Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.976 – 981

RESEARCH ARTICLE

An Efficient Indexing Method for Box Queries in NDDS Spaces using BoND-tree

P Jhansi Rani[1], GK Srikanth[2]
M.TECH, Dept. Of CSE, AVNIET, JNTUH, HYDERABAD, AP
Associate Professor, Dept. Of CSE, AVNIET, JNTUH, HYDERABAD, AP

Abstract— Similarity searches in multidimensional Non-ordered Discrete Data Spaces (NDDS) are becoming increasingly important for application areas such as bioinformatics, biometrics, data mining and E-commerce. Efficient similarity searches require robust indexing techniques. Box queries (or window queries) are a type of query which specifies a set of allowed values in each dimension. Unfortunately, existing indexing methods developed for multidimensional (ordered) Continuous Data Spaces (CDS) such as the R-tree cannot be directly applied to an NDDS. Most of the existing work in this field targets the similarity queries (range queries and k-NN queries). Other indexing methods based on metric spaces such as the M-tree and the Slim-trees are too general to effectively utilize the special characteristics of NDDSs, resulting in non-optimized performance. In this paper, we propose a new dynamic data partitioning- based indexing technique, called the BoND-tree, to exploits exclusive properties of NDDS. Unique characteristics of the NDDS are exploited to develop new node splitting heuristics. The BoND-tree and the Slim-trees for similarity searches in multidimensional NDDSs. For the BoND-tree, we also provide theoretical analysis to show the optimality of the proposed heuristics. Extensive experiments with synthetic data demonstrate that the proposed scheme is significantly more efficient than the existing ones when applied to support box queries in NDDSs. We also show effectiveness of the proposed scheme in a real world application of primer design for genome sequence databases.

Keywords: NDDS, Box Queries, Indexing Methods

I. INTRODUCTION

Box query in NDDS is an important type of query which is defined by specifying a set of allowed values in each dimension. These queries are useful in many diverse applications such as bioinformatics, biometrics, data mining and E-commerce. Each data item is viewed as a set of non-ordered discrete values rather than a vector (in this paper we use the terms 'high-dimensional categorical data' and 'vectors in NDDS' interchangeably). There is an increasing demand for similarity searches in multidimensional Non ordered Discrete Data Spaces (NDDS) from application areas such as bioinformatics, biometrics, data mining and E-commerce. The main characteristic of such a data space is that the data values in each dimension are discrete and have no ordering. Other examples of non-ordered discrete values in a dimension of an NDDS are discrete data types such as gender, complexion, profession and user-defined enumerated types. In general, indexes are used to achieve improved response time for query execution in large databases. In this paper we propose an effective indexing scheme for implementing box queries in NDDS for large databases. There are many existing indexing schemes for large databases for continuous data spaces (CDS). These indexing schemes are not suitable for queries in NDDS because of the fundamental differences between the two spaces. The databases that require searching information in an NDDS can be very large (e.g., the well-known genome sequence database, contains over 80 GB genomic data). To support efficient similarity searches in such databases, robust indexing techniques are needed. Indexing techniques in the CDS rely on the fact that the indexed values can be ordered in each dimension which is not the case in NDDS. However, NDDS has certain value discrimination properties which can be exploited for efficient implementation of indexes in NDDS. The proposed work exploits these properties of NDDS to develop a new indexing scheme, BoND-tree, targeted towards improving the performance of box queries.

In this paper we focus on the application of box queries for primer design in genome sequence databases. A box query in a genome sequence database of q-grams (fixed length overlapping short sequences created from the database of variable length long genome sequences) allows a set of characters in each position of a q-gram. Some essential geometric concepts such as rectangle, sphere, region area, and so on are no longer valid in an NDDS, where data values in each dimension cannot even be labelled on an (ordered) axis. Hence the above techniques cannot be directly applied to an NDDS.

If the alphabet for every dimension in an NDDS is the same, a vector in such a space can be considered as a string over the alphabet. In this case, traditional string indexing method, such as BoND-tree can be utilized. To support efficient similarity searches in an NDDS, we propose a new indexing technique, called the BoND-tree. The key idea is to exploits exclusive properties of NDDS as well as some effective indexing strategies (e.g., node splitting heuristics in the B-tree) in CDSs to NDDSs. There are several technical challenges for developing an indexing method for an NDDS. They are due to:

(1) The limited choices of splitting points on each dimension. The BoND-tree is developed in such a way that these difficulties are properly addressed.
(2) non-applicability of popular continuous distance measures such as Euclidean distance and Manhattan distance to an NDDS;
(3) high probability of vectors to have the same value on a particular dimension in an NDDS; and
(4) no ordering of values on each dimension in an NDDS;

Our extensive experiments and theoretical analysis demonstrate that the ND-tree can support efficient searches in NDDSs. In the process of primer design, a biologist first generates a set of candidate primers which may be degenerate and then eliminate those which cannot be used, by matching the primer against a database of genome sequences.
Traditionally, this search is performed by linearly scanning the genome sequence files. However, an index scheme like the BoNDtree can significantly improve the search performance.

