Optimizing All-Nearest-Neighbor Queries with Trigonometric Pruning

Tobias Emrich, Franz Graf, Hans-Peter Kriegel, Matthias Schubert, and Marisa Thoma

Ludwig-Maximilians-Universität München
Oettingenstr. 67, Munich, Germany
{emrich,graf,kriegel,schubert,thoma}@dbs.ifi.lmu.de

Abstract. Many applications require to determine the k-nearest neighbors for multiple query points simultaneously. This task is known as all-(k)-nearest-neighbor (AkNN) query. In this paper, we suggest a new method for efficient AkNN query processing which is based on spherical approximations for indexing and query set representation. In this setting, we propose trigonometric pruning which enables a significant decrease of the remaining search space for a query. Employing this new pruning method, we considerably speed up AkNN queries.

1 Introduction

Similarity search in databases is an important problem in content-based multimedia retrieval. Additionally, similarity queries are a useful database primitive for speeding up multiple data mining algorithms on large databases. In the majority of approaches for similarity search, the objects are described as points in a high-dimensional data space, where each dimension describes the object value w.r.t. a given characteristic or feature. These feature vectors are compared via metric distance functions such as the Euclidean distance or other L_p-norms. Thus, the distance between two feature vectors v_1 and v_2 models the similarity between the corresponding objects. One of the most important types of similarity queries in this setting are k-nearest-neighbor (kNN) queries, retrieving the k objects in the database which have the smallest distance to a given query vector q. The majority of the approaches developed so far focus on the efficient processing of a single query at a time. However, in many applications like data mining and similarity search, it is previously known that it is necessary to process a large number of kNN queries to generate a result. More precisely, an AkNN query retrieves the k-nearest neighbors in the inner set or database $\boldsymbol{S}$ for each object in the outer or query set $\boldsymbol{R}$. Let us note that the same type of query is also known as kNN join [1].

Multiple computational problems use AkNN queries: In multimedia retrieval, many recent approaches model the image content as a set of local descriptors [2] or point clouds [3]. An AkNN query efficiently retrieves the best matches for a set of local descriptors in a given image database. Another area of application is data mining: [1] surveys multiple data mining algorithms that can be accelerated by efficient methods for AkNN computation like parallel kNN classification and k-means clustering. Furthermore, it is possible to employ AkNN processing for deriving outlier factors like [4].

In this paper, we propose a new approach for processing AkNN queries. As our method uses spherical page regions it employs an SS-Tree [5]. To define the pruning

area around the approximations, we introduce trigonometric pruning which accounts for the fact that the relevant search space is not necessarily symmetric around the query approximation. Since this pruning method is based on trigonometric relationships, we refer to it as trigonometric pruning. One of its advantages is its capability to calculate pruning areas which are based on query approximations and to considerably decrease the remaining search space employed by other approaches like [3, 6].

The rest of the paper is organized as follows. Section 2 surveys related work for processing AkNN queries and kNN joins. In Sect. 3, we specify the problem and describe the methods and data structures our solution is based on. Section 4 introduces our new pruning method and Sect. 5 then describes our new query algorithms. A performance evaluation is presented in Sect. 6. The paper concludes with a summary and some directions for future work in Sect. 7.

2 Related Work

There already exist several approaches for processing AkNN queries. The initial and most simple approach is the computation of a separate kNN query on the inner set $\boldsymbol{S}$ for each point in the outer set $\boldsymbol{R}$. This method is often referred to as nested loop join and can be accelerated by various search methods for kNN query processing. A comprehensive survey can be found in [7]. Since this basic approach obviously is not optimal w.r.t. the number of page accesses and distance computations, several dedicated approaches for processing AkNN queries have been proposed. We first of all have to distinguish index-based from scan-based solutions. In a scan-based solution, we assume that both data sets are not organized in any form of index and thus, we have to scan the complete data set for join processing. Examples for processing kNN joins without the use of index structures are GORDER [8] and HANN [9].

In this paper, we assume that at least $\boldsymbol{S}$ is already organized in an index structure. It is shown in the experiments in [6, 9] that the use of an index structure can considerably speed up kNN join processing. One of the first publications discussing kNN join algorithms was [1]. In this paper, Böhm et al. demonstrate that kNN joins are useful database primitives that can be employed to speed up various data mining algorithms like k-Means clustering. The proposed algorithm is based on a new data structure called multipage index (MuX) which introduces larger pages for fast I/O accesses which are further organized into memory pages aiming at minimizing distance calculations. The proposed join algorithm first retrieves the current page of $\boldsymbol{R}$ and then queries $\boldsymbol{S}$ with the set of all query points. For each query point, the algorithm maintains its own active page list. The current pruning distance is considered as the maximum element of any query of these queues, and pages are refined w.r.t. the minimal distance to any query element. A drawback of this approach is that it has an overhead of distance computations because each object in $\boldsymbol{S}$ has to be compared to all elements in the query page. In [9] the authors discuss two methods for AkNN queries that assume that $\boldsymbol{S}$ is organized in a spatial index structure like the $\boldsymbol{R}$-Tree [10]. The first index-based approach discussed in [9] aims at improving the use of a disk cache when posing a separate NN query for each query point. The second proposed method, called *batched nearest neighbor* search (BNN), groups query points based on a Hilbert curve and poses one query

for each group of points. The method considers the distance to the k-nearest neighbor $\delta_{k,x}$ for each element x in the current query set. The maximum of these distances is now used to extend the minimum bounding rectangle around the query set and thus, describes the current pruning area. In [11], a kNN join algorithm based on the similarity search method iDistance [12] is proposed. The paper describes a basic algorithm called iJoin extending iDistance to the kNN join problem. Furthermore, two extensions are proposed that employ MBR-approximations to reduce distance computations and a dimensionality reduction approach to improve the performance on higher dimensionalities. Recently, two approaches have been published which reduce distance calculations by not considering the particular elements in the current set of query points. Instead, the pruning area is established on a minimum bounding rectangle around the query set. [3] presents an approach for applications on large point clouds in image processing. In this method, the set of query points is approximated by a minimum bounding rectangle (MBR) which is posed as a query to an $\boldsymbol{R}$-Tree organizing $\boldsymbol{S}$. The authors propose to employ the maximum distance that two points in two compared MBRs might have to determine the current pruning range. This criterion is named MaxMaxDist. The authors prove that their algorithm is optimal w.r.t. the number of pages that intersect a query area spanned by the MaxMaxDist. The experiments are focused on the problem of point clouds and thus, are directed at rather low dimensional settings. In [6], the authors propose NXNDIST which is based on the observation that at each side of an MBR, there must be exactly one contained data point which realizes the minimum distance. Thus, NXNDIST decreases the MaxMaxDist by allowing the use of the minimal MaxDist for exactly one dimension. The result is a closer bound for MBRs that is guaranteed to contain at least one nearest neighbor to any query point in the query MBR in the intersecting pages of $\boldsymbol{S}$. In their solution, the authors additionally propose to employ MBR-quadtrees as an index structure for $\boldsymbol{S}$ instead of R-Trees.

In contrast to all discussed approaches, our method uses spherical page approximations instead of MBRs. We employ the SS-Tree [5] for indexing $\boldsymbol{S}$. Our main pruning method, trigonometric pruning, describes the remaining search space based on the query approximations only. However, opposed to previous approaches, the pruning area is not built by symmetrically extending the query approximation in each direction. Instead, we propose the use of an asymmetric pruning area to further limit the search space. Finally, we show how our new approach can be effectively combined with existing pruning methods to achieve a general improvement on data sets of varying characteristics.

