Similarity Search for Adaptive Ellipsoid Queries Using Spatial Transformation

Yasushi Sakurai† *Masatoshi Yoshikawa*§ *Ryoji Kataoka*† *Shunsuke Uemura*§

† NTT Cyber Space Laboratories, Japan
§ Nara Institute of Science and Technology, Japan

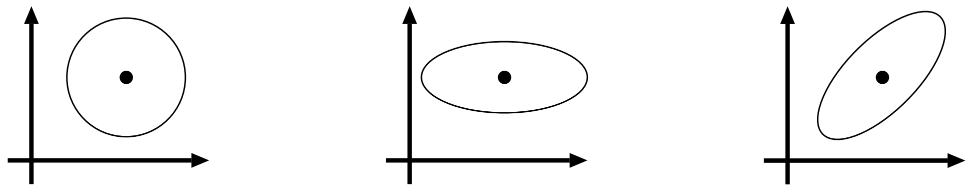
Abstract

Similarity retrieval mechanisms should utilize generalized quadratic form distance functions as well as the Euclidean distance function since ellipsoid queries parameters may vary with the user and situation. In this paper, we present a spatial transformation technique that yields a new search method for adaptive ellipsoid queries. The technique is based on the notion of spatial transformation and efficiently supports adaptive ellipsoid queries with quadratic form distance functions. Although conventional search methods can support ellipsoid queries by using multi-dimensional index structures, these methods incur high CPU-cost for measuring distances between a query point and bounding rectangles with respect to quadratic form distance functions, which exceeds disk access cost on search processing. The basic idea is to transform the bounding rectangles in the original space, wherein distance from a query point is measured by quadratic form distance functions, into spatial objects in a new space wherein distance is measured by Euclidean distance functions. Since bounding rectangles in the original space are transformed into multi-dimensional polygons in the new space, and thus incurring high CPU-cost for distance calculations, our proposed method approximates a polygon by the rectangle that totally encloses the polygon, and measures the distance from the query point to the rectangle instead to the polygon in the Euclidean space. It follows that the spatial transformation technique guarantees no false drops. In contrast to the conventional methods, our proposed method significantly reduces CPU-cost due to the distance approximation by the spatial transformation; exact distance evaluations are avoided for most of the accessed bounding rectangles in the index structures. We also present the multiple spatial transformation technique as an extension of the spatial transformation technique. The multiple spatial transformation technique adjusts the tree structures to suit typical ellipsoid queries; the search algorithm utilizes the adjusted structure. This technique reduces both page accesses and CPU time for ellipsoid queries. Experiments using various matrices and index structures demonstrate the superiority of the proposed methods.

1 Introduction

Multimedia content-based retrieval systems use feature values extracted from multimedia data; they find data objects whose feature values are most similar to those of the query object. These systems include various pattern recognition mechanisms, and the databases on which they operate continue to grow in size. This means that, multimedia systems and spatial databases require (1) information retrieval methods with more general distance functions, and (2) improved search performance.

Since the Euclidean distance space makes all dimensions independent of each other, it fails to adequately represent the user's intention. Therefore, multimedia systems require the use of generalized quadratic form distance functions as well



(a) Euclidean distance (b) weighted Euclidean distance (c) quadratic form distance

Figure 1: Isosurfaces for various distance functions.

as the Euclidean distance function. Since quadratic form distance functions can represent correlations between dimensions, retrieval mechanisms using them have high search quality [HSE+95]. The quadratic form distance function $d_M^2(p,q) = (p-q) \cdot M \cdot (p-q)^t$ is calculated from a similarity matrix M which is positive definite (i.e. $d_M^2(p,q) > 0$), where q is a query point and p is a data object in a data set. In d-dimensional spaces, the Euclidean distance function has circles for isosurfaces, and weighted Euclidean distance functions correspond to iso-oriented ellipsoids, whose major axis is aligned to the coordinate axis. Quadratic form distance functions have arbitrarily oriented ellipsoids that are not necessarily aligned to the coordinate axis (see Figure 1). Quadratic form distance functions are regarded as a generalization of the Euclidean distance function and weighted Euclidean distance functions. MindReader [ISF98] is an example of the application of quadratic form distance functions; based on relevance feedback, it guesses the correlations between dimensions, which reflect the user's preference. Unlike the Euclidean distance function, quadratic form distance functions more faithfully reflect the user's intention.

Given that the size of multimedia databases will continue to grow and that the dimensionality of feature data will continue to increase, high-performance data retrieval methods are essential. Many index methods have been proposed so far [GG98]. They include data-partitioning index trees (e.g. the R*-tree [BKSS90], the X-tree [BKK96] and the A-tree [SYUK00]). Nearest neighbor search methods using such indices have also been proposed [RKV95] [HS95]. In particular, the A-tree is reported to offer good performance for high-dimensional data [SYUK00]. Unfortunately, most spatial access methods were designed for searches based on the Euclidean distance function, so new spatial access methods that suit ellipsoid queries based on quadratic form distance functions are needed. In addition, image retrieval mechanisms using various user-adaptable distance functions [FSA+95] [HSE+95] and relevance feedback mechanisms that guess the user's desires, such as MARS [RHM97] and MindReader [ISF98], deal with queries whose parameters can vary with the user and situation. These mechanisms require search methods that can support the adaptive queries.

The goal of our work is to create a search method for adaptive ellipsoid queries that can find similar objects efficiently. Various metric indices (e.g. the M-tree [CPZ97] and the mvp-tree [BO97]) have been proposed as indexing methods for arbitrary distance functions. However, these indices cannot be applied to systems that handle changeable distance functions and, thus, they are not functionally adequate to support adaptive ellipsoid queries. In [SK97], Seidl et al. presented a search algorithm for adaptive ellipsoid queries on index structures that calculates exact distances between query points and MBRs (Minimum Bounding Rectangles). This search method supports adaptive ellipsoid queries whose distance functions are changeable, by using index structures. However, the calculations of distance between query points and MBRs incurs CPU-costs as high as $O(\omega \cdot d^2)$ time, where d is dimensionality and ω denotes the number of iterations, which exceeds disk access time on retrieval processing. In [ABKS98], Ankerst et al. presented a search method that reduces the number of exact quadratic form distance calculations needed and so reduces the CPU time by using MBB (Minimum Bounding Box) distance functions and MBS (Minimum Bounding Sphere) distance functions. The following definitions formalize the MBB distance function and the MBS distance function for d-dimensional spaces:

$$d^2_{MBB(M)}(p,q) = \max_{i=1}^{d} \left(\frac{(p_i - q_i)^2}{(M^{-1})_{ii}} \right), \tag{1}$$

$$d^2_{MBS(M)}(p,q) = \lambda_{M_{min}} \cdot (p-q)^2, \tag{2}$$

where λ_{M_i} $(i = 1, \cdots, d)$ are the eigenvalues of M, and $\lambda_{M_{min}}$ is the lowest eigenvalue of M. The MBB distance functions approximate an ellipsoid query area by a bounding box that totally encloses the query area. The MBS distance functions use a bounding sphere for the approximation. Both approximation techniques require $O(d)$ time for calculations. The search algorithm in [ABKS98] lowers CPU-cost by using MBB and MBS distance functions as well as exact quadratic form distance functions, which we call the MBB-MBS approximation technique in this paper. However, the MBB-MBS approximation technique has two problems: first, approximation quality degrades as either dimensionality grows or query ellipse becomes flatter. Low approximation quality increases the CPU time. Second, spatial access methods construct index structures to find target objects in the Euclidean space efficiently, and then the MBB-MBS approximation technique invokes ellipsoid queries using the constructed index structures. Due to the use of indices based on the Euclidean distance function, the number of accesses for bounding rectangles and data objects increases as either dimensionality grows or query ellipse becomes flatter; consequently, both CPU-cost and the number of page accesses increase.

