Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle, Washington, June 1998.

Optimal Multi-Step *k*-Nearest Neighbor Search

Thomas Seidl
University of Munich, Germany
Institute for Computer Science
http://www.dbs.informatik.uni-muenchen.de
seidl@dbs.informatik.uni-muenchen.de

Hans-Peter Kriegel
University of Munich, Germany
Institute for Computer Science
http://www.dbs.informatik.uni-muenchen.de
kriegel@dbs.informatik.uni-muenchen.de

Abstract

For an increasing number of modern database applications, efficient support of similarity search becomes an important task. Along with the complexity of the objects such as images, molecules and mechanical parts, also the complexity of the similarity models increases more and more. Whereas algorithms that are directly based on indexes work well for simple medium-dimensional similarity distance functions, they do not meet the efficiency requirements of complex high-dimensional and adaptable distance functions. The use of a multi-step query processing strategy is recommended in these cases, and our investigations substantiate that the number of candidates which are produced in the filter step and exactly evaluated in the refinement step is a fundamental efficiency parameter. After revealing the strong performance shortcomings of the state-of-the-art algorithm for *k*-nearest neighbor search [Korn et al. 1996], we present a novel multi-step algorithm which is guaranteed to produce the minimum number of candidates. Experimental evaluations demonstrate the significant performance gain over the previous solution, and we observed average improvement factors of up to 120 for the number of candidates and up to 48 for the total runtime.

1 INTRODUCTION

More and more applications of database systems require the efficient support of similarity search. Examples include molecular biology [BMH 92], medical imaging [Kor+ 96], CAD/CAM systems [BK 97], and multimedia databases [Fal+ 94] [Haf+ 95] [SK 97] among many others [Jag 91] [AFS 93] [GM 93] [FRM 94] [ALSS 95]. In all of these approaches, similarity is defined in terms of a more or less complex similarity distance function. The smaller the similarity distance value, the more similar are two objects. Typical query types are the similarity range query which is specified by a query object and a similarity distance range $[0, \varepsilon]$, and the *k*-nearest neighbor query which is specified by a query object and a number *k* for the *k* most similar objects to be retrieved.

Whereas single-step algorithms for similarity search already meet the requirements of very large databases, these solutions suffer from the increasing complexity of the objects and of the similarity distance functions. For classic spatial queries such as point queries and region queries, multi-step algorithms have been developed to efficiently support complex objects [OM 88] [BHKS 93]. The paradigm of multi-step query processing has already been extended to complex similarity search, and available algorithms aim at similarity range queries [AFS 93] [FRM 94] and *k*-nearest neighbor queries [Kor+ 96]. However, we observed a bad performance of the latter solution in our experiments on large image and biomolecular databases. Starting from a theoretical analysis of the situation, we develop a novel, optimal multi-step algorithm for *k*-nearest neighbor search that implies a minimum number of exact object distance evaluations.

The paper is organized as follows: In the remainder of this introduction, we specify our problem of complex similarity search. Section 2 is dedicated to algorithms for similarity search and incremental similarity ranking that directly work on index structures in a way they are employed by our new method. In section 3, we present the available multi-step algorithm for *k*-nearest neighbor search of [Kor+ 96] including the significant efficiency shortcomings of the solution. Experiments substantiate that the number of candidates is a fundamental efficiency parameter. We present our novel algorithm in section 4 along with a proof that it exactly generates the minimum number of candidates. The experimental evaluation in section 5 demonstrates the substantial performance improvement before the paper is concluded in section 6.

1.1 Simple and Complex Similarity Distance Functions

Suppose the simple case that the objects of interest are represented by low- or medium-dimensional feature vectors. Then, the similarity distance of two objects is typically defined by an appropriate distance function of the points in the feature space such as the Euclidean distance, for instance. Examples include the section coding approach [BK 97], angular profiles [BMH 92], and 2-D contour features [GM 93]. For such pure feature-based similarity models, single-step system architectures are appropriate: While managing the feature points by a multidimensional access method, query processing is performed by one of the *k*-nearest neighbor search algorithms that are available from the literature (cf. section 2).

Some of the algorithms for similarity search are restricted to similarity models which are completely defined by a distance function of low- or medium-dimensional feature vectors. In practice, however, complex similarity distance functions occur that may not be represented by a simple feature vector distance or that are too high in their dimensionality as they could be efficiently managed by multidimensional index structures. Examples include the Max-Morphological Distance [Kor+ 96], the approximation-based similarity of 3-D surface segments [KSS 97] [KS 98], the error volume of 2-D shapes and of 3-D solids, high-dimensional color histogram similarity [Haf+ 95] [SK 97], etc. If there is no technique available to simplify the complex similarity distance function, query processing has to be performed by a linear scan of the entire database, and the performance obviously suffers a lot. In particular, quadratic form distance functions as they are successfully used for color histograms [Haf+ 95] [SK 97] or shape histograms [Sei 97] [AKS 98] require an evaluation time that is quadratic in the number of dimensions. Figure 1 demonstrates this effect for various dimensions that occur in our example databases. Furthermore, the max-morphological distance of two images of 128×128 pixels is reported to require 12.69 seconds on the average [Kor+ 96].

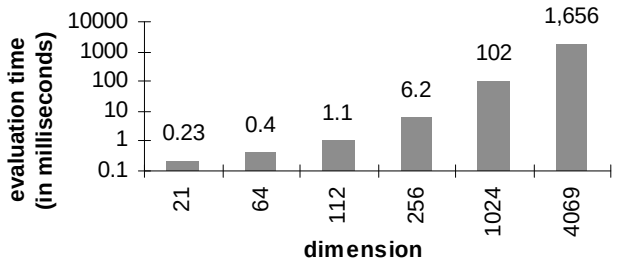

Figure 1. Evaluation time of quadratic form distance functions for various dimensions

A competing approach to avoid multi-step query processing is to use indexing methods for metric spaces that require nothing else than the object distance function. Examples include the M-tree [CPZ 97], SS-tree [WJ 96], FastMap [FL 95] etc. However, these solutions are fixed to distance functions that are available to the system in advance. In particular, they do not support adaptable similarity distance functions which may be interactively adapted to individual user preferences at query time [SK 97] [Sei 97].

1.2 Lower-Bounding Filter Distance Functions

For similarity search in presence of complex high-dimensional or even user-adaptable similarity distance functions, multi-step algorithms are available [FRM 94] [Kor+ 96]. The basic principle of these methods is to employ feature distance functions (also called filter distance functions) that serve as approximations of the complex object distance functions.

An abstraction from the applications leads to the following formalism which represents the essential principle of feature or filter distance functions: By O, let us denote the universe of objects for which the object similarity distance function $d_o\colon O \times O \to \Re_0^+$ is defined. A feature transformation $F\colon O \to \Re^n$ maps every object $o \in O$ onto an n-dimensional feature vector $F(o) \in \Re^n$. The distances in the feature space $\Re^n$ are measured by a feature distance function $d_f\colon \Re^n \times \Re^n \to \Re_0^+$. For notational simplicity, we join the functions d_f and F in order to provide $d_f\colon O \times O \to \Re_0^+$, $d_f(o_1, o_2) = d_f(F(o_1), F(o_2))$ as an abbreviated notation. For similarity search, the feature vectors are typically managed by multidimensional access methods that support efficient *k*-nearest neighbor query processing with respect to the feature distance d_f.

