Proc. of the Fifth International Symposium on Large Spatial Databases (SSD '97), Berlin, Germany, Lecture Notes in Computer Science, Springer, 1997.

3D Similarity Search by Shape Approximation[1]

Hans-Peter Kriegel, Thomas Schmidt, Thomas Seidl

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
http://www.dbs.informatik.uni-muenchen.de
{kriegel | schmidtt | seidl}@dbs.informatik.uni-muenchen.de

Abstract. This paper presents a new method for similarity retrieval of 3D surface segments in spatial database systems as used in molecular biology, medical imaging, or CAD. We propose a similarity criterion and algorithm for 3D surface segments which is based on the approximation of segments by using multi-parametric functions. The method can be adjusted to individual requirements of specific applications by choosing appropriate surface functions as approximation models. For an efficient evaluation of similarity queries, we developed a filter function which supports fast searching based on spatial index structures and guarantees no false drops. The evaluation of the filter function requires a new query type with multidimensional ellipsoids as query regions. We present an algorithm to efficiently perform ellipsoid queries on the class of spatial index structures that manage their directory by rectilinear hyperrectangles, such as R-trees or X-trees. Our experiments show both, effectiveness as well as efficiency of our method using a sample application from molecular biology.

1 Introduction

The problem we focus on in this paper is similarity retrieval of 3D surface segments in a spatial database system. We propose a new similarity measure for surface segments, based on the approximation of the segments by multi-parametric functions. Furthermore, we show how to efficiently evaluate similarity queries based on this measure using a spatial index structure.

Motivation. The task of finding similar surface segments arises in many recent application areas of spatial database systems, such as:

Molecular Biology. A challenging problem in molecular biology is the prediction of protein interactions (the molecular docking problem): Which proteins from the database form a stable complex with a given query protein? It is well known that docking partners are recognized by complementary surface regions. In many cases, the active sites of the proteins, i.e. the docking regions are known and can be extracted from the protein surface and subsequently stored in a database [SK 95]. Then, the problem of finding docking partners reduces to finding similar (complementary) surface segments from a large database of segments for a given query segment.

Medical Imaging. Modern medical imaging technology such as CTI or MRI produces descriptions of 3D objects like organs or tumors by a set of 2D images. These images represent slices through the object, from which the 3D shape can be reconstructed. A

1. This research was funded by the German Ministry for Education, Science, Research and Technology (BMBF) under grant no. 01 IB 307 B. The authors are responsible for the content of this paper.

method for retrieving similar surface segments would help to discover correlations between shape deformations of organs and certain deceases.

Further application fields include CAD and mechanical engineering. In order to meet the typical requirements of these application domains, our method supports invariance against translation and rotation, because the position and orientation of the objects in 3D space does not affect shape similarity. Since the number of objects in a spatial database typically is very large, efficient query processing is important and will be supported by our algorithm. Our method requires the surface segments to be given as sets of points which can be obtained from all common surface representations.

Related Work. In recent years, considerable work has been done on similarity search in database systems. Most of the previous approaches, however, deal with one- or two-dimensional data, such as time series, digital images or polygonal data. However, they do not handle three-dimensional objects.

Agrawal et al. present a method for similarity search in a sequence database of one-dimensional data [AFS 93]. The sequences are mapped onto points of a low-dimensional feature space using a Discrete Fourier Transform, and then a Point Access Method (PAM) is used for efficient retrieval. This technique was later generalized for subsequence matching [FRM 94], and searching in the presence of noise, scaling, and translation [ALSS 95]. However, it remains restricted to one-dimensional sequence data.

Jagadish proposes a technique for the retrieval of similar shapes in two dimensions [Jag 91]. He derives an appropriate object description from a rectilinear cover of an object, i.e. a cover consisting of axis-parallel rectangles. The rectangles belonging to a single object are sorted by size, and the largest ones serve as retrieval key for the shape of the object. Due to a normalization, invariance with respect to scaling and to translation is achieved. Though this method can be generalized to three dimensions by using covers of hyperrectangles, it has not been evaluated for real world 3D data, and furthermore, it does not achieve invariance against rotations.

Mehrotra and Gary suggest the use of boundary features for the retrieval of shapes [MG 93] [GM 93]. Here, a 2D-shape is represented by an ordered set of surface points, and fixed-sized subsets of this representation are extracted as shape features. All of these features are mapped to points in multidimensional space which are stored using a PAM. This method can handle translation, rotation and scaling invariance, as well as partially occluded objects, but is essentially limited to two dimensions.

For retrieving similar 2D polygon shapes from a CAD database system, previous work is presented in [BKK 97] and [BK 97]. This technique applies the Fourier Transform to shape encoding for retrieving similar sections of polygon contours. The polygon sections are stored as extended multidimensional feature objects and a Spatial Access Method (SAM) is used for efficient retrieval [BKK 96] [Ber+ 97]. This approach, however, is also limited to two dimensions.

Korn et. al. propose a method for searching similar tumor shapes in a medical image database [Kor+ 96]. However, they consider only 2D images, and the similarity measure that they investigate is volume based. This might cause problems when applied to surface segments, because surface segments in general do not enclose a volume. Therefore this method cannot be directly applied to the problem we focus on in this paper.

Last but not least, the QBIC (Querying By Image Content) system [Fal+ 94] contains a component for 2D shape retrieval where shapes are given as sets of points. The method is based on algebraic moment invariants and is also applicable to 3D objects [TC 91]. As an important advantage, the invariance of the feature vectors with respect to rigid transformations (translations and rotations) is inherently given. However, the adjustability of the method to specific application domains is restricted. From the available moment invariants, appropriate ones have to be selected, and their weighting factors may be modified. Whereas the moment invariants are abstract quantities, the approximation models that have to be chosen in our approach are concretely visualizable as 3D surfaces, thus providing an early impression of the suitability for certain application domains.

The paper is organized as follows: In section 2, we present the approach of 3D shape approximation and introduce a shape similarity function which will be used for the specification of similarity queries. Section 3 contains the derivation of an efficient method for similarity query processing. In section 4, we present an algorithm for ellipsoid query processing on spatial access methods. Section 5 shows experimental results, and section 6 concludes the paper.

2 3D Similarity based on Shape Approximation

In this section, we present a generic approximation method to represent the shape of 3D surface segments by multi-parametric surface functions. To adapt the shape similarity criterion to the user's application domain, simply instantiate the similarity search method by using an appropriate approximation model. The basic idea of the similarity criterion is that the approximations of the objects of interest are compared mutually. The quantity of (dis-)similarity is defined in terms of the mutual approximation error. Thus, the more an approximation model fits the characteristics of a specific application domain, the more powerful it is to differentiate between a variety of shapes.

The quality of approximation depends on the appropriateness of the approximation model that has been chosen for an actual application. If the real objects actually were instances of the approximation model, approximation errors would only occur due to errors in measurement and roundoff errors. However in general, the real objects of the application are not derived from an artificial approximation model, but the approximation acts as a simplified model for a variety of complex real objects. Thus, choosing an adequate approximation model for specific applications is an important design decision.