A candidate primer can be viewed as a box query having one or more (in case of degenerate primers) characters along each dimension. Further, techniques such as DNA synthesis or PCR (Polymerase Chain Reaction) need two primers to define the region of the sequence that is to be processed (e.g., amplifies in case of PCR). The two candidate primers can be combined together to form a larger box query which can accelerate the search. In this paper we present performance of Bond-tree in primer design applications.

II. BASIC CONCEPTS

In this paper we introduce significant geometric concepts extended from the CDS to the NDDS. Reminiscent of the indexing techniques in, the BoND-tree uses these geometric concepts to optimize the organization of indexed vectors during its construction time. Some essential geometric concepts such as rectangle, sphere, region area, and so on are no longer valid in an NDDS, where data values in each dimension cannot even be labelled on an (ordered) axis. Hence the above techniques cannot be directly applied to an NDDS. If the alphabet for every dimension in an NDDS is the same, a vector in such a space can be considered as a string over the alphabet. The distance measure between two vectors in a data space is important for building a multidimensional index tree. Unfortunately, those widely-used continuous distance measures such as the Euclidean distance cannot be applied to an NDDS.

A *discrete minimum bounding rectangle* (DMBR) of SR is such a discrete bounding rectangle that has the least area among all the discrete bounding rectangles of SR. The *span* of a DMBR R along dimension i is defined as the edge length of R along dimension i. In order to control the contribution of each dimension in the geometric concepts such as the area, a normalization is applied (i.e., the edge length of each dimension is normalized by the domain size of the corresponding dimension).

III. OPTIMIZATION OF INDEX TREES FOR BOX QUERIES IN THE NDDS

TABLE 1: Table of important symbols used in the paper box query I/O for hierarchical indexing structures. Here we discuss the splitting problem of index trees and show that box queries require specifically designed heuristics when building a tree.

Symbol	Explanation
d	Number of dimensions
dd	dimensional NDDS
Ai	Alphabet size of the ith dimension
R	Rectangle in d

Di	Component of R along the ith dimension
SR	Set of rectangles.in d
q	A fixed box query
Q	Random box query in d
w	Query window of q
W	Query window of Q

III.I Box Queries in the NDDS

A box query q on a data set in an NDDS is a query which is specified by listing the set of values that each dimension is endorsed to take. Given a hierarchical indexing structure, assume F (N, q) is
a Boolean function which proceeds true when and only when the query window of a box query q overlaps with the DMBR of a node N in an index tree, box query q is typically evaluated as follows: preliminary from the root node R (let N = R), the query window of q is compared with the DMBRs of all the child nodes of N. Any child node N0 for which F(N0, q) = 1 is recursively evaluate using the same process. However, if q does not overlap with a child node N00 (i.e., F(N00, q) = 0), N00 and its child nodes can be pruned from the search path. Assuming each node occupies one disk block, the query I/O is the total number of nodes accessed during the query process.

III.II Splitting Heuristics

The following heuristics for splitting an overflow node in the NDDS. The heuristics are applied in the order they are specified.

R1: Minimum Overlap

Of all the candidate partitions, heuristic R1 selects the one that results in the minimum overlap between the DMBRs of the newly created nodes. This heuristic is the same as the one used by some of the existing works.

R2: Minimum Span

If R1 generates more than one overlap-free partitions, heuristic R2 selects one of those partitions which is generated from splitting a dimension with the smallest span. This follows directly from theorem 2.

R3: Minimum Balance

Given a splitting dimension u, heuristic R3 chooses the most unbalanced overlap-free partition (i.e., the one that puts as few characters as possible in one node's DMBR and as many characters as possible in the other node's DMBR on dimension u) among all candidate partitions which satisfy the minimum utilization criterion and tied on R2. This follows directly from theorem.

IV. THE BOND-TREE CONSTRUCTION

Here we illustrate the data structure and significant algorithms for constructing the proposed BoND-tree.

