Y. Ioannidis et al. (eds.): EDBT 2006, LNCS 3896, pp. 276-294, 2006.

Similarity Search on Time Series based on Threshold Queries

Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, Peter Kunath, Alexey Pryakhin, Matthias Renz

Institute for Computer Science, University of Munich
{assfalg,kriegel,kroegerp,kunath,pryakhin,renz}@dbs.ifi.lmu.de

Abstract. Similarity search in time series data is required in many application fields. The most prominent work has focused on similarity search considering either complete time series or similarity according to subsequences of time series. For many domains like financial analysis, medicine, environmental meteorology, or environmental observation, the detection of temporal dependencies between different time series is very important. In contrast to traditional approaches which consider the course of the time series for the purpose of matching, coarse trend information about the time series could be sufficient to solve the above mentioned problem. In particular, temporal dependencies in time series can be detected by determining the points of time at which the time series exceeds a specific threshold. In this paper, we introduce the novel concept of *threshold queries* in time series databases which report those time series exceeding a user-defined query threshold at similar time frames compared to the query time series. We present a new efficient access method which uses the fact that only partial information of the time series is required at query time. The performance of our solution is demonstrated by an extensive experimental evaluation on real world and artificial time series data.

1 Introduction

Similarity search in time series data is required in many application fields, including financial analysis, medicine, meteorology, analysis of customer behavior, or environmental observation. As a consequence, a lot of research work has focused on similarity search in time series databases recently.

In this paper, we introduce a novel type of similarity queries on time series databases called *threshold queries*. A threshold query specifies a query time series Q and a threshold τ. The database time series as well as the query sequence Q are decomposed into time intervals of subsequent elements where the values are (strictly) above τ. Now, the threshold query returns these time series objects of the database which have a similar interval sequence of values above τ. Note, that the entire set of absolute values are irrelevant for the query. The time intervals of a time series t only indicate that the values of t within the intervals are above a given threshold τ.

The novel concept of threshold queries is an important technique useful in many practical application areas.

Application 1 For the pharma industry it is interesting which drugs cause similar effects in the blood values of a patient at the same time after drug treatment. Obviously, effects such as a certain blood parameter exceeding a critical level τ are of particular interest. A threshold query can return for a given patient all patients in a database who show a similar reaction to a medical treatment w.r.t. a certain blood parameter exceeding the threshold τ.

Application 2 The analysis of environmental air pollution becomes more and more important and has been performed by many European research projects in the recent years. The amount of time series data derived from environmental observation centers, increases drastically with elapsed time. Furthermore, modern sensor stations record many attributes of the observed location simultaneously. For example, German state offices for environmental protection maintain about 127 million time series each representing the daily course of several air pollution parameters. An effective and efficient processing of queries like "return all ozone time series which exceed the threshold $\tau_1 = 50\mu g/m^3$ at a similar time as the temperature reaches the threshold $\tau_2 = 25°C$" may be very useful. Obviously, the increasing amount of data to be analyzed poses a big challenge for methods supporting threshold queries efficiently.

Application 3 In molecular biology the analysis of gene expression data is important for understanding gene regulation and cellular mechanisms. Gene expression data contains the expression level of thousands of genes, indicating how *active* one gene is over a set of time slots. The expression level of a gene can be up (indicated by a positive value) or down (negative value). From a biologist's point of view, it is interesting to find genes that have a similar up and down pattern because this indicates a functional relationship among the particular genes. Since the absolute up/down-value is irrelevant, this problem can be represented by a threshold query. Each gene provides its own interval sequence, indicating the time slots of being up. Genes with similar interval sequence thus have a similar up and down pattern.

Time series (sometimes also denoted as time sequences) are usually very large containing several thousands of values per sequence. Consequently, the comparison of two time series can be very expensive, particularly when considering the entire sequence of values of the compared objects. However, the application examples above do not need the entire course of the time series, rather "qualitative" course information with respect to a certain threshold is sufficient to determine the correct query results. Consider again the query example of the second application. Let us assume, that we have the information when the ozone values are above $50\mu g/m^3$ for all ozone sequences in form of time intervals. Then, the processing of this query is reduced to compare sequences of time intervals. Usually, the number of intervals is much less than the number of ozone values per ozone sequence. With this aggregated information, obviously the query can be answered more efficiently compared to the approach where the time intervals are not given in advance.

As mentioned above, this is the first contribution to the novel concept of threshold queries for time series databases.

In summary, our contributions are the following:

- We introduce and formalize the novel concept of threshold queries on time series databases.
- We present a novel data representation of time series which support such threshold queries efficiently.
- We propose an efficient algorithm for threshold queries based on this new representation.
- We present a broad experimental evaluation including performance tests of our proposed algorithms and the evidence that the new type of query yields important information and is thus required in several application fields.

The remainder is organized as follows. We give a short overview of the field of similarity search in time series databases in Section 2. Section 3 formally introduces the notion of threshold queries. In Section 4, we show how time series can be represented in order to support threshold queries for arbitrary threshold values efficiently. Section 5 describes an efficient query algorithm based on the proposed representation. The effectiveness and efficiency of our algorithm are evaluated in Section 6. Section 7 concludes the paper with a summary of our findings and an outlook to future extensions.

2 Related Work

In the last decades, time series have become an increasingly prevalent type of data. As a result, a lot of work on similarity search in time series databases has been published. The proposed methods mainly differ in the representation of the time series; a survey is given in [1].

A time series X can be considered as a point in n-dimensional space. This suggests that time series could be indexed by spatial access methods such as the R-tree and its variants [2]. However, most spatial access methods degrade rapidly with increasing data dimensionality due to the "curse of dimensionality". In order to utilize spatial access methods, it is necessary to perform dimensionality reduction and/or to perform multi-step query processing. Standard techniques for dimensionality reduction have been applied successfully to similarity search in time series databases, including Discrete Fourier Transform (DFT) (e.g. [3]), Discrete Wavelet Transform (DWT) (e.g. [4]), Piecewise Aggregate Approximation (PAA) (e.g. [5]), Singular Value Decomposition (SVD) (e.g. [6]), Adaptive Piecewise Constant Approximation (APCA) [1], and Chebyshev Polynomials [7]. In [8], the authors propose the GEMINI framework, that allows to incorporate any dimensionality reduction method into efficient indexing, as long as the distance function on the reduced feature space fulfills the lower bounding property.

