [9], the optimality w.r.t. the number of refined objects necessary for multistep kNN query processing is evaluated and formalized by the concept of R-optimality. An algorithm is defined to be R-optimal, if it produces no more



(a) only with lower-bounding filter distance.



(b) with lower- and upper-bounding filter distances.

Fig. 3. k-nearest neighbor candidates (k=8)

candidates for refinement than necessary. It is shown that a multi-step kNN algorithm is correct and R-optimal iff it exactly retrieves the candidate set $\{o|LB(o, q) \leq \text{nn}_k{-}\text{dist}(q, \mathcal{D})\}$ from the filter step.

3 Generalizing the Definition of Optimality

As indicated above, the algorithm presented in [9] uses only a lower bounding distance estimation in the filter step. However, it is in general sensible to use additional information, in particular an upper bounding filter distance. An upper bounding filter distance estimation UB is always greater or equal to the exact distance, i.e. for any query object q the following *upper bounding property* holds:

$$\forall o \in \mathcal{D} : UB(q, o) \geq \text{dist}(q, o).$$

Using also an upper bounding filter distance yields several important advantages. First, beside pruning true drops with the lower bound we can additionally identify true hits using the upper bounding filter distance. This is illustrated in Figure 3. It depicts for a given query object q the exact distances $\text{dist}(q, o)$ for twelve sample objects $o_1, .., o_{12}$ ($k = 8$). We can distinguish two cases of correct candidate sets returned from the filter-step depending on the distance approximations used: Figure 3(a) shows the case where only a lower bounding filter distance LB is given in the filter-step and Figure 3(b) shows the case where we are given both a lower bounding LB and an upper bounding UB filter distance (illustrated by the bars). In both cases, we marked those objects which have to be returned as candidates from the filter-step. In the first case (cf. Figure 3(a)), we have to refine all objects $o \in \mathcal{D}$ for which the lower bounding distance $LB(q, o)$ is smaller than or equal to the kNN-distance of q, i.e. we have to refine the ten objects $o_1, ..., o_{10}$. In fact, this does not hold for the second case (cf. Figure 3(b)), where the objects o_1, o_2, o_4, o_5 and o_6 can immediately be reported as true hits in the filter step due to the upper bounding distance information. Thus, in contrast to Case 1, the objects o_1, o_2, o_4, o_5 and o_6 need not to be refined.

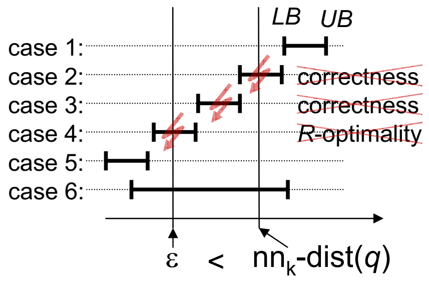
A second advantage of using also an upper bounding filter distance is that the storage requirements of the kNN algorithm can be significantly reduced. As discussed above, [9] uses a ranking algorithm (e.g. [10]). Such a ranking algorithm is usually based on a priority queue. For kNN queries, we can delete true drops (identified using LB) from that queue. Analogously, we can also delete the true hits (identified using UB) from the queue. Thus, the storage cost during query execution are reduced. We will see in our experiments, that using both an upper and a lower bounding filter distance significantly decreases the size of the priority queue (used for producing the ranking sequence) compared to algorithms that use only a lower bound.

Last but not least, a third advantage of using not only a lower bound but also an upper bound in the filter step is the fact that those true hits, identified already in the filter step, can be immediately reported to the user. Thus, the user may receive a part of the complete result directly after the filter step before the query process is completely finished, sometimes even before the exact evaluation of the query predicate for any object has been carried out. The produced hits in the filter step allow the user to inspect the first results very early which is obviously a big advantage in real applications. Unfortunately, none of the existing multi-step query processors provide this feature because none of these methods use suitable distance estimations in the filter step.

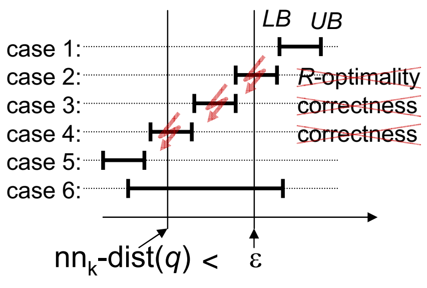
The first obvious question following from these considerations is whether the algorithm proposed in [9] is really R-optimal. We will see that the answer to this question is "yes" *and* "no" – and in fact depends on the type of information (only lower bound or upper and lower bound) available in the filter step. In the traditional sense, a multi-step kNN algorithm is called *R-optimal* if it does not produce more candidates in the filter-step than necessary. As discussed above, the number of candidates that definitely need to be refined depends on the distance approximation available in the filter step. Obviously, it is sensible to define "optimality" in the context of which kind of information I is available in the filter step. In the following, we present the notion of R_I-optimality as a generalization of the traditional R-optimality.

