Efficient Nearest-Neighbour Searches Using Weighted Euclidean Metrics

Ruth Kurniawati*, Jesse S. Jin, and John A. Shepherd

School of Computer Science and Engineering
University of New South Wales
Sydney 2052, Australia

Abstract. Building an index tree is a common approach to speed up the k nearest neighbour search in large databases of many-dimensional records. Many applications require varying distance metrics by putting a weight on different dimensions. The main problem with k nearest neighbour searches using weighted euclidean metrics in a high dimensional space is whether the searches can be done efficiently. We present a solution to this problem which uses the bounding rectangle of the nearest-neighbour disk instead of using the disk directly. The algorithm is able to perform nearest-neighbour searches using distance metrics different from the metric used to build the search tree without having to rebuild the tree. It is efficient for weighted euclidean distance and extensible to higher dimensions.

Keywords. nearest-neighbour search, weighted Euclidean distance, coordinate transformations, R-trees, spatial access methods

1 Introduction

The problem of k nearest-neighbour search using a distance metric can be stated as follows: given a collection of vectors and a query vector find k vectors closest to the query vector in the given metric. Nearest-neighbour search is an important operation for many applications. In image databases utilizing feature vectors [13, 12, 7], we want to find k images similar to a given sample image. In instance-based learning techniques [1, 14], we would like to find a collection of k instance vectors most similar to a given instance vector in terms of target attributes. Other applications include time-series databases [6], non-parametric density estimation [9], image segmentation [2], etc.

In the k-nearest-neighbour search process, we have a region defined by the k-th farthest neighbour found so far. Depending on the distance metric used, this area could be a circle (for unweighted euclidean distance), diamond (for manhattan distance), ellipse (for weighted euclidean distance), or other shapes. At the beginning of the search process, this area is initialized to cover the whole

* Corresponding author: E-mail: ruthk@cse.unsw.edu.au; Telephone: 61-2-9385-3989; Fax: 61-2-9385-1813

data space. As we find closer neighbours, the area will get smaller. We call this area "the nearest-neighbour disk"[1]. The search proceeds downward from the top of the tree, checking tree nodes covering smaller area as it deepens. We usually use a branch-and-bound algorithm and try to choose the nodes which are most likely to contain the nearest neighbour first. This is done in order to shrink the nearest-neighbour disk as soon as possible. We can safely ignore nodes whose bounds do not intersect the nearest-neighbour disk.

Many algorithms have been proposed for finding k nearest neighbours more efficiently than a sequential scan through data. These algorithms involve building a spatial access tree, such as an R-tree [10], PMR quadtree [11], k-d-tree [3], SS-tree [22], or their variants [18, 8, 19, 16]. Spatial access methods using such trees generally assume that the distance metric used for building the structures is also the one used for future queries. However, many applications require varying distance metrics. For example, when the the similarity measure is subjective, the distance metric can vary between different users. Another example comes from the instance-based learning [1, 14], where the distance metric is learnt by the algorithm. The distance metric will change as more instances are collected. The tree structures above cannot be easily used if the distance metric in the query is different to the distance metric in the index tree. One approach is to map the index tree into the new metric space, but this is equivalent to rebuilding the tree, and therefore, computationally expensive. Another approach is to map the k nearest neighbour bounding disk into the index metric. For example, if we do a query using the Minkowski distance metric on a tree built in euclidean distance, the corresponding nearest neighbour bounding region will be of a diamond shape. Similarly, a nearest neighbour disk in the euclidean distance with different weighting will produce an ellipse. Because of the wide variety of shapes than can be produced, calculating the intersection of the new nearest neighbour bounding envelope and the bounding envelopes of index tree nodes is not trivial and can be computationally expensive.

In this paper, we present an efficient k-nearest-neighbour search algorithm using weighted euclidean distance on an existing spatial access tree. We start by describing a commonly used approach [4]. We identify the drawbacks of this approach and introduce our method. The theoretical foundations and detailed implementation of our method in two dimensional space are given. We also give an extension of the algorithm in higher dimensional spaces. The experiments (up to 32 dimensions) were carried out on an R-tree variant, the SS$^+$-tree [16], (although the proposed approach can be used in any hierarchical structure with a bounding envelope for each node).

[1] We use the 2-dimensional term "disk" regardless of the number of dimensions. To be strict, we should use disk only for 2-dimensional spaces, ellipsoid for 3-dimensional spaces and hyper-ellipsoid for spaces with dimensionality greater than 3.

2 Methods for finding the k nearest-neighbours in weighted euclidean distance

Tree structured spatial access methods organize the data space in a hierarchical fashion. Each tree node in these structures has a bounding hyper-box (R-trees), bounding hyper-planes (k-d-trees), or a bounding hyper-sphere (SS-trees), giving the extent of its child nodes. The root node covers the whole data space which is split among its children. The process is repeated recursively until the set of points in the covered space can fit in one node. To simplify the discussion, we use two dimensional terms for all geometric constructs in the rest of this section and next section.