We introduce an approximation technique that efficiently works with high matrix flatness and dimensionality. Motivated by the disadvantages of the MBB-MBS approximation technique for ellipsoid queries, we develop the Spatial Transformation Technique (STT). The basic idea of STT is to transform the MBRs whose distance from a query point is measured by the quadratic form distance functions, into rectangles whose distance is measured by the Euclidean distance functions. With regard to the first problem, even though STT has low CPU-cost, it provides high approximation quality for high dimensionality and matrix flatness. Against the second problem, we provide the multiple spatial transformation technique (MSTT). MSTT adjust the tree structures to suit typical ellipsoid queries; this technique reduces the number of page accesses as well as the CPU-cost since the search algorithm utilizes the adjusted structure.

The remainder of this paper is organized as follows. Section 2 analyzes the MBB-MBS approximation technique based on experiments using real data. Based on the analysis, Section 3 describes the motivation, definitions and algorithms of STT. Section 4 presents MSTT. Section 5 gives the results of a performance evaluation of STT and MSTT. Finally, Section 6 concludes the paper.

Table 1: Variance of eigenvalues.

w_r		1	10	100	1000
σ_M^2	$d = 8$	0.0307	76.489	7998.6	800214
	$d = 27$	64.777	93372	9.29e8	9.29e12

2 Problems of Search Methods for Ellipsoid Queries

This section discusses the properties and problems of the MBB-MBS approximation technique. We evaluated its performance using real data sets with size of 100,000. For the data sets, 8-D and 27-D feature vectors of color histograms were extracted from images. In assessing search performance, the page access number and CPU time were measured by the average of 100 queries. In our evaluation, we used 20-nearest neighbor queries; query data were different from the point data included in the indices, that is, query points were generated randomly and independently of data points. Page size was 8KB. CPU time was measured on a SUN UltraSPARC-II 450MHz. We used the A-tree [SYUK00], which provides superior performance for high-dimensional data, and chose the code with size of 6 bits per dimension for approximating the bounding rectangles and data objects in the A-tree structure. In order to obtain similarity matrices M, we calculated the components m_{ij} of M using the following formula [HSE+95] [ABKS98]:

$$m_{ij} = exp(-\alpha(d_w(c_i, c_j)/d_{max})^2),$$

where α is a positive constant, and $d_w(c_i, c_j)$ denotes the weighted Euclidean distance between the color c_i and c_j. The factors $w = (w_r, w_g, w_b)$ represents the weighting of the red, green and blue components in RGB color space. In our evaluation, α was 10, w_g and w_b were fixed to 1. w_r was varied from 1 to 1,000. We calculated the eigenvalues of every matrix for 8-D and 27-D data. The variance σ_M^2 of eigenvalues was determined as follows:

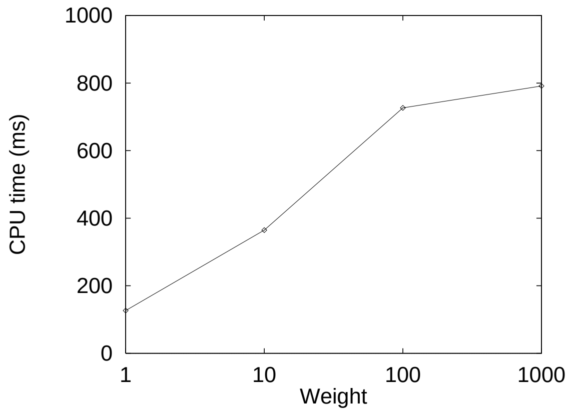
$$\sigma_M^2 = \sum_{i=1}^{d}(\lambda_{M_i} - \overline{\lambda}_M)^2, \quad \overline{\lambda}_M = \sum_{j=0}^{d} \frac{\lambda_{M_j}}{d},$$

where $det(M) = 1$, λ_{M_i} is the i-th dimensional eigenvalue and $\overline{\lambda}_M$ is the average of the eigenvalues of M.

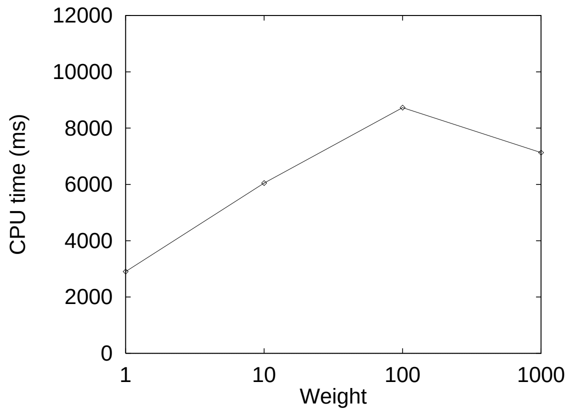
Table 1 enumerates this formula for various values of w_r. Before calculating σ_M^2, all matrices were normalized[1] with $det(M) = 1$. In this paper, the variance σ_M^2 is called the flatness of M. Here, the flatness of the unit matrix that represents searching in the Euclidean space, is 0. As shown in Table 1, matrix flatness increases as w_r grows when α, w_g and w_b are fixed.

Figure 2 shows the CPU time of the MBB-MBS approximation technique for various matrices with different flatness. Throughout this paper, horizontal axis `Weight` denotes w_r. CPU time for ellipsoid queries increases as dimensionality grows, which is similar to searches in the Euclidean distance space. One observation specific to ellipsoid queries is that CPU time increases with matrix flatness. To establish the link between flatness and CPU time on ellipsoid queries, we conducted a detailed

[1] Normalization of matrices is described in Section 4.4.

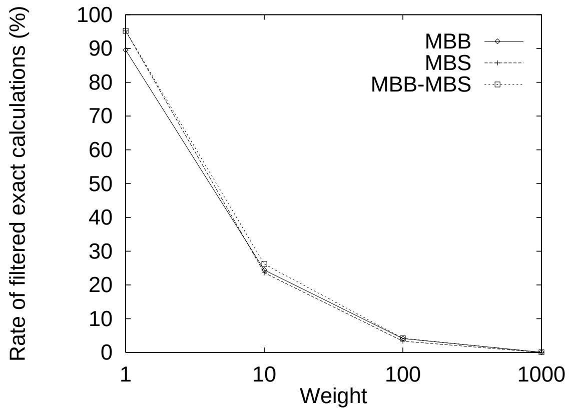


(a) d = 8

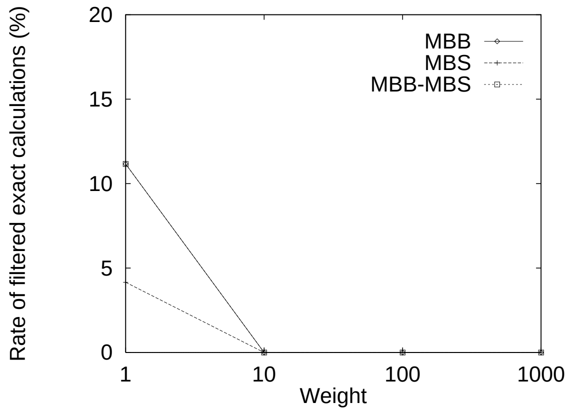


(b) d = 27

Figure 2: CPU time for the MBB-MBS approximation technique.



(a) d = 8

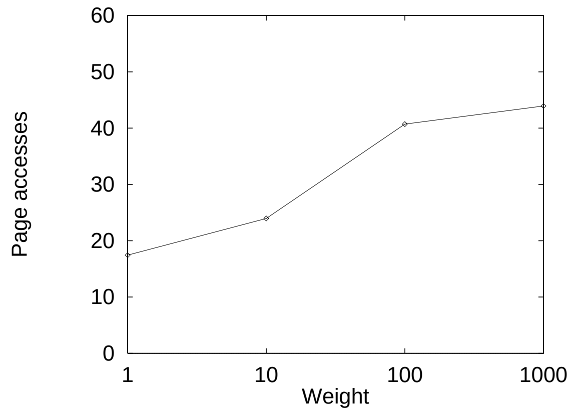


(b) d = 27

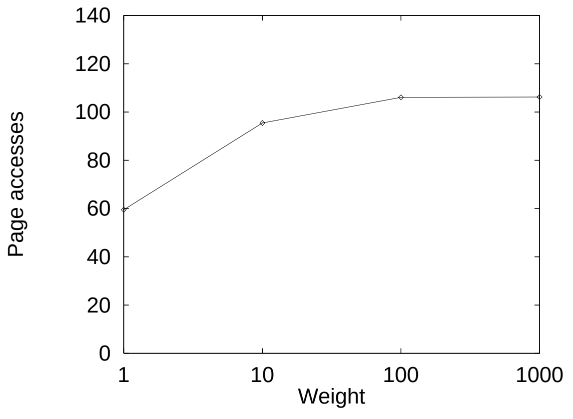
Figure 3: Rate of filtered exact distance calculations for the MBB-MBS approximation technique.

analysis. Figure 3 shows the percentage of exact distance calculations filtered by the MBB-MBS approximation technique. For the calculation of distance between query points and bounding rectangles, Figure 3 demonstrates the characteristics of MBB approximation, MBS approximation and MBB-MBS approximation. The figure shows that approximation quality decreases as flatness increases for both MBB and MBS approximations. Moreover, both approximation techniques are ineffective against 27-dimensional data. Although Equations (1)(2) suggest this property for MBB and MBS approximations, Figure 3 indicates it more clearly.