In a multi-step query processing environment, one or more filter steps produce sets of candidates that are exactly evaluated in one or more subsequent refinement steps. The crucial correctness requirement is to prevent the system from producing false drops. This means that no actual result may be dismissed from the set of candidates. For classic spatial query types such as point queries and region queries, the use of conservative approximations in the filter step ensures the correctness of the algorithms [OM 88] [BHKS 93]. For multi-step similarity search, an analogous criterion is available, the *lower-bounding property* of filter and object distance functions:

Definition 1 (Lower-bounding property). A feature distance function d_f and an object distance function d_o fulfill the *lower-bounding property* if d_f underestimates d_o in any case, i.e. for all objects $o_1, o_2 \in O$:

$$d_f(o_1, o_2) \leq d_o(o_1, o_2)$$

Algorithms that obey this principle have been developed for similarity range queries [FRM 94] as well as for *k*-nearest neighbor queries [Kor+ 96].

The following examples provide an illustration of the wide variety and potential of lower-bounding feature distance functions:

Subsequence Matching. A lower-bounding feature distance function for truncated feature vectors is employed. The feature vectors are obtained from the Discrete Fourier Transform of the original sequence objects [FRM 94].

Max-Morphological Distance. The max-morphological distance of 2-D shapes is lower-bounded by the max-granulometric distance of the corresponding pattern spectra of the 2-D shapes. The low-dimensional feature vectors are obtained by operations from the mathematical morphology [Kor+ 96].

Approximation-based 3-D Similarity. Derive a distance function on key-vectors that underestimates the original complex similarity function for 3-D surface segments [KSS 97].

Reduction of Dimensionality. Project high-dimensional feature vectors to low-dimensional feature vectors, and derive a low-dimensional distance function that lower bounds the high-dimensional distance function. This approach is trivial for the Euclidean distance but also works for adaptable quadratic form distance functions [SK 97].

The principle of lower-bounding feature distance functions is quite general and may be applied to several other complex similarity distance functions.

1.3 *k*-Nearest Neighbor Search

Along with similarity range queries, *k*-nearest neighbor queries are an important query type of similarity search. Similarity range queries are specified by a query object q and a range parameter ε. The result set is defined to be $sim_q(\varepsilon) = \{o \in \mathrm{DB} \mid d(o, q) \leq \varepsilon\}$. For k-nearest neighbor queries, the query object q and a query parameter k have to be provided that specify the retrieval of the k objects from the database that are most similar to q. Conceptual problems occur if several database objects share the same k-th distance value. In this case, most of the available implementations nondeterministically report any k out of the first (more than k) relevant objects. For conceptual reasons, we prefer the definition that any object that is as close or closer to q than any k-th object belongs to the set $NN_q(k)$ of the k nearest neighbors of q. The formal definition is as follows:

Definition 2 (*k*-nearest neighbor query). For a query object $q \in O$ and a query parameter k, the *k-nearest neighbor query* returns the smallest set $NN_q(k) \subseteq \mathrm{DB}$ that contains (at least) k objects from the database, and for which the following condition holds:

$$\forall o \in NN_q(k), \forall o' \in \mathrm{DB} - NN_q(k): d(o, q) < d(o', q)$$

Then, the k-nearest neighbor query is equivalent to a corresponding similarity range query, i.e. for the k-th distance value $\varepsilon_k = \max\{d(o, q) \mid o \in NN_q(k)\}$, both queries return the same result set:

$$NN_q(k) = sim_q(\varepsilon_k)$$

Whereas for now, this connection illustrates our notion of k-nearest neighbor sets, we will later exploit an analogous equivalence on the level of candidate sets and filter distance functions for the correctness and efficiency proof of our new algorithm.

2 SINGLE-STEP *K*-NEAREST NEIGHBOR SEARCH

The multi-step k-nearest neighbor algorithms which we investigate in this paper are based on single-step methods that directly work on multidimensional index structures. We sketch some competing methods and focus on incremental similarity ranking which is required for our optimal multi-step algorithm.

2.1 Directly Index-Based Algorithms

In order to efficiently process k-nearest neighbor queries by directly using multidimensional index structures, several approaches are available from the literature. The proposals include cell-based approaches for nearest neighbor search which are conceptually based on Voronoi cells [PS 93] [AMN 95] [Ber+ 98], branch and bound algorithms for k-nearest neighbor search [FBF 77] [RP 92] [RKV 95], and incremental algorithms for similarity ranking [Hen 94] [HS 95]. Recently, a fast parallel method has been suggested [Ber+ 97]. Also theoretical results have been published concerning the efficiency of nearest neighbor search in high-dimensional spaces. The performance of methods that use *mindist* and *minmaxdist* functions on R-trees was investigated [PM 97], and general cost models have been developed [Spr 91] [BBKK 97]. An important observation is that the use of the *mindist* function guarantees the optimality of the algorithms. The *minmaxdist* function may help to improve the performance of k-nearest neighbor queries for a given k but is of no use for the more general case of incremental ranking.

Most of the available algorithms are tuned to efficiently support k-nearest neighbor queries for a fixed retrieval parameter k. The obvious disadvantage of these methods is that the number k of desired answers has to be specified in advance. If the k results are exhausted but the user is not satisfied with the retrieved objects, there is no chance to obtain a single or several next nearest neighbors without restarting the query from the beginning for a higher k. This problem does not only occur in interactive environments but also in the context of our optimal multi-step algorithm as we will see later: The required number of candidates which will be retrieved from the index cannot be estimated in advance. An approach to overcome this problem is to employ methods for incremental similarity ranking.

2.2 Incremental Similarity Ranking

Incremental similarity ranking is a similarity query type that corresponds to a *give-me-more* facility. After an initialization, the ranked objects may be retrieved by a sequence of *getnext* calls. Formally, performing an incremental similarity ranking means the partial materialization of a q-ranking which may be defined as follows [Sei 97]:

Definition 3 (q-ranking). Given a query object $q \in O$ and a database DB $\subseteq O$ containing $N = |\text{DB}|$ objects, a q-ranking of the database DB is a bijection $ranked_q: \Im_N \rightarrow \text{DB}$ that maps the index set $\Im_N = [1 \ldots N]$ d_q-monotonously onto the database DB, i.e. ascendingly ordered by the distance of the objects to the query object q.

We simplify our notation by writing $ranked_q(i) = o_i$ for the object o_i that is ranked to position i and denote the image of the index set $\Im_k$ by $ranked_q(\Im_k) = \{o_1, \ldots, o_k\}$. Using this abbreviation, the d_q-monotony appears as follows:

$$\forall i, j \in \Im_N: i < j \Rightarrow d(o_i, q) \leq d(o_j, q)$$

When processing incremental similarity ranking queries, the object $o_k = ranked_q(k)$ is reported as response to the k-th *getnext* call. Note that the q-ranking is not totally determined if some objects share the same distance to the query object q.

Let us present an algorithm for incremental similarity ranking which is proven to be optimal with respect to the number of accessed index pages [BBKK 97]. The algorithm was introduced in the context of 2-D geographic information systems and works on PMR quadtrees [HS 95]. In figure 2, we present an adapted version that aims at hierarchical multidimensional access methods [GG 97] and does no longer regard the clipping behavior of PMR quadtrees. For our experiments, we use the X-tree which has been shown to efficiently support dimensions up to 20 [BKK 96].

Note that the actual distance of the query object to the box of the root node of the multidimensional index is not required. Thus, we save the distance evaluation for the root node and insert the root with the distance 0 without affecting the correctness of the procedure.

```
method RTree :: ranking (Object query)
1  PriorityQueue queue;
2  queue.insert (0, root);
3  wait (getnext_is_called);
4  while not queue.isempty() do
5    Element first = queue.pop();
6    case first isa
7      DirNode:
8        foreach child in first do
9          queue.insert (mindist (query, child.box), child);
10     DataNode:
11       foreach object in first do
12         queue.insert (distance (query, object), object);
13     Object:
14       report (first);
15       wait (getnext_is_called);
16   end
17 enddo
```

Figure 2. Incremental ranking query processing on R-trees (adapted from [HS 95])

3 MULTI-STEP *K*-NEAREST NEIGHBOR SEARCH

As already mentioned, a multi-step algorithm for k-nearest neighbor search has already been developed and successfully been applied to similarity search in 3-D medical image databases [Kor+ 96]. After presenting the available solution, we demonstrate its inherent efficiency shortcomings.