However, all techniques which are based on dimensionality reduction cannot be applied to threshold queries because necessary temporal information is lost. Usually, in a reduced feature space, the original intervals indicating that the time series is above a given threshold cannot be generated. In addition, the approximation generated by dimensionality reduction techniques cannot be used

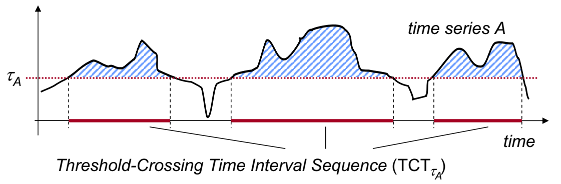

Fig. 1. Threshold-Crossing Time Intervals

for our purposes directly because they still represent the exact course of the time series rather than intervals of values above a threshold.

For many applications, the Euclidean distance may be too sensitive to minor distortions in the time axis. It has been shown, that Dynamic Time Warping (DTW) can fix this problem [1]. Using DTW to measure the distance between two time series t_1 and t_2, each value of t_1 may be matched with any value of t_2. However, DTW is not applicable to threshold queries because it considers the absolute values of the time series rather than the intervals of values above a given threshold.

In [9], a novel bit level approximation of time series for similarity search and clustering is proposed. Each value of the time series is represented by a bit. The bit is set to 1 if the value of the time represented by the bit is strictly above the mean value of the entire time series, otherwise it is set to 0. Then, a distance function is defined on this bit level representation that lower bounds the Euclidean distance and, by using a slight variant, lower bounds DTW. However, since this representation is restricted to a certain predetermined threshold, this approach is not applicable for threshold queries where the threshold is not known until query time.

To the best of our knowledge, there does neither exist any access method for time series, nor any similarity search technique which efficiently supports threshold queries.

3 Threshold Queries on Time Series

In this section, we introduce the novel concept of threshold queries and present techniques allowing for an efficient query processing. We define a time series X as a sequence of pairs $(x_i, t_i) \in \mathbb{R} \times T : (i = 1..N)$, where T denotes the domain of time and x_i denotes the measurement corresponding to time t_i. Furthermore, we assume that the time series entities are given in such a way that $\forall i \in 1, .., N-1 : t_i < t_{i+1}$. Let us note, that in most applications the time series derive from discrete measurements of continuously varying attributes. However, commonly time series are depicted as continuous curves, where the missing curve values (i.e. values between two measurements) are estimated by means of interpolation. From the large range of appropriate solutions for time series interpolation, in this paper we assume that the time series curves are supplemented by linear interpolation which is the most prevalent interpolation method. In the rest of this paper, if not stated otherwise, $x(t) \in \mathbb{R}$ denotes the (interpolated) time series value of time series X at time $t \in T$.

Definition 1 (Threshold-Crossing Time Interval Sequence). *Let $X = \langle (x_i, t_i) \in \mathbb{R} \times T : i = 1..N \rangle$ be a time series with N measurements and $\tau \in \mathbb{R}$ be a threshold. Then the* threshold-crossing time interval sequence *of X with respect to τ is a sequence $TCT_\tau(X) = \langle (l_j, u_j) \in T \times T : j \in \{1, .., M\}, M \leq N \rangle$ of time intervals, such that*

$$\forall t \in T : (\exists j \in \{1, .., M\} : l_j < t < u_j) \Leftrightarrow x(t) > \tau.$$

An interval $tct_{\tau,j} = (l_j, u_j)$ of $TCT_\tau(X)$ is called threshold-crossing time interval.

The example shown in Figure 1 depicts the threshold-crossing time interval sequence of the time series A which corresponds to threshold τ_A.

Definition 2 (Distance between Time Intervals). *Let $t1 = (t1_l, t1_u) \in T \times T$ and $t2 = (t2_l, t2_u) \in T \times T$ be two time intervals. Then the distance function $d_{int} : (T \times T) \times (T \times T) \rightarrow \mathbb{R}$ between two time intervals is defined as:*

$$d_{int}(t1, t2) = \sqrt{(t1_l - t2_l)^2 + (t1_u - t2_u)^2}$$

Intuitively, two time intervals are defined to be similar if they have ”similar” starting and end points, i.e. they are starting at similar times and ending at similar times.

Since for a certain threshold τ a time series object is represented by a sequence or a set of time intervals, we need a distance/similarity measure for sets of intervals. Let us note, that intervals correspond to points in a two-dimensional space, where the starting point corresponds to the first dimension and the ending point corresponds to the second dimension. This transformation is explained in more detail in the next section. Several distance measures for point sets have been introduced in the literature [10]. The Sum of Minimum Distances (SMD) most adequately reflects the intuitive notion of similarity between two threshold-crossing time interval sequences. According to the SMD we define the *threshold-distance d_{TS}* as follows:

Definition 3 (Threshold-Distance). *Let X and Y be two time series and $S_X = TCT_\tau(X)$ and $S_Y = TCT_\tau(Y)$ be the corresponding threshold-crossing time interval sequences.*

$$d_{TS}(S_X, S_Y) = \frac{1}{|S_X|} \cdot \sum_{s \in S_X} \min_{t \in S_Y} d_{int}(s, t) + \frac{1}{|S_Y|} \cdot \sum_{t \in S_Y} \min_{s \in S_X} d_{int}(t, s),$$

The idea of this distance function is to map every interval from one sequence to the closest (most similar) interval of the other sequence and vice versa. Time series having similar shapes, i.e. showing a similar behavior, may be transformed into threshold-crossing time interval sequences of different cardinalities. Since the above distance measure does not consider the cardinalities of the interval sequences, this distance measure is quite adequate for time interval sequences. Another advantage is that the distance measure only considers local similarity.

This means, that for each time interval only its nearest neighbor (i.e. closest point) of the other sequence is taken into account. Other intervals of the counterpart sequence have no influence on the result.

Let us note that the threshold-distance between two time series according to a certain threshold τ is also called τ-similarity.

Definition 4 (Threshold Query). *Let TS be the domain of time series objects. The threshold query consists of a query time series $Q \in TS$ and a query threshold $\tau \in \mathbb{R}$. The threshold query reports the smallest set $TSQ_k(Q, \tau) \subseteq TS$ of time series objects that contains at least k objects from TS such that*

$$\forall X \in TSQ_k(Q, \tau), \forall Y \in TS - TSQ_k(Q, \tau) :$$

$$d_{TS}(TCT_\tau(X), TCT_\tau(Q)) < d_{TS}(TCT_\tau(Y), TCT_\tau(Q)).$$

Let us note, that if not stated otherwise we assume $k = 1$ throughout the rest of this paper.

