To appear in: THE VLDB-JOUNRAL, Dec. 1997.

Using Extended Feature Objects for Partial Similarity Retrieval

Stefan Berchtold, Daniel A. Keim, Hans-Peter Kriegel

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 Munich, Germany
{berchtol, keim, kriegel}@dbs.informatik.uni-muenchen.de
phone: +49-89-2178-2191, fax: +49-89-2178-2192

Abstract

In this paper, we introduce the concept of extended feature objects for similarity retrieval. Conventional approaches for similarity search in databases map each object in the database to a point in some high-dimensional feature space and define similarity as some distance measure in this space. For many similarity search problems, this feature point based approach is not sufficient. When retrieving partially similar polygons, for example, the search cannot be restricted to edge sequences, since similar polygon sections may start and end anywhere on the edges of the polygons. In general, inherently continuous problems such as the partial similarity search cannot be solved using point objects in feature space. In our solution, we therefore introduce extended feature objects consisting of an infinite set of feature points. For an efficient storage and retrieval of the extended feature objects, we determine the minimal bounding boxes of the feature objects in multidimensional space and store these boxes using a spatial access structure. In our concrete polygon problem, sets of polygon sections are mapped to two-dimensional feature objects in high-dimensional space which are then approximated by minimal bounding boxes and stored in an R*-tree. The selectivity of the index is improved by using an adaptive decomposition of very large feature objects and a dynamic joining of small feature objects. For the polygon problem, translation-, rotation-, and scaling-invariance is achieved by using the Fourier transformed curvature of the normalized polygon sections. In contrast to vertex-based algorithms, our algorithm guarantees that no false dismissals may occur and additionally provides fast search times for realistic database sizes. We evaluate our method using real polygon data of a supplier for the car manufacturing industry.

1. Introduction

The problem we focus on is searching a database of 2D-polygons for polygons which are partially similar to a given query polygon. The problem arises in a cooperation with a major supplier of parts for the German car manufacturing industry. The company produces parts called ‘clips’ which are used for holding and joining components in a car. The number of parts manufactured by the company is quite large since many different parts are necessary for each car model. The goal of the cooperation is to reduce the cost of producing new parts by maximizing the reuse of parts. Important for parts to be reusable is that the new part coincides in some detail with previously designed parts which are stored in the parts database of the company. If partially similar parts can be found, the cost for designing and manufacturing a new part can be reduced considerably since the time for designing the part is shortened and it is possible to partially reuse the expensive machinery for manufacturing the part. Finding all partially similar parts for a given query part is therefore the key to cost reduction. Although originating from a rather specific application, the problem of finding all partially similar polygons from a database of 2D-polygons is a general problem which arises in many applications such as CAD, pattern recognition, protein docking, computer tomography and others.

The problem of finding all partially similar parts is a difficult problem. The state-of-the-art approach predominately used in industry is a search based on feature vectors. Feature vectors consist of a fixed number of attributes describing important properties of the polygons. The experience shows, however, that a search based on feature vectors is not sufficient since only a priori defined features can be used in searching for similar parts. What is needed to achieve better results is to allow a similarity search based on the exact geometry of the polygons given by the CAD system.

Unfortunately, the general problem of comparing two polygons under translation-, rotation-, and scaling-invariance is computationally hard. The computational geometry algorithm given in [AB 92] solves a special case of the problem, namely testing two point sets with m points each for congruence under translation- and rotation-invariance, in $O(m^8)$ time. Note that this complexity neither includes scaling-invariance nor testing for similarity. Both, scaling-invariance and similarity-testing, however, are computationally hard problems. Similarity may, for example, be defined as Hausdorff distance, Fréchet distance, percentage of surface superposition, etc. of the two polygons. All those similarity measures are useful for certain applications but none of them is comparable to the human intuition of similarity. The human similarity includes some kind of semantic similarity which considers two parts to be similar if both parts have certain characteristics. In most cases, the characteristics are partial similarities, which means that there exist portions of the two polygons which are similar. Note that even a simple vertex-based partial similarity algorithm which restricts the problem to decide whether there are similar portions of the two polygons starting and ending at any vertex causes the above complexity to increase by a factor of m^2 (all edge sequences of arbitrary length). The ‘continuous’ problem of finding similar portions of the two polygons starting and ending at an arbitrary point (not necessarily a vertex) on any edge of the contour of the polygon is even much harder and, to the authors’ knowledge, there is currently no algorithm solving the problem. Also, in our case we have to solve the 1:n instead of the 1:1 problem, i.e. we have to find all polygons from a database of n parts which are partially similar to a given query polygon.

For solving continuous problems such as the partial similarity problem of polygons, a mapping of the data objects to points in feature space is not sufficient. A polygon section, for example, may

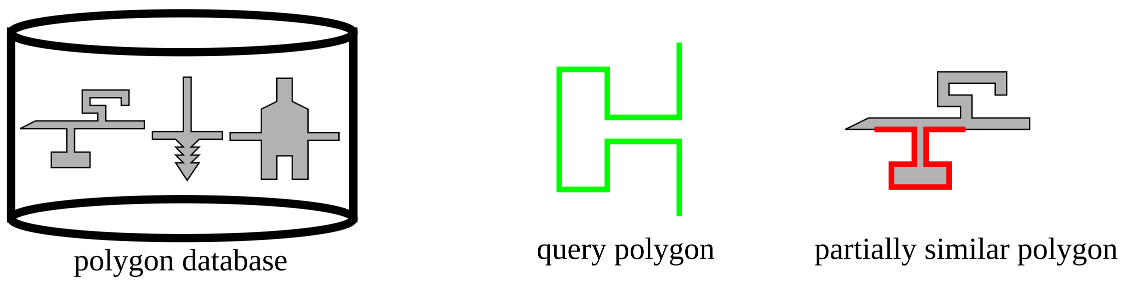


Figure 1: Partial Search Problem

start and end at any point of the polygon contour even between two vertices (cf. Figure 1) and it is therefore not sufficient to map the polygon contour to a single point in feature space. In general, the objects which arise in continuous problems may be described as parameterized functions which depend on one or multiple continuous variables. Our new idea is to map the data objects to extended feature objects in high-dimensional space. Extended feature objects are infinite sets of feature points which are determined by mapping the parameterized functions (corresponding to the data objects) into feature space. In some cases, the data objects have some natural partitioning which can be used to map a data object to a set of extended feature objects instead of a single one. For an efficient storage and retrieval, we determine the minimal bounding boxes of the extended feature objects in multidimensional space and store these boxes using a spatial access structure. The selectivity of the index is improved by using an adaptive decomposition of very large feature objects and a dynamic joining of small feature objects.

In the main portion of the paper, we apply our general idea to the problem of searching a database of 2D-polygons for partially similar polygon sections. We present algorithms which solve the continuous partial similarity problem under translation-, rotation-, and scaling-invariance. In our concrete polygon problem, the natural partitioning of the data objects are the vertices of the polygons. For all edge sequences, we therefore determine the extended feature objects which correspond to the polygon sections starting anywhere on the first edge and ending anywhere on the last edge. Since we have two continuous parameters, the extended feature objects are two-dimensional objects in multidimensional feature space. The two-dimensional feature objects are then approximated by minimal bounding boxes and stored in an R*-tree. Translation-, rotation-, and scaling-invariance are achieved by using a Fourier transformation of the curvature of the polygon. In contrast to vertex-based algorithms, our method guarantees that no false dismissals may occur and provides fast search times for realistic database sizes.

In the literature, there is a lot of work on similarity search of geometry data. In computational geometry, researchers focus on the theoretical aspects of the 1:1 similarity problem. Most of the proposed algorithms are based on similarity measures inadequate for our application [AB 92]. Another area related to similarity retrieval is pattern matching. In pattern matching, the goal is to recognize objects in a scene from a given fixed set of identifiable objects [MG 93, MG 95]. Since the set of identifiable objects is fixed a priori, it is possible to generate models for each of those objects, and try to match the objects in the search scene with those models. For this purpose, a detail with a high significance is sufficient in order to assign a search object to one of these models. In pattern recognition, there are many specialized approaches providing a high invariance against affine and shear transformations (e.g., [WW 80]). Another related area is similarity search in multimedia and pictorial databases. The problem is related since similarity search on images also involves similarity retrieval of the objects in the image. However, there are many differences to our application. The most obvious difference is that in images, the structure and relation of objects is more important than the polygonal shape of the objects and therefore, most of the approaches only use simple similarity measures for comparing the objects [FBFH 94] and do not consider invariances due to a normalized position of known objects [PF 94].

The area of research which is closely related to our approach is similarity search in time series databases. In [AFS 93], an efficient method for similarity search in one-dimensional sequence data has been proposed. The approach maps the data into Fourier space and determines the similarity of two sequences by their Euclidean distance in Fourier space. [FRM 94] extends this idea to include searching for subsequences and [ALSS 95] also considers noise and scaling invariance. All approaches, however, are limited to one-dimensional time series databases. In contrast, our algorithms work for two-dimensional polygon databases and use a distance measure which is completely invariant against affine transformations. Furthermore, our partial similarity search is not limited to edge sequences of the polygons but searches for any section of the polygons starting and ending anywhere on the edges of the polygons even in between vertices.

The rest of this paper is organized as follows: In Section 2, we provide basic definitions and a formal statement of the problem. We further illustrate the problems of other approaches and prove that under scaling-invariance vertex-based algorithms may not avoid false dismissals even if one is only searching for partially identical parts. In section 3, we then describe our algorithm including the analytic calculation of the Fourier transformed curvature of the normalized polygon sections, the mapping of sets of transformed polygon sections to two-dimensional feature objects, and their storage in a spatial access method. We prove the correctness of our algorithm, i.e. we show that false dismissals of partially identical parts may not occur. In section 4, we provide an empirical analysis of our method. For the experiments, we use an R*-tree-based implementation and real ‘clips’ data obtained from our industrial partner. Section 5 summarizes our approach and describes directions for future work.

2. Partial Similarity

In this section, we introduce several notations which are necessary for formalizing the partial similarity problem. Besides defining our notion of polygon data and the partial similarity query, we also discuss essential properties of appropriate distance measures, and we provide examples which show the problems of other approaches. Furthermore, we provide a formal proof that under scaling-invariance vertex-based algorithms may not avoid false dismissals.

2.1 Definition of the Data

A polygon *p* may be defined as a cyclic sequence of edges $\{e_0^p, \ldots, e_{m_p-1}^p\}$ with m_p edges and m_p vertices ($\{v_0^p, \ldots, v_{m_p-1}^p\}$). An edge sequence is defined by $es^p{}_{ij} := \{e_i^p, \ldots, e_j^p\}$ if $i \le j$ and by $es^p{}_{ij} := \{e_i^p, \ldots, e_{m_p-1}^p, e_0^p, \ldots e_j^p\}$ if $i > j$. This definition implies that the edges are concatenated in a cyclic way. The set of all edge sequences for a given polygon *p* may be defined as

$$ES^p = \{es^p{}_{ij} \mid (0 \le i \le m_p - 1) \wedge (0 \le j \le m_p - 1) \wedge (i \ne j)\}.$$

The number of edge sequences is $|ES^p| = m_p \cdot (m_p - 1) \approx m_p^2$ since there are m_p sequences of length m_p, m_p sequences of length *(m_p-1), ...,* m_p sequences of length 2. In the following, we omit the index *p* from all symbols if it is obvious which polygon we refer to i.e., we write e_0 for e_0^p, v_0 for v_0^p, etc. The problem size *N* for the partial similarity task is the set of all edge sequences for all *n* polygons in the database. *N* may be determined as

$$N = \sum_{i=1}^{n} |ES^i| = \sum_{i=1}^{n} m_i \cdot (m_i - 1) \approx \sum_{i=1}^{n} m_i^2 \text{ with } n \cdot m_{max}^2 = \sum_{i=1}^{n} m_{max}^2 \ge N \ge n \cdot \left(\left(\sum_{i=1}^{n} m_i\right) \Big/ n\right)^2 = n \cdot \bar{m}^2$$

where m_{max} is the maximum number of edges of all polygons in the database and $\bar{m}$ is the average number of edges per polygon[1]. The inequality shows that polygons with a high number of edges strongly influence the size of the search space. Note that the edges of the polygons may also be seen as vectors $\vec{e_i} := \overrightarrow{v_{i+1}} - \vec{v_i}$ with their lengths being denoted by $|\vec{e_i}|$. The lengths of the polygon contours are normalized to 2π which means that $\sum_{i=0}^{m-1} |\vec{e_i}| = 2\pi$. The length of an edge sequence may be determined as $|es_{ij}| := \sum_{k=i}^{j} |\vec{e_k}|$. According to Mumford [Mum 87], the set of all possible polygon contours forms an infinite-dimensional space. The reason is that polygon contours may vary in an infinite number of independent ways, which is also an indication for the complexity of the partial similarity search problem of polygon contours.