IV.I Insertion procedure

A BoND-tree is a balanced indexing structure which has the following properties:
(1) All nodes must have at least a given minimum amount of space filled by indexed entries unless it is the root node (the minimum space utilization requirement);
(2) A leaf node entry structure has the form (V, P), where V is an indexed vector (key) and P is the pointer to the relevant tuple in the database corresponding to V;
(3) Each tree node occupies one disk block;
(4) A non-leaf node entry structure has the form (D, P), where D is the DMBR of the entry's corresponding child node and Pis the pointer to that child node.
(5) The root node has at least 2 indexed entries unless it is a leaf node;

Inserting a vector in the BoND-tree involves two steps. First, we find a suitable leaf node L for the new vector. Then we put the vector into L and update L's ancestor nodes' DMBRs as needed. The second step may cause a split of the leaf node (when an overflow occurs), which might trigger cascaded splits all the way to the root node.

i. *Selecting a Leaf Node*
ii. *Splitting an Overflow Node*

IV.II The Node Splitting Problem

Here we analyze how an overflow node N is split in the BoND-tree using heuristic R3. Suppose Sd is the disk block size occupied by each tree node and the minimum space utilization criterion requires that a certain size Smin of each node must be filled. Based on our discussion, the BoND-tree node splitting problem using heuristic R3 could be defined as follows.

Node Splitting Problem of the BoND-tree Using Heuristic R3 (NSP):
Given entry groups G1, G2, . . . , Gn in an overflow node N, suppose the number of characters (along the splitting dimension) and the storage space of each of the groups are GV1, GV2, . . . , GVn and GW1, GW2, . . . , GWn respectively. The BoND-tree splitting algorithm distributes the entry groups to two new nodes N1 and N2 such that,
(1) The total number of characters Vtotal is the maximum.
(2) Both NW1 and NW2 satisfy the minimum space utilization criterion of the tree (i.e., NW1 _ Smin and NW2 _ Smin).

Redefined Node Splitting Problem of the BoND-tree Using Heuristic R3 (RNSP):
Given entry groups G1, G2, . . . , Gn in an overflow node N, suppose the number of characters (along the splitting dimension) and the storage space of all groups are GV1, GV2, . . . , GVn and GW1, GW2, . . . , GWn respectively. The BoND-tree splitting algorithm distributes the entry groups to two new nodes N1 and N2 such that,
(1) The total number of characters Vtotal is the maximum.
(2) The total storage space Wtotal satisfies the constraint Wtotal _ Smax, where Smax .

IV.III The **Node Splitting Algorithm**

As the node splitting problem is mapped to the 0-1 knapsack problem, a dynamic programming solution can be used to solve it optimally and efficiently. Algorithm summarizes all the important steps involved in inserting a new entry into a tree node.
Algorithm 1: insert entry(N**,** E**)**
Input: A node N and an entry E to be inserted inN.
Output: Modified tree structure that accommodates entry E.

Method:
1. **if** N has space for E
2. Insert E in the list of entries in N
3. Update DMBR of N's parent node as needed
4. **else** // We need to split N
5. Record dimensions with span larger than 1 into a list L
6. Sort L based on dimension span in ascending order
7. **for** every dimension i in L **do**
8. Group entries in N based on their component sets on dimension i
9. Calculate each entry group's weight and value //mapped to the 0 − 1 Knapsack Problem
10. **if** N is a leaf node
11. Solve the special case of the 0 − 1 knapsack problem using the greedy approach
12. **else**
13. Solve the 0 − 1 knapsack problem using dynamic programming
14. **end if**
15. **if** a solution satisfying the minimum utilization criterion is found
16. **return** the solution
17. **end if**
18. **end for**
19. **if** no solution that is overlap-free and satisfies the minimum utilization criterion could be found
20. Generate candidate partitions based on the descending order of ri and select a partition with the least overlap
21. **return** the solution
22. **end if**
23. **end if**

Mapping the splitting problem **RNSP** into the 0-1 Knapsack Problem not only provides an efficient way to find the most suitable partition for an overflow node, but also allows the freedom of using different ways to build the BoND-tree based on the particular requirement and purpose of indexing.
IV.IV **Deletion in the BoND-tree**
If removing a vector from a leaf node L does not cause any underflow, the vector is directly removed and DMBRs of L's ancestor nodes are adjusted as needed. If an underflow occurs for L. An update operation can be implemented as a combination of deletion and insertion. In order to update a vector, we first delete it from the database, and insert the modified vector.
IV.V **Box Query on the BoND-tree**
The algorithm for executing box queries on the BoND-tree is implemented as follows. Let q be the query box and N be a node in the tree (which is initialized to root R of the tree). For each entry E in N, if the query window w overlaps with the DMBR of E, entry E is searched. Otherwise, the sub tree rooted at E is pruned.

