Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

An Efficient Point Data Indexing Structure for Multidimensional Range Queries

P. Z. Piah
Department of Computer Science
Kenule Benson Saro-Wiwa Polytechnic
Bori-Rivers State, Nigeria

P. O. Asagba
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria

V. Ejiofor
Department of Computer Science
Nnamdi Azikiwe University
Awka, Nigeria

K.T. Igulu
Department of Computer Science
Kenule Benson Saro-Wiwa Polytechnic
Bori-Rivers State, Nigeria

***Abstract*— Since the main memory is expensive and volatile persistence data cannot be kept in main memory. Most databases utilize the secondary/tertiary storage. The main overhead of using a secondary storage is access time. This is usually high in multi-dimensional databases like OLTP that are characterized by recurrent updates and queries. The use of secondary/tertiary storage inherently suggests indexing. Indexing helps to retrieve/store the required data/ data segments faster than iterating through the table. Indexing enhances speed of querying. In multi-dimensional databases (like OLAP databases) the essential tool for accessing data is the range query (or window query). In extant databases, B-Trees and its variants are the convention for indexing. But the major setback of B-trees is that they are single attribute index structures. Which implies that the record of such database are ordered by a particular attribute usually but not necessarily the primary key. This limits range query restriction on one particular attribute of the table. The use of multiple B-trees indexing for various attributes of a table is the convention for achieving range query for other attributes. The use of multi-indexing is additive and poses so many drawbacks (additional space required and speed is hampered). With the exponential burst of data, there is need for a better data structure with efficient query algorithm that has high storage capacity and also considerably fast (algorithm) for multidimensional range queries. This paper discusses a multidimensional indexing structure that is fast and also consumes virtually equivalent space as though is a single attribute structure. Experiment show that the structure has multiplicative complexities and is immune to the curse of dimensionality.**

Keywords*— Range query, Indexing, multidimensional, database, B-tree, UB-Tree.

I. INTRODUCTION

Complex enterprise applications for instance SAP HANA [1], data mining (DM), data warehousing (DW) and non-standard DBMS applications such as geographical information systems (GIS) and statistical databases have spawned a strong demand for efficiency in the processing of extremely complex queries on large databases. These complex queries of course set new requirements on the query processing and indexing method algorithms for DBMSs. The main purpose of indexing a table of a database is to expedite query execution. This is achieved by utilizing the constraints imposed by a query in order to condense the number of disk accesses. In multi-dimensional databases (like OLAP databases) the essential tool for accessing data is the range query (or window query). In extant databases, Bitmap, B-Trees and its variant are the convention for indexing. But the major setback of B-trees is that they are single attribute index structure. Which implies that the record of such database are ordered by a particular attribute usually but not necessarily the primary key. This limits range query restriction on one particular attribute of the table. The use of multiple B-trees indexing for various attributes of a table is the convention for achieving range query for other attributes.

This paper discusses an access method (indexing structure) for efficient manipulation of range queries called the UB-Tree. It organizes and clusters any table of a database by a certain computation that depends on some or all the attributes of the table. By this, data points that are close by this computation are stored closely in the storage. In other words, spatial proximity is maintained based on all the attributes or some of the attributes of the table.

II. QUERIES

Relational queries are primarily expressed by operators of the relational algebra and they operate either with a single table or span multiple tables [2]. Single table queries restrict, re-arrange or aggregate the tuples of one relation [3]. A query is a predicate φ(x) over the tuples of a relation R. The result set (RS) of a query is the subset of tuples of R sufficiently satisfy the query predicate in (1). The result set size is the cardinality of the result set as given in (2).

$$RS(R,\varphi) = \{x \varepsilon R \mid \varphi(x)\} \quad (1)$$

$$|RS(R,\varphi)| \quad (2)$$

A restriction query is a predicate φ(x) on the tuples 'x' of a relation R. A restriction query can be to a point-exact match query or on some dimension-partial match or partial range query. A range query is a special case of query with restrictions on all the dimensions. Reference [4] categorizes queries into single table queries and multiple table queries. Multiple table queries specifically join single table queries of

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

different tables. Single table queries are further categorized into restriction queries and queries of re-arrangement which can be sorting, projection , grouping and aggregation. Partial range query is a type that gives restriction on some dimensions of the query and some unrestricted. This can be further categorized as exact-match query or range query.