1. The inequation may be proven by induction over n.

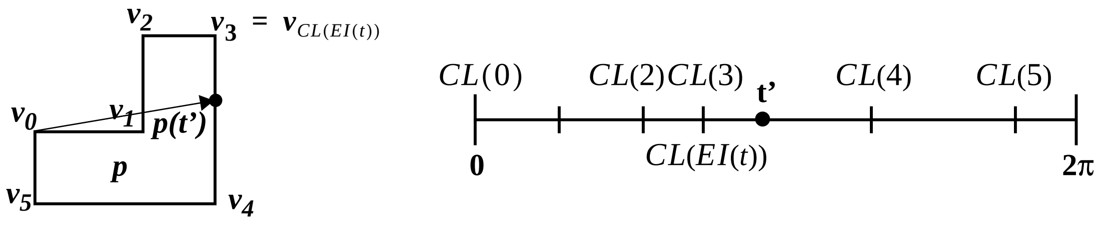


Figure 2: Example for a Polygon as a Parametrized Function

In addition to the vertex/edge-based representation of polygons, we also need a functional definition of polygons. The functional view is necessary for calculating the Fourier transformation of the curvature of the polygon, and it is also more adequate for solving the continuous partial similarity problem. Furthermore, the functional representation of polygons avoids the fundamental problem of the edge representation that there is an infinite number of edge-representations which describe the same polygon contour (e.g., edge sequences with interpolated vertices). In the following definition describing a polygon as a parameterized function, all edge-based polygon representations which describe an identical polygon contour are mapped to the same functional representation — in other words, they fall into a single equivalence class.

Definition 1 *(Polygon contour)*

A polygon contour is defined as a parameterized function $p\colon \Re \rightarrow \Re \times \Re$

$$p(t) = \left(\sum_{k=0}^{EI(t)-1} \vec{e_k}\right) + (t - CL(EI(t))) \cdot \overrightarrow{e_{EI(t)}}\ ,\quad 0 \le t \le 2\pi,$$

where $\quad EI\colon \Re \rightarrow N_0$ is the edge indexing function

$$EI(a) := \min\left\{k \,\middle|\, \sum_{i=0}^{k} |\vec{e_i}| \ge a\right\} \text{ with } 0 \le a \le 2\pi,\ 0 \le k \le m-1 \text{ and}$$

$CL^p\colon N_0 \rightarrow \Re$ is the curve length function $CL(i) := \sum_{j=0}^{i-1} |\vec{e_j}|$ with $0 \le i \le m-1$.

The two functions EI and CL are necessary to define the functional representation of the polygons from the given vertex-based representation. Given a curve length *a* between *0* and 2π, the function EI determines the edge in which the curve ends. The function CL determines the length of the curve up to a given edge *i*, which is the sum of the lengths of all edges from edge *0* to edge *i-1* (c.f. Figure 2). Note that the functional definition is a continuous description of polygon contours which is independent of the number of edges, and therefore avoids the problem of multiple representations describing the same polygon. Note that the edges of the polygon are concatenated in a cyclic

Symbol	Definition
p [$p(t)$]	polygon [parameterized polygon definition]
m_p	number of edges (= number of vertices) of polygon p
e_i^p [e_i]	i-th edge of polygon p
v_i^p [v_i]	i-th vertex of polygon p
α_i	angle between edge i-1 and edge i
$es^p{}_{ij}$	edge sequence (set of all edges from edge i to edge j)
ES^p	set of all edge sequences of p
N	problem size
$CL(i)$	curve length function
$EI(t)$	edge indexing function
PS_{ab}	polygon section starting at curve length a and ending at curve length b
δ	similarity measure
T	affine transformations
λ	polygon transformation function
$c(t)$	curvature function
C	curvature transformation
F	Fourier transformation
a_k, b_k	fourier coefficients
d	degree of the Fourier sum
MBR	minimum bounding rectangle
FO_{ij}	extended feature object corresponding to edge sequence $es^p{}_{ij}$
$FO_{i_1 i_2 j_1 j_2}$	extended feature object corresponding to the set of all edge sequences starting between i_1 and i_2 and ending between j_1 and j_2
V_{max}	maximum volume of the bounding boxes of the extended feature objects

way, which means that the exact definition of a polygon contour is a function with period 2π. Therefore, we assume that all operations on polygon contours are implicitly defined with t modulo 2π. For the rest of the paper, we use the functional definition of polygons — except for the description of shortcomings of edge-based partial similarity retrieval in Section 2.4.

Since we are not only dealing with complete polygon contours but with arbitrary portions of polygon contours, we also need a functional definition of polygon sections. Note that it is not enough to consider edge sequences for solving the continuous partial similarity problem since similar polygon sections may start at any point of a polygon contour — even in between vertices. We therefore need the following functional definition of a polygon section.

Definition 2 *(Polygon Section)*

A polygon section of a polygon *p* with start point *a* and end point *b* ($0 \le a \le b \le 2\pi$) is defined as a parameterized function $PS_{ab} : \Re \to \Re \times \Re$

$$PS_{ab}(t') = p\left(\left(t' \cdot \frac{b-a}{2\pi}\right) + a\right), \quad 0 \le t' \le 2\pi .$$

According to our definition any continuous portion of the polygon is a polygon section. The start and end points are not limited to vertices and, in general, a polygon section is an open-ended portion of the polygon contour. The arc length of polygon sections is defined to be normalized to 2π. Note that there is an infinite number of polygon sections for a given polygon *p*. The complete contour of a polygon is the specific polygon section which results if $a = b$ in which case $PS_{ab}(t) = p(t)$. The definition of polygon sections is important since polygon sections may occur as query polygons and result polygons.

2.2 Definition of Partial Similarity

Before we formally define similarity and partial similarity, we introduce the important notions of congruence and partial congruence. Congruence means that two polygon sections are identical up to affine transformations, and partial congruence means that there are polygon sections which are identical up to affine transformations.

Definition 3 *(Congruence, Partial Congruence)*

Two polygon sections p_1 and p_2 are called *congruent* iff there exists an affine transformation *T* such that $T(p_2) = p_1$.

Two polygon sections p_1 and p_2 are called *partially congruent* iff there exists a section $PS_{ab}^{p_1}$ of p_1 and an affine transformation *T* such that $T(p_2) = PS_{ab}^{p_1}$.

In contrast to (partial) congruence which focuses on finding (partially) identical polygon sections, (partial) similarity also includes all polygon sections which are (partially) similar. For the definition of partial similarity, we need a similarity measure $\delta\colon (P \times P \to \Re)$ [1] which determines the similarity of two polygon sections.

Definition 4 *(ε-Similarity, Partial ε-Similarity)*

Two polygon sections p_1 and p_2 are called $\varepsilon - similar$ with respect to δ iff there exists an affine transformation T such that $\delta(T(p_2), p_1) < \varepsilon$.

Two polygon sections p_1 and p_2 are called $partially\ \ \varepsilon - similar$ with respect to δ iff there exists a section $PS_{ab}^{P_1}$ of p_1 and an affine transformation T such that $\delta(T(p_2), PS_{ab}^{P_1}) < \varepsilon$.

The similarity measure is a function $\delta\colon (P \times P \to \Re)$ [1] which has two polygon sections as input parameters and determines a real value denoting the similarity of the two polygon sections. In the following, we define properties which must be satisfied by any useful similarity measure.

Definition 5 *(Properties of the Similarity Measure)*

(1)	identity-preservation:	$\forall p :\ \delta(p, p) = 0$
(2)	commutativity:	$\forall p_1 \forall p_2 :\ \delta(p_1, p_2) = \delta(p_2, p_1)$
(3)	triangle inequation:	$\forall p_1 \forall p_2 \forall p_3 :\ (\delta(p_1, p_2) + \delta(p_2, p_3) \geq \delta(p_1, p_3))$
(4)	unboundedness:	A similarity measure δ is called *unbounded* iff $\forall p,\ \forall \varepsilon \geq 0,\ \exists q\colon \delta(p,q) > \varepsilon$.

Useful similarity measures must fulfill these four properties. The first three properties are trivial, and the fourth property (unboundedness) means that there is no most dissimilar polygon section for any given query polygon.

Up to now, we have only defined the partial similarity of two polygon sections. The next step is to define the partial similarity query, which is searching a database of polygons for partially similar polygon sections. Given a database *DB* of n polygons p_i with m_i edges each (*i = 0 .. n-1)*, the partial similarity query is defined as follows.

Definition 6 *(Partial Similarity Query)*

Given a query polygon *s*, find all polygons p_i from *DB* which are partially $\varepsilon - similar$ to *s*, i.e. determine

$$\{p_i \in DB \mid \exists PS_{ab}^{p_i} \wedge \exists T\colon \delta(T(s), PS_{ab}^{p_i}) < \varepsilon\}.$$

Note that finding all partially congruent polygons is a subtask of solving the partial similarity query. This means that any algorithm which solves the partial similarity query must at least output all par-

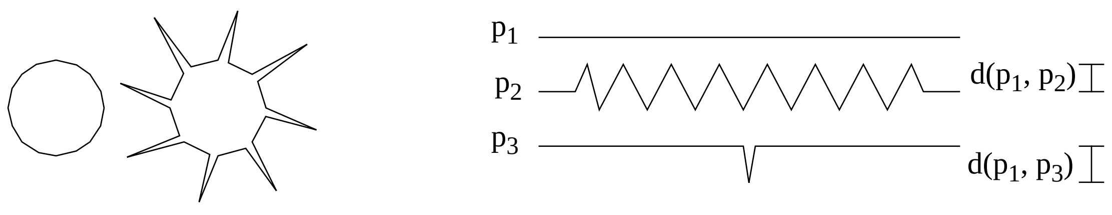
1. Note that δ may also be called a “dissimilarity measure”. We use “similarity measure” since it is more intuitive.

tially congruent polygons. A further observation is that the complexity of the query polygon strongly influences the result. The query polygon should neither be too simple nor too complex, since simple search polygons lead to low selectivities (ultimately to a selectivity of 100% for a single line) and complex query polygons prevent small substructures from having any impact on the result. Consequently, meaningful query polygons will usually be of medium complexity.

2.3 Problems

In solving the partial similarity query, we have to deal with several problems. One important problem of similarity search is that similarity should be largely invariant with respect to translation, rotation and scaling of the polygons. This is true for a wide range of applications including our application in the car manufacturing industry, which requires complete translation-, rotation-, and scaling-invariance. There are several approaches to solving the invariance-problem. The most common solution is a normalization of the polygons to a specific position, orientation, and size. Transforming the polygons into a well-defined position and orientation, however, is very difficult or even impossible for many applications. If there is no unique position and orientation, slight mistakes in the transformation may lead to false dismissal. In general, the uniqueness may only be guaranteed if there is an application- and data-inherent position and orientation, which is only true for a limited number of applications. An obvious possibility to avoid this problem is to use features of the polygons which are translation- and rotation-independent, such as the edge lengths and angles of the polygon or the curvature of the polygon. To achieve scaling-invariance, the traditional approach is to normalize the size of the polygons to some specific size. In our approach, we normalize the arc length of the polygon sections. Using the curvature guarantees translation- and rotation-invariance, and normalizing the arc length guarantees scaling-invariance.

