Efficient Processing of Narrow Range Queries in the R-Tree

Michal Krátký[1], Václav Snášel[1], Jaroslav Pokorný[2], Pavel Zezula[3], Tomáš Skopal[1]

[1]VŠB–Technical University of Ostrava
17. listopadu 15, Ostrava-Poruba
{michal.kratky,vaclav.snasel,tomas.skopal}@vsb.cz

[2]Charles University Prague
pokorny@ksi.ms.mff.cuni.cz

[3]Masaryk University Brno
zezula@informatics.muni.cz

Czech Republic

Abstract

Multi-dimensional data structures are applied in many real index applications, i.e. data mining, indexing multimedia data, indexing non-structured text documents and so on. Many index structures and algorithms have been proposed. There are two major approaches to multi-dimensional indexing. These are, data structures to indexing metric and vector spaces. The R-tree, R*-tree, and UB-tree are representatives of the vector data structures. These data structures provide efficient processing for many types of queries, i.e. point queries, range queries and so on. As far as the vector data structures are concerned the range query retrieves all points in defined hyper box in an n-dimensional space. The narrow range query is a significant type of the range query. Its processing is inefficient in the vector data structures. Moreover, the efficiency decreases from increase dimension of an indexed space. We depict an application of the signature for more efficient processing of narrow range queries. The approach puts the signature into the R-tree but native functionalities are preserved, i.e. the range query algorithm for general range query. The novel data structure is called the Signature R-tree. This data structure is more resistant to the curse of dimensionality.

ARG Technical Report, ARG-TR-01-2004, Amphora Research Group, Department of Computer Science, VSB–Technical University of Ostrava, Czech Republic, *http://www.cs.vsb.cz/arg*
© 2004

1 Introduction

During the last decade, multimedia databases have become increasingly important in many application areas such as medicine, CAD, geography, and molecular biology. An important research issue in the field of multimedia databases is the content-based retrieval of similar multimedia objects such as images, text, and videos. However, in contrast to searching data in a relational database, a content-based retrieval requires the search for similar objects as a basic function of the database system. Most of the approaches addressing the similarity search use a so-called *feature transformation* which transforms important properties of multimedia objects into *high-dimensional points* (*feature vectors*). Thus, the similarity search is transformed into a search of points in the feature space that are close to a given query point in the high-dimensional feature space. Query processing in high-dimensional spaces has therefore been a very prominent research area over the last few years. A number of new index structures and algorithms have been proposed.

Managing multi-dimensional data is needed in many application domains, from CAD, VLSI and geographical databases to multimedia and time series management systems. In particular, indexing spatial data is of foremost importance and has been quite well researched as it is presented in excellent surveys [15], [5] and, recently, [22]. There are a lot of additional applications of *multi-dimensional data structures* [24] e.g., data mining [18], indexing terms [11, 21], XML documents [16, 20], text documents [25], and images [8].

We can divide these data structures into two groups [30], data structures for indexing vector spaces and metric spaces, respectively. The first group includes, for example, n-dimensional B-tree [14], R-tree [17], R*-tree [2], X-tree [4], UB-tree [1], and BUB-tree [13]. The second group includes M-tree [8], for example. A multi-dimensional data structure supports either one or both of the following query types [30]:

- *range/window queries:* "find all objects whose attribute values fall within certain ranges",
- *similarity queries:*
  - *similarity range queries:* "find all objects in the database which are within a given distance from a given object",
  - *k-nearest neighbour (k-NN) queries:* "find the k-most similar object in the database with respect to a given object".

Of course, the *point query* can be considered as a special type of the queries. Now, the range query of a vector data structure is defined.

Definition 1 (Range query).

Let Ω be an n-dimensional discrete space, $\Omega = D^n$, $D = \{0, 1, \ldots, 2^{l_D} - 1\}$, and points (tuples) $T^1, T^2, \ldots, T^m \in \Omega$. $T^i = (t_1, t_2, \ldots, t_n)$, l_D is the chosen length of a binary representation of a number t_i from domain D. The *range query RQ* is defined by a *query hyper box* (*query window*) QB which is determined by two points $QL = (ql_1, \ldots, ql_n)$ and $QH = (qh_1, \ldots, qh_n)$, QL and $QH \in \Omega$, ql_i and $qh_i \in D$, where $\forall i \in \{1, \ldots, n\} : ql_i \leq qh_i$. This range query retrieves all points $T^j(t_1, t_2, \ldots, t_n)$ in the set $T^1, T^2, \ldots, T^m$ such as $\forall i : ql_i \leq t_i \leq qh_i$. ■

The range query may be written as pseudo SQL statement:

```
SELECT * FROM T
   WHERE $ql_1 \leq t_1 \leq qh_1$ AND ... AND $ql_n \leq t_n \leq qh_n$
```

The *narrow range query* is a significant type of the range query.

Definition 2 (Narrow range query).

Let Ω be an n-dimensional discrete space, $\Omega = D^n$. The query hyper box is defined by two points $QL = (ql_1, \ldots, ql_n)$ and $QH = (qh_1, \ldots, qh_n)$, where $\forall i : ql_i \leq qh_i$. Let ψ and ϕ be constants: $min(D) \leq \psi \ll \phi \leq max(D)$. The range query is called the narrow one if:

1. $\forall i : qh_i - ql_i \leq \psi \vee qh_i - ql_i \geq \phi$.
2. Let n_ψ and n_ϕ be the number of dimensions for which formulas $qh_i - ql_i \leq \psi$ and $qh_i - ql_i \geq \phi$, respectively, hold. Furthermore, in the case of the narrow range query it holds $1 < n_\psi < n \wedge 1 < n_\phi < n$. ■

From the first condition the equation turns out to be $n_\phi + n_\psi = n$. In Figure 1 we see examples of query boxes for the narrow range queries in spaces with the dimensions $n = 2$ and $n = 3$.