4 Efficient Management of Threshold-Crossing Time Intervals

The simplest way to execute a threshold query $TSQ_k(Q, \tau)$ is to sequentially read each time series X from the database, to compute the threshold-crossing time interval sequence $S_X = TCT_\tau(X)$ and to compute the threshold-similarity function $d_{TS}(S_X, TCT_\tau(Q))$. Finally, we report the time series which yields the smallest distance $d_{TS}(S_X, TCT_\tau(Q))$. However, if the time series database contains a large number of objects and the time series are reasonably large, then obviously this type of performing the query becomes unacceptably expensive. For this reason we use a convenient access method on the time series data.

In this section, we present two approaches for the management of time series data, both of which efficiently support threshold queries. The key point is that we do not need to access the complete time series data at query time. Instead only partial information of the time series objects is required. At query time we only need the information at which time frames the time series is above the specified threshold. We can save a lot of I/O cost if we are able to access only the relevant parts of the time series at query time. The basic idea of our approach is to pre-compute the $TCT_\tau(X)$ for each time series object X and store it on disk in such a way it can be accessed efficiently.

For the sake of clarity, we first present a simple approach with constant threshold value τ for all queries. Afterwards, we present the general approach which supports arbitrary choice τ.

4.1 Representing Threshold-Crossing Time Intervals with Fixed τ

Let us assume that the query threshold τ is fixed for all queries. Then, we can compute the corresponding $TCT_\tau(X)$ for each time series X. Consequently, each

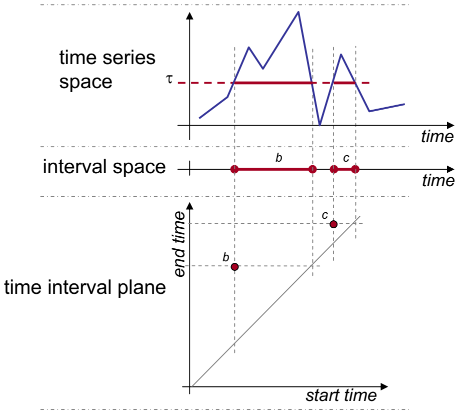

Fig. 2. Mapping of Time Intervals to the Time Interval Plane

time series object is represented by a sequence of intervals. There are several methods to store intervals efficiently, e.g. the RI-Tree [11]. However, they only support intersection queries on interval data but do not efficiently support similarity queries on interval sequences. Besides, they cannot be used for the general approach when τ is not fixed. We propose a solution which supports similarity queries on intervals and which can be easily extended to support queries with arbitrary τ.

Time intervals can also be considered as points in a 2-dimensional plane[12]. In the following we will refer to this plane as time interval plane. The 1-dimensional intervals (*native space*) are mapped to the time interval plane by taking their start and end points as two dimensional coordinates. This representations has some advantages for the efficient management of intervals. First, the distances between intervals are preserved. Second, the position of large intervals, which are located within the upper-left region, substantially differs from the position of small intervals (located near the diagonal). However, the most important advantage is that this plane preserves the similarity of intervals according to Definition 2. Let $t_1 = (x_1, y_1)$ and $t_2 = (x_2, y_2)$ be two time intervals, then the distance between t_1 and t_2 is equal to $d_{int}(t_1, t_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ which corresponds to the Euclidean distance in the time interval plane.

The complete threshold-crossing time interval sequence is represented by a set of 2-dimensional points in the time interval plane. The transformation chain from the original time series to the point set in the time interval plane is depicted in Figure 2. In order to efficiently manage the point sets of all time series objects, we can use a spatial index structure as for instance the R*-tree [13]. In particular, the R*-tree is very suitable for managing points in low-dimensional space which are not equally distributed. Additionally, it supports the nearest neighbor query which will be required to perform the threshold queries efficiently (more details for the query process will be presented in Section 5). Let us note, that each object is represented by several points in the time interval plane. Consequently, each object is referenced by the index structure multiple times.

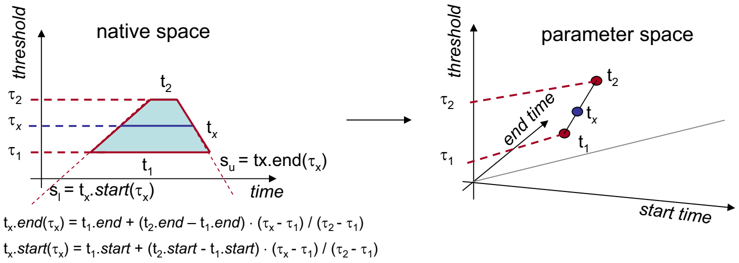
native space
threshold
τ_2
τ_x
τ_1
t_2
t_x
t_1
s_u = tx.end(τ_x)
$s_l = t_x.start(\tau_x)$
time
$t_x.end(\tau_x) = t_1.end + (t_2.end - t_1.end) \cdot (\tau_x - \tau_1) / (\tau_2 - \tau_1)$
$t_x.start(\tau_x) = t_1.start + (t_2.start - t_1.start) \cdot (\tau_x - \tau_1) / (\tau_2 - \tau_1)$
parameter space
threshold
τ_2
τ_1
end time
t_2
t_x
t_1
start time

Fig. 3. Interval Ranges in Parameter Space

4.2 Representing Threshold-Crossing Time Intervals for Arbitrary τ

In contrast to the first approach presented above we will now describe how to manage threshold queries for arbitrary threshold values τ efficiently. First, we have to extend the transformation task of the simple approach, in such a way that the time interval plane representations of the TCT_τs of the time series are available for all possible threshold values τ. Therefore, we extend the time interval plane by one additional dimension which indicates the corresponding threshold values. In the following, we will call this space *parameter space*. A 2-dimensional plane along the threshold axis parallel to the (lower,upper)-plane at a certain threshold τ in the parameter space is called time interval plane of threshold τ.

