Comparative Analysis of Proposed POBBx-Index Structure

K. Appathurai$^1$ and Dr. S. Karthikeyan$^2$

$^1$Department of Information Technology, Karpagam University
Coimbatore, Tamil Nadu
k_appathurai@yahoo.co.uk

$^2$Department of Information Technology, College of Applied Sciences,
Sultanate of Oman
skaarthi@gmail.com

Abstract

In this paper the proposed work is compared with the existing index methods like BBx-index and OBBx index structure. This paper proposes the new index structure called POBBx (Parameterized Optimal BBx) which indexes the positions of moving objects, given as linear functions of time, at any time. The index supports queries that select objects based on temporal and spatial constraints, such as queries that retrieve all objects whose positions fall within a spatial range during a set of time intervals. The proposed work reduces lot of searching efforts done by the existing methods and minimized time complexity. The simulation results shows that the proposed algorithm provides better performance than BBx-index and OBBx index structure.

Keywords: Moving Objects, POBBx index, BBx-trees, OBBx index and Migration.

1. Introduction

Spatio-temporal databases deals with moving objects that change their locations over time. In common, moving objects report their locations obtained via location-aware instrument to a spatio-temporal database server. Spatiotemporal access methods are underground into four categories: (1) Indexing the past data (2) Indexing the current data (3) Indexing the future data and (4) Indexing data at all points of time. All the above categories are having set of indexing structure algorithms [1- 4, 10, 13,21,22]. The server store all updates from the moving objects so that it is capable of answering queries about the past [4, 5, 8, 9, 15]. To predict future positions of moving objects, the spatio-temporal database server may need to store supplementary information, e.g., the objects’ velocities [7, 17]. Many query types are maintained by a spatio-temporal database server, e.g., range queries “Find all objects that intersect a certain spatial range during a given time interval”, k-nearest neighbor queries “Find k restaurants that are closest to a given moving point”, or trajectory queries “Find the trajectory of a given object for the past hour”. These queries may execute on past, current, or future time data. A large number of spatio-temporal index structures have been proposed to support spatio-temporal queries efficiently [12, 13].

2. BBx index Structure

The BBx-index consists of nodes that consist of entries, each of which is of the form (x_rep; tstart; tend; pointer.) For leaf nodes, pointer points to the objects with the equivalent x_rep, where x_rep is obtained from the space-filling curve; tstart denotes the time when the object was inserted into the database (matching to the tu in the description of the Bx-tree), and tend denotes the time that the position was deleted, updated, or migrated (migration pass on to the update of a position done by the system automatically). For non-leaf nodes, pointer points to a (child) node at the next level of the index: tstart and tend are the minimum and maximum tstart and tend values of all the entries in the child node, respectively. In addition, each
node contains a pointer to its right sibling to facilitate query processing. Unlike the Bx-tree, the BBx-index is a group of trees, with each tree having an associated timestamp signature tsg and a lifespan (see Figure 1). The timestamp signature parallels the value tlab from the Bx-tree and is obtained by partitioning the time axis in the same way as for the Bx-tree. The lifespan of each tree corresponds to the minimum and maximum lifespan of objects indexed in the tree. The roots of the trees are stored in an array, and they can be accessed efficiently according to their lifespan. This array is relatively small and can usually be stored in main memory.

3. OBBx INDEX Structure

The main aim of the OBBx (Optimal Broad Bx) algorithm is to decreases the complexity of BBx index structure. Besides the overall performance of the OBBx algorithm is good than BBx index about 40%. The scalability is considered as twice for the better result. The OBBx-index the nodes consist of the form (x_rep; tstart; tend; pointer.) where x_rep is nothing but one dimensional data obtained from the space-filling curve; tstart denotes the time when the object was inserted into the database and tend denotes the time that the position was deleted, updated, or migrated (migration refers to the update of a location done by the system). tstart and tend are the minimum and maximum tstart and tend values of all the entries in the child node, respectively. In addition, each node contains a pointer to its right sibling to facilitate query processing. The OBBx-index is a forest of trees, with each tree having an associated timestamp signature tsg and a lifespan. The timestamp signature parallels the value tlab from the Bx-tree and is obtained by partitioning the time axis in the same way as for the Bx-tree. The lifespan of each tree corresponds to the minimum and maximum lifespan of objects indexed in the tree. The roots of the trees are stored in an array, and they can be accessed efficiently according to their lifespan. This array is relatively small and can usually be stored in main memory. Initially the maximum update interval is found out among all the moving objects.