2.1 Faloutsos' algorithm for k-nn search in a weighted euclidean distance

If the distance metric in the query is consistent with the metric in the index tree, detecting the intersection between the k nearest-neighbour disk and a node can be performed by a simple comparison of the distance of the k-th neighbour to the query vector and the range of the node's bounds in each dimension. When the distance metric is not consistent with the metric in the tree structure, a commonly used technique [4, 5, 15] to find the k-nearest-neighbours is as follows (For easy understanding, we take an example, in which the index tree is built in unweighted euclidean distance and the query is issued in a weighted euclidean distance, and visualize the process in Fig. 1):

- First, we define a distance metric which is a lower bound to the weighted distance. It can be a simple transform to both metrics in the query and in the index tree. Usually, the same metric as in the index tree is used.
- We then search for k nearest neighbours to obtain a bound (LBD in Fig. 1).
- Transferring the distance metric to the query distance metric, we obtain a new bounding shape for the k-nn disk (ED in Fig. 1). The maximum distance D can then be calculated.
- Next, we issue a range query to find all vectors within the distance D from the query point (All vectors in RQD, as shown in Fig. 1). The number of vectors returned from the range query will be greater or equal to k.
- We sort the set from the last step and find the true k nearest neighbours.

From Fig. 1, we can see that the circular range query area (RQD) is much larger than the actual k nearest-neighbour disk. This is not desirable, since a larger query area means we have to examine more tree nodes.

Another drawback of the approach is that we have to search the tree twice. The first scan finds the k nearest-neighbours and the second scan finds all the vectors located within the range D (a range query). Sorting is also needed for selecting first k nearest neighbours within range D.

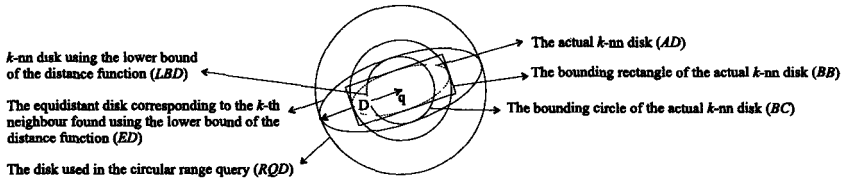

Fig. 1. The evolution of search areas in Faloutsos' approach and our bounding rectangle.

2.2 An efficient k-nn search in variable metrics using the bounding rectangle

We propose an efficient k-nn search with variable metrics. The approach is based on the observation that a transform between the query metric and the index metric exists and transforming the k-nn disk is much simpler than transforming all tree nodes. Using this transformation, we can find the length of the major axis of the k nearest-neighbour disk[2]. However, conducting the search using the ellipse directly is not always desirable. For d-dimensional vectors we will have to calculate the intersection of $(d-1)$-dimensional hyper-planes and a d-dimensional hyper-ellipse. If the intersection is not empty, the result will be a $(d-1)$-dimensional hyper-ellipse. Hence in our solution we use a simpler shape to "envelope" the k nearest-neighbour disk.

We propose a rectangular envelope (the bounding rectangle BB of the k nearest-neighbour disk is shown in Fig. 1). The bounding rectangle can be obtained from calculating the length of all axes of the k-nearest-neighbour disk. Another simpler alternative is by calculating the bounding circle of the k nearest-neighbour disk (BC in Fig. 1). The radius of the bounding circle is equal to the half of ellipse's major axis. Theoretical analysis and detailed calculation will be described in Sect. 3.

Comparing with Faloutsos' algorithm, our approach has two advantages. The bounding rectangle is much compact than Faloutsos' query disk (see RQD and BB/BC in Fig. 1). The advantage is significant if the ratio between the largest and the smallest axes of the nearest neighbour disk is large. There is no need for the second search which is also a significant saving if the level of the tree is high.

3 Theory and implementation

In this paper, we will limit our discussion to the *weighted euclidean distance* $\mathrm{d}(\mathbf{x},\mathbf{y},\mathbf{W}) = \sqrt{((\mathbf{x}-\mathbf{y})^T\mathbf{W}(\mathbf{x}-\mathbf{y}))}$ with a symmetrical weight matrix $\mathbf{W}$. The approach described in this paper can be adapted directly to distances in the

[2] Some books define the axis length of an ellipse as the distance from the centre to perimeter along the axis. The definition we use is the distance from perimeter to perimeter along that axis.

class $\mathrm{d}(\mathbf{x}, \mathbf{y}, \mathbf{W}) = \sum_{i=1}^{d}(x_i - y_i)^p$, where d is the number of the dimensions and $p = 1, 2, 3, \ldots$. We require that the matrix $\mathbf{W}$ is positive definite, i.e. it has positive pivots, which implies that we will always be able to decompose the matrix $\mathbf{W}$ into $\mathbf{Q}^T\mathbf{Q}$ using Cholesky factorization [20]. In practice, we usually deal with symmetrical weight matrices, since non-symmetrical weight matrices means that the distance measured from $\mathbf{x}$ to $\mathbf{y}$ is different from that measured from $\mathbf{y}$ to $\mathbf{x}$.