A. *Range Queries*

Range queries give restriction on all dimensions (attributes) of the query. From the geometric perspective, let P be a set of n points in $Ð^k$ (i.e. k dimensional- $Ð^k = \{d_1, d_2, d_3, \ldots, d_k\}$) and let D be a family of subsets of $Ð^k$ (i.e.$D \subseteq Ð^k$). Let r_i represent a range of the i^{th} dimension ($r_i \subseteq d_i$). We call r_i an interval represented by its lower and upper bound ($r_i = (l,h)$). Elements of D are called ranges (i.e. $D = \{r_1, r_2, \ldots, r_k\}$). Let Δ be points subsumed by the range set D. Given the above, the requirement is to build appropriate data structure that supports range reporting, range count (number of points in the result set or the cardinality of the result set), emptiness query (determining if the result set size is not zero) given in (3), (4) and (5) respectively.

$$\Delta \cap P \quad (3)$$

$$|\Delta \cap P| \quad (4)$$

$$\Delta \cap P = \Phi \text{ or } |\Delta \cap P| = 0 \quad (5)$$

An exact point query shall be regarded as a special case of range query whose intervals are equal values on the various dimensions. i.e. l=h for all the k dimensions. No two points on the plane have the same address (no two points have the same x-and y-coordinates for 2-d space). Range queries are a fundamental problem for all database systems. A range query is specified by an interval for each dimension. If no restriction for a dimension it formally means that the interval is (-∞, +∞) i.e. unbounded. The query is the Cartesian product of the intervals for all dimensions, called the **query box (QB) or Query Volume** of *Q* with the lower and upper bounds *ql* and *qh*. The answer to *Q* is the set of data points-objects in *Q*. Subsequently, this set will be called **Result Set** or simply **RS** of **Q**.

III. THE UB-TREE

The UB-tree is due to Markl [2] in their work on Mistral [3]. The UB-tree as described in [2] is a structure to index multi-dimensional data with linear complexities i.e. using a structure that has linear complexities. The UB-Tree exploits the capabilities of B-tree and Z-curve [7]. Each multi-dimensional data tuple is transformed into an integer (Z-address), which is inserted into the B-tree. Each node is a pair of integer ([α:β]) denoting the lower bound (α) and the upper bound (β) of a region on the plane respectively. It suffices to note at this point that the entire plane is regarded as a Z-region (Super-Z-region). For consistency, all regions will be regarded as simply Z-region. A leaf of the UB-tree which is mapped to a Z-region of the curve holds data (points) or link to the data. Usually, a region mapped to a disk block (or page). Range query can be handled by retrieval points in regions that are perfectly subsumed by the query box or that intersect the query box. Figure 1 shows (a) typical 2-D 8by8 space with 6 Z-regions (b) UB-tree-nodes corresponding to the z-regions in the space. The inner nodes of UB-tree recursively divides the space, such that a hierarchy of nested Z-regions is formed.

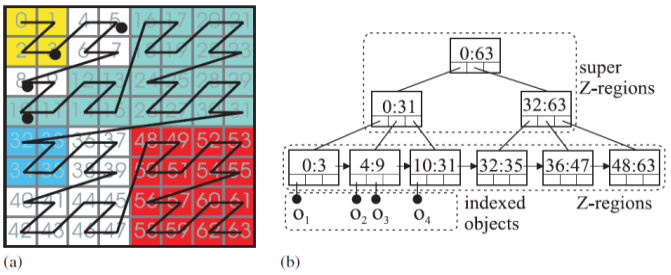


Figure 1: Z-curve with Z-regions and UB-Tree

A. *Data Types and Address Calculation*

The values of the attributes specified in query operations by a user are over specific domains (data types). Some of these domains have natural lexicographical ordering. Domains with explicit lexicographical ordering are directly converted to binary. The interleaving algorithm is then invoked on the binary equivalent of the values. Some domains require transformation to achieve uniqueness and ordering of values over such domain. To achieve this, transformation algorithms are utilized. Then the transformed equivalent is used for interleaving operation. In such domain, bit-lexicographic order on the binary representation of tuples does not correspond to any natural ordering. Transformation is bijective. For Cartesian co-ordinates, the inverse function to the transformation function is applied after bit extraction. Figures 2a and 2b depict this concept. Character string can be transformed to binary by using the ASCII lexicographical ordering of the character. This will produce the binary equivalent of the character string and interleaving invoked appropriately. In programming language, ENUM type defaults to Integer. For the integer default, the ENUM values are mapped to their integer equivalents are interleaved appropriate. ENUM of character string will of course undergo two levels of transformation i.e. mapping and transformation itself.