Definition 2 (Generalized Optimality). *Given an information class I defining a set of distance approximations available in the filter step, a multi-step kNN algorithm is called* R_I-optimal *if it does not produce more candidates in the filter-step than necessary.*

Interesting information classes are $I_l = \{LB\}$, i.e. only a lower bounding distance approximation is available in the filter step, and $I_{lu} = \{LB, UB\}$, i.e.



(a)



(b)

Fig. 4. Illustration of the proof of Lemma 1.

both a lower and an upper bounding distance approximation is available in the filter step. In general, R_{I_l}-optimality corresponds to the traditional concept of R-optimality proposed in [9]. The lemma given in [9] identifies those algorithms which are correct and R_{I_l}-optimal. It states that a multi-step kNN algorithm is correct and R_{I_l}-optimal if and only if it exactly retrieves the candidate set $\{o|LB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})\}$ from the filter step. In [9], such an R_{I_l}-optimal algorithm is presented.

For the information class I_{lu} we can also identify the minimum set of candidates that is produced by a correct and $R_{I_{lu}}$-optimal algorithm.

Lemma 1. *A multi-step kNN algorithm is correct and $R_{I_{lu}}$-optimal, iff it refines the candidate set*

$$\{o \in \mathcal{D}|LB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D}) \leq UB(q,o)\} \qquad \textit{(Case 1)}$$

if there are more than k candidates $c \in \mathcal{D}$ with $LB(q,c) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ and, otherwise, it refines the candidate set

$$\{o \in \mathcal{D}|LB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D}) < UB(q,o)\} \qquad \textit{(Case 2)}$$

from the filter step.

Proof. Assume the following algorithm: For an arbitrary query range ε, we obtain the object set $\mathcal{S} = \{o \in \mathcal{D}|LB(q,o) \leq \varepsilon\}$. The objects in $\mathcal{D} - \mathcal{S}$ are pruned as true drops. Then, we retrieve the candidate set $\mathcal{C} = \{o \in \mathcal{S}|\varepsilon < UB(q,o)\} \subseteq \mathcal{S}$ which has to be refined in the refinement step, the remaining objects in $\mathcal{S} - \mathcal{C}$ are immediately reported as hits. We show that this algorithm can only be correct and $R_{I_{lu}}$-optimal if $\varepsilon = \text{nn}_k\text{−dist}(q,\mathcal{D})$.

1. Let $\varepsilon < \text{nn}_k\text{−dist}(q,\mathcal{D})$:
   Then, there may exist an object $o \in \mathcal{D}$ for which the following estimation

chain holds: $\varepsilon < LB(q,o) \leq \text{dist}(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ (cf. Cases 2-3 in Figure 4(a)). The last inequality implies that $o \in NN(q,k)$. However, due to the first inequality of the chain, we have $o \notin \mathcal{S}$, i.e. o will be pruned as a false drop. This contradicts the correctness of the algorithm.
Furthermore, there may exist an object $o \in \mathcal{D}$ for which the following estimation chain holds: $LB(q,o) \leq \varepsilon < UB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ (cf. Case 4 in Figure 4(a)). The first and second inequalities indicate that $o \in \mathcal{C}$, i.e. o is a candidate that will be refined. However, due to the third inequality, it can be definitely decided that $o \in NN(q,k)$ and, thus, a refinement of the distance between q and o is not necessary which contradicts the $R_{I_{lu}}$-optimality.

2. Let $\varepsilon > \text{nn}_k\text{−dist}(q,\mathcal{D})$:
Then, there may exist an object $o \in \mathcal{D}$ for which $\text{nn}_k\text{−dist}(q,\mathcal{D}) < LB(q,o) \leq \varepsilon < UB(q,o)$ (cf. Case 2 in Figure 4(b)), i.e. $o \in \mathcal{C}$ will be refined. However, due to the lower bounding property, $\text{nn}_k\text{−dist}(q,\mathcal{D}) < LB(q,o) \leq \text{dist}(q,o)$ holds. Thus, $o \notin NN(q,k)$, and the algorithm cannot be $R_{I_{lu}}$-optimal.
Furthermore, there may exist an object $o \in \mathcal{D}$ for which $\text{nn}_k\text{−dist}(q,\mathcal{D}) < \text{dist}(q,o) \leq UB(q,o) \leq \varepsilon$ (cf. Cases 3 and 4 in Figure 4(b)). The second and the third inequalities indicate that $LB(q,o) \leq \varepsilon$ and $o \in \mathcal{S} - \mathcal{C}$, i.e. o is reported as hit without refinement. However, from the first inequality it follows that $o \notin NN(q,k)$ and, thus, the algorithm cannot be correct.