As have been defined informally in Sect. 2, the *nearest-neighbour disk* is the area bounded by the equidistant points with the current k-th nearest point to the query measured using a particular distance metric. The term "point" used here and in the rest of the paper is interchangeable with "vector".

A particular distance metric will define a *space* where the distance is used for measurements. An euclidean distance defines an euclidean space. The k nearest-neighbour disk is circular in the same euclidean space. If the Euclidean metric in query has a different weight on the metric used in the index, the k nearest-neighbour disk is an ellipse in that space.

3.1 Theoretical foundations

Assume we have two points $\mathbf{x_1}$ and $\mathbf{y_1}$ in a space S_1 where we use unweighted euclidean distance (see Fig. 2). Applying a transformation defined by the matrix $\mathbf{Q}$ will bring every point in S_1 into another space S_2. Suppose $\mathbf{x_1}$ and $\mathbf{y_1}$ are mapped into $\mathbf{x_2}$ and $\mathbf{y_2}$, respectively. If the transformation includes scaling and rotation operations, then the distance between $\mathbf{x_1}$ and $\mathbf{y_1}$ in S_1 will equal a weighted euclidean distance between $\mathbf{x_2}$ and $\mathbf{y_2}$ where the weight matrix $\mathbf{W}$ equals $\mathbf{Q}^T\mathbf{Q}$. This means that a circle in S_1 will be transformed into an ellipse in S_2 since the equivalent distance in S_2 is a weighted euclidean distance.

Observation 1. *The weighted distance* $d(\mathbf{x_2}, \mathbf{y_2}, \mathbf{W})$ *with* $\mathbf{W} = \mathbf{Q}^T\mathbf{Q}$ *equals* $d(\mathbf{x_1}, \mathbf{y_1}, \mathbf{I})$, *where* $\mathbf{x_2} = \mathbf{Q}\mathbf{x_1}$, $\mathbf{y_2} = \mathbf{Q}\mathbf{y_1}$, *and* $\mathbf{I}$ *is the identity matrix.*

Proof. Let $\mathbf{W} = \mathbf{Q}^T\mathbf{Q}$, $\mathbf{x_1} = \mathbf{Q}\mathbf{x_2}$, and $\mathbf{y_1} = \mathbf{Q}\mathbf{y_2}$.

$$\begin{aligned}
\mathrm{d}(\mathbf{x_2}, \mathbf{y_2}, \mathbf{W}) &= \sqrt{(\mathbf{x_2} - \mathbf{y_2})^T\mathbf{W}(\mathbf{x_2} - \mathbf{y_2})} \\
&= \sqrt{(\mathbf{x_2} - \mathbf{y_2})^T\mathbf{Q}^T\mathbf{Q}(\mathbf{x_2} - \mathbf{y_2})} = \sqrt{(\mathbf{Q}(\mathbf{x_2} - \mathbf{y_2}))^T\mathbf{Q}(\mathbf{x_2} - \mathbf{y_2})} \\
&= \sqrt{(\mathbf{Q}\mathbf{x_2} - \mathbf{Q}\mathbf{y_2})^T(\mathbf{Q}\mathbf{x_2} - \mathbf{Q}\mathbf{y_2})} = \sqrt{(\mathbf{x_1} - \mathbf{y_1})^T(\mathbf{x_1} - \mathbf{y_1})} \\
&= \mathrm{d}(\mathbf{x_1}, \mathbf{y_1}, \mathbf{I})
\end{aligned}$$

The transformation matrix $\mathbf{Q}$ defines the scaling and rotation operations which map the circular k nearest-neighbour disk in unweighted euclidean space into an ellipse in the weighted euclidean space. This ellipse is defined by all axes and the query point. As described in Sect. 2, to obtain the bounding box, we need to calculate the length of all the axes and the orientation of the major axis. The solution can be derived from Theorem 2.

Theorem 2. *If we apply a transformation* $\mathbf{Q}$ *to all points of a circle (with radius* r*), the resulting points form an ellipse whose centre is the same as the circle and the length of its axes equals* $2r$ *times the square root of eigenvalues of* $\mathbf{Q}^T\mathbf{Q}$.