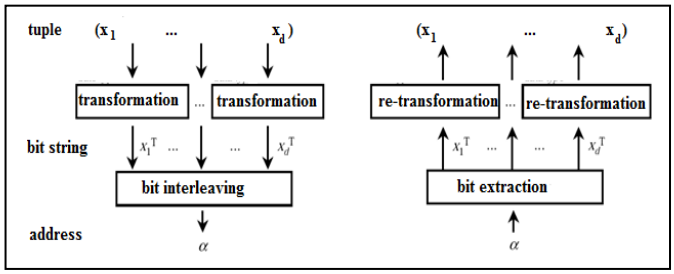


Figure 2: (L): Address Calculation. (R): Cartesian Calculation

B. *Address Calculation and Interleaving Operation*

The address computation of the UB-Tree determines its efficiency. The UB-Tree utilizes the bit-interleaving paradigm. If each attribute x_i of a d-dimensional tuple (x_1, x_2, ..., x_d) consists of 2^r values, it can be regarded as a sequence of bits $x_{i,r}$ ... $x_{i,1}$. Bit-interleaving produces an r-dimensional tuple from the d-dimensional tuple by re-arranging the bits of the tuple in the following way[6]:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

$$\text{Interleave}^{d,r}(x_{1,r}\ldots x_{1,1},\ x_{2,r}\ldots x_{2,1}\ldots x_{d,r}\ldots x_{d,1}) = x_{1,r}\,x_{2,r}\ldots x_{d,r},\ x_{1,r-1}\ldots x_{d,r-1},\ x_{1,1}\ldots x_{d,1}) \quad (6)$$

The value produced by this computation is henceforth known as Z-value. Thus interleave^{3,4} (1110,1010,0111) = (110,101,111,001). If the result of interleave^{d,r} is considered as binary number in the place of an r-dimensional tuple, incrementing this number by 1 yields the Z-Address (Z_{addr}) of a tuple. Thus:

$$Z_{addr}(x_1, x_2, \ldots, x_d) = \text{interleave}^{d,r}(x_1, x_2, \ldots, x_d) + 1 \quad (7)$$

Therefore $Z_{addr}(14,10,7) = Z_{addr}(1110, 1010, 0111) = \text{interleave}^{3,4}(1110,1010,0111) + 1 = (110,101,111,001) + 1 = (110,101,111,010) = 6.5.7.2$. The Z-address can be computed by a function given in [2].

$$Z_{addr}(P) = \sum_{j=0}^{s-1} \sum_{i=1}^{n} P_{i,j}\, 2^{jn+i-1} \quad (8)$$

Pϵ Ω, where the binary representation of each coordinate p_i is denoted as $p_i = p_{i,s-1}, p_{i,s-2}, \ldots, p_{i,o}$. This function does the same thing as bit-interleaving. The inverse function to *interleave*^{d,r} can be computed basically in the same way. This function is called *(interleave*^{d,r}*)*$^{-1}$. Therefore *interleave*^{d,r} *(o) = interleave*^{r,d} (Z_{addr}(o)-1). Only a slight modification of the interleave operation is necessary to support a universe where the domain of each dimension does not consist of the same number of bits *r*. The domains of this work are of equal cardinalities (2^r). The algorithm of bit-interleaving has the CPU-complexity of *O(d*r)*, where *r* stands the length of each attribute in bits and d is the dimensions. The same holds for *(interleave)*$^{-1}$. Switching a tuple between Cartesian representation and address representation can therefore be performed very efficiently. Figures 2 and 3 show the bit-interleaving operations in 2d and 3d respectively. The bit-interleaving does not deteriorate by high dimensionality (immune to the *curse of dimensionality*).