Another critical problem in solving the partial similarity problem is the choice of the similarity measure. In the literature, several similarity measures for polygons have been proposed. Most of them, however, do not seem adequate for our problem. For example, defining the distance of two



a. Surface Superposition b. Hausdorff Metric

Figure 3: Problems with different Distance Measures

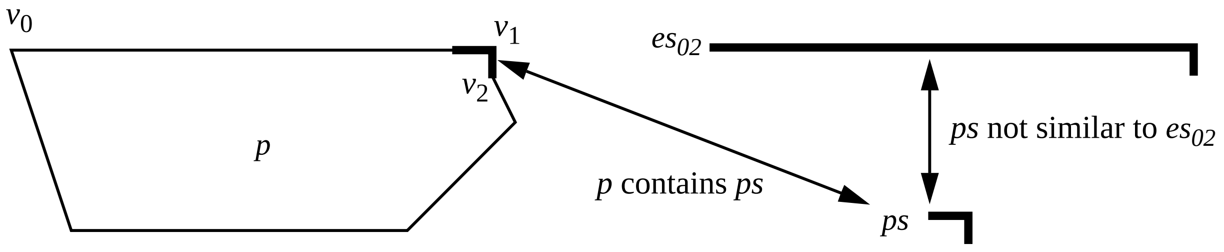


Figure 4: The Problem of Edge-based Algorithms

polygons as the relative percentage of the maximum surface superposition of the two polygons may yield a high similarity for quite different polygons (cf. Figure 3a). Furthermore, the maximum surface superposition is only defined for complete polygons and not for polygon sections. Another similarity measure is the well-known Hausdorff distance which determines similarity as the maximum of the minimal distances between the points of the two polygons[1]. The Hausdorff distance, however, completely ignores the shape of the edge sequence (cf. Figure 3b) and is therefore inadequate for defining similarity. A distance metric which considers the shape of the edge sequence is the Fréchet distance. The Fréchet distance defines the distance of two polygons as the minimum of the maximal distances between the points when walking along the two contours[2]. Both, the Hausdorff and Fréchet distance are defined as some kind of maximum norm. The maximum norm provides good results for low distances in which case the distances correspond to the human intuition of similarity rather well, but for higher distance values the distances may be the result of local effects and do not correspond to the human intuition of similarity at all. As already indicated, our distance measure is using the Fourier transformation of the polygons and determines the distance of two polygons as Euclidean distance in Fourier space. As experimental results show, the distance in Fourier space is a good approximation of the human intuition of similarity [KM 95].

Another problem in solving the partial similarity query is that similarity may be defined differently depending on the user's focus. One user may be interested in the overall contour of the polygons, another user may be interested in the details of the polygons, e.g., whether the contour is a zigzag line or a smooth line, and even another user may be interested in some level of detail in

1. The Hausdorff distance is defined as

$$\delta_H(p1, p2) = max\left(\underset{a \in p1}{sup} \ \underset{b \in p2}{inf} \ d(a, b), \ \underset{b \in p2}{sup} \ \underset{a \in p1}{inf} \ d(a, b) \right)$$

with p1 and p2 being polygons in vertex-representation and d(p1, p2) being the Euclidean distance in the plane.

2. The Fréchet distance is defined as

$$\delta_F(p1, p2) = \left(\underset{\alpha, \beta}{inf} \ \underset{t \in [0, 1]}{sup} \ d(p_1(\alpha(t)), p_2(\beta(t))) \right)$$

with p1(t) and p2(t) being polygons in functional representation and d(p1, p2) being the Euclidean distance in the plane.

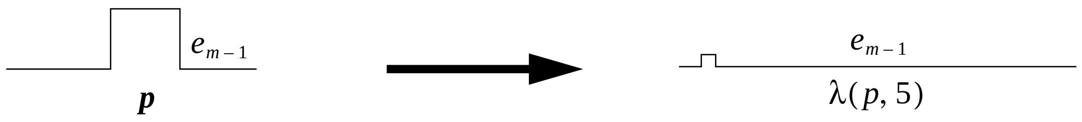


Figure 5: Effect of the λ-function

between. With all currently available approaches, it is impossible to handle similarity being defined for different levels of details. In contrast, our approach is able to deal with different levels of detail and provides useful results over a wide range of levels of detail.

A last but serious problem is that all approaches which are based on the vertices of the polygons may not avoid false dismissals. As already mentioned, any algorithm for partial similarity must at least find all partially congruent polygons. In the following, we show that any algorithm for the partial similarity problem which does not consider the infinite number of all possible polygon sections but only sections starting and ending at the vertices may produce false dismissals which means that it will miss partially congruent polygon sections.

The basic idea of the proof is to show that for any given polygon section *ps*, we are able to construct a polygon *p* such that *p* contains *ps*, but none of the edge sequences in *p* is similar to *ps*. Figure 4 illustrates the basic idea of the proof. For our theoretical considerations, we require the definition of a λ-function. Intuitively, the λ-function transforms a polygon *p* into a polygon *q* such that the polygons *p* and *q* are identical on all vertices but the last vertex. The last edge of *p* is extended in *q* by a factor *k* (c.f. Figure 5). More formally, the function λ: $P \times \Re \to P$, $\lambda(p, k) = q$ transforms a polygon section *p* into the polygon section *q* using an extension factor *k* ($k \in [0, \infty[$) such that

$$(m_q = m_p) \wedge (\forall i, 0 \le i \le m_p - 2\colon\ \overrightarrow{v_i^q} = \overrightarrow{v_i^p}) \wedge (\overrightarrow{v_{m_q-1}^q} = \overrightarrow{v_{m_p-1}^p} + k \cdot \overrightarrow{e_{m_p-1}^p}) .$$

Lemma 1

Any algorithm which tries to solve the partial similarity problem (cf. definition 4) for a given ε and for a given similarity measure fulfilling the properties according to definition 5 by only considering edge sequences ES^p (polygon sections starting and ending at vertices but not between vertices) produces false dismissals.

Proof *(by contradiction):*

Let us assume lemma 1 is false. Then there is an algorithm A which produces no false dismissals for a given fixed ε. If we are able to show that there is no upper bound for ε, we may conclude that A will produce false dismissals and lemma 1 is proved.

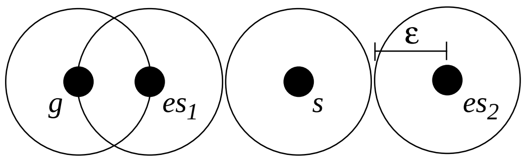


Figure 6: ε-neighborhoods of g, es$_1$, s, and es$_2$

To show that there is no upper bound for ε, we show that for every given ε we are able to construct a polygon p_ε and choose a polygon section s of p_ε such that $\delta(r,s) > \varepsilon \quad \forall r \in ES$. We choose s as $s = \lambda(p_\varepsilon, k),\ 0 \le k \le 1$, which means that s starts exactly on vertex v_0 of p_ε and does not end on a vertex but anywhere on the edge e_{m-1} of p_ε. Since δ is form convergent, lengthening the edge e_{m-1} will cause $\delta(p_\varepsilon,g)$ to converge against 0 for $k \to \infty$. Remember that g is the trivial polygon section consisting of one edge. Without loss of generality we assume that only the edge sequences $es_1 = \{e_0,...,e_m\} = \lambda(p_\varepsilon, 1)$ and $es_2 = \{e_0,...,e_{m-1}\} = \lambda(p_\varepsilon, 0)$ are relevant for s. Now we are able to construct p_ε in a way such that $\delta(p_\varepsilon,g) < \varepsilon$ and $\delta(\lambda(p_\varepsilon, 0),g) > 5\varepsilon$. The existence of p_ε is guaranteed since δ is unbounded. In Figure 6, we show the ε-neighborhoods of g, es_1, s, and es_2. Since $\delta(\lambda(p_\varepsilon, 0),g) > 5\varepsilon$, it follows that there must be a k such that there is no intersection between the ε-neighborhoods of s and es_1 and between the ε-neighborhoods of s and es_2. This, however, implies that $\delta(es_1,s) > \varepsilon$ and $\delta(es_2,s) > \varepsilon$ which means that neither edge sequence es_1 nor edge sequence es_2 will be determined as a result for our query with polygon section s, although s is partially congruent to p_ε since $s = \lambda(p_\varepsilon, k)$. Because our construction of p_ε may be done for any ε, we are able to conclude that there is no upper bound for ε. q.e.d.

Lemma 1 implies that for solving the partial similarity problem it is not enough to consider just the edge sequences of the polygons since there exists a query polygon s which is a polygon section of a polygon p such that $\delta(es,s) > \varepsilon$ for all edge sequences es of the polygon p and any constant ε. Our algorithm which is described in the following section avoids this problem since it is not based on edge sequences but on the continuous representation of the polygons.

3. Our Approach

3.1 Overview of our Algorithm

Our algorithm solves the continuous partial similarity problem and guarantees that at least all partially congruent polygons will be found. The algorithm is designed to support any similarity measure δ which defines similarity as Euclidean distance in some d-dimensional feature space and fulfills the properties according to definition 6. Since the Fourier transformation has proven to be a good measure of similarity, we determine the distance of two polygons as Euclidean distance in

Fourier space. To gain invariance against translation, rotation, and scaling, we use the Fourier transformation of the differential geometric curvature of the polygon instead of the polygon itself and normalize the arc length of the polygon sections to 2π. Using the curvature guarantees translation- and rotation-invariance, and normalizing the arc length guarantees scaling-invariance.

For solving the continuous partial similarity problem, our algorithm maps adjacent sets of polygon sections to two-dimensional objects in multidimensional Fourier space. Let vertex v_s be the starting vertex and v_e be the ending vertex of an edge sequence of polygon p. If we consider all polygon sections starting anywhere between vertex v_s and v_{s+1} terminating at v_e and transform all these polygon sections into Fourier space, we get a one-dimensional object in the *d*-dimensional Fourier space (cf. Figure 7 top). Analogously, if we consider all polygon sections starting between v_s and v_{s+1} and ending between v_{e-1} and v_e, we get a two-dimensional *feature object* in the *d*-dimensional Fourier space (cf. Figure 6 bottom).

For an efficient storage and retrieval of the two-dimensional feature objects in multidimensional space, we determine the minimal bounding boxes of the objects in *d*-dimensional space and store these boxes using a spatial access method (SAM). Basically, any spatial access method such as the R-tree [Gut 84] and its variants (R^+-tree [SRF 87], R*-tree [BKSS 90], P-tree [Jag 90b]), Buddy-tree [SK 90], linear quadtrees [Gar 82], z-ordering [Ore 90] or other space-filling curves [Jag 90a], and grid-file based methods [NHS 84, Fre 87] may be used for this purpose. By using spatial access methods, the problem of finding similar polygon sections has been reduced to the problem of searching two-dimensional extended feature objects in *d*-dimensional space.

Two problems arise with this approach: On the one hand, there is a number of rather big bounding boxes which reduce the quality of the index, and on the other hand there is a large number of quite small boxes which causes the size of the index to increase unnecessarily. The big boxes are a result of the approximation of the feature objects by bounding boxes in high-dimensional space where small variances in some dimension result in large extensions of the bounding boxes, and the small boxes are a consequence of the small changes of the Fourier transformation for certain polygon sections. There are many possible solutions to these problems including using better approximations instead of the bounding boxes (e.g., rotated bounding boxes [BKS 93], polyhedra [Gue 89, Jag 90b], etc.). The most practical and effective approach for our application, however, is to use an adaptive decomposition of the two-dimensional objects recursively partitioning the objects until the volume of each of the resulting boxes is less than a constant value V_{max}. To solve the

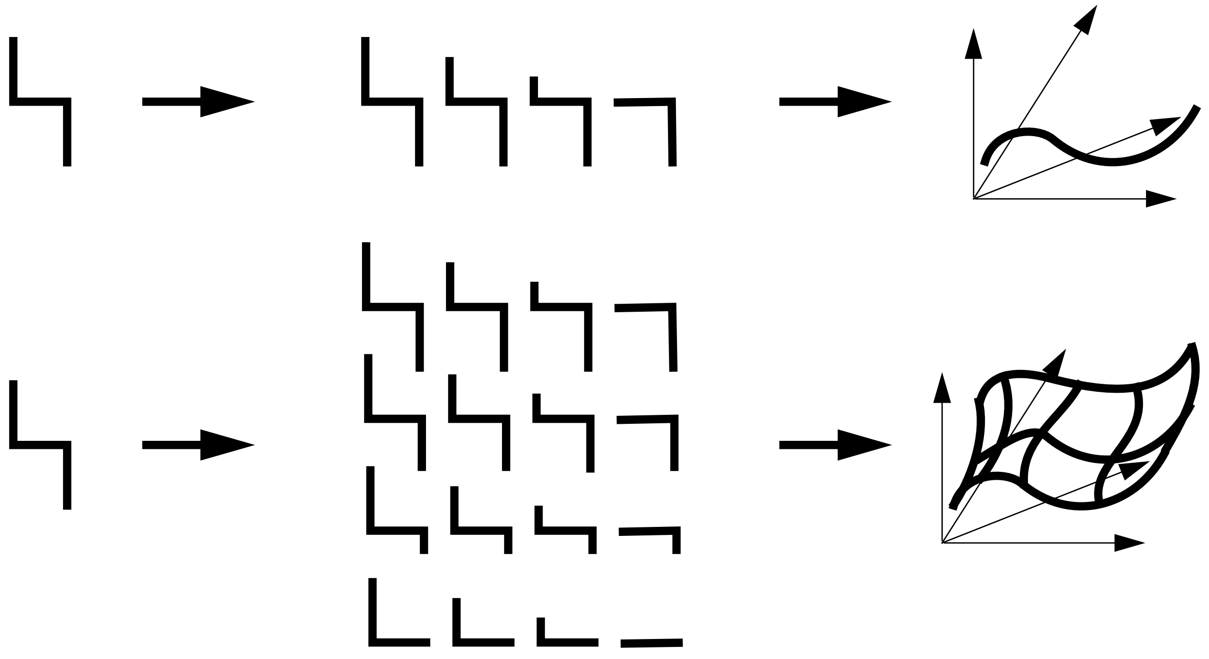


Figure 7: Extended Feature Objects

problem of having too many small boxes, we join as many of these boxes as possible as long as the volume of the bounding box of the joined boxes remains less than V_{max}. Figure 8 provides an overview of the index creation and querying of our partial similarity search algorithm.