3.1 State-of-the-Art Algorithm

In figure 3, we present an adapted version of the multi-step algorithm for k-nearest neighbor search of [Kor+ 96]. The query object is denoted by q, and the parameter k specifies the requested number of neighbors. The basic structure of the algorithm is that it proceeds in two stages: In the first stage, a ***k*-nearest neighbor search** on the index is performed returning the k closest objects with respect to the filter distance function. For these k objects, the maximum $d_{\max}$ of the exact object distances is determined. In the second stage, a **range query** on the index is performed returning all objects that have a filter distance of at most $d_{\max}$. For all of these candidates, the exact object distance is evaluated, and the k closest objects are reported. Figure 4 schematically illustrates the architecture of the algorithm including the communication of the two stages with the index and the object server.

***k*-NearestNeighborSearch (*q*, *k*)** // previous algorithm	
First Stage	1 *Primary Candidates*: Perform a ***k*-nearest neighbor search** on the index around $F(q)$ respecting the filter distance function d_f 2 *Range Determination*: For the primary candidates o, determine $d_{\max} = \max\{d_o(o, q)\}$
Second Stage	3 *Final Candidates*: Perform a **range query** on the index to obtain $\{o \in \text{DB}: d_f(F(o), F(q)) \leq d_{\max}\}$ 4 *Final Result*: Sort the final candidates o according to $d_o(o, q)$, and report the top k objects

Figure 3. Previous multi-step algorithm for k-nearest neighbor search adapted from [Kor+ 96].

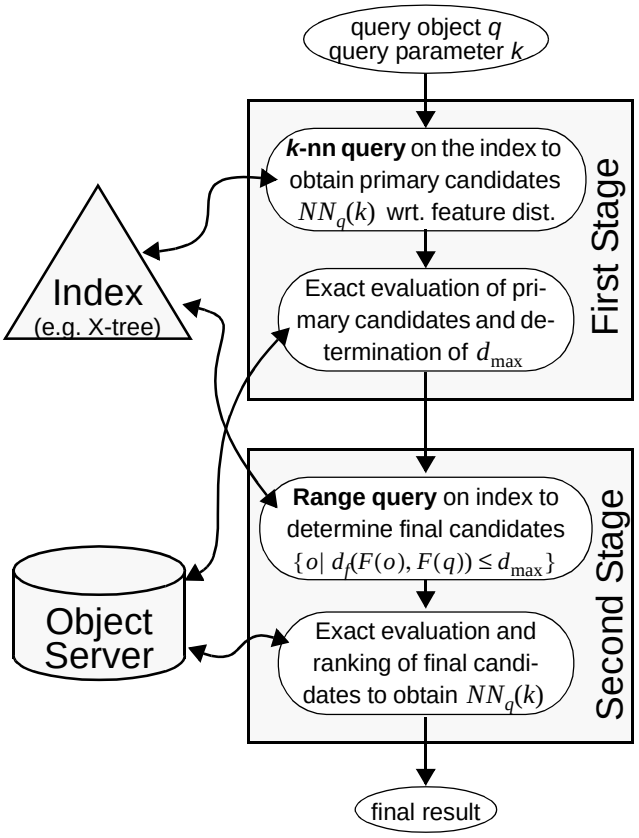

Figure 4. Illustration of the previous *k*-nearest neighbor query processor that proceeds in two stages

Although steps 3 and 4 are combined to a single step in the original version, we prefer the four-step version for conceptual clearness. Following the common terminology of multi-step query processing, the steps 1 and 3 are filter steps since they generate candidate sets from the underlying index structure, whereas 2 and 4 are refinement steps because they perform actual evaluations of the object similarity distance function using the exact representation of the objects.

The following lemma states the correctness of the algorithm. Subsequently, we investigate the performance aspects and analyze the efficiency of the procedure.

Lemma 1. Suppose that the lower-bounding property $d_f(o_1, o_2) \le d_o(o_1, o_2)$ holds for all objects $o_1, o_2 \in O$. Then the multi-step *k*-nearest neighbor algorithm of figure 3 guarantees no false drops.

Proof. See [Kor+ 96].

3.2 Performance Shortcomings

We implemented the *k*-nearest neighbor algorithm from figure 3 and performed some experiments on an image database that contains 64-dimensional color histograms of 12,000 color images [SK 97]. For the present experiments, we employed the Karhunen-Loève Transform (KLT) to reduce the 64-D histograms to 16-D feature vectors which are managed by an X-tree [BKK 96]. Several other techniques for reducing the dimensionality of high-dimensional feature vectors lead to lower-bounding distance functions in lower-dimensional vector spaces [SK 97] [Sei 97].

We performed a sample of 12-nearest neighbor queries which corresponds to a request of 0.1% of the database in each case. For a typical example, we retrieved 307 candidates from the filter step 3 which represent 2.5% of the database, resulting in 307 exact similarity distance evaluations in the refinement step 4. In general, a single refinement evaluation is very expensive, and causes a disk access in most cases since the exact representation of an object may be located anywhere on the disk within the area that contains the database. Moreover, for more complex objects than 64-D color histograms, the CPU time for a single evaluation may substantially exceed the I/O time of a single disk access.

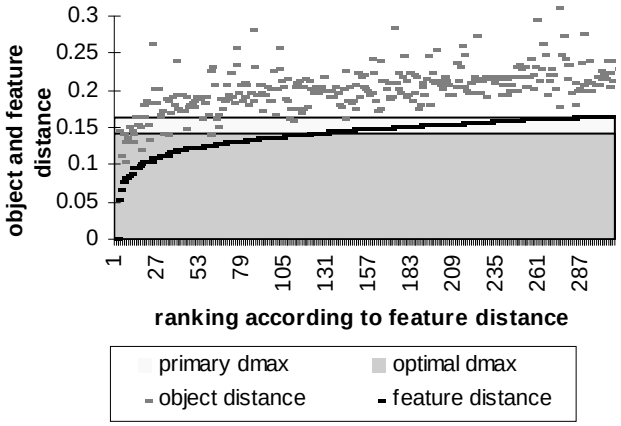

Figure 5. Object and feature similarity distances for typical *k*-nearest neighbor queries, ordered by the feature distance. The *primary* and the *optimal* d_{max} are marked by horizontal lines. For the example ($k = 12$) on the image database, the previous algorithm produces 307 candidates whereas 125 would be optimal.

In figure 5, we demonstrate a typical distribution of object similarity distances and the corresponding feature distances for the mentioned 12-nearest neighbor query, ranked by the feature distance values. In the example, step 2 evaluates d_{max} to approximately 0.164, and in the diagram, this value is depicted as the *primary* d_{max}. While using this primary d_{max} as the similarity query range, the filter step 3 obtains 307 candidates. The result $NN_q(k)$ of the *k*-nearest neighbor query is also retrieved by a corresponding similarity range query using the range $\varepsilon_k = \max\{d(o, q)|\ o \in NN_q(k)\}$. If we would know the value of ε_k already in advance, we would better use ε_k as similarity range in step 3 without producing any false drops. Therefore, we call ε_k the *optimal* d_{max} and depict it in the diagram. In the example, its value is 0.141.