2.1 Approximation of 3D-Segments

The objects our method is designed for are 3D surface segments which are represented by sets of points. Multi-parametric two-dimensional (surface) functions $f_{app}(x,y)$ are used to approximate these objects (cf. figure 1). For example, we use paraboloids and trigonometric polynomials of various degrees as approximation models. The length r of the parameter set $app = \langle a_i \rangle$ is called the dimension of the approximation model.

Definition (*approximation error, approximation, relative approximation error*). Let s be a 3D surface segment given by n points $\{p\}$, and $app = \langle a_i \rangle$ any approximation parameter set for a given approximation model. (i) The *approximation error* is defined in a normalized least squares sense to be $d_s^2(app) := \frac{1}{n}\sum_{p \in s} (F_{app}(p_x, p_y) - p_z)^2$. (ii) A parameter set app_s for which the approximation error for s is minimum, $\forall app$: $d_s^2(app) \geq d_s^2(app_s)$, is called the *approximation* of s. (iii) The *relative approximation error* of an arbitrary approximation parameter set *app'* is defined to be $\Delta d_s^2(app') = d_s^2(app') - d_s^2(app_s)$.

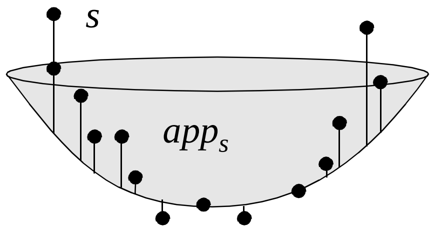

Fig. 1. A 3D surface segment s and its approximation app_s

Notice that in general, the approximation app_s of s is unique. In the worst case, the approximation parameters may vary without affecting the approximation error. Such a case indicates that the approximation model has been chosen inappropriate with respect to the application domain, and should be modified.

In general, even for the approximation app_s of a segment s, the approximation error $d_s^2(app_s)$ will be greater than zero. In order to end up with a similarity function that characterizes similarity of an object to itself by the value zero, we introduced the relative approximation error.

Lemma 1. (i) For any 3D surface segment s and any approximation parameter set *app'*, the relative approximation error is non-negative: $\Delta d_s^2(app') \geq 0$. (ii) The relative approximation error reaches zero. In particular, $\Delta d_s^2(app_s) = 0$ for all segments s.

Proof. (i) Since the approximation error $d_s^2(app_s)$ is defined to be minimum, the estimation $d_s^2(app') \geq d_s^2(app_s)$ holds for all parameter sets *app'*, and thus $\Delta d_s^2(app') \geq 0$. (ii) For $app' = app_s$ we have: $\Delta d_s^2(app_s) = d_s^2(app_s) - d_s^2(app_s) = 0$. ◊

3D-Normalization. In general, the points of a segment s are located anywhere in 3D space and are oriented arbitrarily. Since we are only interested in the shape of s, but not in its location and orientation in 3D space, we transform s by a rigid 3D-transformation into a normalized representation. There are two ways to integrate normalization: (1) *Separate*: First normalize the segment s, and then compute the approximation app_s by least-squares minimization. (2) *Combined*: Minimize the approximation error simultaneously over all the normalization and approximation parameters.

In our experiments, we used the combined normalization approach. For similarity search purposes, only the approximation parameters are used. However, the normalization parameters may be required later for superpositioning of segments.

Numerical Aspects. Approximation by least squares minimization is a numerical task. When restricting to certain approximation models, efficient solutions are available. An appropriate family of approximation models is given as linear combinations of the approximation parameters a_i with arbitrary two-dimensional functions $f_i\,(x,y)$. Examples are paraboloids such as $a_1 \cdot x^2 + a_2 \cdot y^2$ for degree two, or trigonometric polynomials such as $a_1 \cdot cos(x) + a_2 \cdot sin(x) + a_3 \cdot cos(y) + a_4 \cdot sin(y)$. The generic formula of linear combination models is as follows:

$$F_{app}(x,y) = \sum_{i=1\ldots r} app_i \cdot \mathrm{f}_i(x,y)$$

For such linear combination models, [PTVF 92] recommend to perform least-squares approximation by Singular Value Decomposition (SVD) as the method of choice. Besides the r approximation parameters $app = \langle a_i \rangle$, SVD also returns an r-vector w of condition (confidence) factors as well as an orthogonal $r \times r$-matrix V supporting the computation of relative approximation errors. When denoting the rows of V_s by V_{si}, and $A_s = V_s \cdot \mathrm{diag}(w_s)^2 \cdot V_s^{\mathrm{T}}$, the error formula is as follows:

$$\Delta \mathrm{d}_s^2(app') = \sum_{i=1\ldots r} w_{si}^2((app' - app_q) \cdot \mathrm{V}_{si})^2 = (app' - app_s) \cdot A_s \cdot (app' - app_s)^{\mathrm{T}}$$

2.2 Shape Similarity of 3D-Segments

For 3D surface segments, our shape similarity criterion considers two components, the extension of the segment in 3D space, and the shape of the segments in a narrow sense. We define the extension similarity to be the Euclidean distance of the 3D extension vectors which we obtain as the principle moments of inertia. The shape component of the distance function is based on shape approximation, and we exploit more information than only the approximation parameters. This is recommended, since the confidence of the parameters may substantially vary for single segment approximations. Thus, we introduce the method of *mutual approximation errors*. The basic question for shape similarity quantification is the following: How much will the approximation error increase, if segment q would have been approximated by the approximation of segment s, app_s, instead of its own approximation, app_q, and vice versa (cf. figure 2). This approach leads us to the definition of approximation-based shape similarity.

Definition (*shape similarity*). Let s and q be two 3D surface segments, with ext_s and ext_q being their 3D extension vectors in space. Define the mutual approximation distance to be $\mathrm{d}_{\mathrm{app}}^2(s, q) := \frac{1}{2}\Delta \mathrm{d}_s^2(app_q) + \frac{1}{2}\Delta \mathrm{d}_q^2(app_s)$, and the 3D extension distance as the squared Euclidean distance $\mathrm{d}_{\mathrm{ext}}^2(s, q) := (ext_s - ext_q)^2$. With u_{app} and u_{ext} being non-negative weighting factors, the shape similarity function $\mathrm{d}_{\mathrm{shape}}$ is defined to be:

$$\mathrm{d}_{\mathrm{shape}}(s, q) = \sqrt{u_{\mathrm{app}} \mathrm{d}_{\mathrm{app}}^2(s, q) + u_{\mathrm{ext}} \mathrm{d}_{\mathrm{ext}}^2(s, q)}$$

Additionally, for a segment s, we define the r+3-dimensional key vector key_s to be the concatenation of the r-dimensional vector app_s and the 3-dimensional vector ext_s: $key_s = (app_s, ext_s)$.