Thus, only $\varepsilon = \text{nn}_k\text{−dist}(q,\mathcal{D})$ achieves correctness and $R_{I_{lu}}$-optimality does not lead to any contradiction.

Obviously, all objects in the set $\{o \in \mathcal{D} | LB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D}) < UB(q,o)\}$ have to be refined in order to determine, whether they fulfill the query predicate or not (Case 2). All objects $o \in \mathcal{D}$ for which $UB(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ holds, need not be refined, because $\text{dist}(q,o) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ due to the upper bounding property. However, if the number of candidates c with $LB(q,c) \leq \text{nn}_k\text{−dist}(q,\mathcal{D})$ exceeds k (Case 1), we cannot decide whether those objects c for which $UB(q,o) = \text{nn}_k\text{−dist}(q,\mathcal{D})$ holds, are hits or drops. The reason for this is the following: no algorithm can anticipate the real $\text{nn}_k\text{−dist}(q,\mathcal{D})$ and, thus, other candidates c' could have a smaller $\text{dist}(q,c')$. If so, we would have $\text{nn}_k\text{−dist}(q,\mathcal{D}) < UB(q,c)$ and, thus, c would be a true drop. To make this decision in a correct way, c needs to be refined although $UB(q,c) \geq \text{nn}_k\text{−dist}(q,\mathcal{D})$. In tie situations, there may be more such objects c that need to be refined although $\text{nn}_k\text{−dist}(q,\mathcal{D}) \geq UB(q,c)$. □

At first glance, Case 1 of Lemma 1 may appear to be rather arbitrary. However, as discussed in the proof of Lemma 1, there may be some situations where we need to consider both cases. Let o_i be the kNN of a query object q such that $UB(q,o_i) = \text{dist}(q,o_i) = \text{nn}_k\text{−dist}(q,\mathcal{D})$. Let the object o_j be a candidate with $LB(q,o_j) \leq UB(q,o_i) = \text{nn}_k\text{−dist}(q,\mathcal{D})$. Then, o_j cannot be pruned before the exact kNN-distance has been computed. In addition, if we have a tie situation, e.g. $\text{dist}(q,o_j) = \text{nn}_k\text{−dist}(q,\mathcal{D}) = \text{dist}(q,o_i)$, the kNN set of q cannot be determined correctly without the refinement of the object o_i (contradicting Case 2). The reason for this is that we cannot evaluate the query predicate for o_j correctly even if we refine o_j and compute $\text{dist}(q,o_j)$. If $\text{dist}(q,o_i) < \text{dist}(q,o_j)$, then

```
algorithm $k$NN(QueryObject $q$, Integer $k$, DBIndex $I$)
  // Step 1: Initialization
  SortedList *result*;
  SortedList *candidates*;
  initialize *ranking* on $I$ w.r.t. lower bounding distance approximation;
  fetch the first $k$ objects from *ranking* and add them to *candidates*;
  $d_{min} = k^{th}$ smallest lower bound of the elements in *candidates*;
  $d_{max} = k^{th}$ smallest upper bound of the elements in *candidates*;
  $d_{f\_next}$ = lower bounding distance of the next element in *ranking*;
  do {
    update $d_{min}$, $d_{max}$, and $d_{f\_next}$;
    // Step 2: Fetch a candidate
    if $d_{min} \geq d_{f\_next}$ then
      fetch next object from *ranking* $\rightarrow$ *candidates*; // only if $d_{max} \geq d_{f\_next}$
      update $d_{min}$, $d_{max}$, and $d_{f\_next}$;
    // Step 3: Identify true hits and true drops by using $d_{min}$ and $d_{max}$
    for all $c \in$ *candidates* do
      if $UB(q, c) < d_{min}$ then add $c$ to *result*;
      if $LB(q, c) > d_{max}$ then prune $c$;
    // Step 4: Refine a candidate
    if $|$*results*$|+|$*candidates*$| > k \vee d_{f\_next} \leq d_{max}$ then
      for all $c \in$ *candidates* with $LB(q, c) \leq d_{min} \wedge d_{max} \leq UB(q, c)$ do
        if $\mathrm{dist}(q, c) \leq \mathrm{nn}_k{-}\mathrm{dist}(q, \mathit{result})$ then add $c$ to *result*;
    else
      add all remaining $c \in$ *candidates* to *result*;
  } while $(d_{f\_next} \leq d_{max} \vee |$*candidates*$| > 0)$
  return *result*;
```

Fig. 5. R_I-Optimal k-NN Algorithm.