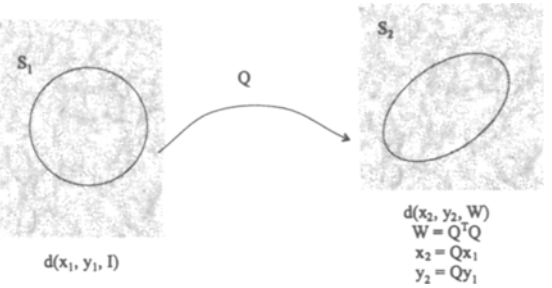

Fig. 2. The transformation between a normal euclidean space and a weighted euclidean space.

Proof. We provide the proof for the two dimensional case. A general proof can be found in [17].

The proof of this theorem depend on a lemma stating that *a real symmetric $n \times n$ matrix has n real eigenvalues, and n linearly independent and orthogonal eigenvectors.* The proof of the lemma can be found in [21].

Let $\mathbf{k_2} = \mathbf{Qk_1}$ where $\mathbf{k_1}$ is an arbitrary point in our original space (S_1 as shown in Fig. 2). Let the circle centre $\mathbf{q_2} = \mathbf{Qq_1}$. The squared distance of $\mathbf{k_2}$ and $\mathbf{q_2}$ is $(\mathbf{Qk_1} - \mathbf{Qq_1})^T \cdot (\mathbf{Qk_1} - \mathbf{Qq_1}) = (\mathbf{k_1} - \mathbf{q_1})^T\mathbf{Q}^T\mathbf{Q}(\mathbf{k_1} - \mathbf{q_1})$. Since $\mathbf{Q}^T\mathbf{Q}$ is real and symmetric, according to the lemma, it has two real eigenvalues and two orthogonal eigenvectors. Let $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ be the unit eigenvectors corresponding to eigenvalues λ_1 and λ_2, respectively.

These orthogonal eigenvectors span $\Re^2$, and hence $(\mathbf{k_1} - \mathbf{q_1})$ can be written as a linear combination of $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. Suppose $(\mathbf{k_1} - \mathbf{q_1}) = r\cos(\theta)\boldsymbol{e}_1 + r\sin(\theta)\boldsymbol{e}_2$.

$$
\begin{aligned}
&(\mathbf{k_2} - \mathbf{q_2})^T(\mathbf{k_2} - \mathbf{q_2}) = (\mathbf{k_1} - \mathbf{q_1})^T\mathbf{Q}^T\mathbf{Q}(\mathbf{k_1} - \mathbf{q_1}) \\
&= (r\cos\theta\boldsymbol{e}_1^T + r\sin\theta\boldsymbol{e}_2^T)(r\cos\theta\mathbf{Q}^T\mathbf{Q}\boldsymbol{e}_1 + r\sin\theta\mathbf{Q}^T\mathbf{Q}\boldsymbol{e}_2) \\
&= (r\cos\theta\boldsymbol{e}_1^T + r\sin\theta\boldsymbol{e}_2^T)(r\cos\theta\lambda_1\boldsymbol{e}_1 + r\sin\theta\lambda_2\boldsymbol{e}_2) \\
&= r^2\lambda_1\cos^2\theta + r^2\lambda_2\sin^2\theta
\end{aligned}
$$

Square distance $r^2\lambda_1\cos^2\theta + r^2\lambda_2\sin^2\theta$ has its extreme values at $\theta =$ multiplies of $\frac{\pi}{2}$ – that is, when we are in $\boldsymbol{e}_1$ or $\boldsymbol{e}_2$ directions. The values at those points are $r^2\lambda_1$ and $r^2\lambda_2$, respectively. Using our definition of the ellipse axis, the length of two axes are $2r\sqrt{\lambda_1}$ and $2r\sqrt{\lambda_2}$, respectively.

□

We can write the weight matrix in diagonal form: $\mathbf{W} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{-1}$, where $\mathbf{E}$ is an orthonormal matrix whose columns are the eigenvectors of $\mathbf{W}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix of the corresponding eigenvalues. Since $\mathbf{E}$ is orthonormal, then $\mathbf{E}^{-1} = \mathbf{E}^T$. Hence:

$$
\begin{aligned}
\mathrm{d}(\mathbf{x}, \mathbf{y}, \mathbf{W}) &= \sqrt{((\mathbf{x} - \mathbf{y})^T\mathbf{W}(\mathbf{x} - \mathbf{y}))} \\
&= \sqrt{((\mathbf{x} - \mathbf{y})^T\mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{-1}(\mathbf{x} - \mathbf{y}))} \\
&= \sqrt{((\mathbf{E}^{-1}\mathbf{x} - \mathbf{E}^{-1}\mathbf{y})^T\boldsymbol{\Lambda}(\mathbf{E}^{-1}\mathbf{x} - \mathbf{E}^{-1}\mathbf{y}))}
\end{aligned}
$$