Since we have chosen to combine $\mathrm{d}_{\mathrm{app}}^2$ and $\mathrm{d}_{\mathrm{ext}}^2$ by the square root over the sum, $\mathrm{d}_{\mathrm{shape}}$ is related to the Euclidean distance in the following way:

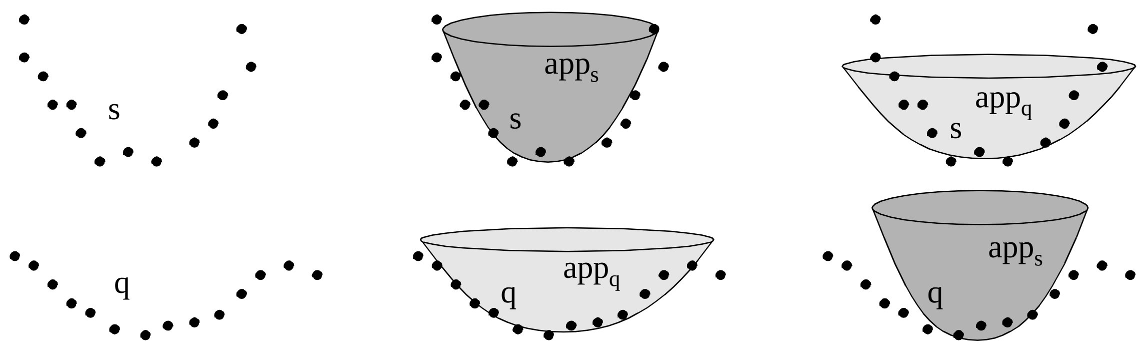

Fig. 2. Similarity quantification by mutual approximation and 3D extension distance

Lemma 2. Let s and q be two segments. If for both approximations, app_s and app_q, all the values w_{si} and w_{qi} are equal to 1, as well as the weighting factors $u_{\text{app}} = u_{\text{ext}} = 1$, the shape similarity $d_{\text{shape}}(s,q)$ equals to the Euclidean distance of the key vectors, key_s and key_q.

Proof. Recall the error formula $\Delta d_s^2(app') = \sum_i w_{si}^2((app' - app_q) \cdot V_{si})^2$, and assume all the w_{si} and w_{qi} being equal 1. Since the orthogonal matrices V_s and V_q represent pure base transformations without any scaling, we obtain $\Delta d_s^2(app') = \sum_i((app' - app_q) \cdot V_{si})^2 = ((app' - app_q) \cdot V_s)^2 = (app' - app_s)^2$. Overall, the following holds: $d_{\text{shape}}(s, q)^2 = 1 \cdot d_{\text{app}}^2(s, q) + 1 \cdot d_{\text{ext}}^2(s, q) = \frac{1}{2}\Delta d_s^2(app_q) + \frac{1}{2}\Delta d_q^2(app_s) + d_{\text{ext}}^2(s, q) =$ $= \frac{1}{2}(app_q - app_s)^2 + \frac{1}{2}(app_s - app_q)^2 + (ext_s - ext_q)^2 = (key_s - key_q)^2$. ◊

In general, the approximation confidence values w_i will be different from 1, and similarity search methods based on Euclidean distance in feature spaces (cf. [AFS 93], [BKK 97]) do not support shape similarity query processing immediately.

As for other similarity functions, a small value of d_{shape} indicates a high degree of similarity, whereas a large value of d_{shape} signals strong dissimilarity. In table 1, we summarize the definitions that we introduced in this section for shape approximation and shape similarity.

2.3 Sample Application

We successfully applied the similarity distance d_{shape} to the area of molecular biology, in particular, molecular recognition (docking). As a sample application, we present the search for similar docking sites in a database of some 6,200 docking segments. The protein data are available from the Brookhaven Protein Data Bank (PDB) which currently provides the atomic coordinates of some 3,000 proteins [Ber+ 77]. From the FSSP (Families of Structurally Similar Proteins) database [HS 94], we selected families of molecules that are similar in their sequence and, hence, are similar in their 3D shape. Examples for our experimental evaluation are the azurin family (PDB code 1AZC-A)

description	symbol	definition
approximation model	$F_{app}(x, y)$	$\sum_{i=1\ldots r} a_i \cdot \mathrm{f}_i(x, y) = (a_1, \ldots, a_r) \cdot (f_1(x, y), \ldots, f_r(x, y))$
approximation error	$d_s^2(app)$	$\frac{1}{n}\sum_{p \in s} (F_{app}(p_x, p_y) - p_z)^2$
(the) approximation	app_s	$\forall app\colon d_s^2(app) \geq d_s^2(app_s)$
relative approximation error	$\Delta \mathrm{d}_s^2(app')$	$d_s^2(app') - d_s^2(app_s) = (app' - app_s) \cdot A_s \cdot (app' - app_s)^{\mathrm{T}}$
key vector	key_s	$(app_s, ext_s) = (a_1, \ldots, a_r, e_1, e_2, e_3)$
(simple) shape similarity	$\mathrm{d}^2_{\mathrm{app}}(s, q)$	$\frac{1}{2}\Delta\mathrm{d}_s^2(app_q) + \frac{1}{2}\Delta\mathrm{d}_q^2(app_s)$
3D extension similarity	$\mathrm{d}^2_{\mathrm{ext}}(s, q)$	$(ext_s - ext_q)^2$
(final) shape similarity	$\mathrm{d}_{\mathrm{shape}}(s, q)$	$\sqrt{u_{\mathrm{app}}\mathrm{d}^2_{\mathrm{app}}(s, q) + u_{\mathrm{ext}}\mathrm{d}^2_{\mathrm{ext}}(s, q)}$ with weighting factors u_{app} and u_{ext}

Table 1. Symbols and definitions used for shape approximation and shape similarity of segments

covering four proteins with a high structural similarity, and the fructose bisphosphatase family (PDB code 1FRP-A) with 18 members.

We queried the database of 6,200 segments with the docking segment of each member of the azurin family. Figure 3 shows how the docking segments of the four azurin molecules rank within the database according to the shape distance $\mathrm{d}_{\mathrm{shape}}$. We compare different approximation models that have 2, 4, 8, and 12 parameters, leading to 5-, 7-, 11-, and 15-dimensional *key* vectors, respectively (cf. table 2). In the diagrams, the abscissa axis indicates the four azurin segments, whereas the ordinate axis depicts the minimum, maximum, and average position that has been achieved by the ranking according to $\mathrm{d}_{\mathrm{shape}}$ within all the 6,200 entries. The experiments support the adequacy of the TRIGO-4 model for the docking-sites application domain. TRIGO-4 is the trigonometric polynomial function $\mathrm{trigo4(x,y)} = \mathrm{a_1 cos\ x + a_2 sin\ x + a_3 cos\ y + a_4 sin\ y}$.

model	formula of approximation model
PARAB-2	$(a_1, a_2) \cdot (x^2, y^2) = a_1 x^2 + a_2 y^2$
TRIGO-4	$(a_1, a_2, a_3, a_4) \cdot (\cos x, \sin x, \cos y, \sin y)$
TRIGO-8	$(a_1, \ldots, a_8) \cdot (\cos x, \sin x, \cos y, \sin y, \cos 2x, \sin 2x, \cos 2y, \sin 2y)$
TRIGO-12	$(a_1, \ldots, a_{12}) \cdot (\cos x, \sin x, \cos y, \sin y, \ldots, \cos 3x, \sin 3x, \cos 3y, \sin 3y)$