Matrices with high flatness require more node accesses. As a result, as shown in Figure 4, the number of page accesses increases as matrix flatness grows. Unlike CPU time, the number of node accesses is quite independent of the approximation quality. The reason for the increase in the number of node accesses is that the index structures focus on searches in the Euclidean space. That is, spatial access methods construct index structures to find target objects in the Euclidean space efficiently; the MBB-MBS approximation technique performs ellipsoid queries using



(a) d = 8



(b) d = 27

Figure 4: The number of page accesses for the MBB-MBS approximation technique.

the constructed index structures. Due to the use of indices based on the Euclidean distance function, the number of accesses for bounding rectangles and data objects increases as matrix flatness grows; this, consequently, leads to increases in both CPU time and the number of page accesses.

Our discussions in this section are summarized below:

(R1) **CPU time**
For both MBB and MBS approximation functions, approximation quality decrease as either dimensionality or matrix flatness grows. As a result, the number of exact quadratic form distance calculations increases, thus leading to high CPU time.

(R2) **Node accesses**
Index structures are constructed to find target objects in the Euclidean space efficiently, and the search algorithms utilize the resulting index structures. Therefore, as matrix flatness grows, the number of node accesses increases. This leads to increases in both CPU time and the number of page accesses.

To cope with (R1), we propose the spatial transform technique in Section 3; the technique achieves high quality approximations and superior performance. To overcome (R2), we present the multiple spatial transformation technique in Section 4; it is an extension of the spatial transformation technique. This technique reduces both CPU time and the number of page accesses.

3 The Spatial Transformation Technique

In this section, we describe the Spatial Transformation Technique (STT) for efficient ellipsoid queries, which approximates the quadratic form distance between a query point and a bounding rectangle. The STT guarantees no false drops like the MBB-MBS approximation technique and so returns exact answers to any query.

3.1 Basic Ideas

In computing the exact distance for quadratic form distance functions, calculating the distance between a query point and MBRs in the index structures incurs high

CPU-cost, moreover, the calculations need to be iterated. That is, the complexity is $O(\omega \cdot d^2)$, where ω denotes the number of iterations. The basic idea of STT is to transform the MBRs, whose distance from a query point is measured by the quadratic form distance functions, into rectangles whose distance is measured by the Euclidean distance functions. STT requires no iteration. The transformation used in STT effectively contributes to reducing CPU-cost. As shown in the evaluation result (R1), the MBB-MBS approximation technique is not effective when either the dimensionality or flatness of query matrices is high. STT requires less CPU-cost, and offers great efficiency even for high dimensionality and matrix flatness due to its high approximation quality. In this section, we first define the spatial transformation, and then describe a spatial transformation technique for bounding rectangles in index structures.

3.2 Definition of Spatial Transformation

Given a query matrix M and a query point q, the quadratic form distance between q and a point p in a d-dimensional space $\mathcal{S}$ is defined as follows:

$$d_M^2(p,q) = (p-q) \cdot M \cdot (p-q)^t. \tag{3}$$

Since M is positive definite, the spectral decomposition of M can be calculated as:

$$M = E_M \cdot \Lambda_M \cdot E_M^t, \tag{4}$$

where E_M is the set of the eigenvectors of M, the diagonal matrix $\Lambda_M = diag(\lambda_{M_1}, \lambda_{M_2}, \ldots, \lambda_{M_d})$ consists of the eigenvalues $\lambda_{M_1}, \lambda_{M_2}, \ldots, \lambda_{M_d}$ of M. From Equations (3)(4), we obtain:

$$d_M^2(p,q) = (p-q) \cdot E_M \cdot \Lambda_M \cdot E_M^t \cdot (p-q)^t. \tag{5}$$

When considering point $p' = (p-q) \cdot E_M \cdot \Lambda_M^{\frac{1}{2}}$ in the Euclidean space $\mathcal{S}'$, Equation (5) denotes that the Euclidean distance between the origin O and p' in $\mathcal{S}'$ is equal to the quadratic form distance $d_M^2(p,q)$ (i.e. $d_M^2(p,q) = p' \cdot p'^t$). Here, the transformation matrix of M is defined as:

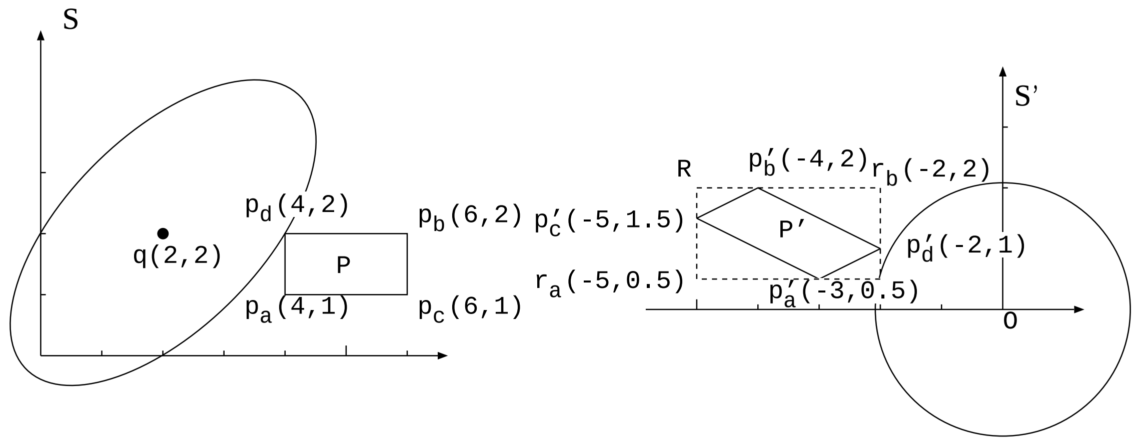
$$A_M = E_M \cdot \Lambda_M^{\frac{1}{2}}. \tag{6}$$

A_M transforms the quadratic form distances within $\mathcal{S}$ into the Euclidean distances within $\mathcal{S}'$. It yields the so called spatial transformation of p into p'.

3.3 Spatial Transformation of Rectangles for Distance Calculation

STT gives the spatial transformation of rectangles in index structures. Figure 5 illustrates the spatial transformation of a rectangle. In this figure, the bounding rectangle P in $\mathcal{S}$ is transformed into the d-dimensional parallelogram P' in $\mathcal{S}'$. Since the calculation of distance between the origin O and polygons in high-dimensional spaces incurs high CPU-cost, STT approximates P' by rectangle R as shown in Figure 5(b). This approximation reduces the calculation cost of ellipsoid queries.

We assume a rectangle P within $\mathcal{S}$ and a query point q. Let p_a and p_b be endpoints of the major diagonal of P, and l_i be the i-th dimensional edge length of



(a) A rectangle in the original space (b) A rectangle calculated by STT

Figure 5: An example of spatial transformation.