We can see the last line of the equation as another weighted distance with diagonal weight matrix $\mathbf{\Lambda}$. Because $\mathbf{\Lambda}$ is diagonal, the nearest neighbour disk in this space has axes parallel to the coordinate axes. The size of the nearest neighbour disk is the same in both spaces since $\mathbf{E}$ is orthonormal. Suppose $\mathbf{x}_\mathrm{l}$ and $\mathbf{y}_\mathrm{l}$ are in this space, then $\mathbf{x}_\mathrm{l} = \mathbf{E}^{-1}\mathbf{x}$ and $\mathbf{y}_\mathrm{l} = \mathbf{E}^{-1}\mathbf{y}$. We can see $\mathbf{E}$ as the rotation matrix that rotates the nearest neighbour disk. Since $\mathbf{x} = \mathbf{E}\mathbf{x}_\mathrm{l}$ and $\mathbf{y} = \mathbf{E}\mathbf{y}_\mathrm{l}$, then the desired rotation is $\mathbf{E}$. The length of the disk's axis in the direction given by the i-th column of $\mathbf{E}^{-1}$ is given by $\sqrt{(\mathrm{d}(\mathbf{x}, \mathbf{y}, \mathbf{W})/\lambda_i)}$.

3.2 Calculating the bounding circle of the k nearest-neighbour disk

In a k nearest-neighbour search using weighted euclidean distance, we start from an initial disk covering the whole data space. After we find k neighbours from searched nodes, a new (smaller) k-nn disk is obtained. We have to calculate the bounding circle of the k-nn disk every time we find a neighbour nearer to the query point than the k-th nearest neighbour found so far. The problem can be stated as follows: given the query point $\mathbf{q}$ and the newly found k-th neighbour we need to calculate the radius of the bounding circle of the ellipse with $\mathbf{q}$ as its centre and $\mathbf{k}$ on its perimeter (see Fig. 3).

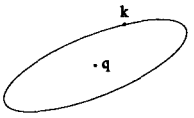

Fig. 3. The query point $\mathbf{q}$ and its k-th nearest-neighbour $\mathbf{k}$.

Based on the discussion in Sect. 3.1, the calculation of the bounding circle of the k nearest-neighbour disk is done as follows.

- Let λ be the largest eigenvalue of matrix $\mathbf{W}$.
- The bounding circle is a circle with the query point $\mathbf{q}$ as its centre and r as its radius, where $r = \mathrm{d}(\mathbf{q}, \mathbf{k}, \mathbf{W})/\sqrt{\lambda_{\min}}$, where $\lambda_{\min}$ is the smallest eigenvalue.

3.3 Calculating the bounding rectangle of the k nearest-neighbour disk

Instead of using bounding circle, we also can use bounding rectangle. The problem, similar to the one described in Sect. 3.2 can be stated as follows: given a query point $\mathbf{q}$ and the newly found k-th neighbour $\mathbf{k}$, calculate the length, the sides and the orientation of the bounding rectangle of the ellipse centred at $\mathbf{q}$ with $\mathbf{k}$ on its perimeter (see Fig. 3).

Let $\mathbf{W}$ is the weight matrix, and $\mathbf{I}$ is the identity matrix, $\mathbf{q}$ is the query point, $\mathbf{k}$ is the k-th nearest neighbour found, $\mathbf{E}$ is an orthonormal matrix whose columns is the eigenvectors of $\mathbf{W}$, λ_i is the eigenvalues corresponding to the

eigenvector in column i of $\mathbf{E}$, and $\boldsymbol{e}_i$ is the vector in column i of $\mathbf{E}^{-1}$. The bounding rectangle of the nearest-neighbour disk can be calculated as follows.

- Let $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ be the eigen-vectors of $\mathbf{W}$ and let λ_1 and λ_2 be the corresponding eigen-values sorted according to their magnitudes.
- Then the bounding rectangle R is defined by the centre $\mathbf{q}$ and dimensional vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ where $r = \mathrm{d}(\mathbf{q}, \mathbf{k}, \mathbf{W})$, and $\boldsymbol{v}_1 = \boldsymbol{e}_1 r/\sqrt{\lambda_1}$ and $\boldsymbol{v}_2 = \boldsymbol{e}_2 r/\sqrt{\lambda_2}$. The corners of the rectangles are:

$$\begin{aligned}
\mathbf{r}_1 &= \mathbf{q} + \boldsymbol{v}_1 + \boldsymbol{v}_2 \\
\mathbf{r}_2 &= \mathbf{q} + \boldsymbol{v}_1 - \boldsymbol{v}_2 \\
\mathbf{r}_3 &= \mathbf{q} - \boldsymbol{v}_1 - \boldsymbol{v}_2 \\
\mathbf{r}_4 &= \mathbf{q} - \boldsymbol{v}_1 + \boldsymbol{v}_2
\end{aligned}$$

Vectors $\boldsymbol{v}_i$ are in the directions parallel to the axes of the nearest neighbour disk. The fact that $\boldsymbol{v}_i$ is equal to $\boldsymbol{e}_i r/\sqrt{\lambda_i}$ can is explained in Sect. 3.1. Note that the rectangle R can be defined by the centre point $\mathbf{q}$ and the directional vectors denoted as $R(\mathbf{q}, \boldsymbol{v}_1, \boldsymbol{v}_2)$ and does not necessarily have any side parallel to the coordinate axes.