After having created the index, querying for similar polygon sections is rather simple. We only have to transform the curvature of the query polygon into a query point in the Fourier space (cf. Figure 8). Using the spatial access method, we are then able to determine all candidate bounding boxes c which are within an ε-neighborhood of the query point. In the refinement step, we finally have to compute the two-dimensional feature object for every candidate *c*, determine the minimal distance of the query point to the feature object, and select all candidates whose minimal distance is less than ε. After sorting the remaining candidates according to their distance we output the result.

3.2 Partial Similarity Search Algorithm

As indicated in the previous subsection, the creation of the partial similarity search index is a crucial part of the algorithm. Note that certain steps of the index creation are also used in the query algorithms, namely the determination of the curvature and Fourier transformation (cf. Figure 8). In this section, we describe the important steps of our partial similarity search algorithm in more detail. In creating the partial similarity index, the first step is to determine all possible edge sequences ES^p for all polygons *p* of the database *DB*. This can be done by successively determining the m_p

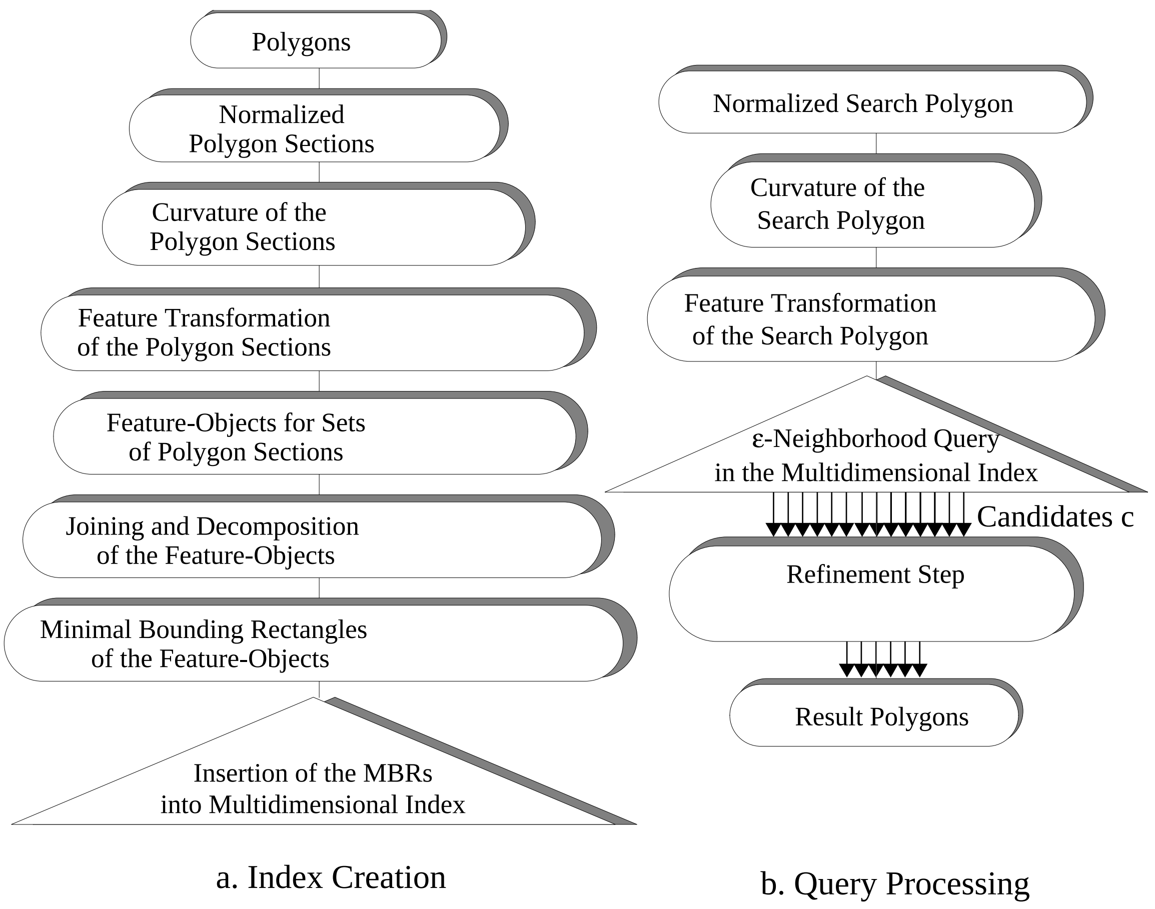

Polygons
Normalized Polygon Sections
Curvature of the Polygon Sections
Feature Transformation of the Polygon Sections
Feature-Objects for Sets of Polygon Sections
Joining and Decomposition of the Feature-Objects
Minimal Bounding Rectangles of the Feature-Objects
Insertion of the MBRs into Multidimensional Index
Normalized Search Polygon
Curvature of the Search Polygon
Feature Transformation of the Search Polygon
ε-Neighborhood Query in the Multidimensional Index
Candidates c
Refinement Step
Result Polygons
a. Index Creation
b. Query Processing

Figure 8: Overview of our Partial Similarity Search Method

sequences of length m_p, the m_p sequences of length $(m_p\text{-}1)$, ..., and the m_p sequences of length 3. Length three is enough since smaller edge sequences may not define significant shapes under partial similarity and translation-, rotation- and scaling-invariance. The next step is to determine the two-dimensional feature objects for each of the edge sequences in ES^p.

3.2.1 Determining the Feature Objects in Fourier Space

The two-dimensional feature objects in *d*-dimensional Fourier space are a result of varying the length of the first and the last edge of the edge sequences and determining the Fourier transformation of the curvature of the resulting polygon sections. More formally, the feature object corresponding to the set of polygon sections created by varying the first and last edges of edge sequence es_{ij} can be described as

$$FO^{es_{ij}}(x, y) = F(C(PS^{es_{ij}}_{(x \cdot |e_i|)\ (2\pi - y \cdot |e_j|)}(t))) \text{ with } x, y \in \Re,\ 0 \le x \le 1,\ 0 \le y \le 1,\ 0 \le t \le 2\pi$$

where $C\colon (\Re \to \Re^2) \to (\Re \to \Re)$ is the transformation determining the curvature function *c(t)* for a given polygon contour function $p(t)$

and $F: (\Re \to \Re) \to \Re^d$ is the Fourier transformation determining the Fourier coefficients for a given curvature function in *d*-dimensional Fourier space.

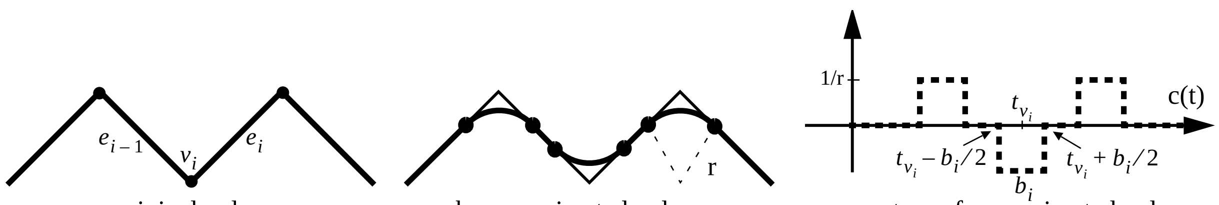
The similarity measure of our approach $\delta_F(T(p_1), p_2)$ is defined as $d_{euclid}(F(C(ps_1)), F(C(ps_2)))$. Note that the normalization of the polygon sections is implicitly contained in the definition of polygon sections PS_{ab} (cf. definition 2). In the rest of this paper, we simply write FO_{ij} for $FO^{es_{ij}}(x, y)$, $0 \le x \le 1, 0 \le y \le 1$. In the following, we describe the curvature transformation *C* and the Fourier transformation *F.*

Determining the Curvature of Polygon Sections

As described in section 2.1, a polygon section may be seen as a parametrized function. The curvature of such a function is mathematically undefined because the second derivative is not continuous. A variety of approaches have been proposed in the literature to overcome this mathematical problem ([HB 86], [KSP 95], [WW 80], etc.). These approaches may be classified into two major groups: Approaches which determine an approximated curvature function in one step, and approaches which approximate the polygon such that the curvature function of the polygon's shape is well-defined and determine the curvature of the approximated polygon in a second step.

An example for an algorithm determining an approximated curvature function in one step is the turning angle algorithm described in [HB 86]. Another possibility is to use a function similar to a curvature plot such as the centroidal profile [KSP 95], the super segments [SM 90], or geometric hashing [RH 92]. Another approach [Ber 97] is to approximate the polygon by a Cubic Spline function and than analytically determine the curvature of the spline. All these algorithms have certain advantages and disadvantages: Plots, for example, such as the turning angle are a good choice if the data is noisy as e.g. in computer vision applications. Furthermore, most techniques require the choice of an appropriate sampling rate which is difficult to choose without any a priori knowledge of the data.

In general, our technique of using extended feature objects is independent of the technique used to determine the curvature function. In principle, any of the curvature functions mentioned above may be used. Which approach is actually used depends on the application domain and its special requirements. In case of CAD data, we decided to use a rather simple curvature function: We substitute all vertices of the polygon with tiny circle segments. From that we get a new geometric object of which at least the first derivative is continuous. The curvature of this structure is defined



a. original polygon b. approximated polygon c. curvature of approximated polygon

Figure 9: Curvature of an approximated polygon

in sections; by concatenating these sections we obtain a non-continuous square wave function. Figure 9 shows our approximation of a polygon section and the according curvature function.

For describing our curvature transformation in more detail, let us focus on two subsequent edges e_{i-1} and e_i. These edges coincide in vertex v_i with an angle α_i. For the edge sequence $es_{(i-1)\,i}$ containing v_i, we may easily compute the curvature function $c_i(t)$ describing the differential geometric curvature of the approximated edge sequence because the curvature of a circle segment with radius r is a constant function $\frac{1}{r}$ and the curvature of a straight line is a constant zero function. We may calculate the arc length of the circle segment substituting vertex v_i by $b_i = |\alpha_i| \cdot r$. For $c_i(t)$, we therefore obtain

$$c_i(t) := \begin{cases} \frac{1}{r} & if \quad (t_{v_i} - b_i/2 > t > t_{v_i} + b_i/2) \\ 0 & else \end{cases} .$$

The curvature of an arbitrary polygon section es_{ij} is $c(t) = \sum_{k=i+1}^{j} c_k(t)$. Figure 9c shows the graph of the curvature function $c(t)$ for the approximation of the polygon section presented in Figure 9a. Figure 10 shows the curvature functions of a square and a circle. In addition, we show two polygons (p_1 and p_2) which are similar under translation-, rotation- and scaling-invariance. Note that the curvature functions of the two polygons have only minor differences in the width of the square waves.