In the parameter space sets of threshold-crossing time intervals can be efficiently represented as follows. As shown in the example depicted in Figure 3, the set of all possible threshold-crossing time intervals of one time series which are left-bounded by the segment s_l and right-bounded by the segment s_u and whose threshold value is between τ_1 and τ_2 can be represented by the segment t_1, t_2 in the parameter space. The management of all threshold-crossing time intervals of a time series can be efficiently handled, as follows: We first identify all groups of *tct*-intervals which start and end at the same time series segment. Then, each group is represented by one segment in the parameter space, which can be efficiently organized by means of a spatial index structure, e.g. the R*-tree. At query time, the time interval plane coordinates of the threshold-crossing time intervals corresponding to the query threshold τ_q can be easily determined by computing the intersection of all segments of the parameter space with the time interval plane P of threshold τ_q.

4.3 Trapezoid Decomposition of Time Series

The set of all time intervals which start and end at the same time series segment can be described by a single trapezoid whose left and right bounds are each congruent with one single time series segment. Let $s_l = ((x_{l1}, t_{l1}), (x_{l2}, t_{l2}))$ denote the segment of the left bound and $s_r = ((x_{r1}, t_{r1}), (x_{r2}, t_{r2}))$ denote the segment of the right bound. The top-bottom bounds correspond to the two threshold-crossing time intervals $tct_{\tau_{top}}$ and $tct_{\tau_{bottom}}$ whose threshold values are

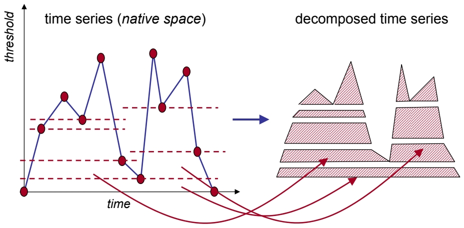

Fig. 4. Time Series Decomposition

computed as follows:

$$\tau_{top} = min(max(x_{l1}, x_{l2}), max(x_{r1}, x_{r2}));$$

$$\tau_{bottom} = max(min(x_{l1}, x_{l2}), min(x_{r1}, x_{r2}));$$

For our decomposition algorithm we can use the following property

Lemma 1. *Threshold-crossing time intervals always start at increasing time series segments (positive segment slope) and end at decreasing time series segments (negative segment slope).*

Proof. Due to Definition 1, all values of X within the threshold-crossing time interval $tct_\tau(X)$ are greater than the corresponding threshold value τ. Let us assume that the time series segment s_l which lower-bounds the time interval at time t_l has a negative slope. Then, all $x(t)$ on s_l with $t > t_l$ are lower than τ which contradicts the definition of threshold-crossing time intervals. The validity of Lemma 1 w.r.t. the right bounding segment can be shown analogously.

Let us note that time series objects can be considered as half-open unimonotone polygons in the time-amplitude plane. In the area of computational geometry there are known several sweep-line based polygon-to-trapezoid decomposition algorithms [14] which can be processed in $O(n \cdot logn)$ time w.r.t. the number of vertices. For this work we adopted one of these decomposition algorithms. Since the time series values are chronologically ordered, our decomposition algorithm can be processed in linear time w.r.t. the length of the sequence.

Figure 4 shows an example of how a time series is decomposed into the set of trapezoids. This algorithm works similar to polygon-to-trapezoid decomposition algorithms known from the area of computational geometry. As we can assume that the time series consist of chronologically ordered pairs (x, t), our decomposition algorithm can be performed in linear time (linear w.r.t. the length of the time series).

4.4 Indexing Segments in the Parameter Space

We apply the R*-tree for the efficient management of the three-dimensional segments representing the time series objects in the parameter space. As the R*-tree index can only manage rectangles, we represent the 3-dimensional segments by rectangles where the segments correspond to one of the diagonals of the rectangles.

For all trapezoids which result from the time series decomposition, the lower bound time interval contains the upper bound time interval. Furthermore, intervals which are contained in another interval are located in the lower-right area of this interval representation in the time interval plane. Consequently, the locations of the segments within the rectangles in the parameter space are fixed. Therefore, in the parameter space the bounds of the rectangle which represents a segment suffice to uniquely identify the covered segment. Let $((x_l, y_l, z_l), (x_u, y_u, z_u))$ be the coordinates of a rectangle in the parameter space, then the coordinates of the corresponding segment are $((x_l, y_u, z_l), (x_u, y_l, z_u))$.

5 Query Algorithm

The query consists of a query time series Q and a query threshold τ (cf. Definition 4). The first step of the query process is to determine the threshold-crossing time interval sequence $TCT_\tau(Q)$. Obviously, this can be done by one single scan through the query object Q. Next, we have to find those time series objects from the database which are most τ-similar to Q according to Definition 3.

5.1 Preliminaries

In this section, we assume that Q denotes the query time series which is represented by its threshold-crossing time interval sequence $S_Q = TCT_\tau(Q)$. Furthermore, $S_X = v_1, .., v_n$ denotes the threshold-crossing time interval sequence $TCT_\tau(X)$ from any database time series X. Since the similarity query is performed in the parameter space (or time interval plane for a certain threshold τ), S_X denotes a set[1] of two-dimensional points.

5.2 Computation of the τ-Similarity

At first, we will consider the computation of the τ-similarities between time series objects in the time interval plane. As mentioned above, the threshold-crossing time interval sequence of a time series object corresponds to a set of points in the time interval plane. In the following, the point set of a time series denotes the time interval plane point representation which corresponds to the threshold-crossing time interval sequence of the time series object.

Given a threshold-crossing time interval, the most similar threshold-crossing time interval in the time space (native space) (w.r.t. Definition 2) corresponds to the nearest-neighbor in the time interval plane.

[1] In our approach it does not make any difference whether $S_X = TCT_\tau(X)$ denotes a sequence or a set of intervals or points, thus for simplicity we consider S_X as a set of intervals or points.

Definition 5 (k-Nearest Neighbor). *Let q be a point in the time interval plane and $S_X = \{v_1, ..., v_n\}$ be a set of points in the time interval plane. The* k-nearest-neighbor $NN_{k,S_X}(q)$ $(k < n)$ *of q in the set S_X is defined as follows:*

$$v = NN_{k,S_X}(q) \in S_X \Leftrightarrow$$

$$\forall v' \in S_X - \{NN_{l,S_X}(q) : l \leq k\} : d_{int}(q, v) \leq d_{int}(q, v').$$

The distance $d_{int}(q, NN_{k,S_X}(q))$ is called k-nearest-neighbor distance. For $k = 1$, we simply call $NN_{1,S_X}(q) \equiv NN_X(q)$ the nearest-neighbor of q in S_X. $NN_{l,\cdot}(q)$ denotes the overall k-nearest neighbor of q, i.e. $NN_{l,\cdot}(q) = NN_{l,\bigcup_{X \in DB} X}(q)$. The set $k - NN_X(q) = \{NN_{l,S_X}(q) : \forall l \leq k\}$ is called k-nearest-neighbors of q.