Note that by a range query that uses the *optimal* $d_{\max}$ range ε_k, only 125 candidates are retrieved which is approximately 40% of 307, the number of candidates that were actually retrieved from the filter step 3. To illustrate the situation in more detail, we consider an additional example of a reduced synthetic data set in figure 6 on which a 10-nearest neighbor query has been performed. In step 1, a primary candidate set is obtained from the index which contains the 10 nearest neighbors of the query object according to the feature distance. From these candidates, the primary value of $d_{\max}$ is determined in step 2 which approximately is 0.54 in our example. The range query in step 3 yields a final candidate set of 32 candidates, from which the top 10 neighbors according to the object distance are determined. The final similarity distance ε_k has an approximate value of 0.34. A similarity range query that is bound by $\varepsilon_k = 0.34$ retrieves a candidate set that contains only 18 candidates which is little more than half of the 32 actual candidates from step 3.

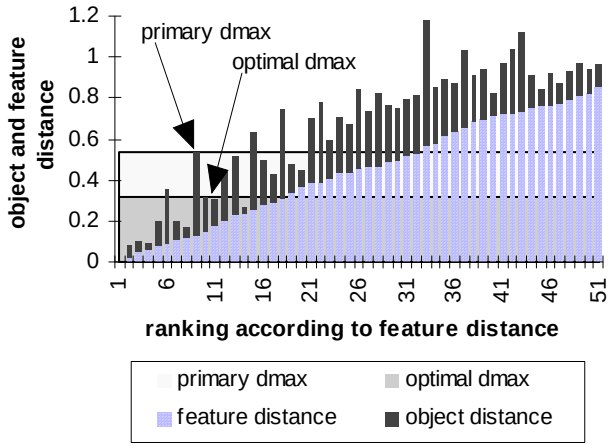

Figure 6. Object and feature similarity distances for *k*-nearest neighbor queries on a synthetic example. Again, the *primary* and the *optimal* d_{max} are marked by horizontal lines. For $k = 10$, the previous algorithm produces 32 candidates whereas 18 would suffice.

Obviously, this behavior of the algorithm is quite unsatisfactory, and we are seriously interested in a better solution that produces a smaller number of candidates for which the exact object distance has to be evaluated. The more time consuming a single exact evaluation is, the more important is the reduction of the number of candidates. As already mentioned in the introduction, we observed evaluation times of up to 1.6 seconds for quadratic forms, and for the max-morphological distance of images, 12.69 seconds are required for a single evaluation on the average [Kor+ 96]. From these measured values, it becomes obvious that the exact distance evaluations represent the most important cost factor for complex similarity search. Therefore, as many exact distance evaluations should be avoided as possible which means that the number of candidates whose exact similarity distances have to be evaluated should be minimized.

4 OPTIMAL MULTI-STEP ALGORITHM

We just identified the number of candidates produced by the filter step as the major cost factor of multi-step similarity search, particularly for complex similarity distance functions. In the following, we provide a formalization of this optimality criterion. Whereas the previous *k*-nearest neighbor algorithm suffers from generating too many candidates resulting in a bad performance, we present a novel algorithm that actually produces the minimum number of candidates thus minimizing the number of time-consuming exact similarity distance evaluations.

4.1 Fundamental Optimality Criterion

By the notion of *r*-optimality, we formalize the fundamental efficiency aspect of multi-step *k*-nearest neighbor algorithms that the number of candidates should be minimal:

Definition 4 (*r*-optimality). A multi-step *k*-nearest neighbor algorithm is called *r-optimal* if it does not produce more candidates in the filter step than necessary.

The question emerges how the *r*-optimality of an algorithm can be ensured. Prior to this problem, however, we have to clear how the *r*-optimality of an algorithm is recognized. How much candidates are actually necessary? By the following lemma, we provide a criterion that answers this question:

Lemma 2. Assume a multi-step *k*-nearest neighbor algorithm for the object similarity distance function d_o such that the filter distance function d_f fulfills the lower-bounding property, i.e. $d_f(o, q) \le d_o(o, q)$ for all objects $o, q \in O$. Such an algorithm is correct and *r*-optimal if and only if it exactly retrieves the candidate set $\{o \mid d_f(o, q) \le \varepsilon_k\}$ from the filter step where $\varepsilon_k = \max\{d_o(o, q), o \in NN_q(k)\}$.

Proof. For an arbitrary query range ε, consider the candidate set $\{o \mid d_f(o, q) \le \varepsilon\}$ which is obtained in the filter step by performing an ε-range query on the underlying access method. We show that for correctness and *r*-optimality of the overall algorithm, $\varepsilon = \varepsilon_k$ has to be fulfilled.

(i) Assume that $\varepsilon < \varepsilon_k$. Then, there may exist an object $o \in \text{DB}$ for which the estimation chain holds: $\varepsilon < d_f(o, q) \le d_o(o, q) \le \varepsilon_k$. The second inequality indicates that this situation is compatible with the lower-bounding property of d_o and d_f for the particular object *o*, and the last inequality implies that $o \in NN_q(k)$. However, due to the first inequality of the chain, the object *o* will not be retrieved by the ε-range query, and therefore, it is a false drop which contradicts the correctness of the algorithm.

(ii) Assume that $\varepsilon > \varepsilon_k$. Then, there may exist an object $o \in \text{DB}$ for which $\varepsilon_k < d_f(o, q) \le \varepsilon$, i.e. *o* is retrieved by the

ε-range query as a candidate that will be exactly evaluated in the refinement step. However, due to the lower-bounding property of d_o and d_f, $\varepsilon_k < d_f(o, q) \le d_o(o, q)$, i.e. the similarity distance of o to the query object q exceeds ε_k, the maximum distance of $NN_q(k)$. Thus, the object o does not rank among the k nearest neighbors of q which contradicts the r-optimality of the algorithm.

At all, only $\varepsilon = \varepsilon_k$ remains without contradiction as appropriate query range for the filter step, and the proposition holds. ◊

As we have seen in the preceding examples, the previous k-nearest neighbor algorithm fails to be r-optimal since it uses the *primary* $d_{\max}$ of step 2 instead of the *optimal* $d_{\max} = \varepsilon_k$. In general, the *primary* $d_{\max}$ of step 2 overestimates the optimal query range ε_k, i.e. $\varepsilon_k \le d_{\max}$, and effects a lot of unnecessary candidates in step 3 as we already observed in the experiments. The essential problem is that the value of ε_k is not known in advance of step 3. Only at the end of step 4, the actual value of ε_k is available.

4.2 Optimal Multi-Step Algorithm

The preceding observation leads us to the basic idea of our new algorithm: The value of $d_{\max}$ is decreased keeping step with the ongoing exact evaluation of the object similarity distance for the candidates. At the end of the step by step refinement, $d_{\max}$ reaches the optimal query range ε_k and prevents the method from producing more candidates than necessary thus fulfilling the r-optimality criterion. Figure 7 provides a pseudocode description of the procedure whereas in figure 8, the algorithm is illustrated schematically.

***k*-NearestNeighborSearch (*q*, *k*)** // optimal algorithm
1 initialize *ranking* = index.increm_ranking ($F(q)$, d_f)
2 initialize *result* = new sorted_list ⟨key, object⟩
3 initialize $d_{\max} = \infty$
4 **while** o = *ranking*.getnext **and** $d_f(o, q) \le d_{\max}$ **do**
5 **if** $d_o(o, q) \le d_{\max}$ **then** *result*.insert ($d_o(o, q)$, o)
6 **if** *result*.length ≥ 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* where key ≤ $d_{\max}$

Figure 7. Optimal multi-step k-nearest neighbor algorithm. The second condition in step 4 as well as the condition in step 5 are optional optimizations.