P. It follows that point p'_a in $\mathcal{S}'$ can be calculated by the spatial transformation of p_a:

$$p'_a = (p_a - q) \cdot A_M. \tag{7}$$

We extract the following components from the components a_{ij} of A_M:

$$\phi_{ij} = \begin{cases} a_{ij} & (a_{ij} < 0) \\ 0 & (\text{otherwise}), \end{cases} \qquad \psi_{ij} = \begin{cases} a_{ij} & (a_{ij} > 0) \\ 0 & (\text{otherwise}). \end{cases} \tag{8}$$

From Equations (7)(8), the rectangle R that totally encloses the d-dimensional parallelogram with respect to the spatial transformation of P can be calculated as [2] :

$$\begin{aligned} R &= (r_a, r_b) \qquad (9) \\ r_{a_j} &= p'_{a_j} + \sum_{i=1}^{d} l_i \cdot \phi_{ij}, \quad r_{b_j} = p'_{a_j} + \sum_{i=1}^{d} l_i \cdot \psi_{ij}, \quad (1 \le j \le d), \end{aligned}$$

where r_a and r_b are endpoints of the major diagonal of R. Since R totally encloses P' in $\mathcal{S}'$, The search algorithm can use the Euclidean distance $d^2(R, O)$ instead of the quadratic form distance $d^2_M(P, q)$ (i.e. $d^2(R, O) \le d^2_M(P, q)$).

For example, as shown in Figure 5, the query point $q = (2, 2)$ and matrix:

$$M = \begin{pmatrix} 1.25 & -0.75 \\ -0.75 & 1.25 \end{pmatrix}$$

are given. When using M, the vertices p_a, p_b, p_c and p_d of the bounding rectangle P in $\mathcal{S}$ are transformed into the vertices p'_a, p'_b, p'_c and p'_d of the parallelogram P' in $\mathcal{S}'$, respectively. Also, $R = (r_a, r_b)$ encloses P'. $d^2_M(q, P)$ is approximated by $d^2(R, O)$, and we can utilize $d^2(R, O)$ instead of $d^2_M(q, P)$.

[2] The proof of Equation (9) is described in Appendix A.

3.4 Search Algorithm

Range queries and k-nearest neighbor queries are useful for multi-dimensional databases. An algorithm based on spatial transformation can efficiently support both types of queries. Since k-nearest neighbor queries are more complex and require higher cost than range queries, we focus on k-nearest neighbor search in this paper, and describe one such algorithm for STT. Note that the idea of STT can be applied to any range query.

We show Equations (6)(7)(8)(9), for spatial transformation in Section 3.2 and Section 3.3. CPU-cost would become excessive if a search required the spatial transformation of all accessed rectangles using these formulas. Therefore, we introduce the following two ideas to reduce CPU-cost.

First, the result of Equations (6)(8) does not depend on the position of the bounding rectangles accessed. Thus, the search algorithm solves these formulas before accessing the rectangles. The result can be applied to the spatial transformation of all rectangles visited.

Second, we reduce the calculation time relative to Equation (9). Note that, in average, half of the components ϕ_{ij} and ψ_{ij} are 0. Therefore, in the implementation, the algorithm searches for all pairs of row number i and column number j whose components are $\phi_{ij} \neq 0, \psi_{ij} \neq 0$ before accessing nodes in index structures. This preprocessing halves the CPU-cost when calculating R (i.e. r_{a_j} and r_{b_j}) using Equation (9).

Let c_{a_j} be the number of components in the j-th column where $\phi_{ij} \neq 0$, and u_{jk} be c_{a_j} row numbers of the components in the j-th column ($k = 1, \ldots, c_{a_j}$). Similarly, for ψ_{ij}, let c_{b_j} be the number of components in the j-th column where $\psi_{ij} \neq 0$, and v_{jk} be c_{b_j} row numbers of the components in the j-that column ($k = 1, \ldots, c_{b_j}$). A function that calculates the position of R with less computation time can be obtained by using u_{jk} and v_{jk}:

$$\begin{aligned} R &= (r_a, r_b) && (10) \\ r_{a_j} &= p'_{a_j} + \sum_{k=1}^{c_{a_j}} l_k \cdot \phi_{(u_{jk})j}, \quad r_{b_j} = p'_{a_j} + \sum_{k=1}^{c_{b_j}} l_k \cdot \psi_{(v_{jk})j}, \quad (1 \leq j \leq d) \end{aligned}$$

where each c_{a_j} and c_{b_j} averages $d/2$.

Figure 6 shows the search algorithm for ellipsoid queries using tree structures of the R-tree family. The search algorithm utilizes the spatial transformation of rectangles to evaluate the distance of a query point to the rectangles. STT and MBB-MBS approximation techniques incur lower CPU-cost than the exact quadratic form distance function for distance calculations. Therefore, the search algorithm first calculates the approximation distance between a query point and a bounding rectangle when evaluating the distance to the bounding rectangle. If the calculated approximation distance is less than or equal to the distance of the query point to the actual k-th nearest neighbor, the exact distance to the rectangle is evaluated using the exact quadratic form distance function.

In Procedure `search` (see Figure 6), for initialization, the transformation matrix is calculated and its components are checked (step 1), and then the pair of a pointer to the root and 0 is stored in the priority queue (step 2). In step 4, the function `dequeue()` dequeues the pair from the top of the priority queue, and extracts a node N. If N is a data node, the MBB-MBS approximation distance of every data object in the node is evaluated. Then, if the approximation distance is less

```
Procedure search(point $query$, matrix $M$, integer $k$)
 1. $\Phi_M$ := analyzeMatrix($M$);
 2. enqueue($a\_pointer\_to\_the\_root$, 0);
 3. for $i = 1$ to $k$, $nnlist[i].dist$ := $\infty$;
 4. while emptyQueue() = false do
 5.   $N$ := dequeue();
 6.   if $N$ is a data node then
 7.     for each $entry \in N$ do
 8.       if $d_{MBB\text{-}MBS(M)}$($query$, $entry$.vector) $\leq$ $nnlist[k].dist$ then
 9.         if $d_M$($query$, $entry$.vector) $\leq$ $nnlist[k].dist$ then
10.           $nnlist[k].id$ := $entry$.id;
11.           $nnlist[k].dist$ := $d_M$($query$, $entry$.vector);
12.           sort $nnlist$ by distance;
13.           pruneQueue($nnlist[k].dist$);
14.         endif
15.   else
16.     for each $entry \in N$ do
17.       if $d_{MBB\text{-}MBS(M)}$($query$, $entry$.rectangle) $\leq$ $nnlist[k].dist$ then
18.         $R$ := spatialTransformation($query$, $entry$.rectangle, $\Phi_M$);
19.         if d($R$, $O$ ) $\leq$ $nnlist[k].dist$ then
20.           if $d_M$($query$, $entry$.rectangle) $\leq$ $nnlist[k].dist$ then
21.             enqueue($entry$.ptr, $d_M$ ($query$, $entry$.rectangle));
22.       endif
23.   endif
24. enddo
25. output($nnlist$);
```

Figure 6: k-nearest neighbor search algorithm for ellipsoid queries.

than or equal to the actual k-th nearest neighbor distance, the exact distance is evaluated (steps 5 to 8), and the data object together with its distance is stored in the nearest neighbor list. (steps 9 to 12). If N is not a data node, the MBB-MBS approximation distance of every bounding rectangle is evaluated (step 16). Then, if the MBB-MBS approximation distance of a rectangle is less than or equal to the actual k-th nearest neighbor distance, the spatial transformation of the rectangle is calculated from Φ_M (step 17). In step 18, the Euclidean distance between O and R obtained by the spatial transformation, is evaluated. If the distance calculated by the spatial transformation is less than or equal to the actual k-that nearest neighbor distance, the exact distance is evaluated (step 19).

In the following experiments, we use not only the A-tree but also the R*-tree. The A-tree is useful for ellipsoid queries as well as queries based on the Euclidean distance function. The A-tree search algorithm differs somewhat from the other methods in the R-tree family. Details of the A-tree search algorithm are described in [SYUK00].

3.5 Dimensionality Reduction

When the flatness of a query matrix is high, there are eigenvectors whose eigenvalue is small. In the space created by spatial transformation, the dimensions corresponding to the eigenvalues contribute less to approximation quality although the dimensions require the same CPU-cost as the others do. The STT with dimensionality reduction eliminates the dimensions whose eigenvalues are small in order to save CPU-cost.