In the time interval plane, the τ-similarity between two time series objects Q and X can be determined by computing for all points of S_Q the nearest neighbor points in S_X and, *vice versa*, for all points in S_X the nearest neighbor points in S_Q.

5.3 Efficient Query Processing

Let us assume that we are given any query threshold τ and the point set of the query object Q in the time interval plane of τ. A straightforward approach for the query process would be the following: First, we identify all parameter space segments of the database objects which intersect the time interval plane of threshold τ. Then we determine the time interval plane point sets of all database objects by computing the intersection between the parameter space segments and the plane of τ. For each database object, we compute the τ-similarity to the query object. Finally, we report the object having the smallest τ-distance to Q. Obviously, this is not a very efficient method since the respective parameter space segments of all time series objects have to be accessed. We can achieve a better query performance by using an R*-Tree index on the parameter space segments to filter out those segments which cannot satisfy the query. For this purpose, we require a lower bound criterion for the τ-distance between two objects.

Lower Distance Bound In the following, we will introduce a lower bound criterion for the threshold-distance d_{TS} on the basis of partial distance computations between the query object and the database objects. This lower bound criterion enables the detection of false candidates very early, i.e. only partial information of the false candidates suffices to prune this object from the candidate list. The amount of information necessary for the pruning of an object depends on the location of the query object and the other candidates.

Let $S_Q = \{q_1, ..., q_n\}$ be the point set corresponding to the query object and $S_X = \{v_1, ..., v_m\}$ be the point set of any object X from the database. Furthermore, we reformulate the τ-distance $d_{TS}(S_Q, S_X)$ between S_Q and S_X of Definition 3 as follows:

$$d_{TS}(S_Q, S_X) = \frac{1}{|S_Q|} \cdot D_1(S_Q, S_X) + \frac{1}{|S_X|} \cdot D_2(S_Q, S_X),$$

where $D_1(S_Q, S_X) = \sum_{i=1..n} d_{int}(q_i, NN_X(q_i))$
and $D_2(S_Q, S_X) = \sum_{i=1..m} d_{int}(v_i, NN_Q(v_i))$.

In the following, we use two auxiliary variables $K_l(q_i)$ and $\bar{K}_l(S_Q)$ which help to distinguish two classes of our objects. $K_l(q_i) \subseteq DB$ denotes the set of all objects S_X which has at least one entity $x \in S_X$ within the set $k-NN_X(q_i)$. Furthermore, $\bar{K}_l(S_Q) \subseteq DB$ denotes the set of all objects which are not in any set $K_l(q_i)$ i.e. $\bar{K}_l(S_Q) = DB - (\bigcup_{i=1..n} K_l(q_i))$.

Lemma 2. *The following inequality holds for any object $S_X \in \bar{K}_l(S_Q)$:*

$$D_1(S_Q, S_X) \geq \sum_{i=1..n} d_{int}(q_i, NN_{l,\cdot}(q_i)).$$

Proof. According to Definition 5 the following statement holds:

$$\forall i \in \{1, .., n\} : d_{int}(q_i, NN_{l,\cdot}(q_i)) \leq d_{int}(q_i, NN_X(q_i)).$$

Therefore,

$$\sum_{i=1..n} d_{int}(q_i, NN_{l,\cdot}(q_i)) \leq \sum_{i=1..n} d_{int}(q_i, NN_X(q_i)) = D_1(S_Q, S_X).$$

The following lemma is a generalization of Lemma 2 and defines a lower bound of $D_1(S_Q, S_X)$ for all database objects $S_X \in DB$ for any $l \in \mathbb{N}$.

Lemma 3. *Let $S_X \in DB$ be any database object and let S_Q be the query object. The distance $D_1(S_Q, S_X)$ can be estimated by the following formula:*

$$d_{min}(S_Q, S_X) = \frac{1}{n} \sum_{i=1..n} \left\{ \begin{array}{l} d_{int}(q_i, NN_X(q_i)),\ \textit{if}\ S_X \in K_l(q_i) \\ d_{int}(q_i, NN_{l,\cdot}(q_i)),\ \textit{else} \end{array} \right\} \leq D_1(S_Q, S_X).$$

Proof. Let $S_X \in DB$ be any database object and S_Q be the query object. According to Definition 5 the following holds:

$$d_{int}(q_i, NN_{l,\cdot}(q_i)) \leq d_{int}(q_i, NN_X(q_i)) \Leftrightarrow X \notin K_l(q_i).$$

Consequently, $d_{min}(Q, X) \leq \frac{1}{n} \sum_{i=1..n} d_{int}(q_i, NN_X(q_i)) = D_1(S_Q, S_X)$.

Pruning Strategy By iteratively computing the l-nearest neighbors $NN_{l,\cdot}(q)$ for all $q \in S_Q$ with increasing $l \in \mathbb{N}$, we can determine the lower bound distances for all objects. The maximal lower bound distance $d_{min}(S_Q, S_X)$ of an object S_X has been achieved as soon as $S_X \in K_l(q_i) : \forall i \in 1, ..., n$. Then, we refine the distance $d_{TS}(S_Q, S_X)$ by accessing the complete object S_X in order to compute the distance $D_2(S_Q, S_X)$. The resulting distance $d_{TS}(S_Q, S_X)$ is then used as new pruning distance d_{prun} for the remaining query process. All objects $Y \in DB - \{X\}$ whose current lower bound distance $d_{min}(S_Q, S_Y)$ exceeds d_{prun} can be omitted from the remaining search steps. The search proceeds by continuing the iterative computations of the next nearest neighbors $NN_{l+1,\cdot}$.