In high dimensional space, care must be taken in calculating the intersection between the bounding hyper-box and tree's bounding envelopes. A naïve calculation is exponential to the dimension. We provide an algorithm with time complexity $\mathrm{O}(d^3)$ of dimension d in Sect. 4.

4 Extension to a higher dimensional space

The algorithm in Sect. 3.3 can be used directly in space with dimensionality greater than 2. It contains three major computations, calculating Cholesky decomposition, finding the bounding hyper-box of the new k-nn hyper-ellipsoid, and checking the intersection of a node and the hyper-box. In this section we describe an efficient extension which gives $\mathrm{O}(d^3)$, $\mathrm{O}(d^2)$ and $\mathrm{O}(d^3)$ complexities to three steps respectively.

Note that checking if a node is intersected with the bounding hyper-box does not need to calculate the corner points of the bounding hyper-box. Otherwise, the approach will not be able to scale up to high dimensions because the number of corners of a hyper-box in a d-dimensional space is 2^d. We can define a bounding hyper-box using a centre point and d directional vectors $(\boldsymbol{r}_i)$. Since the Cholesky decomposition and the eigen matrix are the same throughout the search process, we only need to keep the radius r. The centre of the hyper-box will be the query point $\mathbf{q}$ itself. The cost to calculate the eigen matrix and Cholesky decomposition of a d x d matrix is $\mathrm{O}(d^3)$. We only need to recalculate these matrices whenever the weight matrix changes, which can only happen between queries. The calculation of the bounding rectangle for the query area can be done in $\mathrm{O}(d^2)$.

Our algorithm to check the intersection of tree nodes and the bounding hyper-box of the nearest-neighbour disk has a complexity $\mathrm{O}(d^3)$. If the bounding

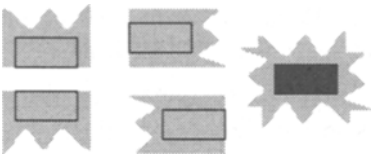

Fig. 4. A bounding rectangle is the intersection of four half-spaces.

hyper-box has sides parallel/perpendicular to the coordinate axes, intersections can be calculated in $O(d)$. We only need to calculate the intersection of each dimension. It can be done by utilizing the fact that a d-dimensional hyper-box is actually an intersection of $2d$ hyper-planes (as shown in the 2D case in Fig. 4).

Suppose the query's bounding hyper-box B is defined by its centre point $\mathbf{q}$ and d orthogonal vectors $\boldsymbol{v}_i$, ie, $B(\mathbf{q}, \boldsymbol{v}_1, ..., \boldsymbol{v}_d)$. The bounding hyper-box of a tree node is A and its closest corner to the origin is $\mathbf{a_{min}}$ and the farthest corner to the origin is $\mathbf{a_{max}}$. We use $\mathbf{r}(j)$ and $\boldsymbol{r}(j)$ to represent the projections of point $\mathbf{r}$ and vector $\boldsymbol{r}$ in dimension j respectively in the following description.

- For all dimensions $j = 1..d$, find the minimum coordinate of the query's bounding hyper-box (can be done in $O(d)$ by subtracting $\boldsymbol{v}_i(j)$ from $\mathbf{q}(j)$). If any minimum is not inside the half-space defined by the equation $x \leq \mathbf{a_{max}}(j)$, then we do not have an intersection. Otherwise, continue checking. This process takes $O(d^2)$ comparisons.
- For all dimension $j = 1..d$, find the maximum coordinate of the query's bounding hyper-box (can be done in $O(d)$ by adding $\boldsymbol{v}_i(j)$ to $\mathbf{q}(j)$). If any minimum is not inside the half-space defined by the equation $x \geq \mathbf{a_{min}}(j)$, then we do not have an intersection. Otherwise, continue checking. This process also takes $O(d^2)$ comparisons.
- If above checking fails, it does not mean that we have a non-empty intersection area. It means that B is not intersected with the hyper-box $\mathbf{a_{min}} \leq x \leq \mathbf{a_{max}}$. The check should also be made from the B rectangle in the query space using all the half-spaces defined by $x \leq \mathbf{q}+\boldsymbol{v}_i$ and $x \geq \mathbf{q}-\boldsymbol{v}_i$. Transformation to the query space takes $O(d^3)$ multiplication and rest comparisons are $O(d^2)$. The total complexity is $O(d^3) + O(d^2) = O(d^3)$.