The approximation of the original polygon and in particular the choice of r influences the curvature function. If we reduce the radius r of the circle segment, $\frac{1}{r}$ will be increased while b_i will be decreased. This causes $c(t)$ to become more narrow and the amplitude of square waves to become higher, while the approximation of the polygon converges against the polygon itself. On the other hand, $c(t)$ becomes difficult to handle numerically. An adequate value for r which has proven useful for our applications is $\frac{\pi}{50}$ for polygon sections with a normalized length of 2π. As our experiments show, the remaining steps of our approach are quite robust against a suboptimal

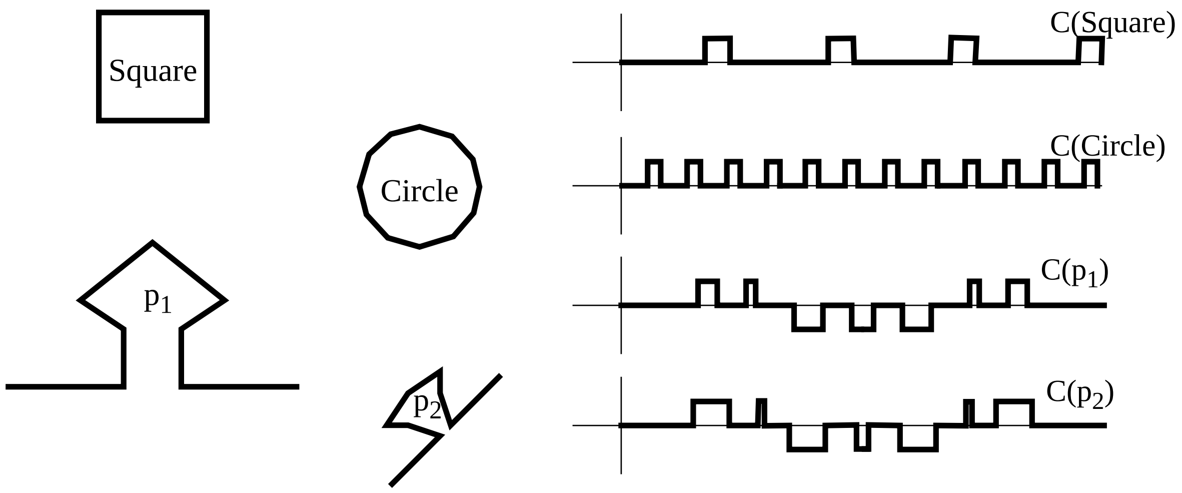


Figure 10: Examples for the Curvature of two Polygons

choice of *r* as long as *r* is smaller than half of the length of the shortest edge in the database since otherwise individual square wave functions may overlap.

Fourier Transformation

The next step is the Fourier transformation F of the curvature. The principle of the Fourier transformation is to approximate a function by summing up *sin* and *cos* functions with adequate parameters. The quality of the approximation is improved by increasing the degree d of the Fourier approximation which means to successively sum up *cos(x), sin(x), cos(2x), sin(2x), ...,* $cos(\frac{d}{2}x)$, $sin(\frac{d}{2}x)$. Experimental results show that the Fourier transformation provides a good approximation of the polygons and their curvature function even for rather small *d*. In Figure 11, we present a polygon together with the Fourier approximations using 2, 4, 6, 32, 64 and 512 coefficients. Note that the polygon in Figure 11 has 8 internal vertices and, therefore, we get 8 spikes in the curvature-function. Our experiments show that six Fourier coefficients are sufficient for our application.

In the appendix on the Fourier theory, we present the formulas for analytically determining the Fourier coefficients of a square wave function (cf. equation 1). For determining the Fourier coefficients of our curvature function *c(t)*, we just have to sum up the formula given in equation 1 for all $c_i(t)$. We obtain the following formulas for the Fourier coefficients of a polygon:

$$a_k = \frac{1}{\pi k r} \sum_{i=0}^{m-2} \frac{\alpha_i}{|\alpha_i|} \left(\sin\left(k\left(t_{i+1} + \frac{|\alpha_i r|}{2}\right)\right) - \sin\left(k\left(t_{i+1} - \frac{|\alpha_i r|}{2}\right)\right)\right)$$

$$b_k = -\frac{1}{\pi k r} \sum_{i=0}^{m-2} \frac{\alpha_i}{|\alpha_i|} \left(\cos\left(k\left(t_{i+1} + \frac{|\alpha_i r|}{2}\right)\right) - \cos\left(k\left(t_{i+1} - \frac{|\alpha_i r|}{2}\right)\right)\right)$$

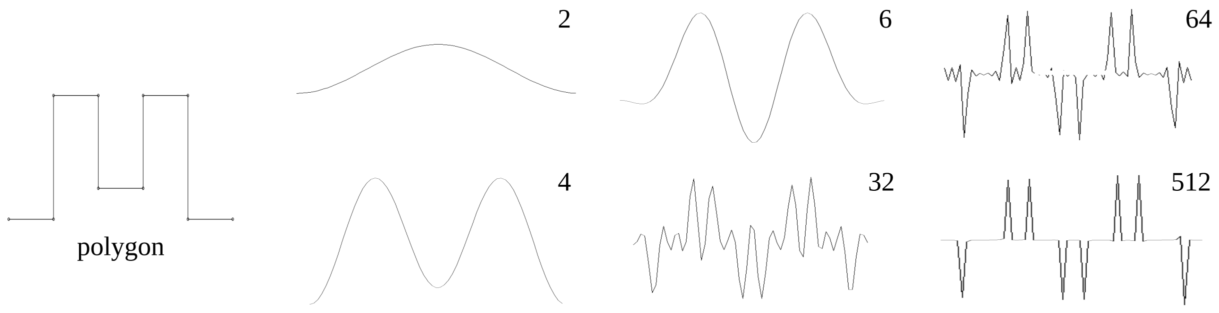


Figure 11: Approximation of Polygons by the Fourier Transformation

The calculation of a_k and b_k can be done in *O(m)* time, the calculation of all coefficients can be done in *O(m* d),* where *d* is the degree of the Fourier sum. Note that we are able to compute the coefficients of the Fourier sum analytically, and therefore we do not run into numerical problems such as choosing the right sample rate. This fact is one of the reasons for being able to use a rather small degree of the Fourier sum without losing too much information.

The Fourier transformation as defined above fulfills the necessary properties of a similarity measure (cf. definition 6). The properties (1) - (3) are trivially fulfilled since each polygon is mapped to a specific point in Fourier space and the distance of polygons is determined by the Euclidean distance of those points in Fourier space. The fourth property (distance measure unbounded) is guaranteed since it is always possible to construct a more dissimilar polygon, at least for infinite *d*. The degree of the Fourier sum *(d)* has to be chosen high enough such that this property is fulfilled for all polygons in the database.

3.2.2 Enhancing Index Selectivity

In the last sections, we described the determination of the two-dimensional feature objects. In the final step of the index creation, we insert the bounding boxes of the feature objects into a multidimensional index structure. However, as indicated in the overview, there are two practical problems that arise with our approach of managing the bounding boxes of the feature objects using a spatial access method: On the one hand, there is a number of rather large bounding boxes which reduce the quality of the index, and on the other hand there is a large number of quite small boxes which causes the size of the index to increase unnecessarily. We solve the problem of having large bounding boxes by using an adaptive decomposition of the two-dimensional feature objects which recursively partitions the feature objects until the volume of each of the resulting boxes is less than a constant value V_{max}; and we solve the problem of having too many small boxes by joining as many of these boxes as possible as long as the volume of the bounding box of the joined boxes remains less than V_{max}. For determining the boxes which are to be decomposed and those which should be joined, we need to calculate the bounding box of the feature object in d-dimensional space. Unfortunately, it is quite difficult to calculate the extremes of a feature object in d-dimensional space analytically. For calculating the extension of a feature object in every dimension of the Fourier space, we have to differentiate the function $FO^{es_{ij}}(x, y)$ and calculate the null values of the derivative. To the authors' knowledge, there exists no analytical solution to this problem, and therefore, we have to use numerical solutions. Unfortunately, the standard method — a modified Newton algorithm —is rather slow in high-dimensional space and requires an appropriate start value to guarantee termination. As our experiments show, however, a simple and very fast algorithm leads to approximately the same results. We simply approximate the feature object by a set of plane segments defined by some points chosen from the feature object. The maximum search on this linearly defined structure is very fast, while the inaccuracy of the approach is negligible.

Decomposition of Feature Objects

The decomposition of the feature objects is necessary since otherwise large bounding boxes will cause an insufficient selectivity of our indexing method. The reason for the large size of the bounding boxes is that bounding boxes are in general bad approximations for low-dimensional objects located in a high-dimensional space, because they usually include a large portion of dead space. Imagine a diagonal line of length 2 in a d-dimensional hypercube. The volume of the bounding box of the line is $(2/\sqrt{d})^d$. If we decompose the original line into two equally sized segments, we obtain two boxes of size $(1/\sqrt{d})^d$, which is a factor of 2^d. The reduction of volume achieved by decomposing the line segment is $2^d/2 = 2^{d-1}$. For d = 6, the volume of the bounding box is reduced from $(2/\sqrt{6})^6 \approx 0.30$ to $(1/\sqrt{6})^6 \approx 0.005$ yielding a reduction factor of $2^5 = 32$. This

```
decompose_feat_obj(FO fo, int ll, int lh, int hl, int hh, set_of_FO result)
{
    int lm, hm;
    if (volume(box(fo)) > Vmax)
    {
        lm = (ll + lh) / 2; hm = (hl + hh) / 2;
        decompose_feat_obj(fo, ll, lm, hl, hm, result);
        decompose_feat_obj(fo, ll, lm, hm, hh, result);
        decompose_feat_obj(fo, lm, lh, hl, hm, result);
        decompose_feat_obj(fo, lm, lh, hm, hh, result);
    }
    else
        result.insert(fo, ll, lh, hl, hh);
}
```

Figure 12: Algorithm for Decomposing Feature Objects

high reduction factor is only true for a diagonal line. If the line is parallel to one of the coordinate axes, there is no reduction. Our experiments show, however, that on the average there is a high reduction potential when dealing with real data.

Decomposing a feature object $FO^{es_{ij}}(x, y)$ is equivalent to splitting the parameter spaces of *x* and *y*. If we divide the parameter spaces into four equally sized subspaces, we obtain four new feature objects. Note that the union of these four feature objects is equal to the original feature object. Our algorithm now recursively decomposes the feature object until the volume of the bounding box of a feature object is lower than a constant value V_{max}. More formally, a feature object $FO^{es_{ij}}([ll, lh], [hl, hh])$ is recursively decomposed into the following four feature objects $FO^{es_{ij}}([ll, lm], [hl, hm])$, $FO^{es_{ij}}([lm, lh], [hl, hm])$, $FO^{es_{ij}}([ll, lm], [hm, hh])$ and $FO^{es_{ij}}([lm, lh], [hm, hh])$, where $lm = (ll + lh)/2$ and $hm = (hl + hh)/2$. In Figure 12, we present the program code of the decomposition algorithm.

Joining Feature Objects

A problem of the index as described so far is that there is a large number of quite small boxes which causes the size of the index to increase considerably. The reason for the large number of rather small boxes is that extending a polygon section by one more edge will provide similar Fourier coefficients if the overall shape of the polygon section fits the Fourier approximation. This effect especially occurs for segments of circular shapes where the Fourier transformation remains approximately constant if the circle segment is extended in both directions. As a consequence, the resulting feature objects which are adjacent to each other have small bounding volumes. Our approach of joining adjacent feature objects reduces the number of index entries (and thus the search time) significantly without compromising index selectivity.

For defining the join operation, we have to generalize the definition of a feature object. In section 3.2.1, a feature object is defined by varying the length of the last and first edge of a polygon section, which means that a feature object FO_{ij} corresponds to all polygon sections starting anywhere on edge e_i and ending on edge e_j. If we join the feature object FO_{ij} with the nearby feature object $FO_{i\,(j+1)}$, the resulting feature object corresponds to all sections starting anywhere on edge e_i and ending on the edge sequence $es_{j\,(j+1)}$. In the generalized definition of a feature object, the polygon sections belonging to the feature object may start anywhere on edge sequence $es_{i_1\,i_2}$ and end anywhere on edge sequence $es_{j_1\,j_2}$:

$$FO_{i_1 i_2 j_1 j_2}(x, y) = F(C(PS^{es_{i_1 j_2}}_{(x \cdot |es_{i_1 i_2}|)\ (2\pi - y \cdot |es_{j_1 j_2}|)}(t))) \text{ with } x, y \in \Re,\ 0 \le x \le 1,\ 0 \le y \le 1,\ 0 \le t \le 2\pi$$

where C is the curvature transformation and F is the Fourier transformation as defined in section 3.2.1. Note that for the special case of $(i_1 = i_2) \wedge (j_1 = j_2)$, this definition is equivalent to the one in section 3.2.1.