$o_j \notin NN(q, k)$, otherwise, if $\mathrm{dist}(q, o_i) = \mathrm{dist}(q, o_j)$, then $o_j \in NN(q, k)$. However, the exact value of $\mathrm{dist}(q, o_i)$ is obviously not known before the refinement of o_i.

From Lemma 1 it follows, that the algorithm proposed in [9] is R_{I_l}-optimal but not $R_{I_{lu}}$-optimal.

4 $R_{I_{lu}}$-optimal Multi-step kNN Search

Based on the above observations, we are able to design an algorithm that is $R_{I_{lu}}$-optimal. The pseudo-code of our algorithm is depicted in Figure 5. The algorithm iteratively reduces the candidate set, where in each iteration it identifies true drops, true hits and/or refines a candidate for which the query predicate cannot be determined without the refinement.

The algorithm starts with the initialization of the incremental ranking on the used index according to the lower-bounding distances of all objects. Then, the first k candidates are fetched from the ranking sequence into the candidate list (Step 1). In order to detect which candidate must be refined, we use two variables d_{min} and d_{max} generating a lower-bounding and an upper-bounding distance estimation of the exact k-NN distance, i.e. $d_{min} \leq \mathrm{nn}_k\mathrm{-dist}(q, \mathcal{D}) \leq d_{max}$. The basic idea of our algorithm is that we can use this restriction of the exact k-NN distance in order to identify those candidates c with $LB(q, c) \leq \mathrm{nn}_k\mathrm{-dist}(q, \mathcal{D}) \leq UB(q, c)$ which must be refined due to Lemma 1. Furthermore, as the stop criterion of the main loop, we initialize the variable d_{f_next} reflecting the lower-bounding distance of the top element of the ranking sequence to q.

In the main loop, we first update the variables d_{min}, d_{max} and d_{f_next} as depicted. Then, we fetch the next candidate o from the ranking sequence into the candidate set *candidates*, if $d_{min} \geq d_{f\_next}$ holds (Step 2). This condition guarantees that we fetch only the next candidate from the ranking query if the variable d_{min} does not guarantee the conservative estimation of the exact k-NN distance any more. This ensures the $R_{I_{lu}}$-optimality of the algorithm and guarantees that our algorithm does not produce unnecessary candidates. Then, the lower-bounding distance estimation of the newly fetched candidate must lie on the new d_{min} value after the update of the d_{min} variable. Hence, the fetch candidate either is a true hit or covers the exact kNN-distance, and thus, must be refined. This guarantees, that our algorithm is optimal w.r.t. the number of fetches from the ranking sequence which in turn is responsible for the optimality according to the number of index accesses. Let us note, that our fetch routine additionally hands over the actual d_{max} value to the ranking query method. This allows us to proceed the exploration of the index only when necessary and to cut the priority queue according to d_{max} in order to decrease the size of the queue. After fetching a new candidate, we have to update the variables d_{min}, d_{max} and d_{f_next} in order to keep the consistency of the used distance estimation variables.

Step 3 of the algorithm identifies the hits and drops according to the d_{min} and d_{max} values. Obviously, all candidates c with $UB(q, c) < d_{min}$ can be returned immediately as hits and all candidates c' with $LB(q, c') > d_{max}$ can be pruned.

Next, if the number of received results plus the remaining number of candidates are greater than k and if the condition $d_{f\_next} \leq d_{max}$ holds, then we refine the next candidate (Step 4). The first condition indicates whether it is still necessary to refine a candidate. The reason for this condition is, that, if the remaining candidates definitely must belong to the query result because there are no concurrent candidates available any more, we can stop the refinement and immediately report the remaining candidates as hits. If both conditions hold, the algorithm refines a candidate c with $LB(q, c) \leq d_{min}$ and $d_{max} \leq UB(q, c)$. As mentioned above, this procedure guarantees the $R_{I_{lu}}$-optimality of this algorithm. We will show later that there must always be a candidate that fulfills the above refinement criterion.

If $d_{f\_next} > d_{max}$, i.e. the top element of the ranking sequence can be pruned as true drop or if there are no more candidates left, the main loop stops.