5 Empirical tests and discussion

We implemented the algorithm described in Sect. 3.3and 3.2 on top of an R-tree variant, the SS$^+$-tree [16]. We compare our algorithms with Faloutsos' method using the unweighted euclidean distance as the lower bound of the weighted distance. We evaluate the performance of the methods using the number of nodes touched/used. For all the experiments in this paper we choose to expand the node whose centroid is closest to the query point with respect to the currently used distance metric. Based on our experience [16], this criterion truncates branches

more efficiently than other criteria, e.g. the minimum distance and the minimum of the maximum distance [18].

All the experiments in Table 1 are in 30 dimensions using 14,016 texture feature vectors extracted from images in Photo Disc's image library using the Gabor filter [12]. The experiments was done using a weight matrices whose ratio between the largest and smallest eigenvalues range from 2 to 32. The fan out of tree's leaf nodes is 66 and the fan out of tree's internal nodes is 22.

We measured the number of leaf nodes that has to be examined (*ltouched*), the number of internal nodes accessed (*inode*), and the number of leaf nodes that actually contain any of the nearest-neighbour set (*lused*). The ratio column is the ratio between square root of the largest and the smallest eigenvalues. All the experiments are 21-nearest-neighbour searches and the tabulated results are the average of 100 trials using a random vector chosen from the dataset.

Table 1. A comparison between the transformation and lower-bound method for the texture vectors of Photo Disc's images (14016 vectors, 30 dimensions, node size: 8192 bytes).

	Bounding (hyper)box			Bounding (hyper)sphere			Faloutsos' method		
ratio	ltouch	lused	inode	ltouch	lused	inode	ltouch	lused	inode
2	25.32	7.06	4.22	19.08	7.32	4.20	36.62	13.78	8.16
4	23.38	6.43	4.17	17.36	7.02	4.24	32.54	12.67	8.09
8	26.04	6.45	4.45	18.04	7.00	4.11	35.63	12.66	8.51
16	24.53	6.76	4.08	17.76	7.16	4.24	33.92	13.43	7.97
32	22.89	6.49	4.15	18.48	7.15	4.12	30.08	12.59	8.01

As can be seen in Table 1, our method outperformed Faloutsos' method in all the ratio values we tried. The number of leaf nodes used (*lused*) by our method is 50% of the number used by Faloutsos' method because our method only need to check the tree once. The same explanation applies to the number of internal nodes accessed (*inode*). The number of leaf nodes that have to be accessed by our method is consistently below 75% of the total leaf nodes accessed by the other method. The data suggest that the ratio between the largest and the smallest eigenvectors does not have any effect in the number of nodes accessed for the texture vectors.

The result experimental results using bounding spheres are somewhat counter intuitive. From our intuition, the results should be worse than the ones using bounding boxes. In two and three dimensions, the bounding rectangle/box of an disk occupy a smaller area/volume than the disk's bounding circle/sphere. Our analytical results [17] supports this empirical results. It shows that in high-dimensional space the query sensitive area resulting from using bounding hyper-spheres is smaller than the one resulting from using bounding hyper-boxes.

The experiments in Table 2 are performed using 100,000 uniformly distributed data in 2, 4, 8, 16, and 32 dimensions. The node size was kept fixed

at 8192 bytes, hence the leaf fan-out (*LFO*) and internal node fan-out (*IFO*) decreases as the dimension increases. The ratio between the largest and smallest eigenvalues was fixed at 4 for all the experiments.

Table 2. A comparison between the transformation and lower-bound method for 100,000 uniformly distributed vectors (ratio used: 4, bucket size: 8192 bytes).

			Bounding (hyper)box			Bounding (hyper)sphere			Faloutsos' method		
dimension	LFO	IFO	ltouch	lused	inode	ltouch	lused	inode	ltouch	lused	inode
2	682	255	1.4	1.4	1	1.4	1.4	1	3.2	2.7	2.0
4	409	146	4.5	2.7	2.2	3.6	2.4	2.3	12.5	5.5	4.5
8	227	78	89.5	8.9	4.8	39.5	7.0	4.0	175.1	17.1	10.4
16	120	40	740.6	25.9	20.4	712.5	25.9	22.0	1489.2	46.0	43.8
32	62	20	1613.0	41.0	87.0	1613	42.0	87.0	3226.0	88.0	174.0

In Table 2, the number of nodes accessed are much higher in high dimensions. This is partly due to the smaller internal/leaf node fan-outs (we kept the node size fixed for all dimensions). Similar to the experimental results using texture vectors, the number of leaf nodes and internal nodes used by our method (*lused* and *inodes*) is approximately half of the number used by Faloutsos' method. In terms of the number of leaf nodes accessed (*ltouch*), our method consistently accessed less than 50% of the number accessed by Faloutsos' method when the dimension is greater than four.