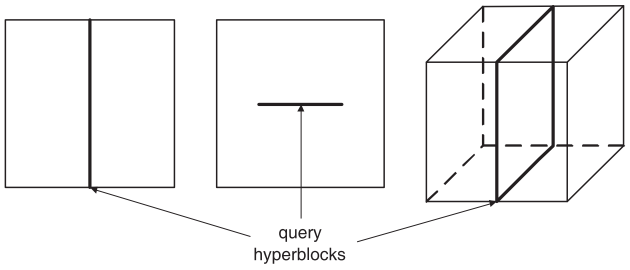

Figure 1: Examples of the narrow range queries in spaces with the dimensions $n = 2$ and $n = 3$.

As far as structures like n-dimensional B-tree, R-tree, and BUB-tree are concerned, processing the narrow range query is inefficient. The efficiency decreases with rising dimension – *curse of dimensionality* [30] takes place. An efficient solution of this problem does not seem to exist. In this work we describe an application of the *signature* [23] for effective processing of a narrow range query over point data index. This approach enriches the existing multi-dimensional data structure but native functionality is preserved. The novel data structure is resistant to the curse of dimensionality. This feature is grounded in better preservation of data distribution achieved by a signature method over multi-dimensional index. Due to the fact that the R-tree is well-known data structure used in many recent database management systems (DBMS), we apply the signature extension to the R-tree.

In Section 2 we review existing multi-dimensional indexes based on the R-tree. Section 3 briefly reviews existing signature methods. In Section 5, we present the newly proposed variant of R-tree with an application of the signature. This variant of the R-tree enables efficient processing of the narrow range query. This novel data structure is called the *Signature R-tree.* In Section 6, we put forward results of experiments. Finally, we conclude with a summary of contributions and a discussion about future work.

2 R-tree and its variants

Since 1984 when Guttman proposed his method [17], R-trees have become the most cited and most used as reference data structure in this area. As is required and expected by applications, they support usual point and range queries, and also some forms of spatial joins.

Another interesting query supported by R-trees, to some extent, is the *k-NN* query.

R-tree can be thought of as an extension of B-trees in a multi-dimensional space. It corresponds to a hierarchy of nested n-dimensional *minimum bounding boxes* (MBB). If $\mathcal{N}$ is an interior node, it contains couples of the form (R_i, P_i), where P_i is a pointer to a child of the node $\mathcal{N}$. If R is its MBB, then the boxes R_i corresponding to the children $\mathcal{N}_i$ of $\mathcal{N}$ are contained in R. Boxes at the same tree level may overlap. If $\mathcal{N}$ is a leaf node, it contains its couples of the form (R_i, O_i), so called *index records*, where R_i contains a spatial object O_i. Each node of the R-tree contains between m and M entries unless it is the root and corresponds to a disk page. Other properties of the R-tree include the following:

- Whenever the number of a node's children drops below m, the node is deleted and its descendants are distributed among the sibling nodes. The upper bound M depends on the size of the disk page.
- The root node has at least two entries, unless it is a leaf.
- The R-tree is height-balanced; that is, all leaves are at the same level. The height of an R-tree is at most $\lfloor \log_m(N) \rfloor - 1$ for N index records $(N > 1)$.

As a dynamic data structure, most attention of previous works on R-trees has been devoted to the split procedure during the adding of new index records into an R-tree. It significantly affects the index performance. Three split techniques (Linear, Quadratic, and Exponential) proposed in [17] are based on a heuristic optimization. The Quadratic algorithm has turned out to be the most effective and other improved versions of R-trees are based on this method. The algorithm uses the following strategy:

- Given a set of $M+1$ entries, each entry is assigned to one of the two produced nodes, according to the criterion of minimum area, i.e., the selected node is the one that will be enlarged the least in order to include the new entry.

Unfortunately, this criterion is taken for granted and not proved to be the best possible. The Quadratic algorithm tends to prefer the group with the largest size and higher population. In most cases this group will be least enlarged. Hence, there is a high chance it will need less area in order to accommodate the next entry, so it will be enlarged again. Over time, this will create a very uneven distribution, with most entries in one node. Also, when one of the groups becomes full, the rest of $M - m + 1$ entries are assigned to the second group without any geometric criteria. A minimum node capacity constraint also exists; thus a number of entries are assigned to the least populated node without any control at the end of the split procedure. This fact usually causes a significant overlap between the two nodes.

R-tree performance is usually measured with respect to the retrieval cost (in terms of disk accesses [24]) of queries. The majority of performance studies concerns point, range, and *k-NN* queries. Considering the R-tree performance, the concepts of node *coverage* and *overlap* between nodes are important. Obviously, an efficient R-tree search requires that both the overlap and coverage are minimized. Minimal coverage reduces the amount of dead area covered by R-tree nodes. The minimal overlap is even more critical than the minimal coverage; searching objects falling in the area of k overlapping nodes, up to k paths to the leaf nodes may have to be executed in such a way.

Variants of R-trees differ in the way they perform the split algorithm during insertions, i.e. which minimization criteria are used. Literature has identified a variety of criteria for the layout of keys on nodes that affect retrieval performance. These criteria are: minimal node area, minimal overlap between nodes, minimal node margins or maximized node utilization. It is impossible to optimize all of these parameters simultaneously. We will briefly put forward two well-known approaches to the R-tree optimization - R*-trees and R$^+$-trees. Authors of [22] put forward, in their recent exhaustive overview, another six variants.