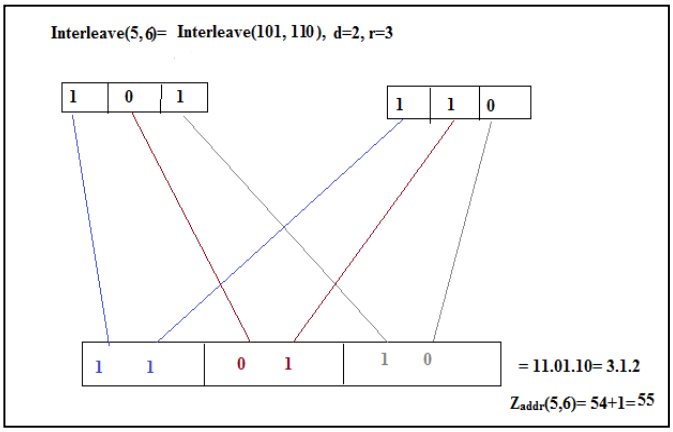


Figure 4: Interleaving process of 2-d tuple [5,6]

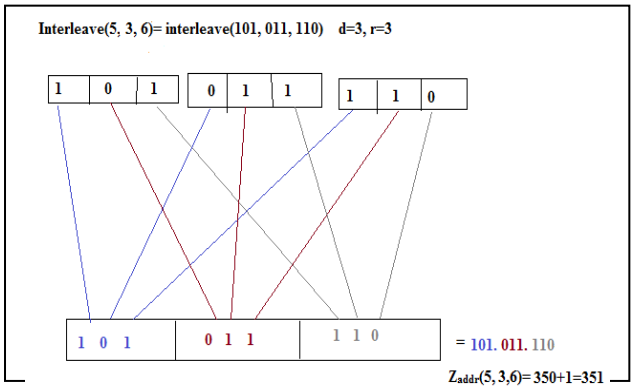
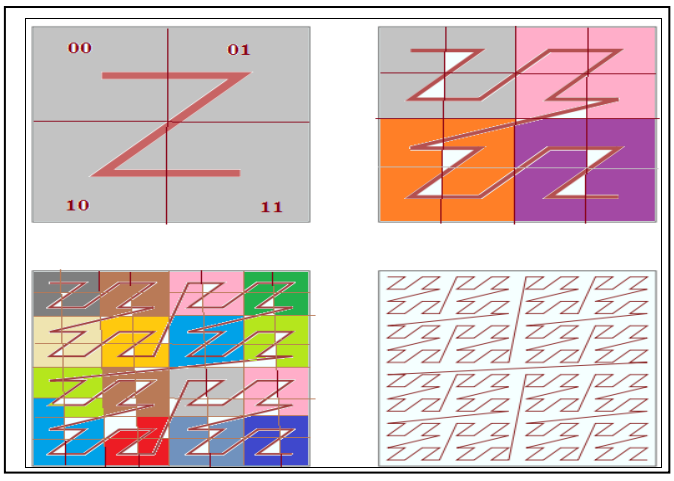


Figure 5: Bit-interleaving in 3d tuple [5,4,6]

C. *Rapid Hyper Quadrant Jump Addressing Approach*

This new approach was introduced in [4]. Investigation show that the approach is just a new method but not different from the original idea. In geometry, the term quadrant depicts an exactly defined quarter of 2-dimensional universe. Equally, 1-dimensional space can be divided into two halves, and for 3-dimensional space eight octants of space. Common to most geometric constructs is a constraint to partition the universe. Moreover, the partitioning is done using halving the space in all existing dimensions. Such information can be generalized by definition of the hyper-quadrant of n-dimensional space.

Let $\Omega = D^n$ be a vector space: The hyper-quadrant (hquad) HQ is a subspace in Ω; i.e. HQ$\subset\Omega$; such that. $HQ = HD_1 \times HD_2 \times HD_n$; where each domain HD_i is the lower or the upper half of the domain D, i.e. HD_i = low(D) or HD_i = up(D). i.e. the lower and upper subspace in each dimension. Each n-dimensional vector space Ω is formed by its 2^n disjoint hquad. i.e. there are distinct 2^n disjoint hquads. If the hquad are designated with the 2^n distinct possible values, a unique code will be computed for each hquad. For instance in n=2, i.e. 2d, there will be four quadrants designated 00, 01, 10 and 11. For 3d, there be 000, 001, 010, 011, 100, 101, 110, 111. The bit-length equals the dimension. Recall the geometric representation of the Z-Order curve which expands its resolution by half-splitting on all the n dimensions. Figure 6 shows the 2d Z-order curve and figures 7 gives the 3d hquad representation.