Table 2. A sample of approximation models of various dimensionalities

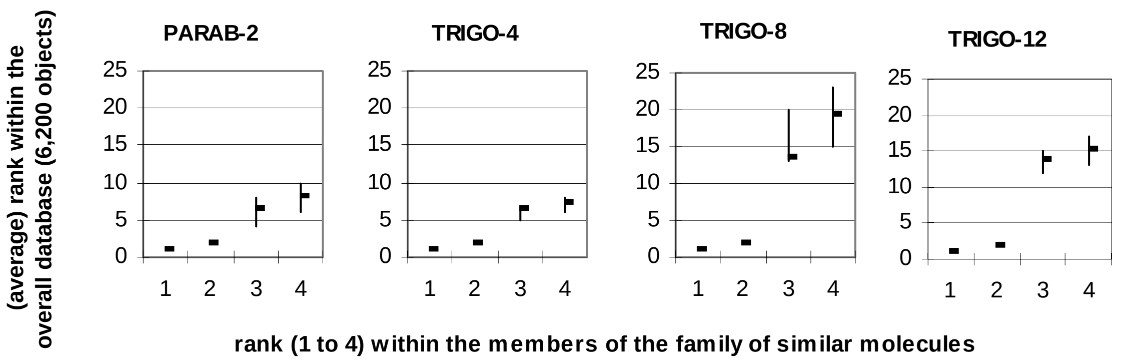

Fig. 3. Ranking by d_{shape} for various approximation models (example: azurin family). The members of the example family rank on top positions, in particular below rank 25 out of 6,200. The best positions are achieved for the TRIGO-4 model.

Additional experiments e.g. for hemoglobin molecules or for trypsin inhibitors show that most of the molecules that were ranked on top positions within the overall database according to d_{shape} also belong to the same family of molecules as the query object.

3 Efficient Similarity Query Processing

There are two types of queries that are relevant with respect to similarity search: range queries and *k*-nearest neighbor (*k*-nn) queries. A range query $SIM_{range}(q, \varepsilon)$ is specified by a query object *q* and a range value ε, defining the answer set to contain all the objects *s* from the database that have a distance less than ε to the query object *q*. A *k*-nearest neighbor query $SIM_{k\text{-}nn}(q, k)$ for a query object *q* and a cardinal number *k* specifies the retrieval of those *k* objects from the database that are most similar to *q*.

Due to the immense and even increasing size of current databases for molecular biology, medical image management, and engineering applications, strong efficiency requirements have to be met. Thus, for the evaluation of complex similarity queries, fast processing is important. As successfully applied for processing of point queries, window queries, and spatial joins, a multi-step query processing architecture is recommended [OM 88] [BHKS 93] [BKSS 94]. Following this paradigm, several filter steps produce and reduce candidate sets from the database, yielding an overall result that contains the correct answer, producing neither false positive nor false negative decisions (no false hits and no false drops). To improve efficiency, filter steps are usually supported by PAMs and SAMs. Refinement steps discard false positive candidates (false hits), but do not reconstruct false negatives (false drops) that have been dismissed by the filter step. Therefore, a basic requirement for any filter step is to prevent false drops.

3.1 Multi-Step Query Processing

For feature-based similarity search, efficient query processing already is available [AFS 93] [Kor+ 96] [BKK 97] . The objects *O* are transformed to feature vectors *F(O)* which are efficiently managed by multi-dimensional point access methods (PAM). The algorithms are shown to work correctly, in particular, producing no false drops, if they use a feature distance function D_{feature} that lower-bounds the object distance functions D_{object}, i.e. $D_{\text{feature}}(F(O_1), F(O_2)) \le D_{\text{object}}(O_1, O_2)$. The efficiency of the algorithms depends on selectivity and the query processing efficiency provided by the underlying PAM. Figures 4 and 5 show the algorithms $\text{SIM}_{\text{range}}(q, \varepsilon)$ for range queries (cf. [FRM 94]), and $\text{SIM}_{\text{k-nn}}(q, k)$ for *k*-nn queries from [Kor+ 96], where the distance function D_{feature} is a filter function $f(s, q)$, and D_{object} is d_{shape}.

```
Algorithm SIM_range (q, ε)
(1) Filter Step. Using the filter function $f(s, q)$, perform a range query on the PAM to
    obtain the candidate set $\{s \mid f(s, q) \le \varepsilon\}$.
(2) Refinement Step. For each of the candidates s, evaluate the shape similarity
    $d_{\text{shape}}(s, q)$, and report those objects that fulfill $d_{\text{shape}}(s, q) \le \varepsilon$
```

Fig. 4. Algorithm for range queries wrt d_{shape}, based on a PAM, cf. [FRM 94]

```
Algorithm SIM_k-nn (q, k)
(1) Primary Candidates. Perform a k-nn query on the PAM to find the k-nn objects with
    respect to the filter function $f(x, q)$.
(2) Range Determination. For all the candidates s obtained from (1), compute the actual
    distance $d_{\text{shape}}(s, q)$ and return the maximum value $d_{max}$.
(3) Final Result. Perform a range query on the PAM to obtain the candidate set
    $\{s \mid f(s, q) \le d_{max}\}$. For all the candidates s, evaluate $d_{\text{shape}}(s, q)$, and report the
    objects having the k smallest values.
```

Fig. 5. Algorithm for *k*-nearest neighbors queries wrt d_{shape}, based on a PAM, cf. [Kor+ 96]

To adapt the algorithms to shape similarity query processing, it suffices to provide a filter function $f(x, q)$ that fulfills the correctness and efficiency requirements, i.e.:

(1) Show the correctness of the filter function, i.e. $f(s, q) \le d_{\text{shape}}(s, q)$.

(2) Provide efficient algorithms to perform queries on the PAM that respect the distance function $f(s, q)$.

3.2 A Lower Bound for Shape Similarity

In order to apply the multi-step query processing paradigm to shape similarity search, we derive an appropriate filter function for the shape similarity function d_{shape}. Besides the correctness, which is ensured by a lower-bounding criterion, we look for a solution that provides efficient support by PAMs. Although a PAM has been developed that efficiently manages feature spaces of dimension 10 to 20 or more [BKK 96], the lower dimensionality promises the better performance.

Let us investigate the similarity function d_{shape} and its components with respect to the number of data values that are required for the evaluation. Table 3 illustrates the situation: Observe that for the evaluation of $d_{\text{shape}}(s, q)$, r^2+r data values are required concerning the segment q, since the formula contains the r-vector app_q as well as the $r \times r$-matrix A_q. Concerning s, only the r-vector app_s is required. For the extension distance $\mathrm{d}^2_{\mathrm{ext}}(s, q) = (ext_s - ext_q)^2$, it is sufficient to provide the 3D extension vectors ext_q and ext_s.

In order to avoid a dimensionality of the index that is quadratic in the dimensionality of the approximation model, in the filter step, only the components 2 and 3 (cf. table 3) will be evaluated. The overall evaluation of $\mathrm{d}_{\mathrm{shape}}$ is deferred to the refinement step.