Let $r = (r_1, r_2, \ldots, r_d)$ be the point in R, which is the closest to O in $\mathcal{S}'$ created by the spatial transformation. When using dimensionality reduction, the distance of R to O can be determined as:

$$\tilde{d}(O,R) = \left(\sum_{i=1}^{n}(r_i)^2\right)^{\frac{1}{2}}, n = COUNT\left(\lambda_i \geq \frac{\eta}{d} \cdot \sum_{i=1}^{d}\lambda_i\right) \quad (n \leq d), \tag{11}$$

where η is a threshold for dimensionality reduction, and λ_i is arranged in descending order (i.e. $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d > 0$). The function $COUNT(\Gamma)$ gives the number of elements that satisfy requirement Γ. This formula shows that the dimensionality for distance calculation in $\mathcal{S}'$ is limited to n. Thus, the STT with dimensionality reduction reduces the calculation time relative to Equations (7)(8)(10) as well as Equation (11) to n/d. As query matrix flatness increases, the reduction rate n/d decreases and higher efficiency is achieved for the distance calculations.

4 The Multiple Spatial Transformation Technique

In this section, we present the Multiple Spatial Transformation Technique (MSTT), which is an extension of STT. STT provides high approximation quality, however, the number of node accesses increases as query matrix flatness grows, since STT, as well as conventional search methods, utilizes the structure constructed by the Euclidean distance function. To overcome this problem, MSTT constructs tree structures based on various quadratic form distance functions, and then chooses a desirable structure for improving search performance; the search algorithm described in Section 3.4 utilizes the chosen structure.

4.1 Basic Ideas

We indicate the problem of node accesses in the evaluation result (R2). Search methods for adaptive ellipsoid queries presented in [SK97] and [ABKS98] use index structures based on the Euclidean distance function. Accordingly, the number of node accesses increases as query matrix flatness grows, which leads to an increase in CPU-cost and the number of page accesses. MSTT overcomes this problem by selecting an arbitrary quadratic form distance function before constructing the index structures; the search algorithm utilizes the resulting structure. MSTT reduces both page accesses and CPU-cost for ellipsoid queries.

MSTT can handle more than one index structure. For multimedia systems that attach importance to retrieval performance and can well afford the disk space, using more than one structure is effective in improving search performance. Figure 7 illustrates a retrieval mechanism based on MSTT. The mechanism first determines a typical ellipsoid query matrix X_i $(i = 1, \ldots, \varepsilon)$ from the user's query logs, and then constructs index structures based on X_i. In query processing, the matrix $X_{similar}$

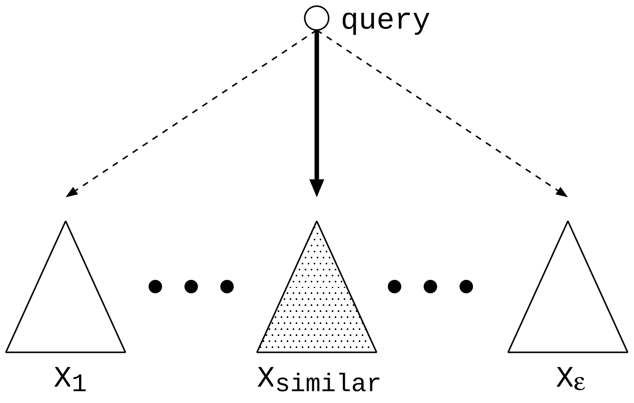


Figure 7: The multiple spatial transformation technique.

closest to the query matrix M is chosen, and target objects are found using the structure constructed by $X_{similar}$. Especially, the query shown in Figure 7 requires search processing based on the Euclidean distance function if $M = X_{similar}$. This retrieval mechanism that adopts multiple indexing can accelerate search performance.

Recently, disk price continues to fall and disk unit capacity is increasing rapidly. [GG97] shows that disk unit capacity and storage cost have increased /decreased hundred times and ten thousand times, respectively; whereas disk access speeds have increased only ten-fold in the last twenty years. The resulting trend is to emphasize disk access speed counts over storage cost. In addition, reducing the search cost has higher priority than reducing the insertion cost in many multimedia databases. It follows from these trends that the retrieval mechanism has a very good reason to use more than one index to accelerate search performance.

4.2 Indexing and Retrieval Mechanisms

Structure Construction:

Let C be a matrix to construct an index structure. The transformation matrix A_C of C is:

$$A_C = E_C \cdot \Lambda_C^{\frac{1}{2}}.$$

All data points included in the data set for constructing an index are transformed by A_C. For instance, A_C transforms a data point p in the data set into $p' = p \cdot A_C$. The MSTT constructs an index structure $\mathcal{I}_C$ based on the transformed data points such as p'. $\mathcal{I}_C$ can well support queries whose matrix is C.

Query Processing by MSTT:

For a query point q and a query matrix M, we first transform q into $q' = q \cdot A_C$ to perform this query using $\mathcal{I}_C$. Considering a new matrix M' for the query processing of M using $\mathcal{I}_C$:

$$M' = A_C^{-1} \cdot M \cdot (A_C^{-1})^t, \tag{12}$$

then the quadratic form distance of M between p and q is expanded as follows:

$$\begin{aligned} d_M^2(p,q) &= (p-q)\cdot M\cdot (p-q)^t \\ &= (p'-q')\cdot A_C^{-1}\cdot M\cdot (A_C^{-1})^t\cdot (p'-q')^t \\ &= (p'-q')\cdot M'\cdot (p'-q')^t. \end{aligned}$$

Thus, the query whose matrix is M' and point is q' using $\mathcal{I}_C$ leads to the search result of the ellipsoid query of M. Especially, if $M = C$, $\mathcal{I}_C$ can well support the query whose matrix is M since M' is unit matrix, which means a search based on the Euclidean distance function.

4.3 Similarity of Matrices

When more than one index structure is constructed, the search process must choose one of them for access. We define the dissimilarity between a query matrix M and an index $\mathcal{I}_C$ using the matrix flatness.

Queries of M using $\mathcal{I}_C$ utilizes M', calculated by Equation (12), as the query matrix. Let $\lambda_{M'_i}$ be the i-th dimensional eigenvalue of M' and $\overline{\lambda}_{M'}$ be the average of the eigenvalues of M'. The variance $\sigma^2_{M'}$ of eigenvalues of M' is determined as follows:

$$\sigma^2_{M'} = \sum_{i=1}^{d}(\lambda_{M'_i} - \overline{\lambda}_{M'})^2, \quad \overline{\lambda}_{M'} = \sum_{j=0}^{d}\frac{\lambda_{M'_j}}{d} \tag{13}$$

We define $\sigma^2_{M'}$ as the dissimilarity between M and $\mathcal{I}_C$. For similarity search using MSTT, the effectiveness of $\mathcal{I}_C$ against M improves as $\sigma^2_{M'}$ decrease.

4.4 Normalization of Matrices

In order to calculate the dissimilarity of queries and indices, all matrices must be normalized, i.e. $det(C) = det(M) = 1$ for matrices C and M. Normalized matrix N of M is obtained by:

$$N = E_M\cdot\Lambda_N\cdot E_M^t, \quad \lambda_{N_i} = \lambda_{M_i}\cdot\left(\prod_{i=1}^{d}\lambda_{M_i}\right)^{-\frac{1}{d}}$$

where the diagonal matrix Λ_N consists of the eigenvalues λ_{N_i} $(i = 1,\ldots,d)$ of N. C can also normalized in the same way.

5 Performance Evaluation

To verify the effectiveness of STT, we implemented the algorithm and compared it to the MBB-MBS approximation technique. We then measured the performance of MSTT. The measurements used the same conditions used for the tests in Section 2. For the dimensionality reduction technique of STT, the best threshold $\eta = 0.01$ from among three alternatives, $\eta = 0.1$, $\eta = 0.01$, $\eta = 0.001$, was chosen. With respect to the matrices created in our experiments, Table 2 shows the dimensions n used for STT.