The algorithm has two basic components: By the incremental ranking query on the underlying access method, candidates are iteratively generated in ascending order according to their feature distance d_f to the query object. We will show

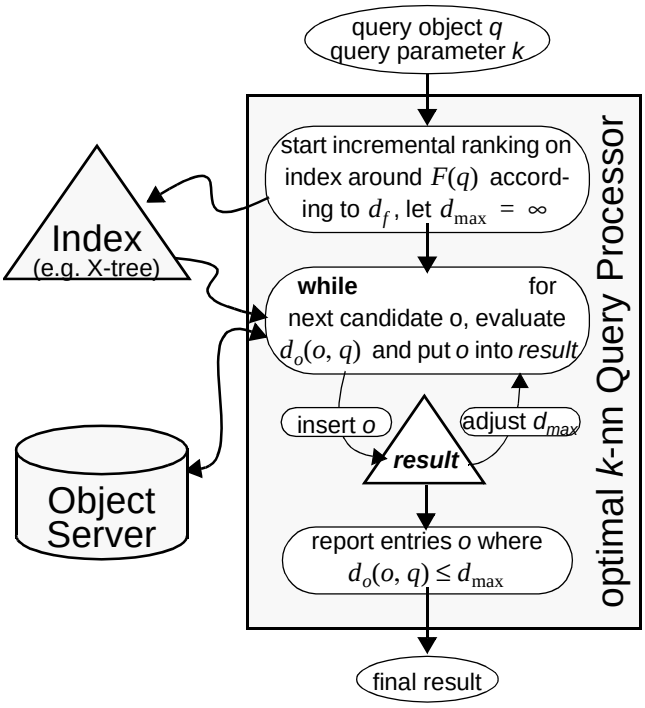

Figure 8. Illustration of the new optimal multi-step query processor for k-nearest neighbor search

that this property ensures the r-optimality of our algorithm. The second major component is the *result* list that manages the k nearest neighbors of the query object q within the current candidate set, keeping step with the candidate generation. The current k-th distance is held in $d_{\max}$ which is set to infinity until the first k candidates are retrieved from the index and evaluated. As we will show in the subsequent, $d_{\max}$ will be decreased exactly down to ε_k. This fact plays an important role in the subsequent analysis since by $d_{\max}$, the retrieval of candidates and the termination of the algorithm is controlled.

4.3 Analysis of the New Algorithm

In this subsection, we show the correctness and r-optimality of our algorithm. We argue about k-th distances and k-nearest neighbors of arbitrary object sets $C \subseteq O$. For this purpose, we slightly generalize our notation which up to now was fixed to a particular object set, the database DB $\subseteq O$. Table 1 lists the symbols and their meanings as they are used in the subsequent.

Definition 5 (q-ranking of C). A *q-ranking* of an object set $C \subseteq O$ is a bijection $r^{q, C}: \Im_{|C|} \to C$ that is monotonous with respect to the distance of the objects from C to the query object q, i.e.

$$\forall i, j \in \Im_N: i < j \Rightarrow d(r^{q, C}(i), q) \le d(r^{q, C}(j), q)$$

Symbol	Description
O	universe of objects
$d\colon O \times O \to \Re_0^+$	similarity distance function
$\Im_n = \{1, \ldots, n\}$	index set
$q \in O$	query object
$k \in \Im_\infty$	query parameter
$d^q\colon O \to \Re_0^+$	distance to the query object, $d^q(o) = d(o, q)$
q-ranking $r^{q,C}$ of $C \subseteq O$	d^q-monotonous bijection $r^{q,C}\colon \Im_{\lvert C\rvert} \to C$
$d^{q,k}(C)$ for $C \subseteq O$	k-th distance of objects from C to q
$NN_q(k)^C$ for $C \subseteq O$	k-nearest neighbors of q within C

Table 1. Symbols in the context of k-nearest neighbor search

Definition 6 (k-th distance of C). For any subset $C \subseteq O$ of objects, i.e. $C \in \wp(O)$ where $\wp(O)$ denotes the power set of O, the function $d^{q,k}\colon \wp(O) \to \Re_0^+$ returns the *k-th distance* of the objects from C to the query object q if C contains at least k elements, and ∞ otherwise:

$$d^{q,k}(C) = \begin{cases} \infty & \textbf{if } |C| < k \\ d(r^{q,C}(k), q) & \textbf{else} \end{cases}$$

Definition 7 (k-nearest neighbor set of C). For any object set $C \subseteq O$, let $NN_q(k)^C$ denote the set of the *k-nearest neighbors* of the query object q within C. We define $NN_q(k)^C$ in terms of a similarity range:

$$NN_q(k)^C = \{o \in C \mid d(o, q) \le d^{q,k}(C)\}$$

In order to show the correctness and r-optimality of our new algorithm, we start by proving the observation that $d_{\max}$ is decreasing with an increasing number of candidates.

Lemma 3. Let a query object $q \in O$ and a query parameter k be given. For any object set $C \subseteq O$ and any additional object $o \notin C$, the following estimation is true:

$$d^{q,k}(C \cup \{o\}) \le d^{q,k}(C)$$

Proof. If $|C| < k$, then $d^{q,k}(C) = \infty$, and the proposition is obvious. For $|C| \ge k$, consider the following cases:

(i) if $d(o, q) > d^{q,k}(C)$, then o does not rank among the k-nearest neighbors of q within $C \cup \{o\}$, i.e. $NN_q(k)^{C \cup \{o\}} = NN_q(k)^C$ and $d^{q,k}(C \cup \{o\}) = d^{q,k}(C)$.

(ii) if $d(o, q) = d^{q,k}(C)$, then o also is a k-nearest neighbor of q within $C \cup \{o\}$, $NN_q(k)^{C \cup \{o\}} = NN_q(k)^C \cup \{o\}$, and $d^{q,k}(C \cup \{o\}) = d^{q,k}(C)$.

(iii) if $d(o, q) < d^{q,k}(C)$, then o ranks among the k nearest neighbors of q within $C \cup \{o\}$, i.e. $o \in NN_q(k)^{C \cup \{o\}}$. If there is a single object $\hat{o} \in NN_q(k)^C$ that has the k-th distance to q, $d(\hat{o}, q) = d^{q,k}(C)$, then o displaces $\hat{o}$ from $NN_q(k)^C$, and all remaining distances are lower than $d^{q,k}(C)$ which implies $d^{q,k}(C \cup \{o\}) < d^{q,k}(C)$. If some objects within C with a rank below k share the same distance $d^{q,k-1}(C) = d^{q,k}(C)$, then all of them remain included in $NN_q(k)^{C \cup \{o\}}$, and $d^{q,k}(C \cup \{o\}) = d^{q,k}(C)$.

At all, we obtain $d^{q,k}(C \cup \{o\}) \le d^{q,k}(C)$ as proposed. ◊

4.4 Proof of the *r*-Optimality

We are now prepared to show the correctness and r-optimality of our new algorithm under the supposition of the lower-bounding property of filter and refinement distance functions. In order to avoid a notational confusion of filter and object distances, we append the subscript index o to $d^{q,k}$ and write $d_o^{q,k}$ in the following.

Theorem. When providing the filter step with a filter distance function d_f that lower-bounds the object similarity distance function d_o, i.e. $d_f(o, q) \le d_o(o, q)$ for all objects $o, q \in O$, the new multi-step k-nearest neighbor algorithm (figure 7) guarantees no false drops and is r-optimal.