3 All-k-Nearest-Neighbors Using Spherical Index Structures

In this section, we will formalize our task of AkNN queries and describe the index structure used for our approach.

3.1 All-k-Nearest-Neighbors

As mentioned before, AkNN queries aim at retrieving the k-nearest neighbors for each element of a given query or outer set $\boldsymbol{R}$ in the inner data set $\boldsymbol{S}$. The set of the k-nearest neighbors of point X in a set $\boldsymbol{X}$ is defined as follows:

Definition 1 (k-nearest neighbors). *Let $X \in \mathbb{R}^n$ be a query vector, let $\boldsymbol{X} \subset \mathbb{R}^n$ be a database of feature vectors and let $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a distance metric on $\mathbb{R}^n$. Then, the set $NN_k(X, \boldsymbol{X})$ of the k-nearest neighbors of X in $\boldsymbol{X}$ for any $k \in \mathbb{N}$ is defined as follows:*

$$(i)\ \ |NN_k(X, \boldsymbol{X})| = k$$
$$(ii)\ \forall\, Y \in NN_k(X, \boldsymbol{X}),\ \hat{Y} \in \{\boldsymbol{X} \setminus NN_k(X, \boldsymbol{X})\} : d(X, Y) \leq d(X, \hat{Y})$$

Definition 2 (AkNN Query). *Let $\boldsymbol{R}, \boldsymbol{S} \subset \mathbb{R}^n$ be two data sets and $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ a distance metric on $\mathbb{R}^n$. An all-k-nearest-neighbor (AkNN) query retrieves the result set $ANN_k(\boldsymbol{R}, \boldsymbol{S}) \subseteq \boldsymbol{R} \times \boldsymbol{S}$ for any $k \in \mathbb{N}$ for which the following condition holds:*

$$\forall\, R \in \boldsymbol{R},\ S \in \boldsymbol{S} : (R, S) \in ANN_k(\boldsymbol{R}, \boldsymbol{S}) \Rightarrow S \in NN_k(R, \boldsymbol{S}) \tag{1}$$

Our algorithm for efficiently processing AkNN queries is built on trigonometric functions, which need to be valid in the underlying featurespace. In this paper we use the Euclidean metric, the most popular distance metric for this kind of applications. Furthermore, we assume that the inner set $\boldsymbol{S}$ is organized in an index structure which is based on spherical page regions. A spherical page region is specified by a centroid $C \in \mathbb{R}^n$ and a radius $r \in \mathbb{R}^+ \setminus \{0\}$ and thus, it describes a hypersphere around C, containing all points P_i with distance $d(C, P_i) \leq r$. Although approximating page regions using minimum bounding rectangles is more common for spatial index structures, there have been several successful index structures employing spherical page regions [5, 13].

3.2 The SS-Tree

As mentioned above, our method employs the SS-Tree [5], an efficient index structure for similarity search based on spherical page regions. In the following, we will shortly review the characteristics of the SS-Tree.

Definition 3 (SS-Tree). *An SS-Tree in the vector space $\mathbb{R}^n$ is a balanced search tree having the following properties:*

- *All nodes besides the root store between m and M entries. The root is allowed to store between 1 and M entries.*
- *The entries of a leaf node are feature vectors in $\mathbb{R}^n$. The entries of an inner node are nodes, i.e. roots of subtrees.*
- *Each entry is bounded by a spherical page region containing all son entries.*
- *For a leaf node $\mathcal{L}$, the center $C_{\mathcal{L}}$ of the containing hypersphere $B_{\mathcal{L}}$ is the centroid of all contained feature vectors. The radius of $B_{\mathcal{L}}$, $r_{\mathcal{L}}$, is the maximum distance between any entry vector $L \in \mathcal{L}$ and $C_{\mathcal{L}}$.*
- *For an inner node $\mathcal{P}$, the center $C_{\mathcal{P}}$ of the containing hypersphere $B_{\mathcal{P}}$, is the centroid of the centroids $C_{\mathcal{Q}}$ of the contained son pages $\mathcal{Q} \in \mathcal{P}$. The radius $r_{\mathcal{P}}$ is chosen as $\max_{\mathcal{Q} \in \mathcal{P}} (d(C_{\mathcal{P}}, C_{\mathcal{Q}}) + r_{\mathcal{Q}})$.*

In general, the structure of the SS-Tree is quite similar to the structure of the R-Tree [10]. However, the page approximations are spherical instead of rectangular. Additionally, the split heuristics for creating the tree are different. Instead of minimizing

the overlap between two pages, the SS-Tree tries to minimize the variance within the entries of its pages. Thus, when splitting a node, the split heuristics determines the dimension having the largest variance. In this dimension, the partition is selected, which minimizes the variance of the resulting pages and for which both new pages contain at least m entries.

3.3 Principles of AkNN Algorithms

Given two data sets $\boldsymbol{R}$ and $\boldsymbol{S}$, where $\boldsymbol{S}$ is organized in an index structure. We want to find the kNN of all elements $R \in \boldsymbol{R}$. The most trivial approach would be to retrieve the kNN for each element R_i separately by using the index structure organizing $\boldsymbol{S}$. Obviously this is not very efficient, e.g. for two query points (in $\boldsymbol{R}$) with a close proximity the same pages of the index have to be read twice. An efficient algorithm needs to group the points in $\boldsymbol{R}$ so that spatially close points fall into the same group. Afterwards these groups, containing distinct subsets of $\boldsymbol{R}$, are used as query sets.

For the ensuing step, it is important to decide which pages of $\boldsymbol{S}$ need to be examined for finding the kNN of each element contained in a subset $\mathcal{R} \subseteq \boldsymbol{R}$. Therefore, the search space, i.e. the space which could contain kNNs of an element $R \in \mathcal{R}$, needs to be defined. To successively minimize the search space, most algorithms start with the search space containing $\boldsymbol{S}$. Defining a pruning area has the opposite intention: define the space which does not have to be further examined. This means, that all pages of $\boldsymbol{S}$ lying completely in the pruning area can be discarded as candidates for the subset $\mathcal{R}$. Keep in mind that pages not completely covered by the pruning area can potentially contain points lying in the search space and therefore need to be resolved. Since all elements in the search space have to be considered in the further processing of $\mathcal{R}$, the goal is to minimize the search space as strongly and quickly as possible.

Minimizing the search space, respectively maximizing the pruning area, is the task of a pruning criterion. The optimal search space would only contain the kNNs of all $R \in \mathcal{R}$. Therefore, a pruning criterion should try to estimate this optimum as well as possible with low effort.

4 Pruning Criteria

In the following section, we describe MaxDist pruning and trigonometric pruning for the case of a 1-nearest neighbor query for a query set $\mathcal{R}$, using its bounding sphere $B_{\mathcal{R}}$. In Sect. 4.2 we will then explain how to extend the pruning criteria to a kNN query with $k > 1$. The notation used in the course of these analyses is summarized in Table 1.