Table 2: Dimensions used for ellipsoid queries.

	w_r	1	10	100	1000
n	$d = 8$	8	8	4	4
	$d = 27$	27	18	9	9

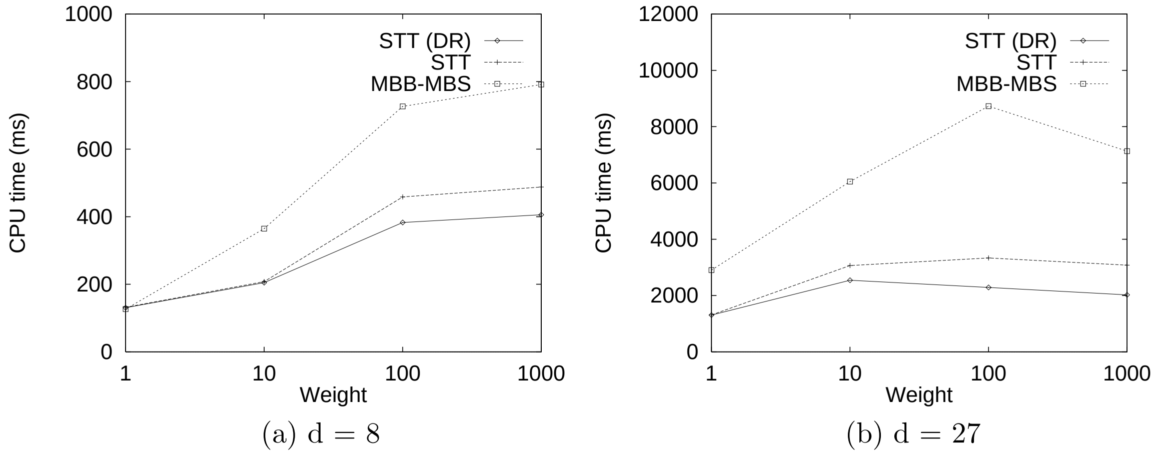


Figure 8: Comparison of STT against the MBB-MBS approximation technique in terms of CPU-cost.

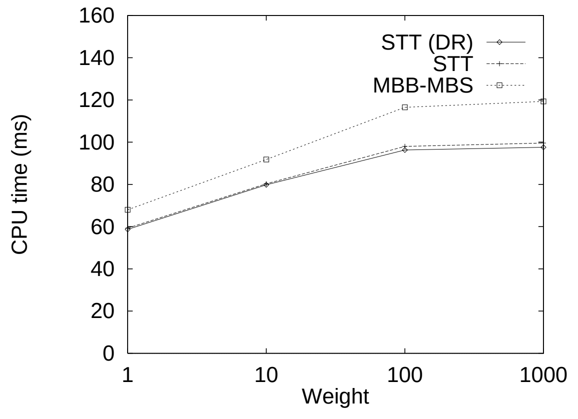
5.1 Search Performance

Figure 8 compares STT to the MBB-MBS approximation technique in terms of CPU-cost. The A-tree was used as an index structure. The symbol `STT(DR)` means the CPU-cost for the STT with dimensionality reduction. The number of page accesses is shown in Figure 4. Since STT and the MBB-MBS approximation technique utilize exact quadratic form distance functions, both require the same number of page accesses to perform ellipsoid queries. Thus, the difference in search time between STT and the MBB-MBS approximation technique depends on calculation complexity. As described in Section 1, ellipsoid queries incur much costs in calculating the distance between bounding rectangles and query points. Figure 8 shows that STT reduces CPU-cost for all data sets, and reveals the superiority of STT. In particular, the effectiveness of STT increases as either dimensionality or matrix flatness grows. Especially, STT achieves a 74 % reduction in CPU-cost for high dimensionality and matrix flatness.

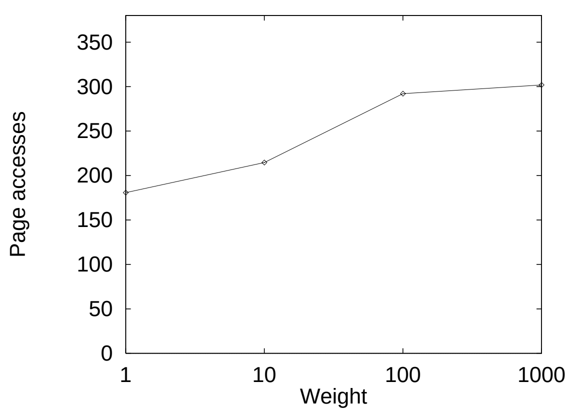
Figure 9(a) compares STT to the MBB-MBS approximation technique in terms of CPU-cost when using the R*-tree. Figure 9(b) shows the number of page accesses. This experiment utilized only 8-D data. With the available hardware the 27-D data lead to excessively long run times. STT using either the R*-tree or the A-tree was superior to the MBB-MBS approximation technique.

5.2 Analysis of Approximation Techniques for Elliptical Queries

STT does not utilize the exact quadratic form distance functions to access bounding rectangles whose approximation distance from the query point exceeds the actual

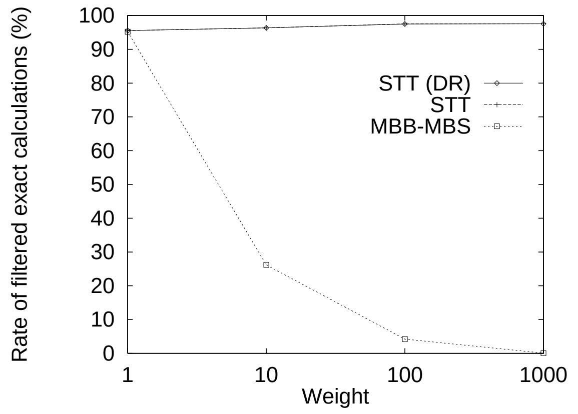


(a) CPU time

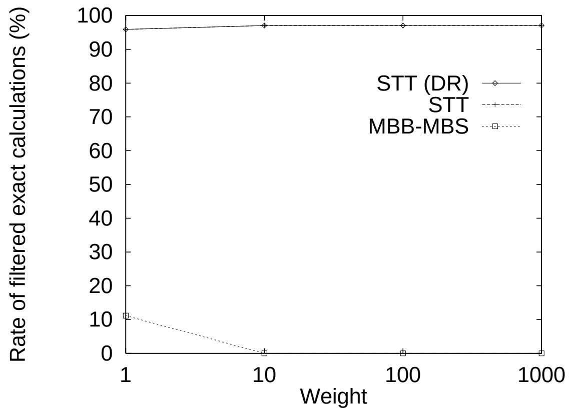


(b) Page accesses

Figure 9: Search performance using the R*-tree.



(a) d = 8

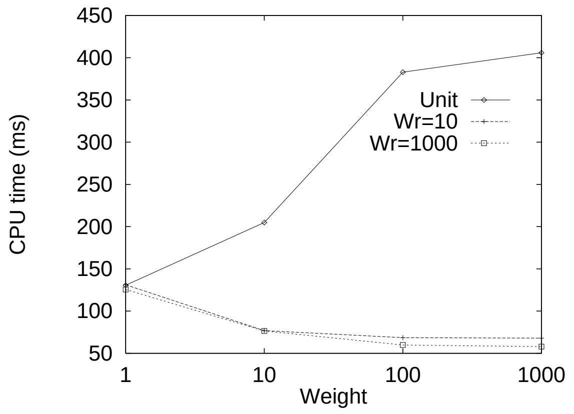


(b) d = 27

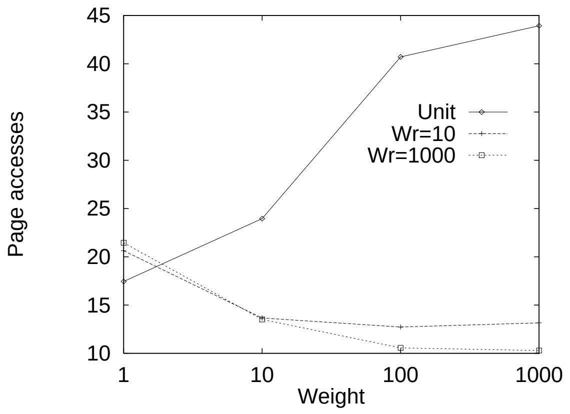
Figure 10: Rate of filtered exact distance calculations for STT.

k-nearest neighbor distance, similar to the MBB-MBS approximation technique. Figure 10 presents the percentage of filtered exact quadratic form distance calculations to the number of bounding rectangles accessed in search processing, that is, this figure illustrates the effectiveness of the approximation techniques. Although the efficiency of the MBB-MBS approximation technique decreases as the flatness of query matrix grows, the STT approximations efficiently filter exact quadratic form distance calculations for all queries. STT proves to be highly effective with high-dimensional data and queries whose matrix flatness is high as well as those with lower dimensionality and flatness. The effectiveness of STT yields low CPU-cost as shown in Figure 8.