Proof. First, let us observe that the algorithm obtains its candidates from the incremental ranking query on the underlying access method. The candidate set is growing step by step and may be regarded as a sequence of iteratively extended subsets of the database, $C_1 \subseteq C_2 \subseteq \ldots \subseteq \mathrm{DB}$. By using the *result* list, the algorithm determines the k-nearest neighbor set $NN_o^{q,k}(C_i)$ for each C_i according to the object similarity distance d_o, and the i-th value of $d_{\max}$ is equal to $d_o^{q,k}(C_i)$. By induction over the increasing sets, lemma 3 justifies the estimation chain $d_o^{q,k}(C_1) \ge d_o^{q,k}(C_2) \ge \ldots$ which is lower-bounded by $d_o^{q,k}(\mathrm{DB})$. Thus, $d_{\max}$ never becomes smaller than $d_o^{q,k}(\mathrm{DB})$. According to lemma 2, this fact ensures that our algorithm produces no false drops, since $d_{\max}$ is used as the upper bound for the feature distance of the candidates that are retrieved from the filter step.

It remains to prove that the filter step does not produce unnecessary candidates. Such a behavior would contradict the r-optimality of the algorithm. From lemma 2, we know that candidates are unnecessary if they have a feature distance which is greater than $d_o^{q,k}(\mathrm{DB})$. Although $d_{\max}$ is iteratively decreased while retrieving new candidates, the boundedness by the final value $d_o^{q,k}(\mathrm{DB})$ applies at least for the last candidate o_{last} that is retrieved from the filter step, i.e.

$d_f(o_{last}, q) \le d_o^{q,k}(DB)$. Now, we exploit the fact that our filter step performs an incremental similarity ranking which effects that the candidates are retrieved in ascending order with respect to their filter distance d_f to the query object, i.e. $d_f(o_1, q) \le d_f(o_2, q) \le \ldots \le d_f(o_{last}, q)$. Thus, no candidates *o'* are obtained from the filter step or even evaluated in the refinement step for which $d_f(o', q) > d_o^{q,k}(DB)$ is true, and the improved multi-step *k*-nearest neighbor algorithm is *r*-optimal as proposed. ◊

Let us observe that d_{max} may incidentally reach its final value quite early among the first few candidates. However, this fact may not be recognized earlier than all candidates are evaluated. On the other hand, it may happen that the actual value of $d_o^{q,k}(DB)$ is only reached for the last candidate. This case occurs e.g. if $d_o(o, q) = d_f(o, q) = d_o^{q,k}(DB)$, and it cannot be excluded until all candidates are evaluated that have a feature distance below $d_o^{q,k}(DB)$.

5 PERFORMANCE EVALUATION

From the analysis in the preceding subsections, we know the theoretical optimality of our new algorithm. In this section, we demonstrate the actual and significant improvement of our method in comparison with the previous algorithm.

The algorithms were implemented in C++, and the experiments were run on an HP C160 under HP-UX 10. For the low- and medium-dimensional feature spaces, we used the X-tree [BKK 96] as an appropriate index structure. Clearly, the number of candidates as the main cost factor does not depend on the index architecture, and a comparable improvement behavior is to be expected for multi-disk index structures such as the parallel X-tree [Ber+ 97]. Presently, we avoid to mix the concepts and defer the parallel case to future work.

5.1 Color Image Database (64-D)

Our first example is a database of 12,100 color images represented by 64-D color histograms [SK 97]. A 12-D X-tree manages the histogram vectors which are reduced to 12 dimensions by the Karhunen-Loève Transform, resulting in an index that contains 240 pages.

We performed a sample of 200 *k*-nearest neighbor queries for $k \in \{2, 4, \ldots, 12\}$ thus retrieving up to 1% of the images from the database. Figure 9 depicts the average number of candidates that are produced by the filter step (top diagram) as well as the overall runtime (bottom diagram). Note that e.g. for $k = 8$, the previous algorithm already retrieves 10% of the database (1,257 candidates) whereas in the optimal algorithm, only 1.4% of all database objects (172 candidates) are read from disk and evaluated exactly. The selectivity improvement factor in our example is approximately 7.2 and does not vary much. The overall runtime is improved by a factor of 5.5 in the average.

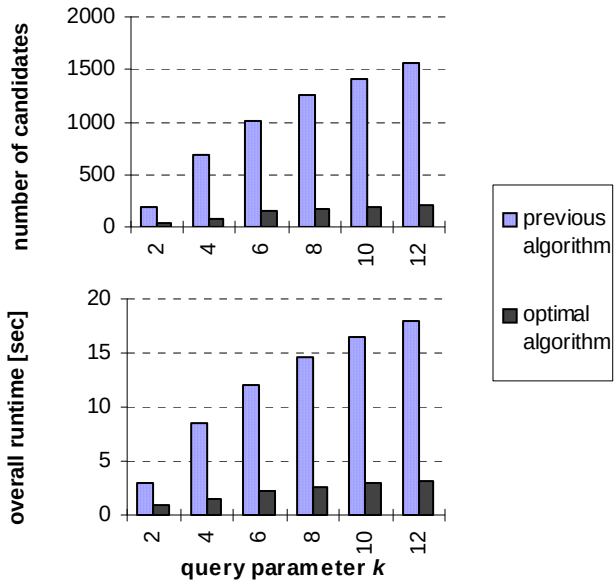

Figure 9. Improvement of filter selectivity (top) and overall runtime (bottom) for a database of 12,100 color images that are represented by 64-D color histograms.

5.2 Leaps in the Filter Selectivity

In figure 10, we demonstrate an effect which cannot be recognized from averaged evaluations of query samples since it becomes only evident for single queries. Recall that the previous algorithm performs a range query on the index using d_{max} as query range. The value of d_{max} is determined as the maximum object similarity distance of the *k* nearest neighbors of the query object with respect to the feature distance. These *k* nearest neighbors correspond to the first *k* of all candidates that were retrieved by the similarity ranking query in the optimal algorithm. Observe that in most cases, the primary d_{max} has the same value for a wide range of *k*, and the corresponding number of candidates increases with *k* in a staircase fashion. In figure 10, query *a* produces such steps where the number of candidates is 3 for $k = 1$; 306 for $k \in \{2, 3\}$; 699 for $k \in \{4, \ldots, 7\}$; 1,408 for $k \in \{8, 9\}$, and 6,255 from $k = 10$ up to $k > 20$. For $k \le 20$, query *b* produces only two steps of 811 for $1 \le k \le 3$ and 3,584 candidates for $k > 3$.

5.3 Color Image Database (112-D)

For our final example, we use a database of 112,700 color images which are represented as 112-D color histograms and indexed by a 12-D X-tree containing 2,387 pages. Again, we performed samples of *k*-nearest neighbor queries for $k \in \{2, 4, \ldots, 12\}$ thus retrieving up to 0.01% of the objects from the database. The top diagram in figure 11 depicts the number of candidates generated by the previous and the opti-

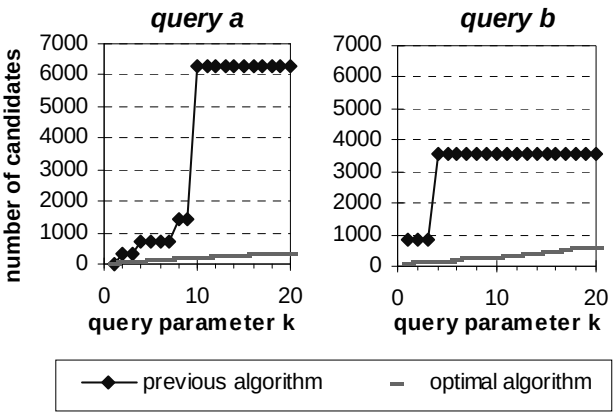

Figure 10. Leaps in the filter selectivity: For two query objects *a* and *b*, the number of candidates out of 12,100 objects depending on the query parameter *k* is depicted for the previous and the optimal algorithm. For the previous algorithm, leaps can be observed.

mal algorithm. The bottom diagram demonstrates the affect of the selectivity improvement onto the overall runtime. Whereas the improvement factor of the filter selectivity is approximately 17, the overall runtime is improved by a factor of 8.5 in the average. The experiments substantiate the general observation that the more complex a object distance function, the stronger is the impact of the filter selectivity onto the overall runtime.