Let S_X be the object with the lowest exact distance to S_Q, i.e. $d_{prun} = d_{TS}(S_Q, S_X)$. The pruning distance can be updated as soon as the next object S_Y which has to be refined is found. In doing so, we have to consider two cases:

```
threshold-query($S_Q$, $DB$, $\tau$) {
  nn := ARRAY[1..—$S_Q$—]; /*array of current nn-objects*/
  $d_{min} - tab$ := LIST of point ARRAY[1..$|S_Q|$]; /*$d_{min}$ table*/
  $obj_{best}$ := null;
  $d_{prune} := +\infty$
  $k$ := 0;
  LOOP
    $k := k + 1$;
    $nn$ = fetch-next-nn($S_Q$,$DB$,$\tau$,$d_{prune}$);
    $d_{min} - tab$.update($nn$);
    if (($o := d_{min} - tab$.object_complete()) ! = null) then {
      load complete object o and compute $d_{TS}(S_Q, o)$; /*refinement-step*/
      if ($d_{TS}(S_Q, o) \geq d_{prune}$) then {
        $obj_{best} := o$;
        $d_{prune} := d_{TS}(S_Q, o)$;
      } else { remove $o$ from the candidate list in $d_{min} - tab$; }}
    for all objects $obj \in d_{min} - tab$ do {
      if ($D_1(S_Q, obj) \geq d_{prune}$) then {
        remove $obj$ from the candidate list in $d_{min} - tab$; }}
    if ($\sum_{q_i \in S_Q} NN_{k,.}(q_i) \geq d_{prune}$) then {
      report $o_{best}$;
      break; }
  end LOOP; }
```

Fig. 5. The threshold query algorithm.

case 1: $d_{TS}(S_Q, S_Y) \geq d_{prune}$ → remove object S_Y from the candidate set,
case 2: $d_{TS}(S_Q, S_Y) < d_{prune}$ → set $d_{prune} := d_{TS}(S_Q, S_Y)$ and remove object S_X from the candidate set.

The search algorithm terminates as soon as all object candidates, except for the best one, have been pruned.

5.4 Query Algorithm

The query algorithm is depicted in Figure 5. The function *threshold-query(S_Q,DB,τ)* computes for a given query object S_Q the database object obj_{best} having the smallest τ-distance $d_{TS}(S_Q, S_X)$. The function *fetch-next-nn(S_Q,DB)* is an iterator function which retrieves the next nearest neighbor for each $q_i \in S_Q$ in each iteration. The nearest neighbors can be efficiently computed by applying the nearest neighbor ranking method as proposed in [15]. Thereby, we maintain for each $q \in S_Q$ a priority queue, each storing the accessed R*-tree nodes in ascending order of their distances to the corresponding query point q.

In this section, we treated the objects as sets of points in the time interval plane. In fact, the database objects are organized within the three-dimensional parameter space (cf. Section 4.4). For the distance computation between the query point $q = (l_i, u_i, \tau)$ and an R*-tree rectangle $r = ((x_l, y_l, z_l), (x_u, y_u, z_u))$, we consider the horizontal distance at threshold τ only, i.e. $d_{int}(q_i, r) = d_{int}((l_i, u_i), ((x_l, y_l), (x_u, y_u)))$.

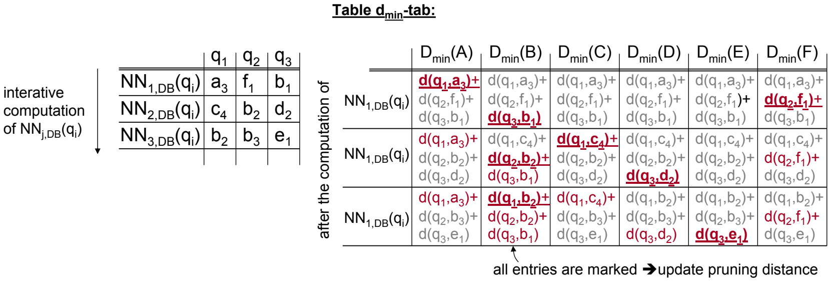

Table d_{min}-tab:

interative computation of $NN_{j,DB}(q_i)$

	q_1	q_2	q_3
$NN_{1,DB}(q_i)$	a_3	f_1	b_1
$NN_{2,DB}(q_i)$	c_4	b_2	d_2
$NN_{3,DB}(q_i)$	b_2	b_3	e_1

after the computation of

	$D_{min}(A)$	$D_{min}(B)$	$D_{min}(C)$	$D_{min}(D)$	$D_{min}(E)$	$D_{min}(F)$
$NN_{1,DB}(q_i)$	**$d(q_1,a_3)+$** $d(q_2,f_1)+$ $d(q_3,b_1)$	$d(q_1,a_3)+$ $d(q_2,f_1)+$ **$d(q_3,b_1)$**	$d(q_1,a_3)+$ $d(q_2,f_1)+$ $d(q_3,b_1)$	$d(q_1,a_3)+$ $d(q_2,f_1)+$ $d(q_3,b_1)$	$d(q_1,a_3)+$ $d(q_2,f_1)+$ $d(q_3,b_1)$	$d(q_1,a_3)+$ **$d(q_2,f_1)+$** $d(q_3,b_1)$
$NN_{1,DB}(q_i)$	$d(q_1,a_3)+$ $d(q_2,b_2)+$ $d(q_3,d_2)$	$d(q_1,c_4)+$ **$d(q_2,b_2)+$** $d(q_3,b_1)$	**$d(q_1,c_4)+$** $d(q_2,b_2)+$ $d(q_3,d_2)$	$d(q_1,c_4)+$ $d(q_2,b_2)+$ **$d(q_3,d_2)$**	$d(q_1,c_4)+$ $d(q_2,b_2)+$ $d(q_3,d_2)$	$d(q_1,c_4)+$ $d(q_2,f_1)+$ $d(q_3,d_2)$
$NN_{1,DB}(q_i)$	$d(q_1,a_3)+$ $d(q_2,b_3)+$ $d(q_3,e_1)$	**$d(q_1,b_2)+$** $d(q_2,b_2)+$ $d(q_3,b_1)$	$d(q_1,c_4)+$ $d(q_2,b_3)+$ $d(q_3,e_1)$	$d(q_1,b_2)+$ $d(q_2,b_3)+$ $d(q_3,d_2)$	$d(q_1,b_2)+$ $d(q_2,b_3)+$ **$d(q_3,e_1)$**	$d(q_1,b_2)+$ $d(q_2,f_1)+$ $d(q_3,e_1)$