4.1 MaxDist Pruning

Definition 4 (MaxDist, MinDist). *Given a point P and a spherical region $B_{\mathcal{R}}$ with radius $r_{\mathcal{R}}$ and center $C_{\mathcal{R}}$, the* MaxDist *of P to $B_{\mathcal{R}}$ is defined as: MaxDist$(B_{\mathcal{R}}, P) = d(C_{\mathcal{R}}, P) + r_{\mathcal{R}}$. The* MinDist *of P to $B_{\mathcal{R}}$ is: MinDist$(B_{\mathcal{R}}, P) = d(C_{\mathcal{R}}, P) - r_{\mathcal{R}}$.*

Table 1. Definition of Parameters

n	The dimension of the feature space
$\boldsymbol{R}$	The outer set ($\subseteq \mathbb{R}^n$)
$\boldsymbol{S}$	The inner set ($\subseteq \mathbb{R}^n$)
$\mathcal{R} \subseteq \boldsymbol{R}$, $\mathcal{S} \subseteq \boldsymbol{S}$	Subsets of the outer set or the inner set
$R \in \boldsymbol{R}$, $S \in \boldsymbol{S}$	Points from the outer set or the inner set ($\in \mathbb{R}^n$)
$B_\mathcal{R}$, $B_\mathcal{S}$	Spheres $\equiv$ blocks around the sets $\mathcal{R}$, or $\mathcal{S}$, respectively ($\in \mathbb{R}^n$)
$\mathcal{P}_{\text{cand}}$	Candidate set of points of the inner set ($\subseteq \boldsymbol{S}$)
$r_\mathcal{R}$, $r_\mathcal{S}$	Radius of sphere $B_\mathcal{R}$ or $B_\mathcal{S}$ around set $\mathcal{R}$ or $\mathcal{S}$
$C_\mathcal{R}$, $C_\mathcal{S}$	Center point of sphere $B_\mathcal{R}$ or $B_\mathcal{S}$ around set $\mathcal{R}$ or $\mathcal{S}$
I_i	Some intersection point on the surface of a sphere

Given a spherical region $B_\mathcal{R}$ and an arbitrary data point $S_0 \in \boldsymbol{S}$, we need to distinguish the points of the inner set $\boldsymbol{S}$ which can potentially be the nearest neighbors of any of the points enclosed by $B_\mathcal{R}$ from the points which do not have to be examined. All points S_i with MinDist$(B_\mathcal{R}, S_i) >$ MaxDist$(B_\mathcal{R}, S_0)$ can be pruned.

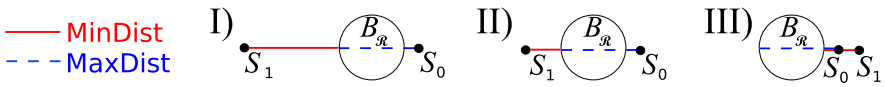


Fig. 1. Example for Min-/MaxDist pruning depending on the spatial position of S_0 and S_1.

Fig. 1 gives an example of comparing MaxDist$(B_\mathcal{R}, S_0)$ with MinDist$(B_\mathcal{R}, S_1)$. In case I), S_1 cannot be the nearest neighbor for any point contained in $B_\mathcal{R}$, because any point in $\mathcal{R}$ is closer to S_0 than to S_1. In case II), there may be points in $B_\mathcal{R}$ (e.g. points lying on the leftmost side of the sphere) which have S_1 as their nearest neighbor and thus S_1 cannot be pruned. Case III) reveals the shortcoming of the pruning criteria mentioned above: Although neither MaxDist$(B_\mathcal{R}, S_0)$ nor MinDist$(B_\mathcal{R}, S_1)$ have changed in comparison to case II), it is clear that S_1 cannot be the nearest neighbor of any point contained in $B_\mathcal{R}$ and could thus be pruned.

This example shows that not only the distances to a query region but also the spatial relations between points of the outer set can play an important role for an optimal pruning criterion.

4.2 Trigonometric Pruning

Hence, we propose to consider the spatial relations of points in order to reduce the search space and thus avoid unnecessary page accesses. Therefore, a point S_i close to the query region $B_\mathcal{R}$ is picked as pruning candidate. By exploiting the trigonometric properties of the spatial relation of S_i and $B_\mathcal{R}$, the search space is reduced.

In order to generalize the correctness of our approach in an n-dimensional space, we will first show the correctness in 2D and afterwards extend the approach to n dimensions. To illustrate our approach, we will use the terms and variable declarations

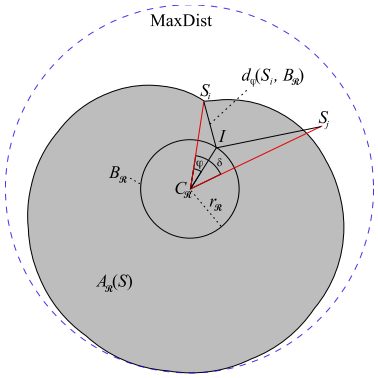


Fig. 2. Trigonometric pruning with active region $A_{\mathcal{R}}(S)$ in comparison with MaxDist pruning.

corresponding to Fig. 2. Afterwards, we show that the distances between a point S and a query region follow the function described in Definition 5 and thus, define a pruning area. Next, we demonstrate how to combine these functions in order to decide whether or not any of the points in $\boldsymbol{S}$ can be pruned (Sect. 4.2). Then, we extend the pruning criterion such that it also holds for pages, meaning it is possible to prune pages and to use pages as pruning candidates. Finally, we give a geometric interpretation of the approach that allows faster calculation of the pruning area.

Defining the Pruning Area The following definition follows directly from the law of cosines, using the triangle $\triangle SC_{\mathcal{R}}I$ between a pruning candidate S, the center of the page $C_{\mathcal{R}}$ and an arbitrary point I on the surface of the page $\mathcal{R}$. This situation is illustrated in Fig. 2. In the following, we will define the Surface Distance describing the distance of S to an arbitrary surface point I:

Definition 5 (Surface Distance $d_\varphi(S, B_{\mathcal{R}})$). *Let $B_{\mathcal{R}}$ be a sphere around the center point $C_{\mathcal{R}}$ with radius $r_{\mathcal{R}}$, containing all elements $R \in \mathcal{R} \subseteq \boldsymbol{R}$ of a subset of the outer set. Let I be a point on the surface of $B_{\mathcal{R}}$, i.e. $\overline{C_{\mathcal{R}}I} = r_{\mathcal{R}}$ and let S be a point from the inner set $\boldsymbol{S}$. The* surface distance *$\overline{SI}$ follows the function*

$$d_\varphi(S, B_{\mathcal{R}}) = \sqrt{\overline{C_{\mathcal{R}}S}^2 + r_{\mathcal{R}}^2 - 2\,\overline{C_{\mathcal{R}}S}\, r_{\mathcal{R}}\, \cos(\varphi)}\,, \tag{2}$$

where $\varphi = \measuredangle SC_{\mathcal{R}}I$ is the angle between $\overrightarrow{C_{\mathcal{R}}S}$ and $\overrightarrow{C_{\mathcal{R}}I}$. If S is equal to the center $C_{\mathcal{R}}$, φ is not defined, thus we set $d_\emptyset(S, B_{\mathcal{R}}) = r_{\mathcal{R}}$ for $S = C_{\mathcal{R}}$.

As illustrated in Fig. 2, the distance between a candidate S and a point I on the surface of the circle around $C_{\mathcal{R}}$ fulfills Equation (2). Thus, we can define the *active region* $A_{\mathcal{R}}(S)$ as the remaining search space after considering S, because any point inside the active region is closer to any position in $\mathcal{R}$ than S is.