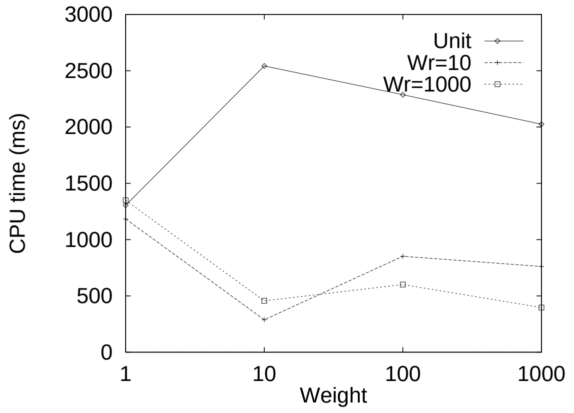
The idea of the dimensionality reduction provided by STT is eliminating dimensions that make only a slight contribution to the approximation of distance between query points and bounding rectangles. This technique is more effective in ellipsoid searches as the flatness of query matrices increases. Since the flatness of query $w_r = 1$ is relatively low, the query uses all dimensions in the search as shown in Table 2. On the other hand, for queries $w_r = 100$ and $w_r = 1000$, both of which have flat ellipsoids, distance calculations are based on lower dimensionality. As Fig-



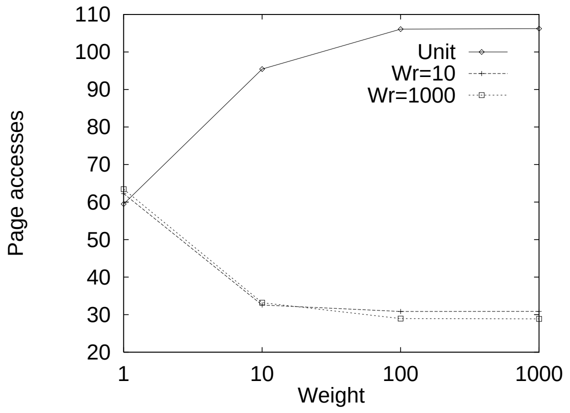
(a) CPU time, d = 8



(b) Page accesses, d = 8



(c) CPU time, d = 27



(d) Page accesses, d = 27

Figure 11: Search performance for MSTT.

ure 10 depicts, STT provides high approximation efficiency both with and without dimensionality reduction; as a result, the STT with dimensionality reduction has superior performance and offers lower CPU-cost.

5.3 Effectiveness of MSTT

Section 4 proposed the MSTT as a development of STT. For a given query matrix, this technique constructs an index structure based on the query matrix in order to support the query more efficiently. In these experiments, we measured the performance of MSTT with the following structures:

(1) **Unit:** the index structures constructed by the unit matrix.

(2) $\boldsymbol{W_r = 10}$**:** the index structures constructed by the matrix $w_r = 10$.

(3) $\boldsymbol{W_r = 1000}$**:** the index structures constructed by the matrix $w_r = 1000$.

Figure 11 depicts the search performance for ellipsoid queries using these index structures. Table 3 shows the dissimilarities between the three index structures and the four ellipsoid queries for 8-D and 27-D dimensions. The dissimilarities of index structures to queries were calculated using Equation (13) in Section 4.3. For any query, choosing the index structure that is most similar to the query minimizes

Table 3: Dissimilarity of matrices.

(a) **Unit**

w_r		1	10	100	1000
$\sigma^2_{M'}$	$d = 8$	0.031	76.490	7999	800214
	$d = 27$	64.777	93372	9.29e8	9.29e12

(b) $\boldsymbol{W_r} = \mathbf{10}$

w_r		1	10	100	1000
$\sigma^2_{M'}$	$d = 8$	71.296	0	194.99	19892
	$d = 27$	42467	0	127814	1.30e9

(c) $\boldsymbol{W_r} = \mathbf{1000}$

w_r		1	10	100	1000
$\sigma^2_{M'}$	$d = 8$	748634	19892	196.01	0
	$d = 27$	4.37e12	1.33e9	132022	0

search cost of the query. MSTT is very successful in reducing CPU-costs and the number of page accesses for any query.

In addition, search cost is not proportional to dissimilarity. For example, queries whose dissimilarity is 0 incur some search cost since similarity searches entail some cost even in the Euclidean distance space. Note that the function is not a cost model. Dissimilarity allows the search algorithm to choose a desirable index structure.

In practical situations, the dissimilarity of a given query matrix to each index must be calculated when choosing from the various index structures. The number of dissimilarity calculations grows as the number of created indices increases since the number of dissimilarity calculations equals the number of indices. However, each calculation incurs only a small CPU-cost: 2 ms for 27-D data in our experiments. Since this is negligible compared to the overall search time, constructing various structures substantially improves search performance.

5.4 Property of the dissimilarity function

In this section, we analyze the property of the dissimilarity function defined in Section 4.3. We created 30 query matrices for 8 and 27 dimensions using weight w_r as follows:

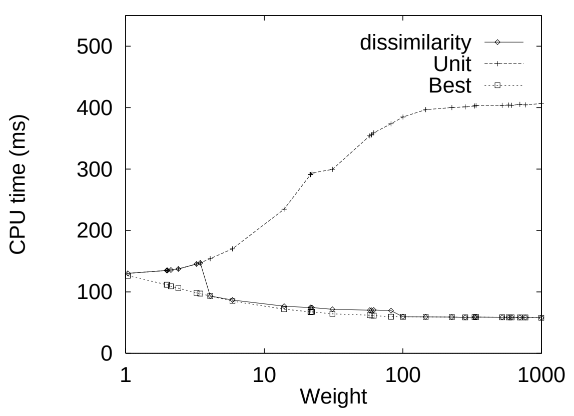
$$w_r = 10^{random},$$

where *random* is a randomly generated number between 0 and 3. Figure 12 shows how the dissimilarity function chooses an index structure. Each search cost for 30 queries was measured by the average of 100 queries. This experiment used three kinds of index, **Unit**, $\boldsymbol{W_r} = \mathbf{10}$ and $\boldsymbol{W_r} = \mathbf{1000}$, such as the experiment shown in Section 5.3. The figure shows the following search costs:

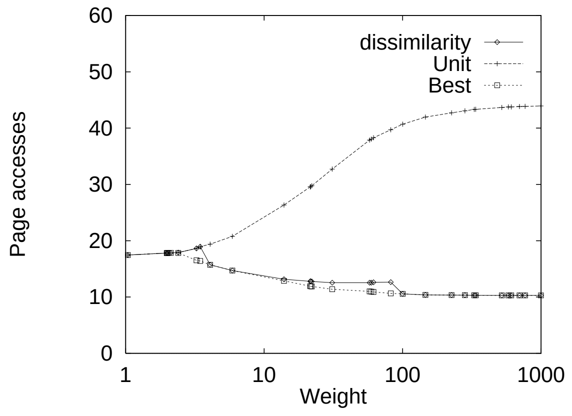
(1) **Dissimilarity:** the cost of search using index structures chose by the dissimilarity function.

(2) **Unit:** the search cost on index structures constructed by the unit matrix.