all entries are marked ➔update pruning distance

Fig. 6. Example of the query processing

The basic functionality of the query algorithm can be explained by the following example which is depicted in Figure 6. In our example, the query consists of three time interval plane points $S_Q = \{q_1, q_2, q_3\}$. The upper table shows the results of the first three states of the incremental nearest-neighbor queries $NN_{1,\cdot}(q_i)$, $NN_{2,\cdot}(q_i)$ and $NN_{3,\cdot}(q_i)$. The state of the corresponding d_{min}-table after each iteration is shown in the table below. The first iteration retrieves the points a_3, f_1 and b_1 of the time series objects A, F, and B, respectively. Consequently, the threshold-distance between q and all objects $S_X \in DB$ can be restricted by the lower bound $d_{min} = \frac{1}{3}(d(q_1, a_3) + d(q_2, f_1) + d(q_3, b_1))$. Next, we insert the actual nearest neighbor distances into the d_{min}-table and mark the corresponding entries (marked entries are underlined in the figure). Let us note, that all unmarked distance entries correspond to the currently retrieved nearest neighbor distances, and thus, need not to be stored for each object separately. After the third query iteration, all nearest neighbor counterparts from S_Q to S_B are found. Consequently, we can update the pruning distance by computing the exact τ-similarity $d_{prune} = d_{TS}(S_Q, S_B)$. We can now remove the column $D_{min}(B)$ from the d_{min}-table.

The runtime complexity of our threshold query algorithm is O$(n_q \cdot n_k \cdot log n_p)$, where n_q denotes the size of the threshold-crossing time interval sequence S_Q, n_k denotes the number of nearest-neighbor search iterations and n_p denotes the overall number of segments in the parameter space. In the experiments (cf. Section 6) we will show that in average n_q is very very small in comparison to the length of the time sequences. Furthermore, we will show that the number of required nearest-neighbor query iterations n_k is very small, i.e. the query process terminates very early. The number n_p of segments in the parameter space is quite similar to the sum n_s of length of all time sequences in the database, but it is slightly smaller than n_s which is also shown in the experiments.

6 Experimental Evaluation

In this section, we present the results of a large number of experiments performed on a selection of different time series datasets. In particular, we compared the efficiency of our proposed approach (in the following denoted by 'R_{Par}') for answering threshold queries using one of the following techniques.

The first competing approach works on native time series. At query time the threshold-crossing time intervals (TCT) are computed for the query threshold and afterwards the distance between the query time series and each database object can be derived. In the following this method will be denoted by 'Seq_{Nat}' as it corresponds to a sequential processing of the native data.

The second competitor works on the parameter space rather than on the native data. It stores all TCTs without using any index structures. As this storage leads to a sequential scan over the elements of the parameter space we will refer to this technique as the 'Seq_{Par}' method.

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8GB RAM. We used a disk with a transfer rate of 100 MB/s, a seek time of 3 ms and a latency delay of 2 ms. Performance is presented in terms of the elapsed time including I/O and CPU-time.

6.1 Datasets

We used several real-world and synthetic datasets for our evaluation, one audio dataset and two scientific datasets. The audio dataset contains time sequences expressing the temporal behavior of the energy and frequency in music sequences. It contains up to 700000 time series objects with a length of up to 300 values per sequence. If not otherwise stated, the database size was set to 50000 objects and the length of the objects was set to 50. This dataset is used to evaluate the performance of our approach (cf. Section 6.2). In Section 6.3, we will show the effectiveness of threshold queries for the two scientific datasets. The scientific datasets are derived from two different applications: the analysis of environmental air pollution (cf. Application 2 in Section 1) and gene expression data analysis (cf. Application 3 in Section 1). The data on environmental air pollution is derived from the Bavarian State Office for Environmental Protection, Augsburg, Germany [2] and contains the daily measurements of 8 sensor stations distributed in and around the city of Munich from the year 2000 to 2004. One time series represents the measurement of one station at a given day containing 48 values for one of 10 different parameters such as temperature, ozone concentration, etc. A typical time series of this dataset contains 48 measurements of station S during day D of parameter P. The gene expression data from [16] contains the expression level of approximately 6,000 genes measured at only 24 different time slots.

6.2 Performance Results

To obtain more reliable and significant results, in the following experiments we used 5 randomly chosen query objects. Furthermore, these query objects were used in conjunction with 5 different thresholds, so that we obtained 25 different threshold queries. The presented results are the average results of these queries.

First, we performed threshold queries against database instances of different sizes to measure the influence of the database size to the overall query time. The

[2] www.bayern.de/lfu

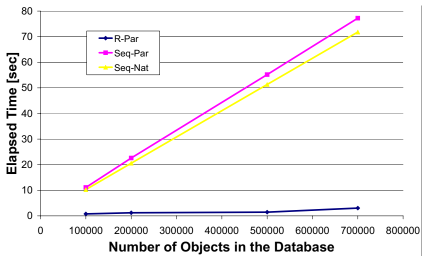

Fig. 7. Scalability against database size.

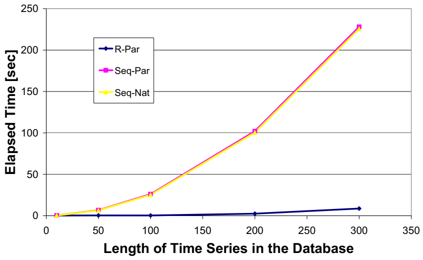

Fig. 8. Scalability against time series length.

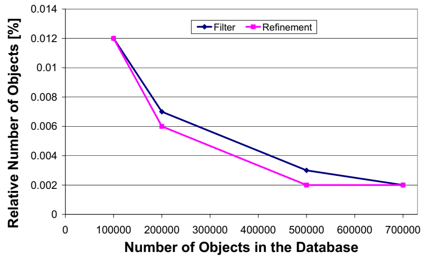

Fig. 9. Pruning power for varying database size.

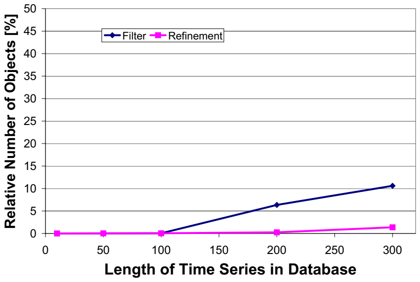

Fig. 10. Pruning power for varying time series length.

elements of the databases are time series of fixed length $l = 50$. Figure 7 exhibits the performance results for each database. It is shown that the performance of both approaches Seq_{Nat} and Seq_{Par} significantly decreases with increasing database size, whereas our approach scales very well even for large databases. Second, we explored the impact of the length of the query object and the time series in the database. The results are shown in Figure 8. Again, our technique outperforms the competing approaches whose cost increase very fast due to the expensive distance computations. In contrast our approach is hardly influenced by the size of the time series objects.