With this generalized definition of a feature object, joining a set of feature objects may be defined as follows: Given an initial set *IS* of feature objects FO_{iijj}, we are looking for a minimal result set *RS* of feature objects $FO_{i_1 i_2 j_1 j_2}$, which fulfills the following conditions:

(1) $\forall FO_{iijj} \in IS, \exists FO_{u_1 u_2 v_1 v_2} \in RS: (u_1 \le i \le u_2) \wedge (v_1 \le j \le v_2)$

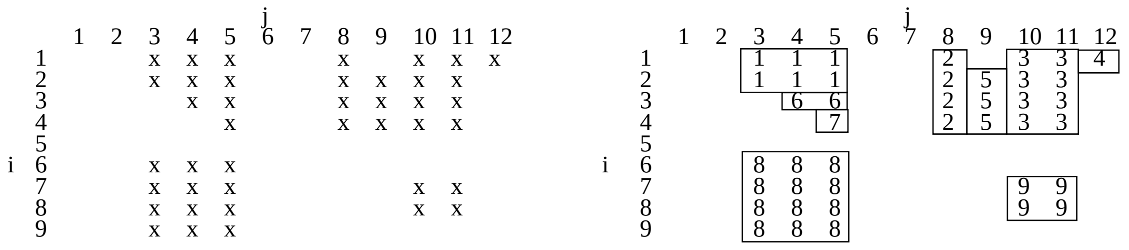
(2) $FO_{u_1 u_2 v_1 v_2} \in RS \Rightarrow (\forall i: u_1 \le i \le u_2\ \ \forall j: v_1 \le j \le v_2\ \ \exists FO_{iijj} \in IS)$

(3) $\forall FO_{u_1 u_2 v_1 v_2} \in RS:\ V_{BBox}(FO_{u_1 u_2 v_1 v_2}) \le V_{max}$

The first condition guarantees that all feature objects of *IS* are fully contained in *RS*, the second condition guarantees that there is no redundancy in *RS*, and the third condition guarantees that the bounding box volume of any feature object of *RS* is smaller than V_{max}, which is important to guarantee a good selectivity of the index.

```
join_feat_obj(set_of_FO IS, set_of_FO RS)
{
    Matrix mm;
    int i, j, x, y;
    bool enlargement_possible;

    ....                                   // initially set up matrix with elements of IS

    while (not mm.is_empty())
    {
        (i, j) = mm.choose_entry(); // choose first occupied position of the matrix

        x = 0; y = 1;                      // initialize test region

        enlargement_possible = TRUE;
        while (enlargement_possible)
        {
            enlargement_possible = FALSE;
            if (mm.cond2(i, i+x, j, j+y) and mm.cond3(i, i+x, j, j+y))
                enlargement_possible = TRUE;
            x++;

            .....                          // analogously test enlargement in j-direction
        }

        ....                               // reset x any y to the last successful values

        for (u = i to i+x)                 // remove joined entries from matrix
            for (v = j to j+y)
                mm.remove(u, v);

        RS.insert(i, i+x, j, j+y);// insert new feature object into RS
    }
}
```

Figure 13: Algorithm for Joining Feature Objects



a. Initial Set of Feature Objects (IS) b. Resulting Set of Feature Objects (RS)

Figure 14: Example for Joining Feature Objects

In Figure 13, we present the program code of the joining algorithm. The algorithm tries to enlarge the feature object alternately to the right and to the bottom as long as conditions (2) and (3) are fulfilled. Condition (1) is fulfilled implicitly since the algorithm continues until all feature objects of *IS* have been considered. Let us explain the algorithm using the following example. Assume, we have a matrix with an entry for all feature objects FO_{iijj} of *IS* which have a bounding box volume smaller than V_{max}. Figure 14a shows an example matrix for a real polygon from the database ($V_{max} = 1$). Every *x* in position *(i, j)* of the matrix corresponds to a feature object FO_{iijj} of the initial set *IS*. A join of feature objects may be seen as a union of rectangular regions in the matrix under the above conditions. Figure 14b shows the resulting set of joined feature objects *RS* computed by our algorithm. Initially, the algorithm chooses the first marked position *(i, j)* in the matrix. Starting from this position, a test region is enlarged first in *i*-direction, then in *j*-direction. The enlargement stops if the new region *(i, i+x, j, j+y)* is not totally filled (condition 2) with feature objects of *IS*, or if the volume of the bounding box of $FO_{i\,(i+x)\,j\,(j+y)}$ exceeds V_{max} (condition 3). Finally, we insert $FO_{i\,(i+x)\,j\,(j+y)}$ into *RS* and remove from *RS* all elements of *IS* which are contained in the joined feature object. The whole procedure is repeated until all feature objects have been removed from *IS*. Note that our algorithm maximizes the product $x \cdot y$ in each step and therefore removes as many entries from *IS* as possible. The resulting *RS* is a good approximation of the overall minimal *RS*.

The last step in creating the index is to insert the bounding boxes of the remaining feature objects into a spatial access method (SAM). Any SAM which is able to index two-dimensional feature objects in *d*-dimensional space may be used. Examples for potential SAMs include the R-tree [Gut 84] and its variants (R^+-tree [SRF 87], R*-tree [BKSS 90], P-tree [Jag 90b]), linear quadtrees [Gar 82], z-ordering [Ore 90] or other space-filling curves [Jag 90a], and grid-file based methods [NHS 84, Fre 87]. In most SAMs, it is sufficient for the indexing step to store the bounding boxes of the feature objects together with pointers to the feature objects and their (i_1, i_2, j_1, j_2)-values.

```
index_creation (set_of_Polygons DB)
{
  set_of_edge_sequences ESSet;
  FO fo;
  set_of_FO DecomposedFO, JoinFO, JoinResultFO;
  for (p in DB)
  {
      DecomposedFO = empty_set; JoinFO = empty_set; JoinResultFO = empty_set;
      ESSet = generate_edge_seqences(p);                // generates all edge sequences of p
      for (es in ESSet)
      {
          fo = calc_feature_obj(es);                    // calculates the feat.-obj. corres. to es
          if (volume(box(fo)) > Vmax)
              decompose_feature_obj(fo,0,1,0,1,DecomposedFO)  // all of decomp. algorithm
          else
              JoinFO.insert(fo);
      }
      join_feature_obj(JoinFO, JoinResultFO);                     // call of joining algorithm
      index_insert(DecomposedFO.union(JoinResultFO));   // insert feat.-obj. into index
  }
}
```

Figure 15: Algorithm for Creating the Partial Similarity Index

3.3 Correctness of our Indexing Technique

In Figure 15, we present the complete algorithm for index creation. The algorithm iterates over all polygons of the database. The first step is to generate all edge sequences of a polygon *p*. Then for each of the edge sequences, the algorithm determines the corresponding feature object. If the volume of the bounding box of the feature object is larger than V_{max}, we call the decomposition algorithm; otherwise, we insert the feature object into the set of feature objects to be joined. After all edge sequences of polygon *p* have been processed, we call the joining algorithm. Finally, we insert the feature objects resulting from the decomposition and joining step into the index.

The algorithm for searching partially similar polygons is presented in Figure 16. In querying for partially similar polygons, we only have to transform the query polygon section into a point in Fourier space. Using the spatial access method, we are then able to determine all bounding boxes *c*

```
partial_similarity_search(polygon_section SearchPolygon, set_of_Polygons ResSet)
{
      CandidateSet CS;
      FourierVector fv;
      fv = fourier(curvature(SearchPolygon));
      CS = index_neighborhood_search(fv, epsilon);                    // filter step
      for (c in CS)                                                   // refinement step
          if (distance(calc_feature_obj(c), fv) < epsilon)
              ResSet.insert(c);
}
```

Figure 16: Partial Similarity Search Algorithm

which are within an ε-neighborhood of the query point. In the refinement step, we have to compute the two-dimensional feature object for every candidate *c*, determine the minimal distance of the query point to the feature object, and select all candidates whose minimal distance is less than ε. In the following, we show the correctness of our algorithm.

Lemma 2

Our algorithm guarantees to find all $partially\ \varepsilon - similar$ polygon sections. In other words, our partial similarity index search does not produce any false dismissals.

Proof *(by contradiction)*

We assume that there is a polygon section $PS_{ab}^{p_i}$ of a database polygon p_i, which is partially similar to the search polygon s but is not in the result of the ε-neighborhood index search. We have to show that this assumption leads to a contradiction. Note that we only have to consider the filter step since in the refinement step there may not be any false dismissals. More formally, the assumption may be expressed as follows

$$\exists p_i\ \exists PS_{ab}^{p_i}\ \exists T\colon\ \delta(T(s), PS_{ab}^{p_i}) < \varepsilon\ \wedge\ PS_{ab}^{p_i}\ not\ in\ \varepsilon\text{-Neighborhood}(T(s)).$$

The term $PS_{ab}^{p_i}\ not\ in\ \varepsilon\text{-Neighborhood}(T(s))$ is equivalent to

$$\Leftrightarrow \forall fo \in Index : \delta_F(s, BBox(fo)) > \varepsilon$$

$$\Leftrightarrow \forall fo \in Index : d_{Euclid}(F(C(s)), BBox(fo)) > \varepsilon$$

The feature objects stored in the index are either decomposed or joined feature objects $(Index = DecomposedFO \cup JoinedFO)$. Therefore, in the next step of our proof we have to show that we may not lose polygon sections by the decomposition and joining algorithms. Since $\forall fo \in DecomposedFO : d_{Euclid}(F(C(s)), BBox(fo)) > \varepsilon$, it is clear that the Euclidean distance of $F(C(s))$ and the original undecomposed feature objects must be larger than ε. In case of the joining algorithm, condition 2 guarantees that

$$\forall FO_{u_1u_2v_1v_2} \in JoinedFO\colon \forall i\colon u_1 \le i \le u_2\ \ \forall j\colon v_1 \le j \le v_2 :\ \exists FO_{iijj} \in FO^{p_i}$$

which implies that $\forall FO_{u_1u_2v_1v_2} \in JoinedFO : \bigcup_{u_1 \le i \le u_2,\, v_1 \le j \le v_2} FO_{iijj} \subseteq FO_{u_1u_2v_1v_2}$. Together with our assumption $\forall fo \in JoinedFO : d_{Euclid}(F(C(s)), BBox(fo)) > \varepsilon$, we may conclude that the Euclidean distance of $F(C(s))$ and each of the unjoined polygon sections must be larger than ε.

Altogether, we have now shown

$$\forall fo \in Index : d_{Euclid}(F(C(s)), BBox(fo)) > \varepsilon$$

$$\Rightarrow \forall fo \in FO^{es_{ij}}, es_{ij} \in ES^{p_i} : d_{Euclid}(F(C(s)), BBox(MF^{es_{ij}})) > \varepsilon$$

$$\Leftrightarrow \forall fo \in FO^{es_{ij}}, es_{ij} \in ES^{p_i} : d_{Euclid}(F(C(s)), BBox(F(C(PS^{es_{ij}}_{(a \cdot |e_i|)\ \ (2\pi - b \cdot |e_j|)})))) > \varepsilon$$

$$\Rightarrow \forall fo \in FO^{es_{ij}}, es_{ij} \in ES^{p_i} : d_{Euclid}(F(C(s)), F(C(PS^{es_{ij}}_{(a \cdot |e_i|)\ \ (2\pi - b \cdot |e_j|)}))) > \varepsilon$$

$$\Leftrightarrow \forall PS^{p_i}_{ab} : d_{Euclid}(F(C(s)), F(C(PS^{p_i}_{ab}))) > \varepsilon$$

$$\Rightarrow \forall PS^{p_i}_{ab} : \delta(T(s), PS^{p_i}_{ab}) > \varepsilon \qquad \text{(since } C \circ F \text{ is invariant against affine transformations)}$$

which is in contradiction to the first part of our assumption. q.e.d.

4. Implementation and Analysis

In this section, we provide an overview of the implementation of our approach. We further present the experimental results of applying our method on a large database of real data. Our experimental results show that the maximum allowed volume of any bounding box in the index (V_{max}) strongly influences the search time and the space requirements of our method. We show that there is an optimal value for V_{max} and determine the optimal value V^{opt}_{max} experimentally. We further show that our method scales well with increasing database size.