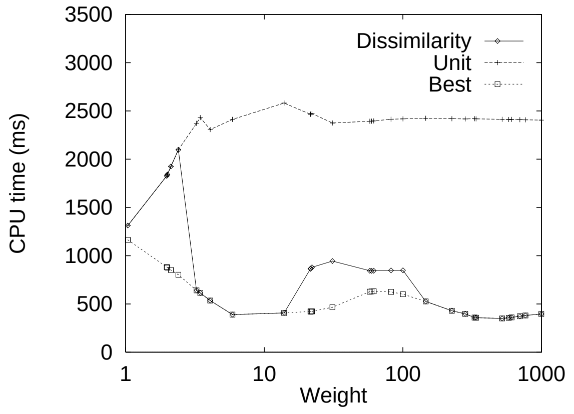
(3) **Best:** the best search cost using the optimal index structure for each query matrix.



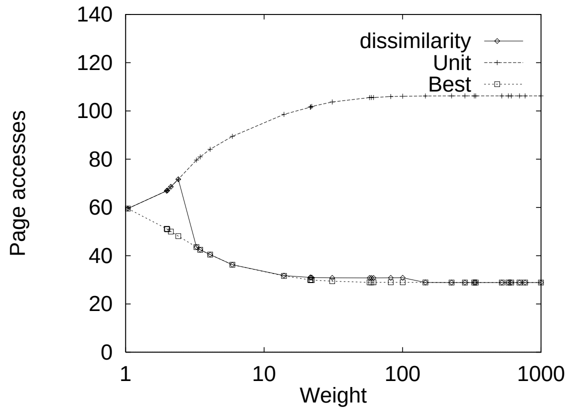
(a) CPU time, d = 8



(b) Page accesses, d = 8



(c) CPU time, d = 27



(d) Page accesses, d = 27

Figure 12: Behavior of the dissimilarity function.

In the experiment using 27-dimensional data, the dissimilarity function chooses the index structure **Unit** for the query matrices whose weight w_r lies between 1 and 3, $\boldsymbol{W_r} = \mathbf{10}$ for the weight between 3 and 100, and $\boldsymbol{W_r} = \mathbf{1000}$ for the weight between 100 and 1,000. The choice of indices for 8-dimensional data is quite similar to that for 27-D. Although search cost determined by the function is not exactly equal to the search cost achieved by the optimal index structures, the function chooses a desirable structure for most of the query matrices. Moreover, compared to the search cost with the index structure based on the Euclidean distance, index structures chosen by the dissimilarity function greatly reduce the search cost. Unlike the previous works presented in [SK97] and [ABKS98] that focus on the index structure based on the Euclidean distance, MSTT constructs various index structures, which allows the dissimilarity function to choose a desirable structure according to the query matrices. This mechanism offers high effectiveness as shown in this figure.

6 Conclusions

This paper has presented the Spatial Transformation Technique (STT); it offers excellent performance when searching for adaptive ellipsoid queries. First, we analyzed the MBB-MBS approximation technique and discussed its problems. Based on this analysis, we developed STT to overcome the problems and so achieve higher search performance.

STT achieves high performance due to its use of spatial transformation. Since the spatial transformation provides highly accurate approximations of the distance between query points and bounding rectangles, STT eliminates exact distance evaluations for most of accessed bounding rectangles in the index structures. The mechanism of STT is remarkably efficient, especially for queries whose dimensionality or matrix flatness is high. This technique guarantees no false drops as does the MBB-MBS approximation technique. In experiments using various matrices and index structures, STT was found to superior to the conventional search method, the MBB-MBS approximation technique.

We also presented a new technique, MSTT. MSTT adjusts tree structures to suit ellipsoid queries; the search algorithm utilizes the adjusted structures. This technique reduces both page access number and CPU-cost for ellipsoid queries.

MSTT can support ellipsoid queries efficiently because one or more index structures can be used. In the future, we plan to consider an algorithm that determines matrices from a log of user's queries to create various indices. We will present a matrix decision algorithm whose parameters are a log of queries and the number of indices that can be stored on disk.

References

[ABKS98] Mihael Ankerst, Bernhard Braunmüller, Hans-Peter Kriegel, and Thomas Seidl: "Improving Adaptable Similarity Query Processing by Using Approximations", in *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, pp. 206–217, New York City, NY, August 1998.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel: "The X-tree: An Index Structure for High-Dimensional Data", in *Proc. of the 22nd International Conference on Very Large Data Bases (VLDB)*, pp. 28–39, Bombay, September 1996.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger: "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", in *Proc. ACM SIGMOD Conf.*, pp. 322–331, Atlantic City, NJ, May 1990.

[BO97] Tolga Bozkaya and Meral Ozoyoglu: "Distance-Based Indexing for High-Dimensional Metric Spaces", in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 357–368, May 1997.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula: "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces", in *Proc. of the 23rd International Conference on Very Large Data Bases (VLDB)*, pp. 426—435, Athens, August 1997.

[FSA$^+$95] M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker: "Query by image and video content: the QBIC system", *IEEE Computer*, Vol. 28, No. 9, pp. 23–32, September 1995.

[GG97] Jim Gray and Goetz Graefe: "The Five-Minute Rule Ten Years Later and Other Computer Storage Rules of Thumb", *SIGMOD Record*, Vol. 26, No. 4, pp. 63–68, December 1997.

[GG98] Volker Gaede and Oliver Günther: "Multidimensional Access Methods", *ACM Computing Surveys*, Vol. 30, No. 2, pp. 170–231, June 1998.

[HS95] G. R. Hjaltason and H. Samet: "Ranking in Spatial Databases", in *Proceedings of the 4th Symposium on Spatial Databases*, pp. 83–95, Portland, Maine, August 1995.

[HSE+95] James L. Hafner, Harpreet S. Sawhney, William Equitz, Myron Flickner, and Wayne Niblack: "Efficient Color Histogram Indexing for Quadratic Form Distance Functions", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 7, pp. 729–736, July 1995.

[ISF98] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos: "MindReader: Querying databases through multiple examples", in *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, pp. 218–227, New York City, NY, August 1998.

[RHM97] Y. Rui, T. S. Huang, and S. Mehrotra: "Content-based Image Retrieval with Relevance Feedback in MARS", in *Proc. of IEEE International Conference on Image Processing*, pp. II–815–818, October 1997.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent: "Nearest Neighbor Queries", in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 71–79, May 1995.

[SK97] Thomas Seidl and Hans-Peter Kriegel: "Efficient User-Adaptable Similarity Search in Large Multimedia Databases", in *Proc. of the 23rd International Conference on Very Large Data Bases (VLDB)*, pp. 506—515, Athens, August 1997.

[SYUK00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima: "The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation", in *Proc. of the 26th International Conference on Very Large Data Bases (VLDB)*, pp. 516–526, Cairo, Egypt, September 2000.

A Proof of Equation (9)

Let t be an arbitrary vertex of P. The i-th dimensional coordinate value of t is:

$$t_i = p_{a_i} + l_i \cdot \beta_i,$$

where β_i is the number of 0 or 1. Since the spatial transformation of t is $t' = t \cdot A_M$, the j-th dimensional coordinate of t' is:

$$\begin{aligned} t'_j &= \sum_{i=1}^{d} t_i \cdot a_{ij} \\ &= p'_{a_j} + \sum_{i=1}^{d} l_i \cdot \beta_i \cdot a_{ij}. \end{aligned}$$

Here, we assume that t'_k is the highest coordinate value of P' for k dimensions. t'_k can be represented as:

$$\begin{aligned} t'_k &= p'_{a_k} + \sum_{i=1}^{d} l_i \cdot \beta_i \cdot a_{ik}, \\ \beta_i &= \begin{cases} 1 & (a_{ik} > 0) \\ 0 & (\text{otherwise}). \end{cases} \end{aligned}$$

Therefore, $t'_k = r_{b_k}$. We can say that r_b has the highest coordinate values of P' for all dimensions. Similarly, If t'_k is the lowest coordinate value of P' for k dimensions, t'_k can be represented as:

$$\begin{aligned} t'_k &= p'_{a_k} + \sum_{i=1}^{d} l_i \cdot \beta_i \cdot a_{ik} \\ &= r_{a_k}, \end{aligned}$$

where

$$\beta_i = \begin{cases} 1 & (a_{ik} < 0) \\ 0 & (\text{otherwise}). \end{cases}$$

Therefore, r_a has the lowest coordinate values of P' for all dimensions. □