Now, for any given query object q, let us compose the $(r+3)\times(r+3)$-matrix A'_q from the $r\times r$-matrix A_q and the 3×3 unit matrix I_3, resulting in $A'_q = \begin{bmatrix} \frac{1}{2}u_{\mathrm{app}}A_q & 0 \\ 0 & u_{\mathrm{ext}}I_3 \end{bmatrix}$. Recall the definition of the $r+3$ key vectors $key_s = (app_s, ext_s)$, and define the function $f_q\colon \Re^{r+3} \to \Re$ as follows:

$$f_q(key_s) = \sqrt{(key_s - key_q) \cdot A'_q \cdot (key_s - key_q)^{\mathrm{T}}}$$

	term	formula	data of q	data of s
1	$\Delta\mathrm{d}_s^{2}(app_q)$	$(app_q - app_s) \cdot A_s \cdot (app_q - app_s)^{\mathrm{T}}$	r	r^2+r
2	$\Delta\mathrm{d}_q^{2}(app_s)$	$(app_s - app_q) \cdot A_q \cdot (app_s - app_q)^{\mathrm{T}}$	r^2+r	r
3	$\mathrm{d}^2_{\mathrm{ext}}(s, q)$	$(ext_s - ext_q)^2$	3	3

Table 3. Number of data values required for the evaluation of the similarity function $\mathrm{d}_{\mathrm{shape}}(s, q)$

The following lemma states the lower-bounding property of f_q with respect to $\mathrm{d}_{\mathrm{shape}}$:

Lemma 3. Given an r-dimensional approximation model with two positive weighting factors u_{app} and u_{ext}, and two arbitrary segments s and q. The filter function $f_q(key_s)$ lower-bounds the shape distance $\mathrm{d}_{\mathrm{shape}}(s, q)$, i.e. $f_q(key_s) \le \mathrm{d}_{\mathrm{shape}}(s, q)$.

Proof. Observe that $f_q(key_s)^2 = \frac{1}{2}u_{\mathrm{app}}\Delta\mathrm{d}^2_q(app_s) + u_{\mathrm{ext}}\mathrm{d}^2_{\mathrm{ext}}(s, q)$, and consider the difference $\mathrm{d}_{\mathrm{shape}}(s, q)^2 - f_q(key_s)^2 = \frac{1}{2}u_{\mathrm{app}}\Delta\mathrm{d}^2_s(app_q)$, which is greater or equal to zero, according to Lemma 1. Since $f_q(key_s)^2 \le \mathrm{d}_{\mathrm{shape}}(s, q)^2$, also $f_q(key_s) \le \mathrm{d}_{\mathrm{shape}}(s, q)$ due to Lemma 1. ◊

Now, the obvious strategy is as follows: Construct a PAM on the $r+3$-dimensional key vectors key_s of all the segments s in the database. Then provide efficient processing methods for filter queries on the PAM, i.e. range queries $f_q(x) \le \varepsilon$, and k-nn queries using the filter distance function $f_q(x)$.

4 Ellipsoid Queries on Point Access Methods

The efficiency of all the algorithms mentioned above is due to the efficiency provided by the underlying PAM. In the following, we focus on access methods that manage the secondary storage pages by rectilinear hyperrectangles, e.g. minimum bounding rectangles (MBR), for forming higher level directory pages. For instance, this paradigm is realized in the R-tree [Gut 84] and its derivatives, R^+-tree [SRF 87], R*-tree [BKSS 90], as well as the X-tree [BKK 96], which has already been used successfully to support query processing in high-dimensional spaces.

4.1 Ellipsoids as Query Objects

Observe that $f_q(x)^2 - c = 0$ with $x \in \Re^{r+3}$, $c \in \Re$, is a quadratic equation that represents an r+3-dimensional ellipsoid for a given level c. Thus, the query region specified by the range criterion $f_q(x) \le \varepsilon$ which is equivalent to $f_q(x)^2 \le \varepsilon^2$, is an r+3-dimensional ellipsoid of the level ε^2 (cf. figure 6).

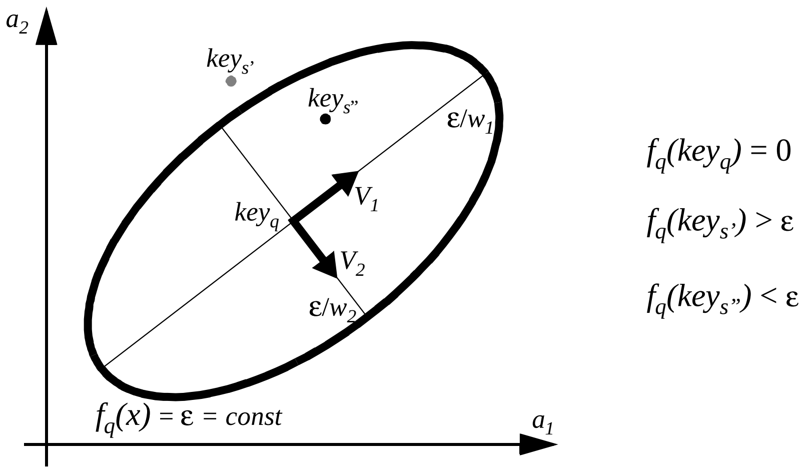

Fig. 6. Ellipsoid for distance range ε in a two-dimensional parameter space

Query processing on MBR-based access methods requires operations for query criterion evaluation of both, data objects (e.g. *query-contains-object*) and MBRs (e.g. *query-intersects-box*). For a query object q, we instantiate the query ellipsoid $ellip_q$. A method *eval* is provided for the evaluation of the filter function $f_q(p)$: $ellip_q.eval(p) \equiv f_q(p)$. In case of a range query, $\mathrm{SIM}_{\mathrm{range}}(q, \varepsilon)$, the query object is an ellipsoid around q whose boundary is determined by ε.

In figure 7, we give an illustration of the operations in case of a three-dimensional parameter space. The default way for range query processing would be to compute the MBR of the query ellipsoid, and to perform the standard range query algorithm as filter step. However in general, an MBR is a bad approximation of an ellipsoid, since it introduces a large portion of dead space. The percentage of dead space rapidly grows with increasing dimensionality and affects the selectivity of the index in a negative way.

Therefore, we suggest to perform ellipsoid queries in a direct way: The *ellip-contains-object* operation simply consists of an evaluation of the ellipsoid function, $ellip_q.eval(p) \le \varepsilon$. The *query-intersects-box* operation is more complex: The intersection predicate evaluates to true iff there exists a point that is contained in both the box and the ellipsoid. Our approach is to transform the intersection criterion into an optimization problem: Let p_0 be the point within the box that minimizes the ellipsoid function f_q, i.e. $f_q(p_0) = \min\{f_q(p) \mid p \in \mathrm{box}\}$, and test whether $f_q(p_0) \le \varepsilon$. Since we are only interested in the fact of intersection but not in the actual minimum value, the algorithm may stop if $f_q(p)$ is below the query range ε, thus improving efficiency.