Definition 6 (Active Region $A_{\mathcal{R}}(S)$). *Let $B_{\mathcal{R}}$ and $S \in \boldsymbol{S}$ be defined as above. The region $A_{\mathcal{R}}(S)$ contains all points P for which the distance to any point I on the surface*

of $B_\mathcal{R}$ is less than or equal to the distance of S to I:

$$A_\mathcal{R}(S) = \{P \in \mathbb{R}^2 \mid \exists\, I \text{ on } B_\mathcal{R} : d(P,I) \leq d(S,I)\} \tag{3}$$

$$= \{P \in \mathbb{R}^2 \mid \exists\, \varphi \in [0;2\pi] : d_{\varphi-\delta}(P, B_\mathcal{R}) \leq d_\varphi(S, B_\mathcal{R})\}\,, \tag{4}$$

where $\delta = \measuredangle P C_\mathcal{R} S$ is the angle between $\overrightarrow{C_\mathcal{R} S}$ and $\overrightarrow{C_\mathcal{R} P}$.

This definition is equivalent to the union of all circles starting at an intersection point I of $B_\mathcal{R}$ with radius $d(S,I)$. $A_\mathcal{R}(S)$ is visualized as the gray shaded area in Fig. 2. Since the distance condition of (3) automatically holds for points I' *within* $B_\mathcal{R}$, $A_\mathcal{R}(S)$ contains only the points $P \in \boldsymbol{S}$ which can still be closer or equally close to any point $R \in B_\mathcal{R}$ than S and thus it forms a valid search space.

Extension to n Dimensions The definition for the active region can be extended to $n > 2$ dimensions. The points S_i, S_j and $C_\mathcal{R}$ span a 2D hyperplane H in any dimension.

Lemma 1. *Let $A_\mathcal{R}(S_i)$ be the active region of an $S_i \in \boldsymbol{S}$ for a data sphere $B_\mathcal{R}$ with $\mathcal{R} \subseteq \boldsymbol{R}$ and let $S_j \in \boldsymbol{S}$ be another data point. If $S_j \in \boldsymbol{S}$ is not in $A_\mathcal{R}(S_i)$ in the hyperplane H spanned by S_i, S_j and $C_\mathcal{R}$, there cannot be any $R \in B_\mathcal{R}$ s.t. $d(R,S_j) \leq d(R,S_i)$ and S_j can be pruned.*

The proof is omitted due to space limitations. The idea is to reduce the test on whether or not S_j is within the active region of S_i to the test of (3), which implicitly takes place on the common 2D hyperplane H.

Pruning Points Given two candidate points S_i, S_j and a query region $B_\mathcal{R}$ with center $C_\mathcal{R}$ and radius $r_\mathcal{R}$. Let δ be the angle $\measuredangle S_i\, C_\mathcal{R}\, S_j$. Then S_j can be pruned if

$$d_\varphi(S_i, B_\mathcal{R}) < d_{\varphi-\delta}(S_j, B_\mathcal{R}); \forall \varphi \in [0; 2\pi[\,. \tag{5}$$

If $$d_\varphi(S_i, B_\mathcal{R}) > d_{\varphi-\delta}(S_j, B_\mathcal{R}); \forall \varphi \in [0; 2\pi[\tag{6}$$

holds, S_i can be pruned. If none of the conditions hold, neither S_i nor S_j can be pruned. W.l.o.g. φ can be limited to $[0; 2\pi[$. If neither (5) nor (6) holds, there is at least one φ, where $d_\varphi(S_i, B_\mathcal{R}) = d_{\varphi-\delta}(S_j, B_\mathcal{R})$ and thus both S_i and S_j are NN candidates for the outer set $\mathcal{R}$ (see Fig. 3). This condition can be transformed to finding the roots of function $g_{S_i,S_j,B_\mathcal{R}}(\varphi) = d_\varphi(S_i, B_\mathcal{R}) - d_{\varphi-\delta}(S_j, B_\mathcal{R})$. A root of $g_{S_i,S_j,B_\mathcal{R}}(\varphi)$ indicates that $d_\varphi(S_i, B_\mathcal{R})$ and $d_{\varphi-\delta}(S_j, B_\mathcal{R})$ intersect such that neither of the points can be pruned safely. In the case of $C_\mathcal{R} = S_i$ or $C_\mathcal{R} = S_j$, we only need to compare the according MinDists.

As the computation of the roots of $g_{S_i,S_j,B_\mathcal{R}}(\varphi)$ is quite expensive, we calculate the extrema by examining the roots of $g'_{S_i,S_j,B_\mathcal{R}}(\varphi)$ as defined in Equation (7) and test them on opposite signs.

$$g'_{S_i,S_j,\mathcal{R}}(\varphi) = 2\,\overline{C_\mathcal{R} S_i}\, r_\mathcal{R}\, \sin(\varphi) - 2\,\overline{C_\mathcal{R} S_j}\, r_\mathcal{R}\, \sin(\varphi - \delta) \tag{7}$$

$$g'_{S_i,S_j,\mathcal{R}}(\varphi) = 0 \Rightarrow \varphi_{1,2} = \arctan\left(\frac{\overline{C_\mathcal{R} S_i}\sin(a\pi + \delta)}{\overline{C_\mathcal{R} S_j}\cos(a\pi + \delta) - \overline{C_\mathcal{R} S_i}}\right) \tag{8}$$

$$\text{with } a = 0, 2$$

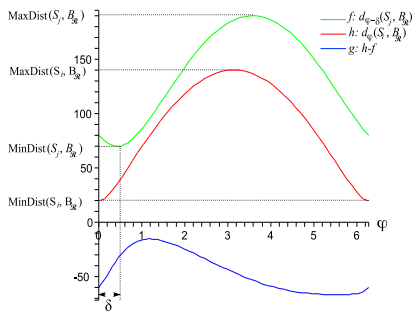


Fig. 3. Distance functions $d_\varphi(S_i, B_\mathcal{R})$, $d_\varphi(S_j, B_\mathcal{R})$ of two candidate points S_i and S_j with $\delta = \measuredangle S_i\, C_\mathcal{R}\, S_j$.

Obviously, there are only two possible roots φ_1 and φ_2 in the domain $[0; 2\pi[$ for $g'_{S_i,S_j,\mathcal{R}}$ because the numerator can only be zero if $\sin(a\pi + \delta) = 0$. Using the signum function, we can now differ between the following cases:

1. $\text{sgn}(g_{S_i,S_j,B_\mathcal{R}}(\varphi_1)) = \text{sgn}(g_{S_i,S_j,B_\mathcal{R}}(\varphi_2))$:
   $\Rightarrow \forall\varphi\colon g_{S_i,S_j,B_\mathcal{R}}(\varphi) \neq 0$ and thus $\forall\varphi\colon d_\varphi(S_i, B_\mathcal{R}) \neq d_{\varphi-\delta}(S_j, B_\mathcal{R})$.
2. $\text{sgn}(g_{S_i,S_j,B_\mathcal{R}}(\varphi_1)) \neq \text{sgn}(g_{S_i,S_j,B_\mathcal{R}}(\varphi_2))$:
   $\Rightarrow \exists\varphi \in [0; 2\pi[\colon d_\varphi(S_i, B_\mathcal{R}) = d_{\varphi-\delta}(S_j, B_\mathcal{R})$.