The main feature of R*-trees [2] involves the node-splitting policy. Therefore, the R*-tree differs from the R-trees mainly in the insertion algorithm. Although original R-tree algorithms tried only to minimize the area covered by MBBs, the R*-tree algorithms also take the following objectives into account:

- The *overlap* between MBBs at the same (non-leaf) tree level should be minimized. The lesser overlap, the smaller the probability that one has to follow multiple search paths.
- *Perimeters* (*margins*) of MBBs should be minimized. For example, in 2D the preferred rectangle is the square, since this is the most compact rectangular representation.
- *Storage utilization* should be maximized. Nodes should store as many entries as possible so that the height of the tree is kept low.

According to the R*-tree split algorithm, the split axis is the one that minimizes a cost value S (S being equal to the sum of all margin values of the different distributions). Then the distribution which achieves minimum overlap-value is selected to be the final one along the chosen split axis. On the other hand, the distinction between the "minimum margin" criterion to select a split axis and the "minimum overlap" criterion to select a distribution along the split axis, followed by the R*-tree split algorithm, could cause the loss of a "good" distribution if, for example, that distribution belongs to the rejected axis. The design of the R*-tree also introduces a policy called *forced reinsert*: If a node overflows, it is not split in the right away. Moreover, p entries, $p > 0$, are removed from the node and reinserted into the tree. Authors of [2] suggest p should be about 30% of the maximal number of entries per page. Through all above mentioned techniques they reached performance improvements of up to 50% compared to the basic R-tree.

Clipping-based schemes do not allow any overlaps between bucket regions; they have to be mutually disjoint. A typical access method of this kind is the R^+-tree [26], a variant of the R-tree which allows no overlap between regions corresponding to nodes at the same tree level and an object can be stored in more than one leaf node. R^+-trees are considered to be one of the most efficient indexes for supporting point and range queries.

Other approaches to an improvement of original R-trees release some of their basic features. For example, the MBBs have been replaced by minimum bounding spheres or polygons. In [3] R^+-trees are extended to support *k-NN* queries. We do not mention the other ones as they do not have a direct impact on ideas presented in this paper. Special attention should be devoted to the use of signatures in connection with R-trees. The approach [9] offers an RS-tree that consists of an R-tree and an S-tree [10], i.e. a well-know hierarchical signature file. The main application of this data structure is an improvement of incremental *k-NN* query algorithm.

3 Signature methods

The signature file method has widely been advocated as an efficient access method to deal with many applications demanding a large volume of textual databases, such as libraries, office information, and medical information systems [6, 7]. Therefore, the signature file approach has become a well-known concept for implementing associative retrieval on data files kept in stable storage. Recently, the use of signature files was extended to support multimedia data, such as images, voice, and video [2]. Many recent DBMS support multimedia data and require a dynamic storage structure which performs not only retrieval operations, but also insertion, deletion, and update operations in an efficient manner. As a result, several dynamic signature files have been proposed, for example S-tree or Quick filter [31].

The *signature file* is an abstraction which acts as a filtering mechanism to reduce the number of block accesses and CPU time to execute a query. A signature is a bit string formed from the terms which are used to index a record in a data file. Signature files typically make use of the *superimposed coding* technique in order to create a record signature [12]. When we assume that a record consists of n terms, each term is converted into a bit string, called the *term signature*, using a hash function. The record signature is formed by superimposing (inclusive ORing) the n term signatures. The number of 1's in the signature S is called the *weight* $\gamma(S)$. To answer a query, we first examine the signature file rather than the data file, to immediately discard non-qualifying records. For this, a set of terms in a query is hashed to form a *query signature* in the same way used for the record signature. If the record signature contains 1's in the same position as the query signature (i.e. the query signature is included in the record signature), the record can be considered as a potential match. However, there can be a case where the record signature may qualify for a query signature, but the record itself does not satisfy the query. This is called the *false drop*.

4 Narrow range query processing in multi-dimensional data structures

In general, multi-dimensional data structures divide an n-dimensional space into sub-spaces (*regions*). In the case of the R-tree and (B)UB-tree, the tuples are clustered to MBBs and Z-regions, respectively. The index is made by hierarchies of the regions (so called super-regions). Consequently, the tuples of the region are stored in one leaf node. The inner nodes contain a definition of super-regions, MBBs in the case of R-tree again. An algorithm of range query filters the irrelevant tree node (regions), only leaf nodes intersected by the query box are searched.

Example 1 *(Reason of inefficiency processing of the narrow range query in R-tree).*

Let us take a 2-dimension space which contains points (4,1), (4,5), and (6,4). These points define MBB (4,1):(6,5) (see Figure 2) and they are stored on a single leaf node. Now, a range query is defined by the query box $(1,2) : (5,2)$. The region is intersected by the query box and it will be searched. Consequently, this region is relevant to the query box from the R-tree point of view, but it contains no point from the query box. ■

Definition 3 (intersect and relevant regions, relevant ratio).

Let RQ be the range query defined by the box HQ.

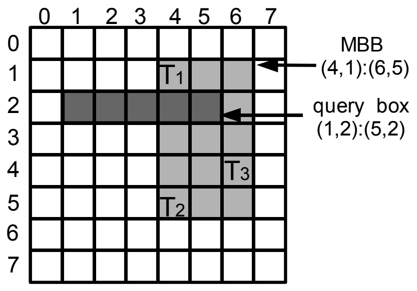

Figure 2: Points T_1, T_2 a T_3 in MBB $(4,1):(6,5)$ and the narrow range query $(1,2):(5,2)$.

Regions intersecting a query box during the processing of a range query are called *intersect regions* and regions containing at least one point of the query box are called *relevant regions*. We denote their number by N_I and N_R, respectively. The *relevance ratio* is $c_R = \frac{N_R}{N_I}$. ■

An experiments show (see Section 6), the ration c_R nears to zero and $c_R \ll 1$ for a narrow range query in the case of R-tree. The efficiency of the narrow range query processing is not optimal. A lot of irrelevant regions must be searched, therefore a lot of extra disk accesses must be performed.