In the following, we show that our algorithm R_I-Optimal kNN is (1) fetch optimal in the number of fetches from the ranking sequence, (2) correct, and (3)

$R_{I_{lu}}$-optimal. Let us note, that fetch optimal corresponds to a minimal number of disk accesses of the underlying index on which the ranking sequence is computed when using access optimal ranking query algorithms (e.g. [10]).

We start with showing that the variables d_{min} and d_{max} conservatively and progressively approximate the exact kNN-distance $\mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S})$, where $\mathcal{S} \subseteq \mathcal{D}$ is the set of candidates in a particular iteration of the algorithm.

Lemma 2. *Let q be a query object and $\mathcal{S} \subseteq \mathcal{D}$ be the set of candidates in a particular iteration of the algorithm. Then, $d_{min} \leq \mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S}) \leq d_{max}$.*

Proof. d_{min} is the k^{th} lower-bounding distance of objects from $\mathcal{S}$ to q and d_{max} is the k^{th} upper-bounding distance of objects from $\mathcal{S}$ to q.

First, we show that $d_{min} \leq \mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S})$. We know that there are at least k objects $o \in \mathcal{S}$ with $\mathrm{dist}(q, o) \leq \mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S})$. Consequently, there must be at least k objects $o \in \mathcal{S}$ with $LB(q, o) \leq \mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S})$, and thus, $d_{min} \leq \mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S})$.

The second property $\mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S}) \leq d_{max}$ can be shown in a similar way. We know that there are at least k objects $o \in \mathcal{S}$ with $UB(q, o) \leq d_{max}$. Consequently, there must be at least k objects $o \in \mathcal{S}$ with $\mathrm{dist}(q, o) \leq d_{max}$, and thus, $\mathrm{nn}_k\!-\!\mathrm{dist}(q, \mathcal{S}) \leq d_{max}$. $\square$

Fetch-optimality. In order to verify that our novel algorithm is fetch optimal, we have to show that the lower-bounding distance estimation LB of the newly fetched candidate in Step 2 is equal to the new d_{min} value after the update of the d_{min} variable. d_{min} corresponds to the k^{th}-smallest lower-bounding distance of the candidates which are already fetched from the ranking sequence. Let c denote the already fetched candidate for which $LB(q, c) = d_{min}$ actually holds. We only fetch the next candidate c' if $LB(q, c') \leq d_{min}$. Then, either $LB(q, c') = d_{min}$ which trivially fulfills the criterion, or $LB(q, c') < d_{min}$. In the last case, $LB(q, c)$ would not be the k^{th}-smallest lower-bounding distance estimation anymore, because c' is an additional already fetched candidate with $LB(q, c') < LB(q, c)$. Hence, d_{min} has to be set to $LB(q, c')$. Consequently, as mentioned above, the fetched candidate c' either is a true hit or certainly covers the kNN-distance and, thus, must be refined.

Correctness. Due to Lemma 2, the candidates c with $UB(q, c) \leq d_{min}$ can safely be reported as hits because $\mathrm{dist}(q, c) \leq UB(q, c) \leq d_{min}$. Similarly, candidates c with $LB(q, c) > d_{max}$ can be safely pruned, since $\mathrm{dist}(q, c) \geq LB(q, c) \geq d_{max}$. In summary, our algorithm is correct, i.e. does not produce false hits or false drops.

$R_{I_{lu}}$-optimality. We can prove that our algorithm is $R_{I_{lu}}$-optimal by showing that we only refine candidates whose lower- and upper-bounding filter distances cover the exact kNN-distance. In fact, we only refine those candidates c with $LB(q, c) \leq d_{min}$ and $d_{max} \geq UB(q, c)$. Thus, according to Lemma 2, the R_I-optimality is guaranteed. However, this works only if in Step 4 of the algorithm there exists at least one candidate c with $LB(q, c) \leq d_{min}$ and $d_{max} \leq UB(q, c)$.

Table 1. Summary of real-world test datasets.

Dataset	description	# objects	distance	ratio of the cost of filter vs. refinement
San Joaquin	road network	18,263 nodes	Dijkstra	1/300
Protein	protein graph	1,128 proteins	graph kernel	1/2,000
Plane	voxelized 3D CAD	35,950 voxel	Euclidean	1
Timeseries	audio timeseries	2400 clips	DTW	1/150