Only case 1 allows one of the points to be pruned. If $g_{S_i,S_j,B_\mathcal{R}}(\varphi) > 0$ for all φ, S_i can be pruned since its active region $A_\mathcal{R}(S_i)$ completely contains the active region of S_j. Analogously, S_j can be pruned if $g_{S_i,S_j,B_\mathcal{R}}(\varphi) < 0$ holds.

Pruning Pages In order to reduce page accesses, we now extend trigonometric pruning to prune pages without accessing the page itself.

Lemma 2. *Let $B_\mathcal{R}$ be a query region, $B_\mathcal{S}$ a page from the inner set, let S_0 be a pruning candidate for the query region $B_\mathcal{R}$ and let δ be the angle $\measuredangle S_0\, C_\mathcal{R}\, C_\mathcal{S}$. Then, $B_\mathcal{S}$ can be pruned if $d_\varphi(S_0, B_\mathcal{R}) < d_{\varphi-\delta}(C_\mathcal{S}, B_\mathcal{R}) - r_\mathcal{S}, \forall\varphi \in [0; 2\pi[$.*

Proof. According to Sect. 4.2, a point S can be pruned if $g_{S,S_0,B_\mathcal{R}}(\varphi) > 0\ \forall\varphi \in [0; 2\pi[$. Since every point $S \in \mathcal{S}$ is at most $r_\mathcal{S}$ away from the center $C_\mathcal{S}$, the following condition holds because of the triangle inequality:

$$d_{\varphi-\delta}(C_\mathcal{S}, B_\mathcal{R}) - r_\mathcal{S} \leq d_{\varphi-\delta}(S, B_\mathcal{R}), \forall S \in \mathcal{S}.$$

Therefore, all points $S \in \mathcal{S}$ and thus the page bounded by $B_\mathcal{S}$ can be pruned if

$$d_\varphi(S_0, B_\mathcal{R}) < d_{\varphi-\delta}(C_\mathcal{S}, B_\mathcal{R}) - r_\mathcal{S}, \forall\varphi \in [0; 2\pi[\tag{9}$$

□

Figure 4 illustrates the initial situation with a query region $B_\mathcal{R}$, a pruning candidate S_0 and a page $B_\mathcal{S}$ containing points of the inner set. I is a point on the surface of

$B_{\mathcal{R}}$. The page $B_{\mathcal{S}}$ can be safely pruned if its distance to any point I is larger than $\overline{S_0 I}$. Since we have a region $B_{\mathcal{S}}$ rather than a point S, we use MinDist($I, B_{\mathcal{S}}$). The decision whether a page $B_{\mathcal{S}}$ can be pruned, is based on the existence of at least one φ for which $d_\varphi(S_0, B_{\mathcal{R}}) \geq d_{\varphi-\delta}(C_{\mathcal{S}}, B_{\mathcal{R}}) - r_{\mathcal{S}}$. As in the previous section, we compute the maxima of the difference of the two functions:

$$g_{S_0,B_{\mathcal{S}},B_{\mathcal{R}}}(\varphi) = d_\varphi(S_0, B_{\mathcal{R}}) - (d_{\varphi-\delta}(C_{\mathcal{S}}, B_{\mathcal{R}}) - r_{\mathcal{S}}) \tag{10}$$

$$\Rightarrow g_{S_0,B_{\mathcal{S}},B_{\mathcal{R}}}(\varphi) = g_{S_0,C_{\mathcal{S}},B_{\mathcal{R}}}(\varphi) + r_{\mathcal{S}} \tag{11}$$

$$\Rightarrow g'_{S_0,B_{\mathcal{S}},B_{\mathcal{R}}}(\varphi) = g'_{S_0,C_{\mathcal{S}},B_{\mathcal{R}}}(\varphi) \tag{12}$$

Subtracting $r_{\mathcal{S}}$ from $d_{\varphi-\delta}(C_{\mathcal{S}}, B_{\mathcal{R}})$ causes the distance of $C_{\mathcal{S}}$ to the intersection point I to be translated along the y-axis by $-r_{\mathcal{S}}$ units. Hence, the locations of the extrema of $g_{S_0,B_{\mathcal{S}},B_{\mathcal{R}}}(\varphi)$ are not different from the extrema of $g_{S_0,C_{\mathcal{S}},B_{\mathcal{R}}}(\varphi)$ such that $g'_{S_0,B_{\mathcal{S}},B_{\mathcal{R}}}(\varphi)$ can be used for calculating the values of φ for testing (9) on whether or not the page $B_{\mathcal{R}}$ can be pruned.

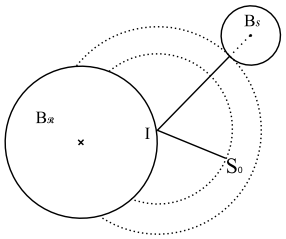


Fig. 4. Distances of the candidates S_0 and B_S to I for $B_{\mathcal{R}}$.

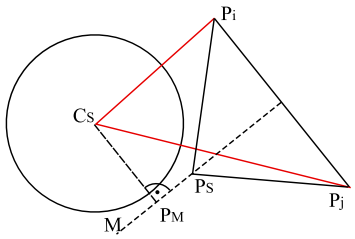


Fig. 5. Geometric interpretation of the pruning criterion. P_j can be pruned, if M does not intersect the circle around C_S.

Geometric Interpretation A computationally cheaper and thus faster result can be achieved by using the following geometric interpretation which is illustrated in Fig. 5: Let P_i be a pruning candidate, let C_S be the center of a page B_S with radius r_S and let P_j be a point to be tested on whether it can be pruned or not. Then, P_j can be pruned if $d_\varphi(P_i, B_S) < d_\varphi(P_j, B_S)$ (as shown in 4.2). This condition is fulfilled if P_i, P_j and P_S build an isosceles triangle $\triangle P_i\, P_j\, P_S$, with $[P_i, P_S]$ and $[P_j, P_s]$, representing the two sides with equal length, so that P_S is located on the perpendicular bisector M of $[P_i, P_j]$. If M does not intersect the circle around C_S, there is no P_S for which $\overline{P_i P_S} = \overline{P_j P_S}$ holds. Thus, P_j can be pruned if $\overline{C_S P_M} > r_S$ with P_M being the orthogonal projection of C_S onto M (cf. Fig. 5). $\overline{C_S P_M}$ can be calculated by Equation (13). Obviously, no additional coordinates than the ones given by P_i, P_j and C_S are needed.

$$\overline{C_S P_M} = \frac{-\overline{C_S P_i}^2 + \overline{C_S P_j}^2}{2\overline{P_i P_j}} \tag{13}$$

Picking Pruning Candidates In this section, we explain which points should be selected as pruning candidates. Generally, points close to the query region result in a smaller active region, so that these points are preferable. The active region is further reduced by comparing a point S_j to all pruning candidates $S_i \in \mathcal{S}_{cand}$, since the resulting active region is equivalent to an intersection of the active regions of all S_i. Hence, more candidate points result in fewer page accesses at the price of an increasing number of distance calculations. The largest benefit is achieved if the pruning candidates are equally distributed around C_S because the resulting active region is minimized in this case.