In the next experiment we present the speed-up of the query process caused by our pruning strategy. We measured the considered number of result candidates during the query processes and the number of finally refined objects. Figure 9 and Figure 10 show the results relatively to the database size and object size. Only a very small portion of the candidates has to be refined to report the result. An interesting point is that very large time series lead to lower pruning power than smaller time series.

Furthermore, we examined the number of nearest-neighbor search iterations which were required for the query process for varying length of the time series and varying size of the database. We observed, that the number of iterations was between 5 and 62. The number of iterations increases linear to the length of the time series and remains nearly constant w.r.t. the database size. Nevertheless, only a few iterations are required to report the result.

6.3 Results on Scientific Datasets

The results on the air pollution dataset were very useful. We performed 10-nearest neighbor threshold queries with randomly chosen query objects. Interestingly, when we choose time series as query objects, that were derived from rural sensor stations representing particulate matter parameters (M_{10}), we obtained only time series representing the same parameters measured also at rural stations. This confirms that the pollution by particle components in the city differs considerably from the pollution in rural regions. A second interesting result was produced when we used M_{10} time series of working days as queries. The resulting time series were also derived from working days representing M_{10} values.

The results on the gene expression dataset were also very interesting. The task was to find the most similar gene with $\tau = 0$ to a given query gene. The intuition is to find a gene that is functionally related to the query gene. We posed several randomized queries to this dataset with $\tau = 0$ and evaluated the results w.r.t. biological interestingness using the SGD database [3]. Indeed, we retrieved functionally related genes for most of the query genes. For example, for query gene CDC25 we obtained the gene CIK3. Both genes play an important role during the mitotic cell cycle. For the query gene DOM34 and MRPL17 we obtained two genes that are not yet labeled (ORF-names: YOR182C and YGR220C, respectively). However all four genes are participating in the protein biosynthesis. In particular, threshold queries can be used to predict the function of genes whose biological role is not resolved yet.

To sum up, the results on the real-world datasets suggest the practical relevance of threshold queries for important real-world applications.

7 Conclusions

In this paper, we motivated and proposed a novel query type on time series databases called *threshold query*. Given a query object Q and a threshold τ, a *threshold query* returns time series in a database that exhibit the most similar threshold-crossing time interval sequence. The threshold-crossing time interval sequence of a time series represents the interval sequence of elements that have a value above the threshold τ. We mentioned several practical application domains for such a query type. In addition, we presented a novel approach for managing time series data to efficiently support such threshold queries. Furthermore, we developed a scalable algorithm to answer *threshold queries* for arbitrary thresholds τ. A broad experimental evaluation demonstrates the importance of the new query type for several applications and shows the scalability of our proposed algorithms in comparison to straightforward approaches.

For future work, we plan to develop suitable approximations which represent the novel time series data in a compressed form in order to apply efficient filter steps during the query process. Furthermore, we plan to extend our approaches to data mining tasks, such as clustering.

[3] http://www.yeastgenome.org/

References

1. Keogh, E., Chakrabati, K., Mehrotra, S., Pazzani, M.: ”Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’01), Santa Barbara, CA. (2001)
2. Guttman, A.: “R-Trees: A Dynamic Index Structure for Spatial Searching”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’84). (1984)
3. Agrawal, R., Faloutsos, C., Swami, A.: ”Efficient Similarity Search in Sequence Databases”. In: Proc. 4th Conf. on Foundations of Data Organization and Algorithms. (1993)
4. Chan, K., Fu, W.: ”Efficient Time Series Matching by Wavelets”. In: Proc. 15th Int. Conf. on Data Engineering (ICDE’99), Sydney, Australia. (1999)
5. Yi, B.K., Faloutsos, C.: ”Fast Time Sequence Indexing for Arbitrary Lp Norms”. In: Proc. 26th Int. Conf. on Very Large Databases (VLDB’00), Cairo, Egypt. (2000)
6. Korn, F., Jagadish, H., Faloutsos, C.: ”Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’97), Tucson, AZ. (1997)
7. Cai, Y., Ng, R.: ”Index Spatio-Temporal Trajectories with Chebyshev Polynomials”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’04), Paris, France). (2004)
8. Faloutsos, C., Ranganathan, M., Maolopoulos, Y.: ”Fast Subsequence Matching in Time-series Databases”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’94), Minneapolis, MN. (1994)
9. Ratanamahatana, C.A., Keogh, E., Bagnall, A.J., Lonardi, S.: ”A Novel Bit Level Time Series Representation with Implication for Similarity Search and Clustering”. In: Proc. 9th Pacific-Asian Int. Conf. on Knowledge Discovery and Data Mining (PAKDD’05), Hanoi, Vietnam. (2005)
10. Eiter, T., Mannila, H.: ”Distance Measure for Point Sets and Their Computation”. In: Acta Informatica, 34. (1997) 103–133
11. Kriegel, H.P., Pötke, M., Seidl, T.: ”Object-Relational Indexing for General Interval Relationships”. In: Proc. Symposium on Spatial and Temporal Databases (SSTD’01), Redondo Beach, CA. (2001)
12. Gaede, V., Günther, O.: “Multidimensional Access Methods”. Computing Surveys **30** (1984)
13. Beckmann, N., Kriegel, H.P., Seeger, B., Schneider, R.: ”The R*-tree: An Efficient and Robust Access Method for Points and Rectangles”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’90), Atlantic City, NJ. (1990)
14. Fournier, A., Moniwno, D.Y.: ”Triangulating simple polygons and equivalent problems”. In: ACM Trans. Graph., 3, 2. (1984) 153–174
15. Hjaltason, G., Samet, H.: “Ranking in Spatial Databases”. In: Proc. Int. Symp. on Large Spatial Databases (SSD’95), Portland, OR. (1995)
16. Spellman, P., Sherlock, G., Zhang, M., Iyer, V., Anders, K., Eisen, M., Brown, P., Botstein, D., Futcher, B.: ”Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization.”. Molecular Biolology of the Cell **9** (1998) 3273–3297