Lemma 3. *Let q be the query object and $\mathcal{S} \subseteq \mathcal{D}$ be a set of candidates for which the lower-bounding and upper-bounding distance estimations ($LB(q,c)$ and $UB(q,c)$ for all $c \in \mathcal{S}$) are known. Furthermore, let d_{min} denote the k^{th}-smallest lower-bounding distance estimation and d_{max} denote the k^{th}-smallest upper-bounding distance estimation in $\mathcal{S}$. Then, the following statement holds:*

$$\exists o \in \mathcal{S} : LB(q,o) \leq d_{min} \leq d_{max} \leq UB(q,o).$$

Proof. Obviously, there must exist at least one candidate $o \in \mathcal{S}$ with $d_{min} = LB(q,o)$ and at least one candidate $p \in \mathcal{S}$ with $d_{max} = UB(q,p)$. Let us assume, that the statement in Lemma 3 does not hold, then for all candidates $c \in \mathcal{S}$ it holds that $LB(q,c) > LB(q,o) \vee UB(q,c) < UB(q,p)$, i.e. this also holds for o and p. Thus, if $LB(q,o) < LB(q,p)$ and $UB(q,o) < UB(q,p)$ it follows that $o \neq p$.

As a consequence, for $k = 1$, we have $d_{min} = LB(q,o)$ and $d_{max} = UB(q,o) \neq UB(q,p)$ which contradicts the assumption about d_{max}.

Analogously, for $k > 1$, there must be at least k candidates $c \in \mathcal{S}$ with $LB(q,c) \leq LB(q,o)$ and there must be at most $k-1$ candidates $c \in \mathcal{S}$ with $UB(q,c) < LB(q,p)$. Consequently there must be at least one candidate $c \in \mathcal{S}$ with $LB(q,c) \leq LB(q,o)$ and $UB(q,c) \geq LB(q,p)$, which contradicts our above assumption. $\square$

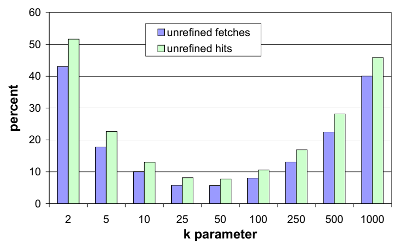
In summary, assuming that a lower- and upper-bounding filter distance is available for each processed object, our novel multi-step kNN algorithm is correct, requires the minimal number of index page accesses and is optimal w.r.t. the number of refinements required to answer the query.

5 Experimental Evaluation

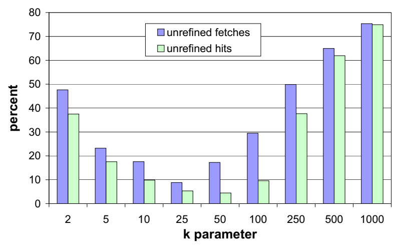
We conducted our experiments on Windows workstations with a 32-bit 3.2 GHz CPU and 4 GB main memory. All evaluated methods were implemented in Java.

5.1 Setup

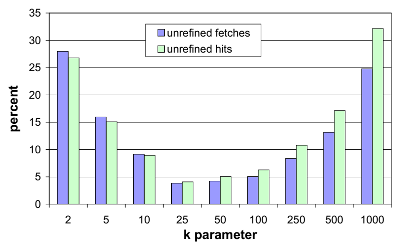
Our experimental testbed contains four real-world datasets with different characteristics summarized in Table 1. We applied a special form of Lipschitz embedding [11] for the first three datasets using randomly chosen singleton reference



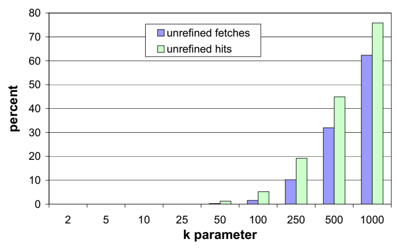
(a) San Joaquin



(b) Protein



(c) Plane



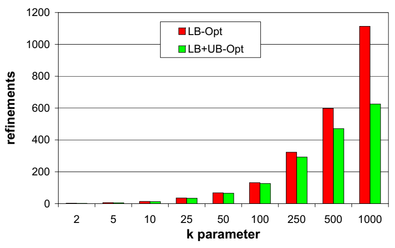
(d) Timeseries

Fig. 6. Relative number of unrefined candidates.

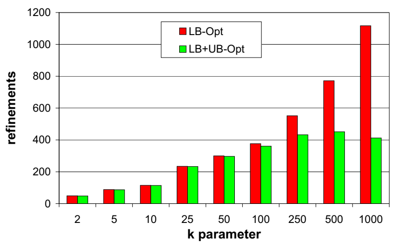
sets in order to derive upper and lower bounds. For the timeseries dataset, we generated lower- and upper-bounding distance approximations for the Dynamic Time Warping (DTW) distance as described in [12] where we set the size of the Sakoe-Chiba band width to 10%. Let us note, that for many applications there may exist filter distance measures that yield an even better pruning power in the filter step. We processed 50 randomly selected kNN queries for the particular dataset and averaged the results.