Extending the idea of trigonometric pruning from ANN to AkNN is very simple: A point or page can be pruned if it is pruned by at least k pruning candidates. Hence, at least k pruning candidates have to exist before defining a search space smaller than the whole space $\mathbb{R}^n$. Resulting from these findings, we define the parameter ϵ, which controls the maximum number of pruning candidates that should be considered for pruning elements from $\boldsymbol{S}$, by setting this number to $k \cdot \epsilon$.

5 The Trigonometric Pruning Algorithm

Algorithm 1 gives an overview of our query algorithm. In the following, we explain the employed data structures and describe the complete algorithm. Finally, we propose an extension for employing multiple pruning techniques.

5.1 Data Structures

Algorithm 1 requires the following data structures:

PQ: A priority queue handling all yet unconsidered pages for a query set $\mathcal{R} \subseteq \boldsymbol{R}$. PQ is organized as a heap with the element with the smallest MinMinDist (extension of MinDist for two pages) to the query region on top.

RES_HT: A hash table storing one priority queue for each query point. Each queue has a capacity of k and maintains the k-nearest neighbors for its query point. The call RES_HT.add($\mathcal{R}$, S_i) adds S_i to the priority queue of $\mathcal{R}$.

PCLIST: A list containing the pruning candidates, ordered by their MinDist to the query region. The maximum size of the list is defined by $\epsilon \cdot k$ with k being the amount of nearest neighbors that should be computed and $\epsilon \geq 1$ being an adjustable parameter. The larger ϵ is chosen, the more pages can be pruned (as PCLIST grows and provides more pruning power) which saves time consuming I/O-accesses at the cost of distance calculations.

5.2 The AkNN Algorithm

Our algorithm starts with grouping the query set $\boldsymbol{R}$ into compact spherical approximations. We use BIRCH [14] to produce a list of compact regions in linear time and organize them in a list. This way, the resulting query regions can be described by compact hyper spheres having a given maximum radius. Since the pruning area also decreases with the radius of a query region, we use the algorithm proposed in [15] to calculate the

Algorithm 1 The AKNN algorithm

```
AkNN(LIST_queryRegions, SSTREE_innerSet, k, RES_HT)
0  : forall R ∈ LIST_queryRegions do
1  :    PCLIST        := new PCLIST(ε · k)
2  :    PQ            := new PQ()
3  :    PQ.add(SSTREE_innerSet.root, -1)
4  :    while PQ.hasElements() do
5  :       if PCLIST.size = ε · k and PQ.smallestDistance > PCLIST.MaxDist_k
6  :          break
7  :       node := PQ.removeFirst()
8  :       if not canBePruned(node, R, PCLIST)
9  :          forall child ∈ node
10 :             if child is an inner node                  // i.e. a B_S from the SS-Tree
11 :                PQ.add(child, MinDist(child, R))
12 :             else                                       // i.e. an S_i ∈ S
13 :                if not canBePruned(child, R, PCLIST)
14 :                   PCLIST.add(child)
15 :                   forall queryPoint ∈ R                // i.e. an R_i ∈ R
16 :                      RES_HT.add(queryPoint, child)
```

approximate smallest enclosing ball (SEB) for the data content of each region. Computing SEBs costs extra CPU time, but it pays off fast with growing $|\boldsymbol{S}|$, as the resulting SEBs' radii are about 10% - 20% smaller than the radii of the original spheres.

The main algorithm now receives a list of query regions from $\boldsymbol{R}$ and the number of nearest neighbors k as input and proceeds as follows:

The query regions are processed successively and independently from each other. First, a new query region is taken from the list. Afterwards, a list for storing pruning candidates (PCLIST) and a priority queue (PQ) are initialized and the root of the SS-Tree storing $\boldsymbol{S}$ is put into the page list PQ. The pages in PQ are ordered by the MinMinDist to the currently processed query region ($B_{\mathcal{R}}$). The SS-Tree is then traversed in a best-first manner, as proposed by Hjaltason and Samet [16]. This is done by always removing the first node from the priority queue PQ processing it and putting all child nodes (if available) back to PQ, as long as the queue is not empty or the stopping criterion is triggered: The search for other kNN-candidates can be stopped if the MinMinDist of the current node to $\mathcal{R}$ is larger than the MaxDist of the k-th pruning candidate. In that case, the node and all its successors in the PQ can be pruned (cf. Sect. 4.1).

If the algorithm cannot be terminated in this way, it tests whether the current node can be pruned. This is done by the canBePruned() function, which prunes the node or point w.r.t. the pruning techniques introduced in Sect. 4.2. If the node cannot be pruned and its children are nodes of the SS-Tree, they are all added to the page list PQ. If the children are data points, another test on whether or not they can be pruned ensues. If the test is negative, they are added to the PCLIST and to all priority queues (via RES_HT) of query points contained in $\mathcal{R}$. Let us note that it is possible that the points will be eliminated from both data structures at a later point of time.

5.3 Combining Trigonometric Pruning with Other Criteria

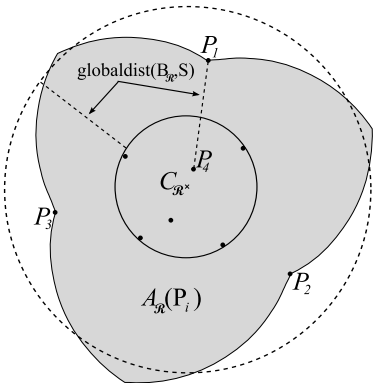


Fig. 6. The gray shaded area shows the active region used by trigonometric pruning (see Definition 6) which results from the intersection of the three pruning areas of P_1, P_2, P_3. The dashed line $[P1, P4]$ indicates the GlobalDist$(B_{\mathcal{R}}, S)$ which defines the pruning area (dashed outer circle) used by BNN in case of a 1NN query.

In this section, we will discuss the use of different pruning criteria to further increase the performance. Even if employing additional pruning techniques causes additional computational cost, the cost is negligible if the combination yields a significant decrease of the search space and thus saves page accesses. In order to maximize the effect of combining different pruning criteria, it is important that the combined criteria perform always at least as well as one criterion alone. In our case, this can be achieved when trigonometric pruning (TP) is combined with a pruning criterion that also defines a pruning region so that the intersection of both regions can be used as a new pruning region.

In our experiments, we combine TP with BNN [9] which turned out to be one of the best pruning criteria we examined. The fact that the original BNN algorithm is proposed to be applied on rectangular page regions is not a problem, as it can be transferred to spherical page regions, such that BNN can be applied to the SS-Tree as well. The authors of BNN propose to query the inner set $\mathcal{S}$ with compact groups $B_{\mathcal{R}i} \in \mathcal{R}$ whereas each $R_i \in B_{\mathcal{R}i}$ organizes its NNs and according maximum-NNDist$(R_i, \boldsymbol{S})$. The largest value of NNDist$(R_i, \boldsymbol{S})$ of all $R_i \in B_{\mathcal{R}i}$ defines the GlobalDist$(B_{\mathcal{R}i}, \boldsymbol{S})$ of $B_{\mathcal{R}i}$. Pages $\mathcal{S}$ can then be pruned if MinDist$(B_{\mathcal{R}i}, \mathcal{S}) >$ GlobalDist$(B_{\mathcal{R}i}, \boldsymbol{S})$ as $\mathcal{S}$ cannot contain a NN for any $R_i \in B_{\mathcal{R}i}$. The disadvantage of the BNN algorithm compared to TP is that the spatial relation to other NN candidates is ignored. This can lead to the case shown in Fig. 6 where GlobalDist(C, S) degenerates and converges to MaxDist in the worst case.