The result experimental results using bounding spheres (up to dimension 16) further support the fact that the query sensitive area resulting from using bounding hyper-spheres will get smaller than the one resulting from using bounding hyper-boxes. But the experimental results in 32 dimension shows that the performance is almost the same for both method. This is due to the inherent problem with hierarchical tree access method in high dimension. In this case, the search actually has degraded into linear search (the variations are due to the randomization used in the experiments).

The experiments are verified by comparing the results of the k nearest neighbour search with the results obtained via simple linear search and comparing the results of the two methods.

6 Conclusion

Nearest-neighbour search on multi-dimensional data is an important operation in many areas including multimedia databases. The similarity measurement usually involves weighting on some of the dimensions and weighting may vary between queries. Existing tree structured spatial access methods cannot support this requirement directly. We have developed an algorithm to support variable distance metrics in queries. The algorithm uses the bounding hyper-box of the k-nn hypersphere instead of calculating intersections directly. It has $O(d^3)$ complexity. We

derive the theoretical foundation of the algorithm and give a detailed implementation in the 2D case. We also extend the algorithm to higher dimensions and give a heuristic algorithm to detect the intersection between the query disk and the bounding envelopes of the tree nodes. We provide analytical and empirical results regarding the performance of our approach in terms of the number of disk pages touched. The experiments go up to 32 dimensions. The algorithm has a significant impact on k-nn search and various index trees. There are also research issues in speeding up the transformation. As we can see from the derivation of the algorithm, the major computational cost is in cross-space transformation.

References

1. David W. Aha. A study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations (dissertation). Technical Report ICS-TR-90-42, University of California, Irvine, Department of Information and Computer Science, November 1990.
2. S. Belkasim, M. Shridhar, and M. Ahmadi. Pattern classification using an efficient KNNR. *Pattern Recognition*, 25(10):1269–1274, 1992.
3. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.
4. Christos Faloutsos. *Searching Multimedia Databases by Content.* Advances in Database Systems. Kluwer Academic Publishers, Boston, August 1996.
5. Christos Faloutsos, William Equitz, Myron Flickner, Wayne Niblack, Dragutin Petkovic, and Ron Barber. Efficient and effective querying by image content. *J. of Intelligent Information Systems*, 3:231–262, July 1994.
6. Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 419–429, May 1994.
7. Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Bryan Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, pages 23–32, September 1995.
8. Jerome H. Friedman, Jon Louis Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Math. Software (TOMS)*, 3(3):209–226, September 1977.
9. Keinosuke Fukunaga and Larry D. Hostetler. Optimization of k-nearest-neighbor density estimates. *IEEE Transactions on Information Theory*, IT-19(3):316–326, May 1973.
10. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.
11. Gisli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In Max J. Egenhofer and John R. Herring, editors, *Advances in Spatial Databases, 4th International Symposium, SSD'95*, volume 951 of *Lecture Notes in Computer Science*, pages 83–95, Berlin, 1995. Springer-Verlag.
12. Jesse Jin, Lai Sin Tiu, and Sai Wah Stephen Tam. Partial image retrieval in multimedia databases. In *Proceedings of Image and Vision Computing New Zealand*, pages 179–184, Christchurch, 1995. Industrial Research Ltd.

13. Jesse S. Jin, Guangyu Xu, and Ruth Kurniawati. A scheme for intelligent image retrieval in multimedia databases. *Journal of Visual Communication and Image Representation*, 7(4):369–377, 1996.
14. D. Kibler, D. W. Aha, and M. Albert. Instance-based prediction of real-valued attributes. *Computational Intelligence*, 5:51–57, 1989.
15. Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, and Eliot Siegel. Fast nearest-neighbor search in medical image databases. In *International Conference on Very Large Data Bases*, Bombay, India, Sep 1996.
16. Ruth Kurniawati, Jesse S. Jin, and John A. Shepherd. The SS$^+$-tree: An improved index structure for similarity searches in a high-dimensional feature space. In *Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases V*, volume 3022, pages 110–120, San Jose, CA, February 1997.
17. Ruth Kurniawati, Jesse S. Jin, and John A. Shepherd. Efficient nearest-neighbour searches using weighted euclidean metrics. Technical report, Information Engineering Department, School of Computer Science and Engineering, University of New South Wales, Sydney 2052, January 1998.
18. Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, California, May 1995.
19. Robert F. Sproull. Refinements to nearest-neighbour searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.
20. Gilbert Strang. *Introduction to applied mathematics.* Wellesley-Cambridge Press, Wellesley, MA, 1986.
21. Gilbert Strang. *Linear algebra and its applications.* Harcourt, Brace, Jovanovich, Publishers, San Diego, 1988.
22. David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proc. 12th IEEE International Conference on Data Engineering*, New Orleans, Louisiana, February 1996.