Definition 4 (quality ratio of a range query algorithm).

Let us take a range query algorithm, which searches N_{RQ} regions for a query RQ. The *quality ration of a range query algorithm* is $c_Q = \frac{N_R}{N_{RQ}}$. ■

In optimal case, $c_Q = 1$. The value c_Q decreases sharply for increase dimension of indexed space in current multi-dimensional data structures. Consequently, the processing of a narrow range query is ineffective. Note, the number of inner nodes $\ll$ the number of leaf nodes (the number of regions) in the case of tree data structure. Since such ratios take hold of efficiency of a range query algorithm rather precisely.

A probability that the irrelevant region is matched decreases with downward region volume. The reducing of region volume is a way for an efficiency improvement of processing the narrow range query. We need to insert a piece of information into a data structure for the reducing of region volume and, consequently, for better filtration of irrelevant tree nodes (regions). The result of increasing c_Q is the decrease of disk access cost and data structure overhead. In this case we apply the *n-dimensional signature* as a piece of information. In this work we describe such extension of the R-tree and the novel data structure is called the *Signature R-tree.*

5 Efficient processing of narrow range queries in the R-tree

As far as the Signature R-tree is concerned the n-dimensional signature helps to filter irrelevant parts of an R-tree preferably during a narrow range query processing. The n-dimensional signature can be applied to various multi-dimensional data structures. Here, we put forward the extension of well known R-tree.

Definition 5 (n-dimensional signature).

Let Ω be an n-dimensional discrete space, $\Omega = D^n$, $|D| = 2^{l_D}$. Let us take a set of m points (tuples) $T^1, T^2, \ldots, T^m$, where $T^i = (t_1, t_2, \ldots, t_n)$, $T^i \in \Omega$, ${T^i}_j = t_j \in D$, $1 \leq i \leq m$, $1 \leq j \leq n$. Let F be a mapping creating a signature: $\{0,1\}^{l_D} \rightarrow \{0,1\}^{l_S}$. **$n$-dimensional signature** $S^n(T^1, T^2, \ldots, T^m) = (S_1, \ldots, S_n) = (F({T^1}_1)$ `OR` $\ldots$ `OR` $F({T^m}_1), \ldots, F({T^1}_n)$ `OR` $\ldots$ `OR` $F({T^m}_n))$, where S_i is a signature $\in \{0,1\}^{l_S}$, l_S is the length of the signature, $n \times l_S$ is the length of the n-dimensional signature. The weight of the n-dimensional signature $\gamma(S^n) = \sum_{k=1}^{n} \gamma(S_k)$, where $\gamma(S_k)$ is the weight of the signature S_k. ■

We can discover the absence of relevant points (the points belonging to a query box) using the `AND` operation for n-dimensional signature of a query and the n-dimensional signature of points in a region during the processing of a range query. Consequently, we apply the `AND` operation to elements of points how it is usual in signature methods.

Definition 6 (Range query processing with the n-dimensional signature).

Let us take the range query defined by two points of an n-dimensional space $QL = (ql_1, \ldots, ql_n)$ and $QH = (qh_1, \ldots, qh_n)$. Let us create the n-dimensional signature of the query box ${S^n}_{qb} = (S_{qb_1}, \ldots, S_{qb_n})$: if $ql_i = qh_i$, or $ql_i \neq qh_i$ and $qh_i - ql_i \leq \psi$, then $S_{qb_i} = F(ql_i) = F(qh_i)$ and $S_{qb_i} = F(ql_i)$ `OR` $F(ql_i + 1)$ `OR` $\ldots$ `OR` $F(qh_i)$, respectively. If $qh_i - ql_i \geq \phi$ then $S_{qb_i} = 2^{l_S} - 1$ (the number with only true bits). Let us take the n-dimensional signature $S^n = (S_1, \ldots, S_n)$ of points $T^1, T^2, \ldots, T^m$. The points generating the n-dimensional signature can belong to the query box if all partial signatures S_i and S_{qb_i}, $1 \leq i \leq n$, are matched by the `AND` operation. A partial signatures S_i and S_{qb_i} are matched if:

- for $ql_i = qh_i$ and $qh_i - ql_i \geq \phi$ it holds S_i `AND` $S_{qb_i} = S_{qb_i}$.
- for $ql_i \neq qh_i$ and $qh_i - ql_i \leq \psi$ it holds $\gamma(S_i$ `AND` $S_{qb_i}) \geq 1$.

Consequently, the n-dimensional signatures S^n and

${S^n}_{qb}$ are matched by the AND operation if all partial signatures S_i and S_{qb_i}, $1 \leq i \leq n$, are matched. ■

Of course, if S_{qb_i} contains only true bits, the operation AND can be omitted. If $\gamma(S_{qb_i}) \rightarrow l_S$ a probability of the `false drop` is close to one (see Chapter 5.2). Consequently, this algorithm is possible to apply only for small values of ψ.