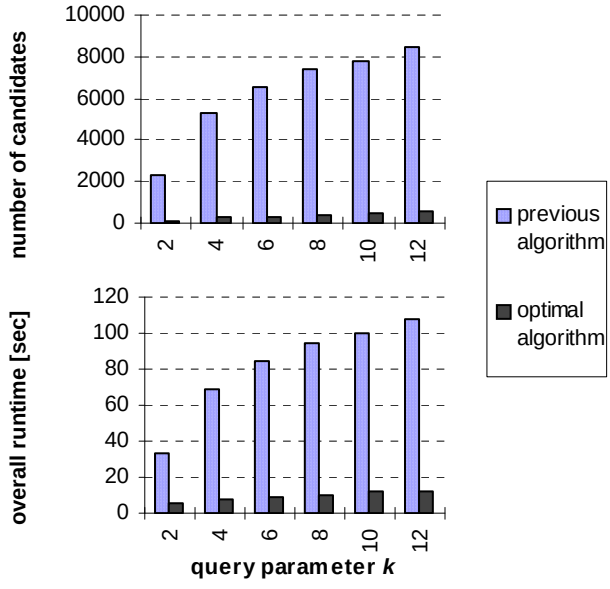

Figure 11. Improvement of filter selectivity (top) and overall runtime (bottom) for a 112-D image database of 112,700 color images.

Our optimal algorithm does not only reduce the number of candidates whose exact representation are read from disk and whose object similarity distance to the query object is evaluated exactly, but also reduces the number of page accesses in the index. This behavior results from the fact that the previous algorithm performs a range query over the primary $d_{\max}$ value that overlaps a larger portion of the data space than the optimal query range $d_o^{q,k}(\mathrm{DB})$ to which the optimal multi-step algorithm restricts its search in the index. At $k = 8$ for example, the previous algorithm accesses 50% of the index pages (1,200 of 2,387) whereas the optimal algorithm reads only 22% of the index pages (533 of 2,387). The improvement factor is greater than 2.0 and does not vary significantly with the increasing query parameter *k* in our experiments.

5.4 Pixel-Based Shape Similarity (1,024-D)

The next experiments are performed on a 1,024-D database of 10,000 grayscale images as an example for the adaptable pixel-based shape similarity [AKS 98] [Sei 97]. In the example, we used neighborhood influence weights for the neighborhood area (9,1) around each pixel. The resulting distance function is a quadratic form according to the model, and we measured an average evaluation time of approximately 100 milliseconds for a single image distance. By applying the Karhunen-Loève Transform (KLT) in order to reduce the dimensionality [SK 97], various indexes were created for the reduced dimensions 16, 32, 48, and 64. Figure 12 depicts the performance results for a sample of 50 *k*-nearest neighbor queries ($k = 5$). Whereas the number of candidates monoto-

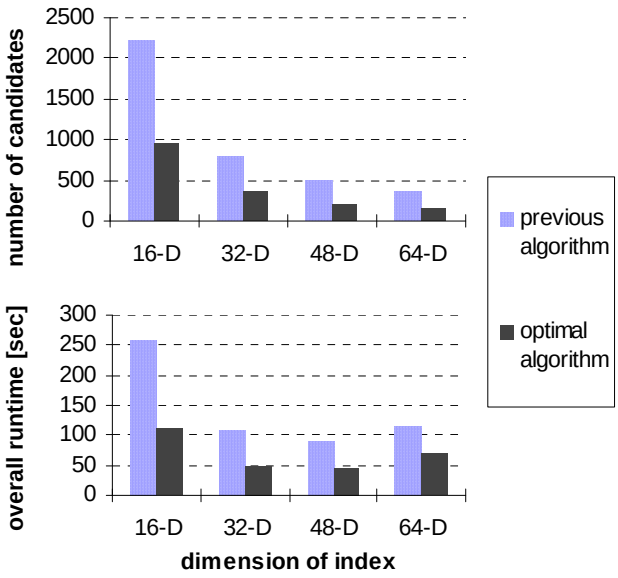

Figure 12. Filter selectivity and overall runtime for a 1,024-D image database of 10,000 grayscale images.

nously decreases with increasing dimension of the index, the overall runtime is minimum for the 48-D index in our example. The reason is the quadratic nature of the filter distance function in the index and the well-known curse of dimensionality for high-dimensional index structures. Nevertheless, the new optimal algorithm outperforms the previous two-stage algorithm by a factor of 2.3 for the number of candidates and a factor of 1.6 to 2.3 for the overall runtime.

5.5 Uniformly Distributed Data (20-D)

For the next experiments, we synthetically created a database of 100,000 objects uniformly distributed in the 20-D space. An index of dimension 15 was used, and we performed a sample of 200 *k*-nearest neighbor queries for $k = 10$. As similarity distance function, we employed artificially generated quadratic forms that represent the Euclidean distance (sim-id), a weighted Euclidean distance (sim-1-0) and a more general quadratic form (sim-2-2). Figure 13 demonstrates that we obtained average improvement factors of 72, 120, and 64 for the number of candidates. These reductions lead to an acceleration of the total runtime by factors of 26, 48, and 23 on the average for the sample queries.

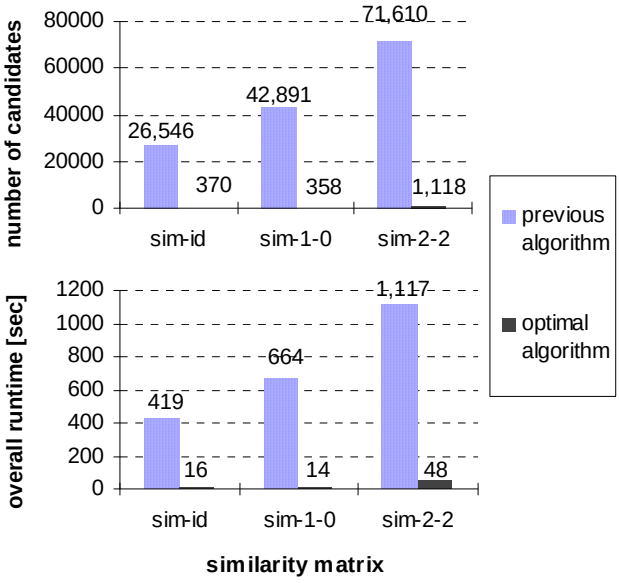

Figure 13. Improvement of filter selectivity (top) and overall runtime (bottom) for *k*-nearest queries ($k = 10$) on a 20-D dataset of 100,000 uniformly distributed objects.

6 CONCLUSIONS

We developed a new multi-step algorithm for *k*-nearest neighbor search which clearly outperforms the state-of-the-art algorithm. In addition to the significant performance gain, we have theoretically shown that our algorithm is optimal with respect to the number of candidates that are retrieved from the underlying index. The number of candidates is identified to be an important parameter for the overall runtime efficiency since the exact evaluation of complex, high-dimensional and adaptable similarity distance functions is the dominating cost factor of multi-step similarity query processing. Along with the CPU time, nearly every candidate causes a random disc access in the refinement step since the exact representations of the objects are in general spread over the database. For the filter step, the impact of a smaller number of candidates is that a smaller number of index pages has to be accessed.