V. COMPRESSION TECHNIQUE FOR THE BOND-TREE

V.I The Compressed BoND-tree Structure

In a non-leaf node entry of the compressed BoND-tree, we use one additional bit to indicate if the DMBR is full or not on each dimension. Only when it is not full, we record the occurrence of each character on that dimension. As the space requirement of a single DMBR is reduced, the fanout of the node increases. This high fanout results in reduction in the height of the tree and reduced I/O at the time of querying.

Note that the compression of DMBRs applies only to non-leaf nodes because the leaf node entry in the BoNDtree has only one character along each dimension. Thus the performance gain of the compressed BoND-tree is achieved through a more effective representation of DMBRs in the nonleaf nodes, especially nodes at higher levels of the tree.

V.II **Effect of Compression on Splitting Overflow Non-leaf Nodes**

When a non-leaf node entry's DMBR is split along one dimension, the resulting DMBRs may also shrink along other (full) dimensions. In a non-leaf ode N, the need for its splitting comes when one of its node entries E gets replaced with two new entries E0 and E00 (due to the split of a child node NE).

VI. OVERVIEW

The BoND-tree was implemented in C++. Experiments were conducted on machines with Intel Xeon quad-core processors with 8 GB ECC DDR2 RAM running SuSE Enterprise Linux 10 in a high performance computing cluster system. Performance of the proposed BoND-tree (with and without compression) was evaluated using synthetic data with various dimensions, alphabet sizes and database sizes (the number of vectors indexed).

Tree construction time- Issues:

- Impact of each heuristic on performance
- Effect of Different Database Sizes
- Effect of Different Numbers of Dimensions
- Effect of Alphabet Size
- Effect of Different Query Box Sizes
- BoND-tree with skewed data
- Comparison of running time

Application in Primer Design:

As explained earlier, box queries in NDDS are useful in primer design for genome sequence databases. In this section we present results of applying the BoND-tree for this application. In order to enable a sub-sequence search, the index is built of all possible overlapping sub-sequences (Q-grams) of a genome sequence having the given primer length.

Performance of the Compressed BoND-tree:

We also examined the performance of BoND-tree using the proposed compression strategy. First we show the performance gain for varying number of dimensions. The database size used for this set of tests is 5 millions. The number of vectors indexed is fixed at 5 millions, the number of dimensions is set to 16 and the query box size is 2. This set of tests demonstrates the effectiveness of the compression strategy when indexing NDDSs with different alphabet sizes.

VII. CONCLUSION

In this paper, we have presented a new indexing structure, called the BoND-tree, which exploits exclusive properties of the NDDS. Theoretical analysis of box queries in the NDDS shows that a better filtering power could be achieved using new splitting heuristics adopted by the BoND-tree. Our extensive experimental results using different alphabet sizes, database sizes, dimensions and query box sizes demonstrate that the BoND-tree isscp bon significantly more efficient than existing techniques such as the ND-tree and the 10% linear scan. Effectiveness of the BoND-tree in a real world application involving genome sequence databases is demonstrated. We also present the use of compression in the NDDS to further improve performance of the BoND-tree.

REFERENCES

1. J. Cl´ement, P. Flajolet, J. Cl´ement, B. Vall´ee, B. Vall´ee, T. G. Logiciel, and P. Algo, "Dynamical sources in information theory: A general analysis of trie structures," *Algorithmica*, vol. 29, pp. 307–369, 1999.
2. G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, November 1975.
3. Guttman, "R-Trees: a dynamic index structure for spatial searching, "*Proceedings of ACM SIGMOD*, pp. 47–57, 1984.
4. W. Loots and T. H. C. Smith, "A parallel algorithm for the 0−1 knapsack problem," *Int. J. Parallel Program.*, vol. 21, no. 5, pp. 349–362, 1992.
5. S. Meyn and R. Tweedie, *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993.
6. Robinson, J. T. 1981. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In Proc. of SIGMOD. 10{18.
7. J. L. Bentley and R. Sedgewick, "Fast algorithms for sorting and searching strings," *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pp. 360–369, 1997.
8. S. Berchtold, D. Keim, and H.-P. Kriegel, "The X-tree: an index structure for high-dimensional data," *Proceedings of the 22nd International Conference on VLDB*, pp. 28–39, 1996.

AUTHORS

P Jhansi Rani , I am pursuing Post Graduate in Master of Technology with specialization of Computer Science &Engg. at AVN Inst. of Engg.& Tech, Hyderabad, AP, India. My interested research area is Data warehousing & Data Mining, Network Security and Data Structures.

Mr. GK Srikanth is the Associate Professor, Dept. of CSE, AVN Inst. of Engg.& Tech, Hyderabad, AP, India. He received his M.Tech. from JNTU Hyderabad . His expertise areas are Network Programming and Networking Security.