Example 2 *(Usage of the n-dimensional signature for filtration of irrelevant tree pages).*

Let us express the creation and application of a simple n-dimensional signature for better filtration of irrelevant tree nodes. Let us take points from Example 1. The first coordinate of the n-dimensional signature contains superimposed first coordinates of the points: 4 (100) OR 4 (100) OR 6 (110). The second coordinate equals: 1 (001) OR 5 (101) OR 4 (100). In this way, the n-dimensional signature (110,101) is created. Since the second coordinates of both query box points contain the same values (the number 2) then all relevant points contain value 2 in the second coordinate. Consequently, the n-dimensional signature of the query hyper box is (111,010). The region (MBB (4,1):(6,5)) is recognized as irrelevant by the signature operation (111,010) AND (110,101). Since (010) AND (101) $\neq$ (010) then the region is irrelevant, in spite of the query box intersecting the region is searched during the narrow range query processing in the classical R-tree. ■

5.1 The Signature R-Tree

The Signature R-Tree is the R-tree data structure with added n-dimensional signature for better filtration of irrelevant tree nodes. A general structure of the Signature R-tree is presented in Figure 3. Leaf nodes include indexed tuples, which are clustered into regions – MBBs. The MBBs of leaf points can be hierachized to MBBs again and, in this way, super-regions are created. A definition of the regions and super-regions (two points in n-dimensional space) is stored in inner tree nodes. The n-dimensional signature is assigned to each region. The node's item with the definition of a super-region holds an n-dimensional signature, superimposed with signatures of the direct node's children. Consequently, such a tree contains two hierarchies, the hierarchy of MBBs and of n-dimensional signatures.

Operations of the R-tree are preserved, and of course we apply the n-dimensional signature for better filtration of irrelevant nodes upon processing a narrow range query. The signature helps to examine the intersect algorithm of which the node is/is not relevant to a user's query box. Note, the number of inner nodes $\ll$ the number of leaf nodes in the case of tree data structure. Since the n-dimensional signatures are inserted into inner nodes only, enlargement of a data structure is not enormous (see Section 6). Now, operations of the Signature R-tree shall be described.

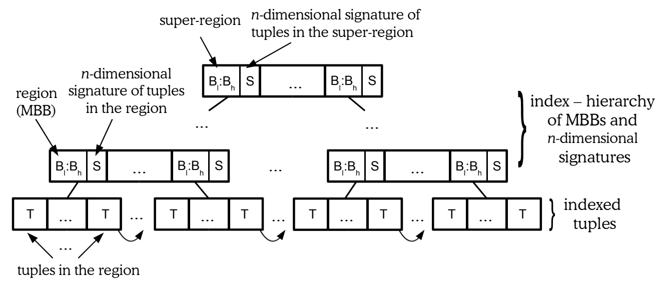
super-region
n-dimensional signature of tuples in the super-region
n-dimensional signature of tuples in the region
region (MBB)
B_l:B_h
S
...
index – hierarchy of MBBs and n-dimensional signatures
T
indexed tuples
tuples in the region

Figure 3: Structure of the Signature R-Tree.

5.1.1 Operations of the Signature R-tree

Operations `Insert`, `Delete`, and `Find` (point query) are handled by algorithms of the selected R-tree variant. Consequently, an arbitrary splitting algorithm or an algorithm, with various complexity, can be chosen for `Insert` operation, see [17, 26, 2]. Moreover, in the case of the Signature R-tree's `Insert` and `Delete` operations, a change of tuples in a leaf node must be reflected by changes of n-dimensional signatures in all inner nodes of a current path. The *Hamming distance* [10] is applied for measuring a similarity of signatures. The propagation of the changes to the root node is finished if the Hamming distance of old and new n-dimensional signatures equals 0 in some nodes.

5.1.2 Range query operation for a narrow hyper box

An advantage of described approach is that the algorithm of range query is not changed for a general range query. Of course, for processing a narrow range query, we apply the n-dimensional signature for a better filtration of any irrelevant tree node. Let us suppose the well-known `Intersection` operation, which ascertains whether a MBB is intersected by a query box in linear time.

Input: tuples $\mathtt{T}_1$,$\mathtt{T}_2$ which define the query box
Output: a set of tree tuples in the query box stored in an array R
Variables: a node N, a stack Z which contains a current path in the tree

```
begin
Z.Remove()
R.Remove()
N = the root node
```

```
Z.Push(N)
while Z is not empty do
begin
  if N is not leaf
  then begin
    if there is the next MBB, mbb, in N with
        non-empty MBB ∩ QB
    then begin
          determine whether mbb can be relevant
          using AND operation on $S^n_{qb}$ and $S^n_{mbb}$.
          if it is matched
          then begin
            Z.Push(N)
            read a child of region's item into N
          end
          else N = Z.Pop()
    end
    else N = Z.Pop()
  end
  else begin
    if N contains points of the query box
    then add such points into R
    N = Z.Pop()
  end
end
end
```

The increase of the space after adding of n-dimensional signatures is not enormous, but the time complexity of the algorithm is improved (see Section 6). The important issue is that the algorithm stays without change for a general range query.

In general, the volume of a region is reduced using a signature by following of perfect distribution of points. The n-dimensional signature forms a spatial region as well. Let $\mathcal{R_S}$ be a signature region formed for a set of points with the MBB $\mathcal{R_{MBB}}$. The intersection and signature operations are applied to filtration of irrelevant regions. Consequently, regardless to the shape of the signature region: $\mathcal{R_{MBB}} \cap \mathcal{R_S} \neq \emptyset$, $\mathcal{R_{MBB}} \cap \mathcal{R_S} \subseteq \mathcal{R_{MBB}}$. The formula turns out to be $N_{RQ} \leq N_I$. The most important issue is that the efficiency of the signature extension is always better or equal in comparison to the R-tree. Our experiments (see Chapter 6) prove that the efficiency of the Signature R-tree is always better for real data.

5.2 Signature generating

Now, we shall describe a method for generating an n-dimensional signature suitable for more effective processing a narrow range query. In the case of the multi-dimensional data structure, point clustering is controlled by principles of appropriate structure. As far as the R-tree is concerned, points are clustered into the MBB. In the case of signature data structures (e.g. S-tree [10]) signatures with a minimal Hamming distance are clustered. Of course, the signatures of points clustered in a region of a multi-dimensional data structure do not have minimal Hamming distances. If the signature of points would contain more true bits, then the signature of region's tuples would contain almost only true bits. Consequently, such a signature does not filter any irrelevant regions, because the probability of `false drop` is close to one. Evidently, the weight of a signature must be the smallest. Therefore, a hash function mapping each value to only one bit in a signature is used. Of course, we can not thicken the signature of a region by adding extra true bits for the reduction of `false drop` probability, as it is known in the S-tree. In this case, the n-dimensional signatures of superior tree's level would contain almost only true bits again.