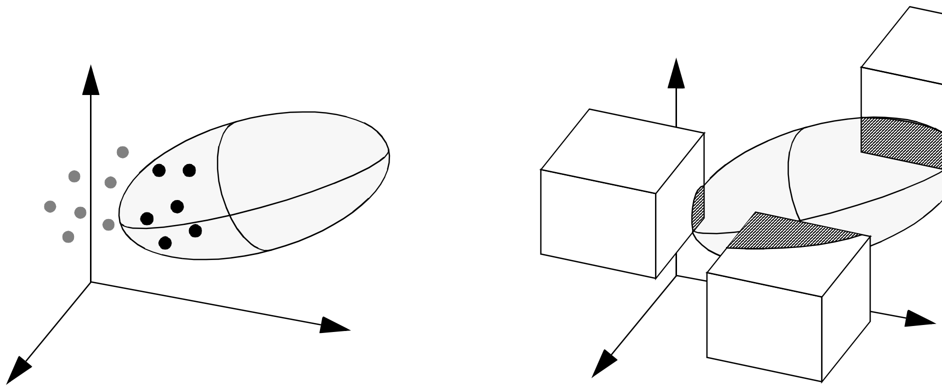

Fig. 7. Problems *ellipsoid contains point* and *ellipsoid intersects box*

For k-nn queries, $\mathrm{SIM}_{k\text{-nn}}(q, k)$, we fall back on the algorithms of [RKV 95] or [HS 95] which provide efficient processing of k-nn queries on PAMs for the Euclidean distance. Both approaches are based on a ranking of distance values. For the processing of ellipsoid queries, we modify the algorithms such that they use the filter distance function f_q instead of the Euclidean distance.

As before, the ellipsoid function, $ellip_q.eval\,(p) = f_q(p)$, returns the distance of the query ellipsoid to a database object p. For the MBRs, the required operation is *mindist*[1], yielding the minimum distance of the box with respect to the query ellipsoid. Thus, once more the ellipsoid function minimization can be applied, since $mindist(ellip, box) = \min\{ellip_q.eval(p) \mid p \in box\}$.

Since the two box operations are closely related, we generalize them and provide a single basic operation for ellipsoids concerning boxes, the distance function *ellip.distance*(*box, limit*) for a (hyper-)rectangle *box* and a float parameter *limit*. The function *distance* returns the minimum distance d_{min} from *ellip* to *box* if $d_{min} \geq limit$, and an arbitrary value below *limit* if $d_{min} < limit$. The relationship of *distance* to the operations *intersect* and *mindist* is as follows:

Lemma 4. The function *ellip.distance*(*box, limit*) fulfills the following correspondences:

(i) bool *intersect* (*box*, *ellip*, ε) ≡ *ellip.distance* (*box*, ε) ≤ ε

(ii) float *mindist* (*box, ellip*) ≡ *ellip.distance* (*box*, 0)

Proof. (ii) Since $d_{min} > 0$, *ellip.distance*(*box,* 0) always returns the actual minimum which is never below the *limit* = 0. (i) By definition, the inequality $ellip.distance(box, \varepsilon) \leq \varepsilon$ holds if and only if the minimum distance d_{min} of *ellip* to *box* is lower or equal to ε. On the other hand, $d_{min} \leq \varepsilon$ holds if and only if the (hyper-)rectangle *box* intersects the ellipsoid *ellip* of level ε. ◊

In order to test intersection, Lemma 4 helps to improve runtime efficiency, since the exact value of mindist is not required as long as it is smaller than the limit parameter. In case of intersection, an approximate value suffices and may save iterations.

1. In [RKV 95], also a *minmaxdist* operation is investigated. Due to space limitation, we drop the explanation how to adapt minmaxdist to ellipsoid queries.

4.2 Basic Ellipsoid-and-Box Algorithm

The specification of *ellip.distance*(*box, limit*) is to return the minimum distance $d_{min} = \min\{ellip_q(p) \mid p \in box\}$ from the ellipsoid *ellip* to the (hyper-)rectangle *box* if d_{min} is greater than *limit,* and to return an arbitrary value lower than *limit* otherwise.

For the evaluation of *ellip.distance*(*box, limit*), we combined two paradigms, the steepest descent method and iteration over feasible points. The steepest descent as a general optimization technique is applicable to the ellipsoid function since at every location in space, the gradient can be determined. The steepest descent works correctly and stops after a finite number of iterations [PTVF 92]. The feasible points paradigm is adapted from the linear programming algorithm of [BR 85]. The basic idea is that every point that is visited on the way downto the minimum should belong to the feasible domain. In our case, we start within the box, and ensure that our point does not leave the box. Thus, if the algorithm reaches a point that also lies within the ellipsoid of level ε, the ellipsoid and the box intersect (cf. figure 8).

```
function ellip :: distance (box, limit) —> float;
1   $p_0$ := box.closest (this.center);                      // starting heuristics
2   loop
3       if (this.eval ($p_i$) ≤ limit) break;                 // ellipsoid is reached
4       g := – this.gradient ($p_i$);                         // descending gradient
5       g := box.truncate ($p_i$, g);                         // truncate by boundary
6       if (|g| = 0) break;                                   // no feasible progress
7       s := this.linmin ($p_i$, g);                          // linear minimization
8       $p_{i+1}$ := box.closest ($p_i$ + s*g);               // projection onto box
9       if (this.eval ($p_i$) ≈ this.eval ($p_{i+1}$)) break; // no more progress
10  endloop
11  return this.eval ($p_i$);
end distance;
```

Fig. 8. The basic algorithm *ellip.distance (box, limit)* iterates over feasible points p_i within the *box* until *limit* or the minimum of the ellipsoid *this* is reached

The algorithm *distance* works as follows: The iteration with gradient computation (step 4), linear minimization (step 7), and termination criterion (step 9) performs the steepest descent. The projection in step (8) ensures that *p* does not leave the box, whereas steps (1) and (5) apply heuristics to accelerate the iteration, as we will explain below.

The linear minimization *ellip.linmin (p,g)* in step (7) returns the scaling factor *s* for which $p+s \cdot g$ evaluates to a minimum with respect to the function value of *ellip*, which is fulfilled in case of $\nabla_{\text{ellip}}(p + sg) \cdot g = 0$. This equation is immediately solved to $s = -(\nabla_{\text{ellip}} p \cdot g) / (\nabla_{\text{ellip}} g \cdot g)$.

The feasibility property of the algorithm is ensured by the closest point operation provided by the box type. Every point that is visited by the iteration is projected into the box, or even to the box boundary. Since the box boundary is given by iso-oriented hyperplanes, the projection can be performed in linear time with respect to the number of dimensions (cf. figure 9).

```
box.closest(p)[d]  ≡  if    (p[d] < box.lower[d]) then box.lower[d]
                      elsif (p[d] > box.upper[d]) then box.upper[d]
                      else p[d];
```

Fig. 9. Projection of a point to a box (closest point) for each dimension d

The algorithm works for any starting point that is feasible, i.e. which is located within the box, e.g. the center of mass or any of the corner points. As a heuristics to save iteration steps, we suggest to start at the point of the box that is closest to the ellipsoid center point. This may be a corner of the box, but in general, it is not. Since the boxes are iso-oriented, the closest point of a box with respect to any given location in space is easily determined by projecting each of the coordinates onto the corresponding box range. If the ellipsoid center lies within the box, it is closest to itself, and iteration will stop immediately at the beginning. Note that the closest point is not yet the global minimum we look for, because in general, the ellipsoid distance function is not compatible with the homogeneous Euclidean distance function.