5.2 $R_{I_{lu}}$-Optimality vs. R_{I_l}-Optimality

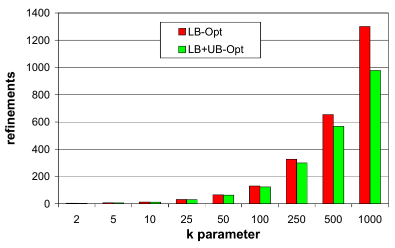
In the first experiment we demonstrate the superiority of our novel $R_{I_{lu}}$-optimal algorithm based on lower- and upper-bounding distance estimations over the traditional R_{I_l}-optimal algorithm that uses only lower-bounding distance estimations in the filter step. Figure 6 shows the results of multi-step kNN queries on our four datasets for different settings of the k parameter. In particular, the number of hits and the number of fetches (in percent) that both need no refinement are depicted. Note, that an R_{I_l}-optimal algorithm has to refine all fetched candidates, and thus, produces zero unrefined candidates. The results show that, due to the use of an upper-bounding filter distance, a significant amount of the



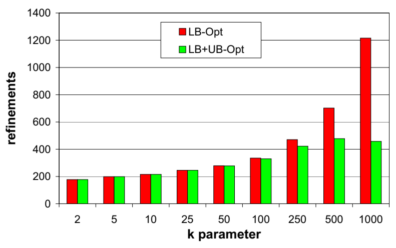
(a) San Joaquin



(b) Protein



(c) Plane

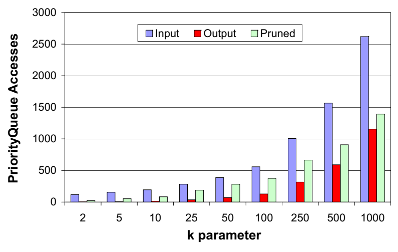


(d) Timeseries

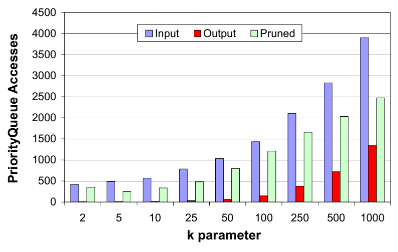
Fig. 7. Absolute number of needed refinements of R_{I_l} and $R_{I_{lu}}$ approach.

hits does not need to be refined. If we consider that the filter step is 150 (time series), 300 (road network), and 2,000 (proteins) times faster than the refinement step, the runtime improvement is drastic. For three datasets we observe that the amount of unrefined hits and fetches first decreases with increasing k and then again increases. This is due to the characteristics of the datasets and the used Lipschitz embedding: When increasing k, the distances first increase very quickly, then stabilize at some point and finally increase again very quickly when k converges to the number of objects in the dataset. As a consequence, for very low and very high values of k, the distance approximations produce rather selective stop criteria, whereas for medium values of k, the pruning power decreases.

Figure 7 compares the $R_{I_{lu}}$-optimal algorithm and an R_{I_l}-optimal algorithm according to the absolute number of refinement operations. The refinement reduction using the upper-bounding filter distance was clearly improved. In particular, for high k settings we have to refine only about half of the objects in comparison to competing techniques using only the lower bound filter. For all four datasets, we can also observe that the approximation qualities of the upper bound and the lower bound are rather similar.



(a) San Joaquin



(b) Plane

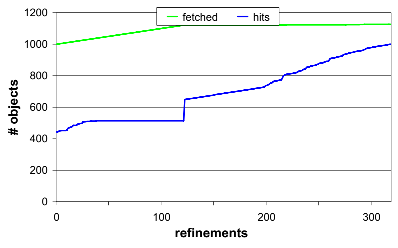
Fig. 8. Pruning of the priority queue by means of d_{max}.

5.3 Size of the Priority Queue of the Ranking Query

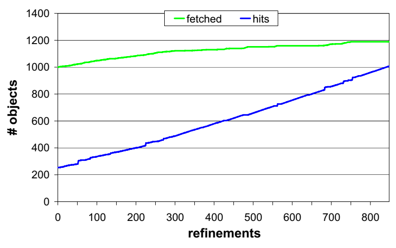
In the next experiment, we examine the influence of the d_{max} value on the size of the priority queue of the ranking query. Figure 8 depicts the size of *Input*, *Output* and *Pruned*. *Input* denotes the objects that are inserted into the priority queue while traversing the index. *Output* denotes the objects removed from the priority queue while fetching the next candidate. *Pruned* denotes the objects in the priority queue that can be pruned according to the d_{max} value during the execution of our novel algorithm. It can be observed that we achieve a significant reduction of the priority queue. On the average, we can save more than 50% of memory space when pruning the queue using d_{max}.