Let $F : D \rightarrow H$ be a hash function. Let us take a domain to be $D = \{0, 1, \ldots, 2^{l_D} - 1\}$ and a range to be $H = \{0, 1, \ldots, 2^{l_S} - 1\}$ (see Definition 5). The hash function F is created by a generator of pseudo-random numbers (e.g. generator with a normal distribution). If $|D| = l_S$ then the mapping is suitable to define F as a simple one. If $H = \{2^0, 2^1, \ldots, 2^{l_S - 1}\}$, consequently only one bit is generated for each value and $\gamma(S^n)$ is the smallest. If $ql_i = qh_i$ for a query box's coordinate, then $\gamma(S_i) = 1$. For pure detection of irrelevant regions it must hold $\gamma(S_i) =$ the number of node items. Taken into consideration a hierarchy of n-dimensional signatures, it must hold $l_S = |D|$ for pure detection of irrelevant tree nodes. Such a length of a signature is not possible in real cases (often $|D| = 2^{32}$). The suitable length of the n-dimensional signature is a subject for experiments.

An n-dimensional signature is created by the superimposing of signatures independently for each dimension. Consequently, this case can arise, e.g., the region containing points $(2, 3, 4)$ and $(1, 5, 1)$ is relevant to the query box $QL = (2, 5, 0)$, $QH = (2, 5, max(D))$ from the signature filtering point of view. Experimental results show that this is not the case for the R-tree, because the clustering does not work in this way. Of course, it depends on data distribution.

5.3 Cost analysis

The complexity is not modified for basic operation `Find`, `Insert`, and `Delete` in the case of the Signature R-tree. A policy of node splitting or complexity of splitting algorithm depends on the chosen R-tree variant or a selected splitting algorithm [17, 26, 2]. In the case of the Signature R-tree, a change of tuples in a leaf node must be propagated to changes of n-dimensional signatures in all inner nodes of the current path. Consequently, the complexity is preserved.

The complexity of the general range query algorithm is $O(N_I \times \log_c m)$, where N_I is the number of intersect regions, c is the node's capacity. It holds the value $c_R \ll 1$ (see Definition 3) for a narrow range query (particularly for increasing dimension of indexed space). In the case of the Signature R-tree the complexity is $O(N_{RQ} \times \log_c m)$, where N_{RQ} (see Definition 4) is the number of searched regions (leaf nodes). Our experiments show $N_I \gg N_{RQ} \geq N_R$, consequently $c_Q \to 1$ in the case of the Signature R-tree. In other words, the space complexity of the algorithm is enhanced for reducing the time complexity. The R-tree clusters points of n-dimensional space into regions and follows an approximate distribution of data. The properties of clustering weaken the curse of dimensionality. Since the signatures follow a perfect distribution of data, the signatures eliminate the curse of dimensionality explicitly.

6 Experimental results

In [19, 20] a multi-dimensional approach to indexing XML data [29] was depicted. In this approach an XML document is represented as a set of paths. The paths are modelled as points in an n-dimensional space, where n is directly proportional to the maximal length of a path in an XML tree. Such points are inserted into a multi-dimensional data structure and XML queries are processed by the narrow range queries.

Table 1: A characterization of the test data collection and the size of index file

Dimension	**Number of**	**Index size** [MB]
n	**points**	**R*-tree**
7	8,268,357	478.6
9	8,739,522	603.1

Dimen-	**Index size** [MB]		
sion	**Signature R*-tree**		
n	$n \times 32$	$n \times 64$	$n \times 128$
7	493 [+3%]	512.2 [+7%]	536 [+12%]
9	651.4 [+8%]	680.7 [+13%]	711.7 [+18%]

The Protein Sequence Database XML document [27] was used for the experiments[1]. The document size is 683 MB. It includes 21,305,818 elements and 1,290,647 attributes. Approximately 17 mil. paths were obtained from this document. With respect to the frequency of the path lengths, two multi-dimensional indices indexing spaces of dimension $n = 7$ and $n = 9$ (for detail see [21]) were created to indexing XML data. Domain cardinalities of the spaces $|D| = 2^{32}$. The R*-tree and Signature R*-tree data structures were used for indexing the spaces. The lengths of n-dimensional signatures were chosen $n \times 32$, $n \times 64$, and $n \times 128$. Tables 1 and 2 summarize a characterization of data collection, index size, and index multi-dimensional data structures, respectively. Square brackets include the increase of index volume for Signature R*-trees. An average utilisation of 62% was reached in all cases.

[1]The experiments were executed on an Intel Pentium ®4 2.4GHz, 512MB DDR333, under Windows XP.

Table 2: A characterization of index multi-dimensional data structures

Dimen-	**Number of inner nodes**			
sion	**R***	**Signature R*-tree**		
n	**tree**	$n \times 32$	$n \times 64$	$n \times 128$
7	15,731	22,456	33,186	65,412
9	24,750	36,451	55,750	112,412

Dimen-	**Number of leaf nodes**			
sion	**R***	**Signature R*-tree**		
n	**tree**	$n \times 32$	$n \times 64$	$n \times 128$
7	256,520	257,124	258,187	260,741
9	318,370	320,741	331,474	335,846

Dimen-	**Height of tree**			
sion	**R***	**Signature R*-tree**		
n	**tree**	$n \times 32$	$n \times 64$	$n \times 128$
7	5	5	6	8
9	5	6	8	9

Inserted n-dimensional signatures enlarge the size of inner node items and a capacity of the inner nodes decreases (for equal node size) for the growing length of the signature. Consequently, the tree height increases. For more correct comparison, we chose the same size of node 2048 B for all trees. Of course, the node size can be extended and properties of the Signature R-tree can be improved (the height will be lower). We can see that the index size of Signature R*-trees extends to 3–18% in comparison to the R*-tree. An overhead of a Signature R*-tree escalates for growing signature length. We must choose an appropriate rate between the lesser number of searched regions during range query processing and an accrual of data structure overhead. The consequential results show the index size magnifies by the units of percentage for the signature length which filters the irrelevant tree nodes very well.