4.1 Implementation

To test our partial similarity index, we integrated the algorithms into our prototype CAD-database-system ***S3*** (Similarity Search System). ***S3*** is a database system for the management of industrial parts which are described by a polygonal CAD model. The system stores polygon contours for each part of the database and allows the user to create indexes using different indexing methods, providing a testbed for comparing the query performance of the different indexing

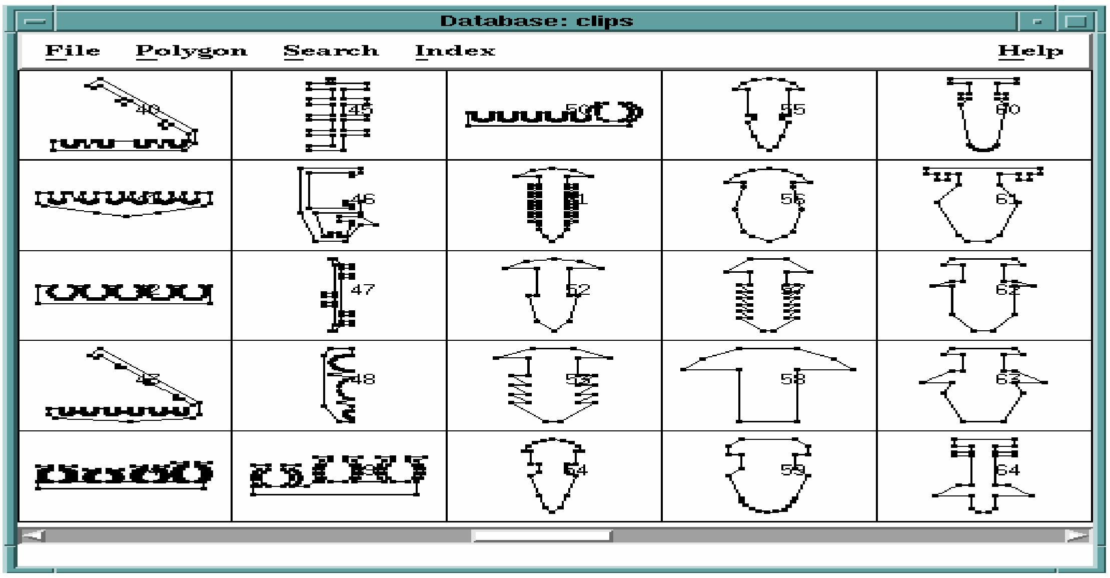


Figure 17: Polygon Database

methods. ***S3*** has been implemented in C++ using X11 / OSF Motif, and is running under HP-UX as well as Linux. All experimental results presented in the next sections are computed on an HP 715/64 workstation with 64 MBytes of main memory and several GBytes of secondary storage. The spatial access method used is an object-oriented version of the R*-tree which is implemented as a C++-template to support different data types. The R*-tree allows different types of queries such as partial range queries and nearest neighbor queries.

4.2 Data, Query Polygons and Results

As indicated in the introduction, our algorithms have been designed to process real data from a major supplier of parts for the German car manufacturing industry. The data used in our experiments are the 2D-contours of parts called ‘clips’ which are used in the car manufacturing industry for holding and joining components in a car. Since, in general, ‘clips’ are made of plastic material and produced by injection molding, the variety of shapes is nearly unlimited. The ‘clips’ data used for our experiments result from real CAD drawings from our industrial partner. In Figure 17, a small portion of our ‘clips’ database is presented.

The queries used to test our algorithms are a set of real query polygon sections provided by our industrial partner. In Figure 18, we show an example of such a query together with the corresponding result generated by our program. Note that the leftmost of the retrieved polygons (polygon number 15) is partially congruent to the query polygon section, the others are only partially similar sorted with respect to increasing distance. The similarity of the retrieved polygons turned out to correspond well with the human intuition of similarity. A more objective study of the quality of our similarity measure is difficult since it involves humans and their intuitive understanding of similarity.

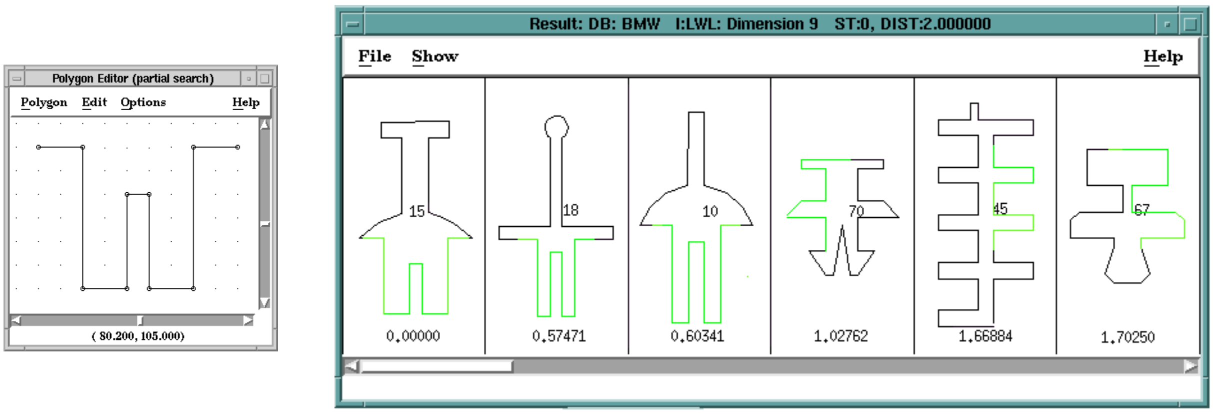

Polygon Editor (partial search)
Polygon Edit Options Help
(80.200, 105.000)
Result: DB: BMW I:LWL: Dimension 9 ST:0, DIST:2.000000
File Show Help
15
0.00000
18
0.57471
10
0.60341
70
1.02762
45
1.66884
67
1.70250

Figure 18: Example for a Query and the Result Produced by our Index-Search

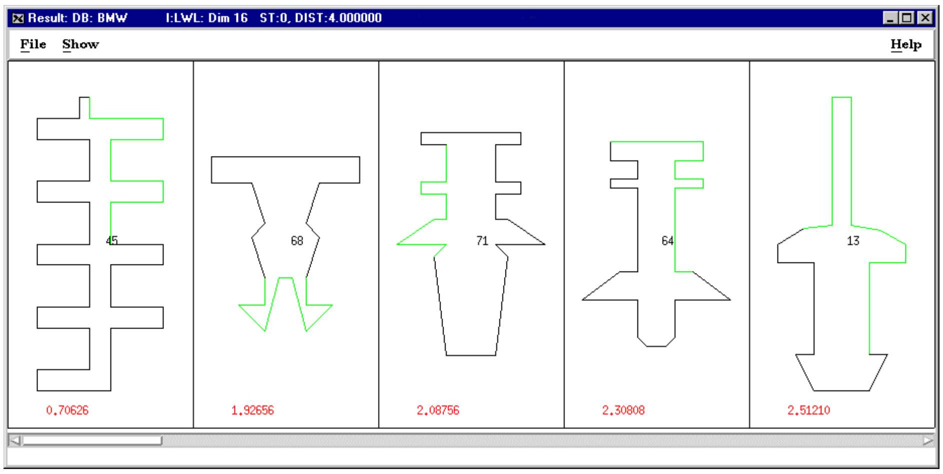
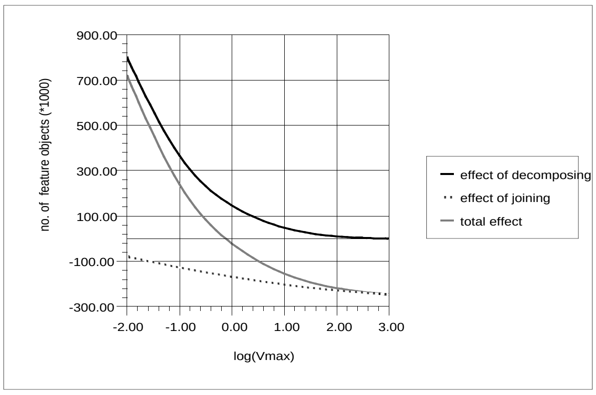


Figure 19: Result Provided by the Gary-Mehrotra Approach [MG 93, MG 95]

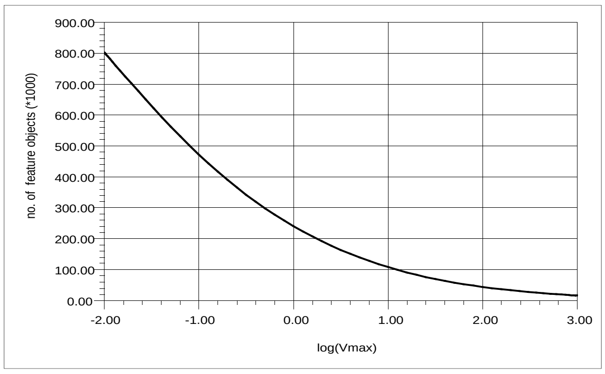
To show the advantage of a continuous solution of the partial similarity search problem in contrast to an edge-based solution, we also implemented one of the best existing methods - the well-known Gary-Mehrotra approach [MG 93, MG 95]. In Figure 19, we show the result of querying the same database by using the query polygon shown in Figure 18a. It is clear that the Gary-Mehrotra does induce false dismissals, i.e. it does not find all partially congruent polygons (e.g. result polygons 1, 2, and 3 in Figure 18) since it does not consider polygon sections starting and ending between vertices.

4.3 Experimental Results

In this subsection, we present an experimental performance evaluation of our algorithms. An important parameter of our algorithms is the constant value V_{max} which largely influences the size of the generated index. In Figure 20a, we show the effect of the decomposition and joining algorithms for a varying value of V_{max}. As expected, the joining algorithm reduces the number of index entries while the decomposition algorithm increases the number of index entries. The effect of the decomposition algorithm dominates for small values of V_{max} since many feature objects have to be decomposed. For larger V_{max} the number of feature objects which have to be decomposed decreases rapidly and converges against 0. Inversely, the effect of the joining algorithm is rather low for small V_{max} (only few feature objects may be joined) but is increasing for larger V_{max} since many feature objects may be joined. For a V_{max} of about 1 (=10^0) the two algorithms are neutralizing each other. Note that the decomposition algorithm improves the quality of the filter step significantly since the volume of bounding boxes is decreased, and the joining algorithm does not worsen the quality substantially since the joined boxes remain rather small.



a. Effect of Decomposition and Joining



b. Total Number of feature objects

Figure 20: Index Size Depending on V_{max}

Figure 20b shows the development of the index size with an increasing value of V_{max}. The decrease of the index size follows the curve describing the total effect of decomposition and joining (cf. Figure 20a). Note that we use a logarithmic scale for V_{max}, which means that the reduction of the index size is very high until V_{max} reaches a value of 10^1 while the increase of the index size is super-exponential for $V_{max} \rightarrow 0$.