5.4 Early Output of Result Tuples

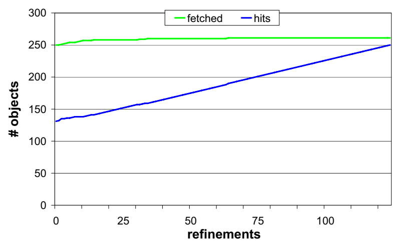
As mentioned in Section 4, the proposed upper bound filter allows the early output of some true hits even before refinement. Our last experiment evaluates the capability of early outputs for queries with $k = 1000$ and $k = 250$. Figure 9 depicts the number of fetches and true hits detected by our $R_{I_{lu}}$-optimal algorithm against the number of refinements. The number of refinements corresponds to the number of iterations in the main loop of our algorithm (cf. Figure 5). It can be observed that already about 45% of the results of the *Protein* dataset can be reported before starting the refinement of the first object. On that dataset this corresponds to a speed-up of approximately 2,000 for each of these objects. After the 25^{th} refinement, we have reported 500 of 1000 results. Similar results can be seen for the experiments on the *San Joaquin* and *Plane* datasets. In summary, a significant portion of the result could be reported very early. Very few refinements are sufficient in order to report more than half of the entire results. Note, that the traditional R_{I_l}-optimal algorithm proposed in [9] could be adapted such that it would generate the first results before the need to refine the first k candidates. However, it would be impossible to report more results than



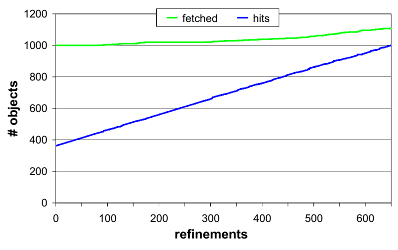
(a) Protein (k=1000)



(b) Plane (k=1000)



(c) San Joaquin (k=250)



(d) San Joaquin (k=1000)

Fig. 9. Number of reported results against the number of query iterations.

there are refined candidates. As we can see in the experiments, our algorithm reports significantly more results than required refinements. Thus, the user can already evaluate the results before the query execution is finished. In an application scenario where the first results are already sufficient for the user, e.g. a doctor wants to confirm his diagnosis drawn from an X-ray image by comparing the actual image to some of the k most similar images in his database, our algorithm would yield a very high performance gain as very few refinements would be necessary before the user stops the query execution procedure after getting enough intuition about the result.

6 Conclusions

In this paper, we generalized the traditional notion of R-optimality in order to capture the optimality of multi-step kNN query processing using both lower and upper bounding filter distances. We proposed a novel kNN multi-step query algorithm and showed that this algorithm is R-optimal in the generalized sense, correct and fetch-optimal, i.e. requires a minimum number of fetch operations on the underlying ranking algorithm. In our experiments, we demonstrated the superiority of our novel query processing algorithm in comparison to state-of-the-art competitors. In particular, we showed that our approach drastically reduces the number of refinement operations and, thus, the query execution time since the refinement is usually three orders of magnitude slower than the filter step. Our approach features a considerably decreased storage requirement compared to existing solutions and can be used to report first hits as early as possible even before any object has been refined.

References

1. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD. (1984) 47–57
2. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: Proc. SIGMOD. (1990) 322–331
3. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: an efficient access method for similarity search in metric spaces. In: Proc. VLDB. (1997)
4. Orenstein, J., Manola, F.: Probe spatial data modelling and query processing in an image database application. IEEE Trans. on Software Engineering **14**(5) (1988) 611–629
5. Brinkhoff, T., Horn, H., Kriegel, H.P., Schneider, R.: A storage and access architecture for efficient query processing in spatial database systems. In: Proc. SSD. (1993)
6. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: Proc. FODO. (1993)
7. Faloutsos, C., adn Y. Manolopoulos, M.R.: Fast subsequence matching in time series database. In: Proc. SIGMOD. (1994)
8. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z.: Fast nearest neighbor search in medical image databases. In: Proc. VLDB. (1996)
9. Seidl, T., Kriegel, H.P.: Optimal multi-step k-nearest neighbor search. In: Proc. SIGMOD. (1998)
10. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: Proc. SSD. (1995)
11. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann (2006)
12. Keogh, E.: Exact indexing of dynamic time warping. In: Proc. VLDB. (2002)