Two set of queries were tested for each space. The first set includes queries with a smaller result set (< 10), the second set holds queries with a larger result set (about 10^3). Each query processes a simple XPath query [28] for values of elements and attributes of the XML document like `ProteinDatabase/ProteinEntry [reference/refinfo/citation='Nature']`. In Ta-

Table 3: A characterisation of narrow range query sets

Query set	**Dimension** n	n_ψ (see Definition 2)	**Result size**
1	7	2	5
2	7	2	3,397
3	9	2	8
4	9	2	2,794

Query set	N_I	N_R	c_R
1	828	1	0.0012
2	1,542	717	0.47
3	641	7	0.01
4	136	136	1

ble 3 a characterization of the narrow range query sets is shown. Note, the results were given an average for all tests. Values N_I, N_R, and c_R are given for the R*-tree. Naturally, the relevance ratio c_R (ratio of the relevant and intersect regions) is approximately the same for all data structures. We see that the ratio is rather low ($\ll 1$) for the narrow range queries. Consequently, efficiency of query processing is not optimal for current multi-dimensional data structures.

Table 4: Experimental results of processing the narrow range queries – N_{RQ}

Query set	N_{RQ}			
	R* tree	**Signature R*-tree**		
		$n \times 32$	$n \times 64$	$n \times 128$
1	828	162	16	10
2	1,542	1,142	792	761
3	641	258	44	36
4	171	165	136	136

The efficiency of narrow range query processing was measured by the number of searched leaf nodes (regions) N_{RQ}, ratio c_Q, disk access cost (DAC, number of all tree nodes read during a query processing), and time of query processing. In Table 4 the N_{RQ} values are presented for R*-tree and Signature R*-tree for more lengths of an n-dimensional signature. Table 5 presents the ratio between N_{RQ} to the number of all leaf nodes. We can see the ratio is lesser in the case of Signature R*-tree. Of course, the percentage is lesser for longer n-dimensional signatures. Another view of the trend is the higher values of c_Q ratio in Table 6. We see the ratio is closer to one more than in the case of R*-tree. Consequently, the improvement of the Signature R*-tree (compare to the R*-tree) is 1.2–83× (see the second table).

In Table 7 the DAC is presented for processing of the test queries. The number of searched leaf nodes (inner nodes as well) is lower in the case of the Signature R*-tree. In spite of the n-dimensional signature escalates the data structure overhead, the DAC is decreased. We see that we must choose a compromise between a better quality of longer signature and lower data structure overhead. The optimal length of the n-dimensional signature is $n \times 64$ in this case. Table 8, which contains times of query processing supports the conclusion. We see that the Signature R*-tree provides a better efficiency of processing the narrow range queries.

Table 5: Experimental results of processing the narrow range queries – the ratio of searched leaf nodes

Query set	**Ratio of searched leaf nodes** [%]			
	R* tree	**Signature R*-tree**		
		$n \times 32$	$n \times 64$	$n \times 128$
1	0.32	0.061	0.006	0.004
2	0.60	0.45	0.310	0.300
3	0.20	0.76	0.013	0.011
4	0.05	0.03	0.040	0.040

Table 6: Experimental results of processing the narrow range queries – c_Q ratio

Query set	c_Q			
	R* tree	**Signature R*-tree**		
		$n \times 32$	$n \times 64$	$n \times 128$
1	0.0012	0.006	0.06	0.1
2	0.47	0.63	0.91	0.94
3	0.01	0.03	0.16	0.2
4	0.80	0.96	1	1

Query set	**Improvement of c_Q ratio**		
	Signature R*-tree		
	$n \times 32$	$n \times 64$	$n \times 128$
1	5×	53×	83×
2	1.3×	2×	2×
3	3×	16×	20×
4	1.2×	1.3×	1.3×
Average	2.6×	18×	27×

7 Conclusion

We have presented the application of the signature for efficient processing the narrow range queries in multi-dimensional data structures. The n-dimensional signatures are inserted into well known R-tree data structure, the novel data structure is called the Signature R-tree. The signature helps to filter the irrelevant tree nodes better than simply the intersection algorithm. Experimental results prove an efficiency of such approach. For example, the Signature R-tree proves an

Table 7: Experimental results of processing the narrow range queries – the disk access cost

Query set	**DAC**			
	R* tree	**Signature R*-tree**		
		$n \times 32$	$n \times 64$	$n \times 128$
1	960	478	74	108
2	1,671	1,393	1,043	1,285
3	817	396	477	340
4	220	200	184	201

Query set	**Improvement ratio of DAC**		
	Signature R*-tree		
	$n \times 32$	$n \times 64$	$n \times 128$
1	2×	13×	9×
2	1.2×	1.6×	1.3×
3	2×	1.7×	1.4×
4	1.1×	1.2×	1.1×
Average	1.6×	4.3×	3.2×

Table 8: Experimental results of processing the narrow range queries – time of query processing

Query set	**Time of query processing** [s]			
	R* tree	**Signature R*-tree**		
		$n \times 32$	$n \times 64$	$n \times 128$
1	0.08	0.05	0.015	0.03
3	0.19	0.17	0.094	0.16
2	0.20	0.14	0.140	0.15
3	0.017	0.015	0.015	0.015

Query set	**Improvement ratio**		
	Signature R*-tree		
	$n \times 32$	$n \times 64$	$n \times 128$
1	1.7×	5×	3×
2	1.1×	2×	1.2×
3	1.5×	1.5×	1.3×
4	1.1×	1.1×	1.1×
Average	1.4×	2.4×	1.7×

improvement of the disk access cost up to 4.3× in our experiments. Moreover, the results prove resistance to the curse of dimensionality. Due to the fact that presently generated signatures do not allow for the search of just the relevant nodes, we would like to improve this approach in our future work as well.