For determining an optimal value of V_{max}, we have to consider not only the space requirements but also the development of the search times depending on V_{max}. Figure 22a shows the CPU-time[1] of the filter step. Obviously, we get shorter filter times for higher V_{max} since the index sizes are also smaller. Note that the decrease of the search time in an R*-tree is not continuous since there is a discontinuity each time the depth of the R*-tree is decreasing. Figure 22b shows the time required for the refinement step, which is the time needed to recompute the feature objects of the candidates and test the distances of the candidates against the query polygon. The number of candidates is inversely proportional to the selectivity of the index. For a higher value of V_{max} the bounding boxes in the index become bigger, which means that the selectivity of the index decreases and the number of candidates increases. Since the time curve for

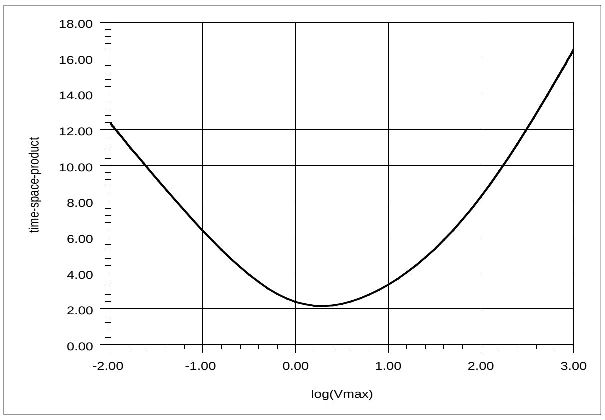
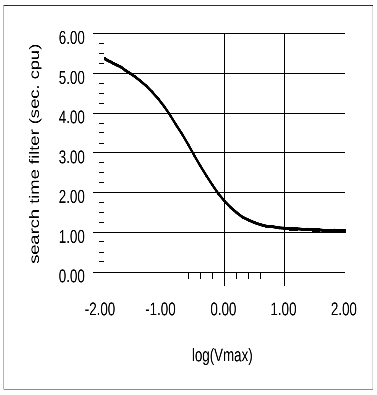
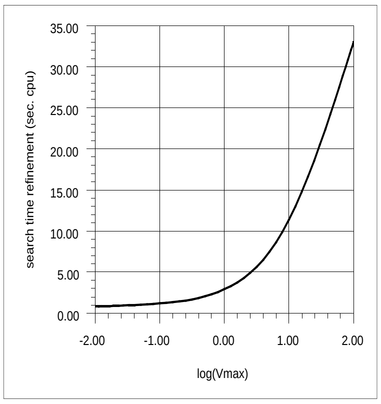


Figure 21: Time-Space Requirements

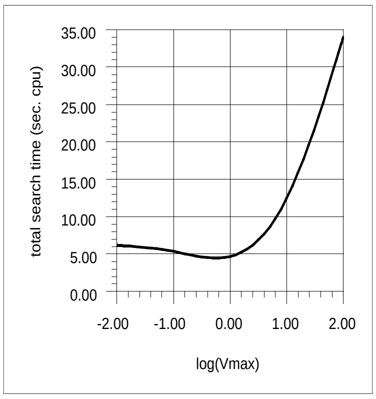
1. Note that the search time of our method is mainly CPU-bound and therefore, the CPU-time directly corresponds to the elapsed time.



a. Filter Step



b. Refinement Step



c. Total Search Time

Figure 22: Search Time

the filter and refinement step are almost inverse, the total search time has a minimum for V_{max} between 0.1 (=10^{-1}) and 1 (=10^{0}) (cf. Figure 22c). To determine the optimal value for V_{max}, we have to consider space and time requirements of our algorithm. As shown in Figure 20b, the index size decreases with an increasing V_{max}. Unfortunately the search time explodes for high values of V_{max} (cf. Figure 22c). To consider both space and time performance of our algorithm in determining the optimal value for V_{max}, we use the weighted product of the two curves (cf. Figure 21) with a higher weight for the time curve since the search time is more important for our application than the space requirements. The quadratic regression of the product curve provides

$$f(x) = 2.033 \cdot x^2 - 1.282 \cdot x + 1.971 .$$

The minimum of the regression curve is 0.315 which means that the optimal value for V_{max} is $V_{max}^{opt} = 10^{0.315} \approx 2.07$.

For the experiments reported so far, we used a database of 800 polygons with about 50 edges on the average, resulting in about 275,000 edge sequences (problem size $N \approx 275000$). The generated extended feature object indexes had a size of up to 70 MBytes (for V_{max} =10^{-2}). In our final experiments, we examined the scale-up of our algorithm with increasing N (for

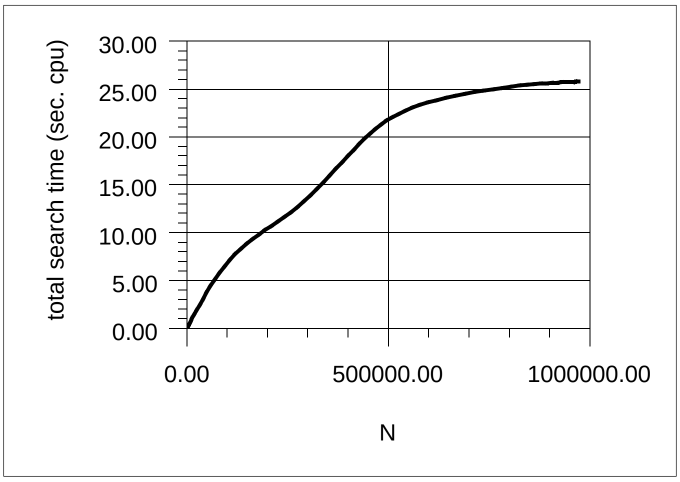


Figure 23: Search Time Depending on *N*

$V_{max} = V_{max}^{opt}$). Figure 23 shows the resulting curve for the total search time. As expected, the curve is sublinear with increasing *N*. The undulations of the curve are again due to the jumps in the height of the R*-tree.

5. Conclusions

In this paper, we presented an efficient solution for the continuous partial similarity search problem of 2D-polygons. We introduced the new technique of extended feature objects which — in contrast to conventional algorithms — guarantees that there are no false dismissals. The method is based on a transformation of the polygon sections into extended feature objects in a high-dimensional space. To enhance the selectivity of the index, we developed an algorithm for decomposing objects with large bounding volumes into smaller ones, and to decrease the size of the index, we developed an algorithm for joining adjacent objects. In our experimental evaluation, we have determined the optimal value for V_{max}, the threshold value for the decomposition and joining algorithm. We have further shown that our method scales well even for large databases of polygons.

The advantages of our new approach are: It is provably correct (cf. subsection 3.3) and it is efficient since it is based on a multidimensional indexing of the resulting extended feature objects. A disadvantage is the large number of edge sequences $\left[O\left(\sum_{i=1}^{n} m_i^2\right)\right]$ and the high preprocessing time needed for determining the extended feature objects of the edge sequences. Both disadvantages, however, are the price which has to be paid in exchange for obtaining a solution of the computationally hard continuous partial similarity search problem. The parameters of our approach are important to adapt our techniques to the specific needs of different applications. The maximum volume V_{max} may be used to find the optimal trade-off between index size and filtering selectivity, and the radius *r* of our curvature approximation may be used to increase the quality of the curvature approximation. Both parameters may be varied depending on the application.

Our future work includes a practical evaluation of the system ***S3*** by our industrial partner. The evaluation will include a detailed comparison of different types of CAD data and different types of queries. We also intend to apply the technique of extended feature objects in other application contexts such as pattern recognition or protein docking. Our future work includes an examination of alternative index structures on their suitability for high-dimensional data and their performance for large databases. Also interesting will be an examination of how an adaptive decomposition of large objects into smaller ones may improve the performance of the indexing methods. Another area of future work will be the partial similarity problem for contours of 3D-polyhedra. This, however, is a difficult problem since there are many obstacles that arise in going from 2D to 3D.

Acknowledgment

We thank Rainer Urian for his support in developing the details of the Fourier theory. We are also thankful to Thomas Seidl, Thomas Schmidt, and the anonymous referees for their comments on previous versions of the paper.

Appendix: Mathematics of the Fourier Transformation

Any periodic function $f: \Re \to \Re$ with period 2π can be approximated by the sum

$FS(x) = \frac{a_0}{2} \sum_{k=1}^{n} (a_k \cos(kx) + b_k \sin(kx))$, where the coefficients a_k and b_k are defined as

$$a_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(kx) dx \quad \text{and} \quad b_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(kx) dx .$$

The coefficients *a* and *b* can be seen as vectors in an *n*-dimensional vector space. The base of the vector space is build by *cos(x), sin(x), cos(2x), sin(2x), ..., cos(nx), sin(nx)*. In general, integrals of the form $\int f(x) \sin(x)\ \mathrm{dx}$ are difficult to solve analytically. For the special case of *f(x)* being a square wave function however, $\int f(x) \sin(x)\ \mathrm{dx}$ can be determined easily. Let us assume that *f(x)* has a value of $\frac{1}{r}$ in the interval $[u, v]$ and is zero elsewhere. Since the value of the integral is zero outside of $[u, v]$, we just have to integrate from *u* to *v*. Therefore, we are able to calculate a_k and b_k as:

$$a_k = \frac{1}{\pi} \int_u^v \frac{1}{r} \cos(kx) dx = \frac{1}{\pi k r} (\sin(kv) - \sin(ku)) \qquad \text{(equation 1a)}$$

$$b_k = \frac{1}{\pi} \int_u^v \frac{1}{r} \sin(kx) dx = -\frac{1}{\pi k r} (\cos(kv) - \cos(ku)) \qquad \text{(equation 1b)}$$

For a detailed treatment of the Fourier theory see [Wei 80].

References

[AB 92] Alt H., Blömer J.: *'Resemblance and Symmetries of Geometric Patterns'*, Data Structures and Efficient Algorithms, in: LNCS, Vol. 594, Springer, 1992, pp. 1-24.

[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, 1993, LNCS 730, pp. 69-84

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

[Ber 97] Berchtold S.: '*Geometry based search of similar parts*', (in german), Ph.D. thesis, University of Munich, 1997.

[BKS 93] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[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.

[FBFH 94] Faloutsos C., Barber R., Flickner M., Hafner J., et al.: *'Efficient and Effective Querying by Image Content'*, Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.

[Fre 87] Freeston M.: *'The BANG file: A new kind of grid file'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco, CA, 1987, pp. 260-269.

[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.

[Gar 82] Gargantini I.: *'An Effective Way to Represent Quadtrees'*, Comm. of the ACM, Vol. 25, No. 12, 1982, pp. 905-910.

[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.

[Gue 89] Günther O.: *'The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases'*, Proc. 5th Int. Conf. on Data Engineering, Los Angeles, CA, 1989, pp. 598-605.

[HB 86]] Horn P., Berthold K.: '*Robot vision*', MIT Press, Cambridge, MA, 1986

[Jag 90a] Jagadish H. V.: *'Linear Clustering of Objects with Multiple Attributes'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 332-342.

[Jag 90b] Jagadish H. V.: *'Spatial Search with Polyhedra'*, Proc. 6th Int. Conf. on Data Engineering, Los Angeles, CA, 1990, pp. 311-319.

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

[KM 95] Kehrer, L., Meinecke, C. (in press): *'Perceptual Organization of Visual Patterns: The Segmentation of Textures'*, In W. Prinz, B. Bridgeman (Eds.): Handbook of Perception and Action: Vol. 1: Perception, Chapter 2., London: Academic Press, 1995.

[KSP 95] Kauppinen H., Seppänen T., Pietikäinen M.: *'An Experimental Comparison of Autoregressive and Fourier-Based Descriptors in 2D Shape Classification'*, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, No. 2, 1995.

[MG 93] Mehrotra R., Gary J.: *'Feature-Based Retrieval of Similar Shapes'*, Proc. 9th Int. Conf. on Data Engeneering, 1993.

[MG 95] Mehrotra R., Gary J.: *'Feature-Index-Based Sililar Shape retrieval'*, Proc. of the 3rd Working Conf. on Visual Database Systems, 1995.

[Mum 87] Mumford D.: *'The Problem of Robust Shape Descriptors'*, Proc. of the IEEE First International Conf. on Computer Vision, 1987.

[NHS 84] Nievergelt J., Hinterberger H., Sevcik K. C.: *'The Grid File: An Adaptable, Symmetric Multikey File Structure'*, ACM Trans. on Database Systems, Vol. 9, No. 1, 1984, pp. 38-71.

[Ore 90] Orenstein J., : *'A comparison of spatial query processing techniques for native and parameter spaces'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 326-336.

[PF 94] E. Petrakis, C. Faloutsos: *‘Similarity Searching in Large Image DataBases’*, Technical Report CS-TR-3388, University of Maryland, 1994.

[RH 92] Rigoutsos I., Hummel R.: *’Massively Parallel Model Matching: Geometric Hashing on the Connection Machine’*, IEEE Computer, Vol. 25, 2, pp.33-42, 1992

[SK 90] Seeger B., Kriegel H.-P.: *‘The Buddy Tree: An Efficient and Robust Access Method for Spatial Data Base Systems’*, Proc. 16th Int. Conf. on Very Large Data Bases, Brisbane, Australia, 1990, pp. 590-601.

[SM 90] Stein F., Medioni G.: *‘Efficient Two Dimensional Object Recognition’*, 10th. Int. Conf. on Pattern Recognition, Atlantic City, 1990, pp. 13-17.

[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.

[Wei 80] Weisstein, N.: *‘The Joy of Fourier Analysis’*, In C.S. Harris (Edt.), Visual coding and adaptability, Hillsdale, NJ: Erlbaum, 1980.

[WW 80] Wallace T., Wintz P.: *‘An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors’*, Computer Graphics and Image Processing, Vol. 13, pp. 99-126, 1980.