6 Experimental Evaluation

In the following, we outline the evaluation process of the methods suggested in this paper. We compare them w.r.t. to I/O cost to the state-of-the-art methods BNN [9] and MBA [6]. In order to make the approaches competitive for data sets of larger dimensionality, we adapted the algorithm to the X-Tree [17] and the SS-Tree [5] instead of using an R-Tree [10]. We display results for the following algorithmic settings:

- MP: AkNN-algorithm, using MaxDist on the SS-Tree
- XBNN: BNN, using **S** indexed in an X-Tree
- SSBNN: BNN, using **S** indexed in an SS-Tree
- TP: AkNN-algorithm, using TP (see Algorithm 1)

– TP+SSBNN: AkNN-algorithm combining TP and BNN in an SS-Tree
– MBA: MBA algorithm [6]

We have evaluated the algorithms on the following three real-world data sets, which were also used in the evaluation of [6]:

– TAC: Twin Astrographic Catalog Version 2 [18]: a library of stellar coordinates resulting in a set of 705 099 two-dimensional star representations.
– FC: Forest Cover Type data set, retrieved from the UCI KDD repository [19]: 581 012 data points with 10 real-valued attributes, each representing a 30x30 meter square of a forest region.
– COREL: Corel color histograms, which are also available at the UCI KDD repository [19]. They consist of 32-dimensional color histograms for 68 040 images.

For all experiments we used a third of the data sets as outer set, the rest as inner set. The page size was set to 1KB for TAC and 4KB for FC and COREL.

All experiments were run on a 64bit Intel® Xeon™CPU (3GHz) with 16G RAM, under Microsoft Windows 2003 R2, with Service Pack 1.

6.1 Results

In this section, we present the results of our experimental evaluation for the named algorithms and data sets. Additionally, we will describe the effect of the employed parameters on the query performance. Due to space limitations, we cannot display all settings for all data sets, but we give representative examples of the parameters involved.

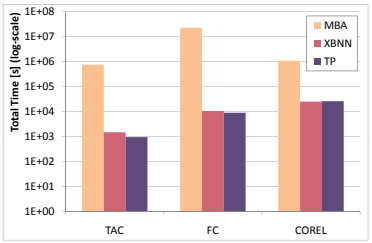
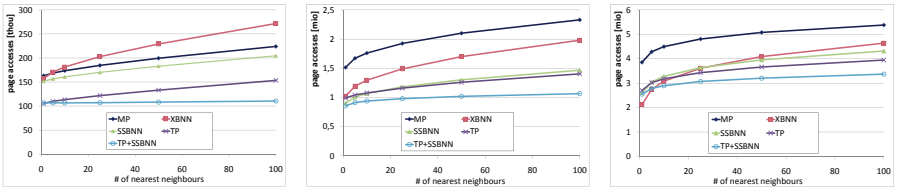


Fig. 7. Comparison of MBA, XBNN and TP for $k = 10$. The figure displays the sum of IO and CPU times for one AkNN query.

Comparison to Other Approaches In order to compare the different approaches, we performed all-10-NN queries with the mentioned AkNN-algorithms on the three data sets and measured the CPU-time and the number of page accesses. To combine these two measures, we considered each logical page access with 8ms. Fig. 7 illustrates that TP and BNN perform orders of magnitudes better than MBA on all used data sets in matters of the overall runtime. Similar results were obtained with the parametric settings used for the experiments described in the following paragraphs. Therefore, we excluded the results of MBA in the further diagrams because the required scaling would conceal interesting effects.

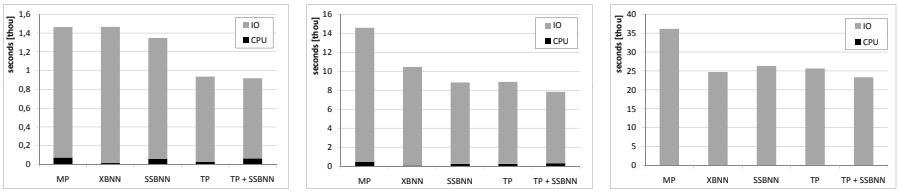
Dependency on k In Fig. 8 we compare the named algorithm w.r.t. the number of retrieved neighbors k on all three data sets. Though most methods compare their performance on rather small values of k, larger values of k are often of large practical

relevance. In the context of search engines, it is quite common to provide large result sets which are ordered with respect to the similarity to the query. Since the performance mainly relies on the I/O operations, only page accesses are measured. While TP and BNN perform comparable for smaller values of k, TP clearly shows a better performance for larger values of k. For all data sets, our new combined approach (TP + SSBNN) performed significantly better than all compared approaches.



(a) AkNN on TAC (b) AkNN on FC (c) AkNN on COREL

Fig. 8. Page accesses for AkNN queries on different data sets by all algorithms increasing k.



(a) A-10-NN on TAC (b) A-10-NN on FC (c) A-10-NN on COREL

Fig. 9. Performance of 10-NN queries on the different dataset with all mentioned algorithms.

Dependency on the Choice of ϵ The ϵ parameter regulates the number of pruning candidates and thus, it is the most important parameter for tuning the AkNN-algorithm based on TP. Larger ϵ cause a smaller search space by the price of an increased number of distance calculations. In Fig. 10, we summarize the effect of ϵ on the performance. As expected, the number of distance calculations grows with increasing ϵ due to the larger number of pruning candidates. Contrarily, more pruning candidates lead to a tighter pruning effect, and thus the number of page accesses is reduced when increasing ϵ. For all experiments in this paper we set $\epsilon = 5$.

Overall Runtime Performance The direct comparison in matters of runtime is shown in Fig. 9 and yields the following insights:

As most operations in the field of databases, the AKNN operation is I/O bound, as CPU time is mainly consumed by distance calculations and represents only a small portion of the overall performance.

Comparing MP and TP shows that TP reduces the page accesses by 30% - 40% in contrast to MP. This is a consequence of the reduction of the search space as shown in

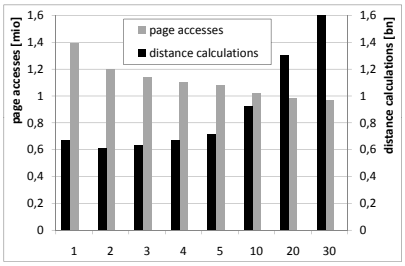


Fig. 10. ϵ of TP for an A-10-NN query on FC.

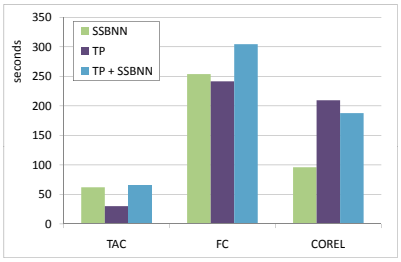


Fig. 11. CPU-times on different data sets.

Fig. 2. This effect is very stable over all data sets, suggesting its independence from the amount of dimensions.

The implementations of BNN on different underlying index structures (XBNN and SSBNN) show that the SS-Tree is at least comparable to the X-Tree in the performed experiments. Only for the 32-dimensional COREL data set, the X-Tree leads to a slightly better performance. Considering that the SS-Tree is not as suitable for large dimensions as the X-Tree, these results demonstrate that spherical page regions are well-suited for the AkNN problem.