References

[1] R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of WWCA'97, Tsukuba, Japan*, 1997.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331.

[3] A. Belussi, E. Bertino, and B. Cataniac. Using spatial data access structures for filtering nearest neighbor queries. *Data & Knowledge Engineering*, 40(1):1–31, 2002.

[4] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, San Francisco, U.S.A., 1996. Morgan Kaufmann Publishers.

[5] C. Böhm, S. Berchtold, and D. Keim. Searching in High-dimensional Spaces – Index Structures for Improving the Performance Of Multimedia Databases. *ACM Computing Surveys*, 3(3):322–373, 2001.

[6] J. Chang, J. Lee, and Y. Lee. Multikey Access Methods Based on Term Discrimination and Signature Clustering. In *Proceedings of 12th ACM SIGIR, USA*, pages 176–185, June, 1989.

[7] W. Chang and H. Schek. A Signature Access Method for the Starburst Database System. In *Proceedings of 15th VLDB Conference, Netherlands*, pages 145–153, Aug. 1989.

[8] P. Ciaccia, M. Pattela, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of 23rd International Conference on VLDB*, pages 426–435, 1997.

[9] H.-J. K. D.-J. Park, S.Heu. The RS-tree: An efficient data structure for distance browsing queries. *Information Processing Letters*, 80:195–203, 2001.

[10] U. Deppisch. S-Tree: A dynamic balanced signature index for office retrieval. In *Proceedings ACM Conf. Research and Development Information Retrieval, Pisa, Italy*, pages 77–87, 1986.

[11] V. Dohnal, C. Gennaro, and P. Zezula. A Metric Index for Approximate Text Management. In *Proceedings of IASTED International Conference Information Systems and Database – ISDB 2002*, 2002.

[12] C. Faloutsos and S. Christodooulakis. Signature Files: An Access Method for Documents and its Analytic Performance Evaluation. *ACM Transactions on Information Systems*, 2(4):267–288, 1984.

[13] R. Fenk. The BUB-Tree. In *Proceedings of 28rd VLDB International Conference on VLDB, Hongkong, China*, 2002.

[14] M. Freeston. A General Solution of the n-dimensional B-tree Problem. In *Proceedings of SIGMOD International Conference, San Jose, USA*, 1995.

[15] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[16] T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD 2002, Madison, USA*, June 4-6, 2002.

[17] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.

[18] N. Karayannidis, A. Tsois, T. Sellis, R. Pieringer, V. Markl, F. Ramsak, R. Fenk, K. Elhardt, and R. Bayer. Processing Star Queries on Hierarchically-Clustered Fact Tables. In *Proceedings of VLDB Conf. 2002, Hongkong, China*, 2002.

[19] M. Krátký, J. Pokorný, T. Skopal, and V. Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conferences, EurAsia-ICT 2002, Shiraz, Iran.* Springer–Verlag, LNCS 2510, 2002.

[20] M. Krátký, J. Pokorný, and V. Snášel. Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In *Accepted at International Workshop on Database Technologies for Handling XML information on the Web, DataX, Int'l Conference on Extending Database Technology (EDBT 2004), Heraklion - Crete, Greece*, 2004.

[21] M. Krátký, T. Skopal, and V. Snášel. Multidimensional Term Indexing for Efficient Processing of Complex Queries. *Kybernetika, Journal of the Academy of Sciences of the Czech Republic, accepted*, 2003.

[22] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. R-trees Have Grown Everywhere. *Submitted to ACM Computing Surveys*, 2003.

[23] Y. Manolopoulos, A. Nanopoulos, E. Tousidou, and Y. Manopoulos. *Advanced Signature Indexing for Multimedia and Web Applications.* The Kluwer International Series on Advances in Database Systems, 2003.

[24] Y. Manolopoulos, Y. Theodoridis, and V. Tsotras. *Advanced Database Indexing.* Kluwer Academic Publisher, 2001.

[25] T. Skopal, P. Moravec, M. Krátký, V. Snášel, and J. Pokorný. An Efficient Implementation of the Vector Model in Information Retrieval. In *Proceedings of the fifth National Russian Research Conference, RCDL'2003, Digital Libraries: Advanced Methods and Technologies, Digital Collections, Saint-Petersburg, Russia*, pages 170–179. Saint-Petersburg State University Published Press, 2003.

[26] C. F. T. Sellis, N. Roussopoulos. The R^+-Tree: A Dynamic Index For Multi-Dimensional Objects. In *Proceedings of the 23. Int. VLDB Conference*, pages 507–518, 1997.

[27] University of Washington's database group. The XML Data Repository, 2002, `http://www.cs.washington.edu/research/xmldatasets/`.

[28] W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.

[29] W3 Consortium. Extensible Markup Language (XML) 1.0, 1998, `http://www.w3.org/TR/REC-xml`.

[30] C. Yu. *High-Dimensional Indexing.* Springer–Verlag, LNCS 2341, 2002.

[31] P. Zezula, F. Rabitti, and P. Tiberio. Dynamic Partitioning of Signature Files. *ACM Transactions on Information Systems*, 9(4):336–369, Oct. 1991.