International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

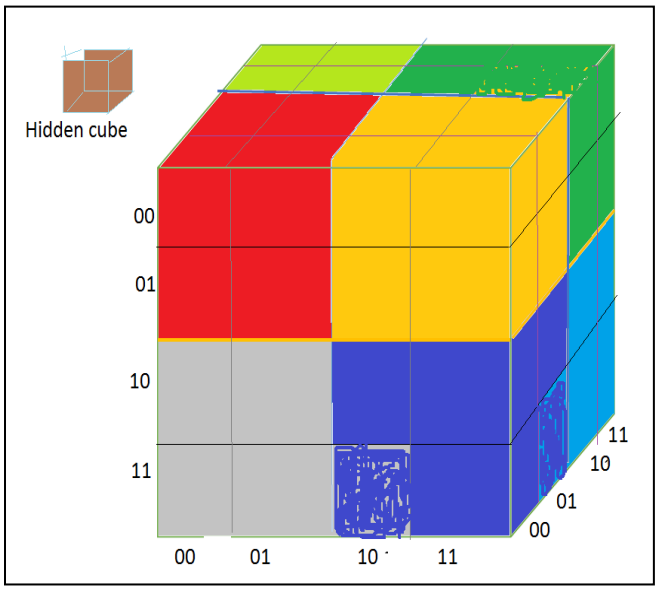


Figure 7: 3d case with Visualization of the various Hquads.

The Z-Order curve is a finite image of its self by recursively splitting along the various dimensions of the plane. Since the Z-Order is an image of itself, we can jump from one supper quadrant to its specializations given the bit string address of such quadrant. For Z-curve in 2d, the first two bits of the Z-address represent the super quadrant where the point can be found. The second two bit string specifically identifies the second level of the splitting and so on. The same applies to d-dimensional space. This approach combines the interleaving and the searching together and has a great performance impact for all the operations on the UB-Tree[4].

D. Insertion Algorithm

A point P to be inserted into the universe Ω is specified by its Cartesian coordinates $(x_1\ x_2\ , ...,\ x_d)$ with address Z-value = $Z_{addr}(x_1\ x_2\ , ...,\ x_d)$. P belongs to the unique region [α:β] fulfilling α≤ Z-value ≤β. It should be noted that Z-value must be computed only to a precision sufficient enough to determine the proper region. The addresses are linearly ordered by '≤'. P is inserted into the leaf-page corresponding to such region, which is found by an exact-match query. Since pages can store only a maximum number *M* of pointers or objects, pages may overflow and are split like in B-trees. [α:β] is split by a new area with address γ satisfying α< γ< β. The region [α:β] is divided by γ into [α:γ] and [γ+1:β]. The objects or object pointers in *page* ([α:β]) are distributed onto *page*([α:γ]) and *page*([γ+1:β]) accordingly. γ is created by increasing *area*(α) as follows: Add to area α sub-cubes from [α:β] in increasing order until the number of the objects in [α:γ] is between ½*M*-∂ and ½*M*+∂ . If the sub-cube that follows in this process contains much objects, it undergoes a recursive subdivision until the condition can be met. The parameter ∂ is used to get shorter split addresses, which are favorable for the UB-Tree performance especially of the range query algorithm[2].

E. Split Point Algorithm

The constraint of SFCs, (Z-order precisely) is that regions are disjoint (regions don't overlap). The non-overlapping of regions has direct performance impact on range query. Range query is succinctly finding those region that intersect the query box (QB). If splitting is not optimal, the general performance of the range query degrades. Another effect of region or page splitting is to avoid post filtering as much as possible. If the split point for a split is not optimally computed, there could be overlapping of pages and this directly impacts the access time during an operation that affects such region. Reference[2] uses a tree to keep track of split points. Reference [4] showed that the underlining one dimensional tree used in [2] is redundant and introduced a new optimal split point algorithm. The algorithm proposed in [4] ensures that;

i. The left region should be half full.

ii. A perfect partition that creates at least the maximum number of point's larger page/region for the left page. After the split, the left region has a restriction of the number that can be inserted. This avoids early split.