The effect of TP compared to BNN is dependent on the data set. On the TAC data set, TP clearly outperforms SSBNN by about 30%. On the FC and the COREL data set, both pruning criteria perform similar and are both outperformed by the combination of TP and SSBNN which reduces the costs by $8\% - 15\%$.

This demonstrates that the proposed combination nicely compensates the additional computation costs and that the better approximation of the search space can significantly decrease the amount of page accesses.

CPU Consumption As seen in Sect. 6.1, all AkNN-algorithms are clearly I/O bound. However, through the use of buffers and caching strategies, the page accesses can be dramatically reduced. Therefore, it is important to show that TP and TP+SSBNN apply only a negligible computational overhead compared to SSBNN. The results shown in Fig. 11 support this claim. Only at the high-dimensional COREL data set, SSBNN is significantly faster with respect to CPU-time. Let us note that also in the presence of buffers, the main challenge for AkNN-algorithms is to reduce the I/O operations. Therefore, it is often beneficial for the overall runtime to reduce I/O operations by the cost of CPU-time.

For all our compared methods it is necessary to group the outer set $\boldsymbol{R}$ into compact query regions. In [9], the authors use a grouping approach based on a bulk-load using the Hilbert order with a maximum threshold for the MBR volume and group size. They argue that this procedure runs in linear time and leads to better page approximations than, for instance, the data pages resulting from indexing $\boldsymbol{R}$ like $\boldsymbol{S}$. This grouping procedure can also be transferred to spherical page approximations. However, we chose BIRCH [14], which is a clustering algorithm with linear runtime. Even though this procedure uses spherical page approximations, we experienced that the groupings result-

ing from BIRCH show a better suitability even for the AkNN-algorithm based on the X-Tree (XBNN) than those resulting from Hilbert grouping. Fig. 12 shows the experiments on the three data sets for the two grouping algorithms. The Hilbert groups have been created by limiting the group volume to the average data page size of the inner set's X-Tree. The parametric setting of BIRCH has been chosen such that the number of groups is comparable to the Hilbert grouping. For the algorithms based on spherical page approximations, the results showed even more evidence for using BIRCH as grouping algorithm.

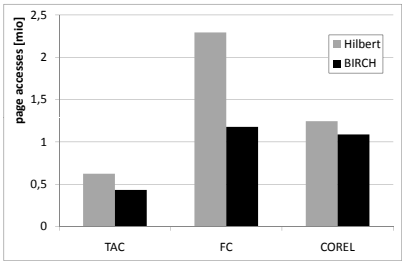


Fig. 12. Grouping procedures of the outer set, validated via XBNN.

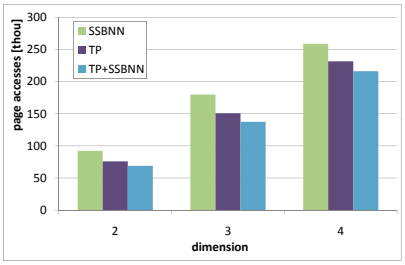


Fig. 13. A-10-NN queries for synthetic clustered data sets of varying dimensions

Synthetic Data Sets We have also compared SSBNN, TP and TP+SSBNN on several synthetic data sets. Fig. 13 displays the results of A-10-NN queries on clustered datasets of varying dimensions. The 3000 clusters in the example data set of size 600,000 have been generated using a Gauss-like process: For each cluster a centroid has been sampled from a uniform distribution. The standard deviation used for generating the cluster points was chosen uniformly for each dimension. The experiments show that the combination of TP and SSBNN outperforms TP by approximately 10% and that it outperforms SSBNN by 20 to 30%. We note that this effect is stable for all dimensions – also for dimensions not displayed here due to space limitations.

7 Conclusions

In this paper, we introduced a novel approach for processing AkNN queries. Unlike previous approaches, our method is based on spherical page regions and thus, we apply it using an SS-Tree. To exclude pages from the search as early as possible, our algorithm introduces trigonometric pruning which allows to consider an asymmetric pruning area around a given query approximation. We propose a new AkNN algorithm which is based on this new pruning method. Afterwards, we further extend AkNN processing to employ multiple pruning criteria. Thus, it is possible to construct even tighter bounds around the remaining search space and thus to further decrease the number of necessary page accesses. In our experimental evaluation, we demonstrate that our proposed methods decrease the all-over runtime as well as the page accesses necessary to process AkNN queries on three real-world data sets. Especially for larger values for k, our new method considerably improves the query times.

The application of our pruning principle on spherical page regions allows a simple computation. In future work, we aim at extending the principle of trigonometric pruning to non-spherical page regions. We believe that transferring the principle to other approximations, such as MBRs, will open up new possibilities. Additionally, we investigate the transfer of our approach to other problems involving similarity search.

Acknowledgements

This research has been supported in part by the THESEUS program in the MEDICO and CTC projects. They are funded by the German Federal Ministry of Economics and Technology under the grant number 01MQ07020. The responsibility for this publication lies with the authors.

References

1. Böhm, C., Krebs, F.: The k-nearest neighbor join: Turbo charging the KDD process. KAIS **6**(6) (2004) 728–749
2. Lowe, D.: Object recognition from local scale-invariant features. In: International Conference on Computer Vision, Corfu, Greece. (1999) 1150–1157
3. Sankaranarayanan, J., Samet, H., Varshney, A.: A fast all nearest neighbor algorithm for applications involving large point-clouds. Comput. Graph. **31**(2) (2007) 157–174
4. Breunig, M.M., Kriegel, H.P., Ng, R., Sander, J.: LOF: Identifying density-based local outliers. In: Proc. SIGMOD. (2000)
5. White, D.A., Jain, R.: Similarity indexing with the SS-tree. In: Proc. ICDE. (1996) 516–523
6. Chen, Y., Patel, J.: Efficient evaluation of all-nearest-neigbor queries. In: Proc. ICDE. (2007)
7. Samet, H.: Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics). Morgan Kaufmann (2006)
8. Xia, C., Lu, H., Ooi, B.C., Hu, J.: GORDER: An efficient method for KNN join processing. In: Proc. VLDB. (2004)
9. Zhang, J., Mamoulis, N., Papadias, D., Tao, Y.: All-nearest-neighbors queries in spatial databases. In: Proc. SSDBM. (2004)
10. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD. (1984) 47–57
11. Yu, C., Cui, B., Wang, S., j. Su: Efficient index-based KNN join processing for high-dimensional data. Information and Software Technology **49**(4) (2007)
12. Jagadish, H.V., Ooi, B., Tan, K.L., Yu, C., Zhang, R.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM TODS **30**(2) (2005)
13. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: an efficient access method for similarity search in metric spaces. In: Proc. VLDB. (1997)
14. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In: Proc. SIGMOD. (1996) 103–114
15. Bâdoiu, M., Clarkson, K.L.: Smaller core-sets for balls. In: SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2003) 801–802
16. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: Proc. SSD. (1995)
17. Berchtold, S., Keim, D.A., Kriegel, H.P.: The X-Tree: An index structure for high-dimensional data. In: Proc. VLDB. (1996)
18. Zacharias, N., Zacharias, M.I.: The twin astrographic catalog on the hipparcos system. The Astronomical Journal **118**(5) (1999) 2503–2510
19. Hettich, S., Bay, S.D.: The UCI KDD archive. http://kdd.ics.uci.edu (1999)