Our new algorithm optimally supports multi-step *k*-nearest neighbor search. The performance is no longer affected by hot spots of d_o among the first *k* of the d_f-candidates, but only depends on the quality of the filter step: The higher the values of a feature distance function, the better is the exact value of the object distance function estimated, and the less is the expected number of candidates that are obtained from the filter step. The best filter selectivity is achieved by using the greatest of all possible lower-bounding feature distance functions. On top of the theoretical analysis that proves the *r*-optimality of our new algorithm, experimental evaluations demonstrate the significant performance gain of the novel technique over the previous solution.

REFERENCES

[AFS 93] Agrawal R., Faloutsos C., Swami A.: *'Efficient Similarity Search in Sequence Databases'*, Proc. 4th. Int. Conf. on Foundations of Data Organization and Algorithms (FODO'93), Evanston, IL, in: Lecture Notes in Computer Science, Vol. 730, Springer, 1993, pp. 69-84.

[AKS 98] Ankerst M., Kriegel H.-P., Seidl T.: *'Pixel-based Shape Similarity Search in Large Image Databases'*, submitted for publication.

[ALSS 95] Agrawal R., Lin K.-I., Sawhney H. S., Shim K.: *'Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases'*, Proc. 21th Int. Conf. on Very Large Databases (VLDB'95), Morgan Kaufmann, 1995, pp. 490-501.

[AMN 95] Arya S., Mount D. M., Narayan O.: *'Accounting for Boundary Effects in Nearest Neighbor Searching'*, Proc. 11th Annual Symposium on Computational Geometry, Vancouver, Canada, 1995, pp. 336-344.

[BBKK 97] Berchtold S., Böhm C., Keim D. A., Kriegel H.-P.: *'A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces'*, Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS), Tucson, AZ, 1997, pp. 78-86.

[Ber+ 97] Berchtold S., Böhm C., Braunmüller B., Keim D. A., Kriegel H.-P.: *'Fast Parallel Similarity Search in Multimedia Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, AZ, 1997, pp. 1-12, Best Paper Award.

[Ber+ 98] Berchtold S., Ertl B., Keim D. A., Kriegel H.-P., Seidl T.: *'Fast Nearest Neighbor Search in High-dimensional Spaces'*, Proc. 14th Int. Conf. on Data Engineering (ICDE'98), Orlando, Florida, 1998.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems'*, Proc. 3rd Int. Symp. on Large Spatial Databases (SSD'93), Singapore, 1993, in: Lecture Notes in Computer Science, Vol. 692, Springer, pp. 357-376.

[BKK 96] Berchtold S., Keim D. A., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data'*, Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB'96), Mumbai, India, 1996, pp. 28-39.

[BK 97] Berchtold S., Kriegel H.-P.: *'S3: Similarity Search in CAD Database Systems'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, AZ, 1997, pp. 564-567.

[BMH 92] Badel A., Mornon J. P., Hazout S.: *'Searching for Geometric Molecular Shape Complementarity Using Bidimensional Surface Profiles'*, Journal of Molecular Graphics, Vol. 10, 1992, pp. 205-211.

[CPZ 97] Ciacca P., Patella M., Zezula P.: *'M-tree: An Efficient Access Method for Similarity Search in Metric Spaces'*, Proc. 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece, 1997, pp. 426-435.

[Fal+ 94] Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D., Equitz W.: *'Efficient and Effective Querying by Image Content'*, Journal of Intelligent Information Systems, Vol. 3, 1994, pp. 231-262.

[FBF 77] Friedman J. H., Bentley J. L., Finkel R. A.: *'An Algorithm for Finding the Best Matches in Logarithmic Expected Time'*, ACM Transactions on Math. Software, Vol. 3, 1977, pp. 209-226.

[FL 95] Faloutsos C., Lin K.-I.: *'FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Data'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, 1995, pp. 163-174.

[FRM 94] Faloutsos C., Ranganathan M., Manolopoulos Y.: *'Fast Subsequence Matching in Time-Series Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 419-429.

[GG 97] Gaede V., Günther O.: *'Multidimensional Access Methods'*, ACM Computing Surveys.

[GM 93] Gary J. E., Mehrotra R.: *'Similar Shape Retrieval using a Structural Feature Index'*, Information Systems, Vol. 18, No. 7, 1993, pp. 525-537.

[Haf+ 95] Hafner J., Sawhney H. S., Equitz W., Flickner M., Niblack W.: *'Efficient Color Histogram Indexing for Quadratic Form Distance Functions'*, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, No. 7, 1995, pp. 729-736.

[Hen 94] Henrich, A.: *'A Distance-Scan Algorithm for Spatial Access Structures'*, Proc. 2nd ACM Workshop on Advances in Geographic Information Systems, Gaithersburg, Maryland, 1994, pp. 136-143.

[HS 95] Hjaltason G. R., Samet H.: *'Ranking in Spatial Databases'*, Proc. 4th Int. Symposium on Large Spatial Databases (SSD'95), in: Lecture Notes in Computer Science, Vol. 951, Springer, 1995, pp. 83-95.

[Jag 91] Jagadish H. V.: *'A Retrieval Technique for Similar Shapes'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 208-217.

[Kor+ 96] Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z.: *'Fast Nearest Neighbor Search in Medical Image Databases'*, Proc. 22nd VLDB Conference, Mumbai, India, 1996, pp. 215-226.

[KS 98] Kriegel H.-P., Seidl T.: *'Approximation-Based Similarity Search for 3-D Surface Segments'*, GeoInformatica, Kluwer Academic Publishers, 1998, to appear.

[KSS 97] Kriegel H.-P., Schmidt T., Seidl T.: *'3D Similarity Search by Shape Approximation'*, Proc. Fifth Int. Symposium on Large Spatial Databases (SSD'97), Berlin, Germany, Lecture Notes in Computer Science, Vol. 1262, 1997, pp.11-28.

[OM 88] Orenstein J. A., Manola F. A.: *'PROBE Spatial Data Modeling and Query Processing in an Image Database Application'*, IEEE Trans. on Software Engineering, Vol. 14, No. 5, 1988, pp. 611-629.

[PM 97] Papadopoulos A., Manolopoulos Y.: *'Performance of Nearest Neighbor Queries in R-Trees'*, Proc. of the 6th International Conference on Database Theory, Delphi, Greece, 1997, LNCS 1186, pp. 394-408.

[PS 93] Preparata F. P., Shamos M. I.: *'Computational Geometry. An Introduction'*, Texts and Monographs in Computer Science. 5th, corr. ed., Springer, 1993.

[RKV 95] Roussopoulos N., Kelley S., Vincent F.: *'Nearest Neighbor Queries'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, 1995, pp. 71-79.

[RP 92] Ramasubramanian V., Paliwal K. K.: *'Fast k-Dimensional Tree Algorithms for Nearest Neighbor Search with Application to Vector Quantization Encoding'*, IEEE Transactions on Signal Processing, Vol. 40, No. 3, March 1992, pp. 518-531.

[Sei 97] Seidl T.: *'Adaptable Similarity Search in 3-D Spatial Database Systems'*, PhD thesis, Institute for Computer Science, University of Munich, 1997.

[SK 97] Seidl T., Kriegel H.-P.: *'Efficient User-Adaptable Similarity Search in Large Multimedia Databases'*, Proc. 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece, 1997, pp. 506-515.

[Spr 91] Sproull R.F.: *'Refinements to Nearest Neighbor Searching in k-Dimensional Trees'*, Algorithmica 1991, pp. 579-589.

[WJ 96] White D. A., Jain R.: *'Similarity Indexing with the SS-tree'*, Proc. 12th Int. Conf. on Data Engineering (ICDE), 1996, pp. 516-523.