F. Point Query Algorithm

Point Queries are also called "exact-match queries". They are specified by the Cartesian coordinates $(x_1, x_2, ..., x_d)$ of the point P. In OLAP these co-ordinates are usually called dimensions or dimension attributes. To find P, the address z-value := $Z_{addr}(x_1, x_2, ..., x_d)$ of P with sufficient precision to find the unique region [α:β] with the property a*b and fetch *page*([α:β]). This is realized by searching the UB-tree with the address Z-value as the search key. *page*([α:β]) must contain point *P* with the additional information or the identifier *Id(P)* that is used to reference *P*. *P* can be found in $O(\log_k N)$ time, where *N* is the number of objects in the universe Ω and *k* is the order of the tree. Since UB-trees are balanced and searched exactly like the B-tree used as the underlying data structure for the UB-tree. Thus the point query performance of a UB-Tree is similar to that of a B-Tree index. The additional address calculation overhead is negligible.

G. Range Query Alogorithm

To answer a range query, only those regions, which perfectly intersect the query box, must be fetched from the database and thus from the disk. Initially the range query algorithm computes and retrieves the first region that is overlapped by the query-box. Then the next intersecting region is computed and retrieved. This is repeated until a minimal cover for the query box has been constructed, i.e., the region that contains the ending point of the query box has been retrieved. At first, Z-address of the query box lower and upper bounds are computed. Using this value, a leaf page of the UB-tree is retrieved and searched for relevant data tuples. Next, the following query-intersected leaf is retrieved and so on. The algorithm will finish as soon as the β of the actual Z-region gets greater than Z-address of the query box upper bound, i.e.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

when $\beta >$ Z-value$_h$. With the exception of cases where the query box degenerates to a hyper-plane of the universe, the number of regions, which must be fetched from the database, is for sufficiently large databases proportional to the volume of *Q* and therefore proportional to the size of the answer to the range query.

IV. CONCLUDING REMARKS

Extant commercial DBMS do either not support multidimensional indexes at all or only use them as an add-ons. A widespread approach to handling multi-dimensional range queries consists of the successive application of such single key structures, one for each dimension. Concatenated or compound single structures are used to handle multi-dimensional range queries. Unfortunately, this approach can be very inefficient. Since each index is accessed independently of the others. It's apparent that high selectivity in one dimension to narrow down the search in the remaining dimensions is not possible. Generally, there is no easy and apparent approach to expand single key structures to handle multi-dimensional data. Other shortcomings of existing multidimensional indexing structures for databases is that they don't guarantee for spatial proximity and they suffer of the curse of dimensionality. Their performance also degrade because of memory consumption for creating indexes for the various attributes in the case of compound indexing and post-filtering (matching of tuples to build the result set). For concatenated indexing, queries favor the first attributes. More also every query using such indexing must include the first group of attributes. Experience also show that existing indexing structures for complex range queries possess additive complexity whereas the approach discussed here possesses multiplicative complexity which impacts performance. The enforcement of tuple clustering which also translates to page clustering based on majority dimensions (attributes) of the tuple reduces disk accesses for complex range queries.

REFERENCES

[1] SAP-Group, 2015. [Online]. Available: www.sap.com.
[2] V. Markl, "Processing relational queries using a multidimensional acces technique," *Dissertations in Database and Information Systems-Infix,*, vol. 59, 1999.
[3] "MISTRAL Project," 1999. [Online]. Available: http://mistral.informatik.tu-muenchen.de.
[4] P.Z. Piah, "An Efficient Range Searching Algorithm in Complex Geometric Space", PhD thesis, University of Port Harcourt, 2015.
[5] T. Skopala, M. Kratkyb, J. Pokornya and V. Snaselb, "A new range query algorithm for Universal B-trees," *Information Systems*, vol. 31, p. 489–511, 2006.
[6] R. Bayer, "The universal B-tree for multidimensional indexing: general concepts," in *International Conference on Worldwide Computing and Its Applications, Springer*, Berlin, 1997.
[7] J. Lewder, "The Application of Space-the Storage and Retrieval of Multi-dimensional Data," London, 1999.