Another heuristics to save iterations is the gradient truncation (step 5). Obviously, starting at a point from the box boundary, we are not allowed to proceed in any direction that would leave the box. Only directions along the boundary, or towards the box interior, are feasible. Therefore, we truncate the gradient by nullifying components that would leave the box. Formally, we decompose the gradient g into two components $g = g_{remaining} + g_{leaving}$, and proceed only in the direction $g_{remaining}$ that does not leave the box (cf. figure 10).

```
box.truncate (base, dir)[d]  ≡  if    (base[d] = box.lower[d] and dir[d] < 0) then 0
                                elsif (base[d] = box.upper[d] and dir[d] > 0) then 0
                                else dir[d];
```

Fig. 10. truncation of vector *dir* by *box* boundary where *dir* is affixed to *base* (for each dim. d)

5 Experimental Results

We implemented our algorithms for similarity range query processing in C/C++ and performed experiments on an HP9000/735 under HP-UX 9.0. For our test database, we used the atomic coordinates from the PDB [Ber+ 77] and computed some thousands of protein surfaces [SK 95]. From these surfaces, we extracted 12,000 segments; 6,200 of them are known docking sites. As approximation model, we chose TRIGO-4, leading to a 7D index over the key vectors of the segments. The weighting factors u_{app} and u_{ext} are set to 1. We managed the index using an R*-tree with a pagesize of 4 kbytes. As query objects, we used docking sites from the azurin and the fructose family.

Figure 11 shows performance results for similarity range queries on the database of 12,000 segments where ε ranges from 0 to 2.5 (ε=0 corresponds to a point query). We compare two evaluation strategies: Direct evaluation of ellipsoid queries by the ellip-

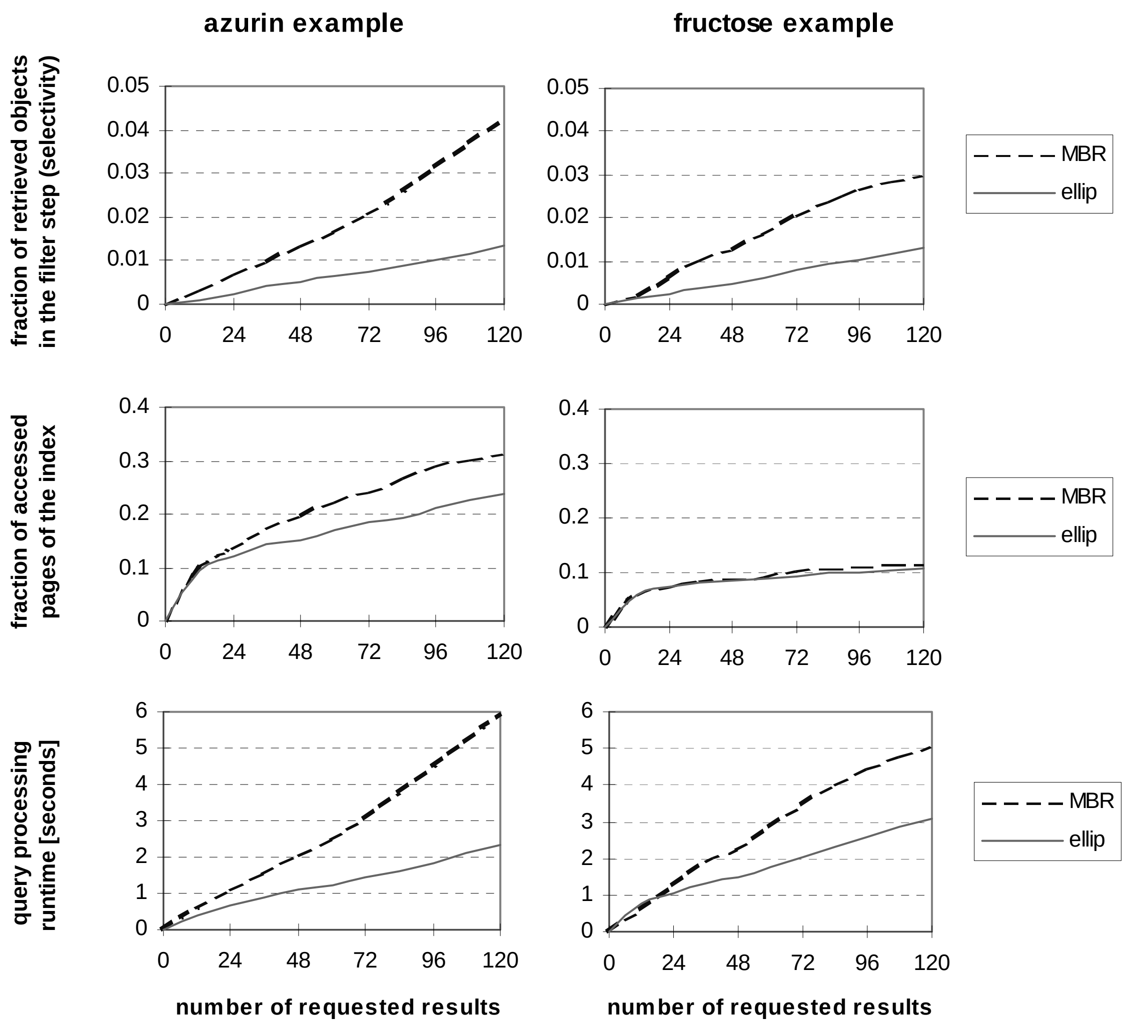

Fig. 11. Performance results for similarity range queries on a database of 12,000 segments using the query examples azurin (left hand side) and fructose (right hand side). The abscissa axis indicates the number of results (up to 120 objects, i.e. 1% of the data). *Top:* Selectivity of the filter step vs. number of results. *Middle:* Fraction of accessed pages vs. number of results. *Bottom:* Overall runtime of shape similarity query processing (in seconds) vs. number of results.

soid-and-box algorithm, and evaluation using a range query where the ellipsoid is approximated by an MBR. In each diagram, the abscissa axis indicates the number of results obtained from the refinement step. In the top diagram, we show the selectivity of the filter step vs. the number of results, i.e. the percentage of the data objects from the database that were selected by the filter step. The selectivity of our ellipsoid method is clearly better than the selectivity of the standard MBR method. In the middle, the percentage of disk pages that were read by the filter step vs. the number of results is plotted. Again, the ellipsoid method is better than the result of the MBR technique. In the bottom diagram, the overall runtime for shape similarity query processing (in seconds) vs. the number of results is depicted. It shows that the ellipsoid method outperforms the MBR approach if considering both, page accesses and CPU time. The main advantage of the direct ellipsoid method is the reduced number of candidates that are obtained from the filter step.

In figure 12, we present the overall runtime as well as the average runtime per result vs. the size of the database (0 to 12,000 surface segments). These experiments ran for various query ranges (from 0.5 to 2.0). Whereas the number of results increases with a growing query range as expected, the average runtime is nearly constant.

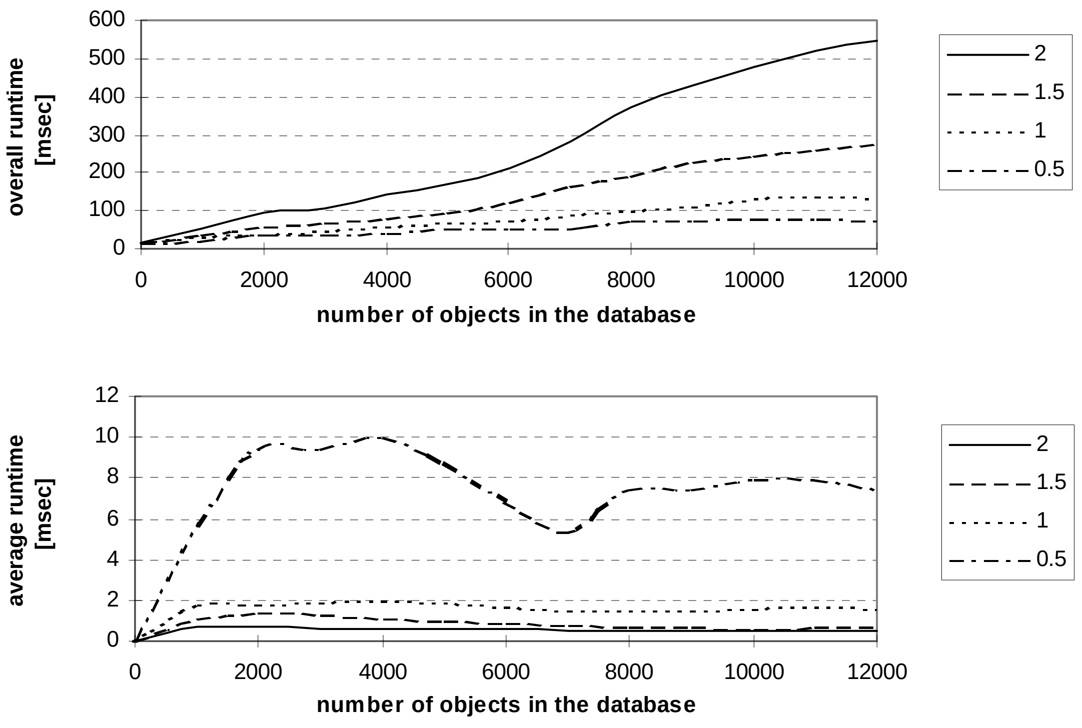

Fig. 12. Overall runtimes (above) and average runtimes per result (below) in seconds vs. database size (0 to 12,000 segments) for various query ranges (from 0.5 to 2.0)

Finally, our experiments have shown that the number of iterations in the *distance* algorithm for ellipsoids and boxes typically ranges from 1 to 8, depending on the data. We never observed an iteration number greater than 10.

6 Conclusion

In this paper, we presented a new approach for similarity search in spatial database systems. First, we introduced the notion of approximation-based similarity for 3D surface segments where similarity is quantified with respect to an application-specific approximation model. Second, to support efficient processing of shape similarity queries, we introduced a new query type, the ellipsoid query. An algorithm for efficient evaluation of ellipsoid queries on PAMs is given.

Experimentally, we could show that our method for ellipsoid query processing outperforms the standard technique using the MBR approximation. Up to now, we mainly investigated databases with known docking segments. We plan to extend our database by arbitrary segments as required for docking prediction systems. This will result in some 100,000 and 1,000,000 data objects which have to be managed by the system.

In our future work, we plan to extend the method of approximation-based similarity from surface segments to overall surfaces of molecules or mechanical parts. Thereby, more complex surfaces will occur which may be managed by two approaches: First, use more complex approximation models. This method leads to an increased dimensionality of the key vectors. In order to efficiently support ellipsoid queries on high-dimensional vectors, techniques to reduce the dimensionality are required. Second, split complex segments into simple pieces. For this purpose, decomposition techniques and new retrieval methods have to be developed. Additionally, we will investigate the promising potential of the concept of ellipsoid queries when applied to other similarity distance functions.

Acknowledgements

We would like to thank the anonymous reviewers who carefully read this paper and gave us helpful hints to improve the presentation of our approach.

References

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

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

[Ber+ 77] Bernstein F. C., Koetzle T. F., Williams G. J., Meyer E. F., Brice M. D., Rodgers J. R., Kennard O., Shimanovichi T., Tasumi M.: *‘The Protein Data Bank: a Computer-based Archival File for Macromolecular Structures’,* Journal of Molecular Biology, Vol. 112, 1977, pp. 535-542.

[Ber+ 97] Berchtold S., Böhm C., Braunmüller B., Keim D., Kriegel H.-P.: *‘Fast Parallel Similarity Search in Multimedia Databases’,* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997.

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

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

[BKK 97] Berchtold S., Keim D., Kriegel H.-P.: *‘Using Extended Feature Objects for Partial Similarity Retrieval’,* accepted for publication in VLDB Journal.

[BK 97] Berchtold S., Kriegel H.-P.: *‘S3: Similarity Search in CAD Database Systems’,* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *‘The R*-tree: An Efficient and Robust Access Method for Points and Rectangles’,* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *‘Efficient Multi-Step Processing of Spatial Joins’,* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 197-208.

[BR 85] Best, M. J., Ritter K.: *‘Linear Programming. Active Set Analysis and Computer Programs’*, Englewood Cliffs, N.J., Prentice Hall, 1985.

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

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

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

[Gut 84] Guttman A.: *‘R-trees: A Dynamic Index Structure for Spatial Searching’,* Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[HS 94] Holm L., Sander C.: *‘The FSSP database of structurally aligned protein fold families’*, Nucl. Acids Res. 22, 1994, pp. 3600-3609.

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

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

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

[MG 93] Mehrotra R., Gary J. E.: *‘Feature-Based Retrieval of Similar Shapes’,* Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 108-115.

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

[PTVF 92] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *‘Numerical Recipes in C’*, 2nd ed., Cambridge University Press, 1992.

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

[SK 95] Seidl T., Kriegel H.-P.: *‘A 3D Molecular Surface Representation Supporting Neighborhood Queries’*, Proc. 4th Int. Symposium on Large Spatial Databases (SSD ‘95), Portland, Maine, USA, Lecture Notes in Computer Science, Vol. 951, Springer, 1995, pp. 240-258.

[SRF 87] Sellis T., Roussopoulos N., Faloutsos C.: *‘The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects’*, Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp 507-518.

[TC 91] Taubin G., Cooper D. B.: *‘Recognition and Positioning of Rigid Objects Using Algebraic Moment Invariants’,* in *Geometric Methods in Computer Vision,* Vol. 1570, SPIE, 1991, pp. 175-186.