The maximum interval value is making it as twice for scalability. Figure 1 shows a BBx-index with n = 2. Objects inserted between timestamps 0 and 0:5tmu are stored in tree T1 with their positions as of time 0:5tmu; those inserted between timestamp 0:5tmu and tmu are stored in tree T2 with their positions as of time tmu; and so on. Each tree has a maximum lifespan: T1’s lifespan is from 0 to 1:5tmu because objects are inserted starting at timestamp 0 and because those inserted at timestamp 0:5tmu may be alive throughout the maximum update interval tmu, which is thus until 1:5tmu; the same applies to the other trees [10].

1. Find out the maximum update interval for each object and the maximum interval value is stored in ui.
2. The maximum update interval Ui is multiplied by two and then based on this scalability the linear array is formed for ts1,ts2,ts3, etc.,
3. Array of n equal intervals of ts1, ts2, ts3, etc
4. Each object lifespan are find out that is stored in LE.
5. Based on the lifespan the data are stored in the tree.
6. If the insertion node C is lesser than the node N then the node C inserted on left else inserted on right. If already the nodes are there the same way created and stored. The insertion time for each object is stored in the variable Arr and total object is inserted is stored in the variable Tot
7. For each move from one tree to another, While Arr not equal to Null, it is checked whether all the moving objects are reached to the new tree or not, if it is reached call the function update or else all the function migration.

Figure 1: Algorithm to Tree Construction, Object Insertion, Updation and Migration

Each tree has lifespan after that the tree values are updated to next tree. So first check whether all the objects are reached or not if it is reached then update all the objects to next tree and then the objects are removed or deleted from the existing old tree because to avoid duplication of index. The following algorithm shows how the updation takes place in OBBx. In this algorithm first identify the tree where the update object is located and then find out the position of the object in that tree and then the object is removed and updated in new tree from old tree.

Update Node[i] to ts[Pos-1]

Algorithm Update(Eo; En)
Input: Eo and En are old and new objects respectively
tindex ← time Eo is indexed in the tree
find tree Tx whose lifespan contain tindex
posindex ← position of Eo at tindex
keyo ← x-value of the posindex
locate Eo in Tx according to keyo
modify the end time of Eo’s lifespan to current time

Figure 2 : Algorithm for Update

4. Statement of Problem
In BBx index and OBBx index structure the searching process is one of the major calamity, during updation and migration process the searching took more time in BBx and OBBx index. Took more effort and time for the whole process of indexing. Due to this high effort the memory space utilization, processor utilization, execution time and cost increases. Besides in tree the node insertion, deletion also complex process when the number of moving objects is high.

5. Proposed Algorithm
The main aim of the proposed work is to reduce the searching process so that the efficiency is improved and got better result than BBx and OBBx index structures. While in case of BBx and OBBx index during node value updation or migration first it find tree Tx whose lifespan contain tindex again in that tree it find the position of the node and then based on the key value it find the node there the end time is changed to current time. So for each updation or migration the searching is the major role. In this proposed work the scalability is similar to OBBx
but not BBx index structure and during values transferred from one tree to another the old tree address and position also passed along with the moving object. So during updation or migration from one tree to another no need to search old tree and the position. So lot of searching time and effort is reduced. Due to this reduction of searching the node accesses also reduced, besides the utilization of memory also reduced and automatically the processing speed improved than OBBx index. All these are clearly mentioned in performance studies section.

The proposed algorithm of POBBx index is as follows,

1. Find out the maximum update interval for each object and the maximum interval value is stored in ui.
2. The maximum update interval Ui is multiplied by two and then based on this scalability the linear array is formed for ts1, ts2, ts3, etc.,
3. Array of n equal intervals of ts1, ts2, ts3, etc
4. Each object lifespan are find out that is stored in LE. In this Multi dimensional points representing object paths by Hilbert curve coordinates.
5. Based on the lifespan the data are stored in the tree.
6. If the insertion node C is lesser than the node N then the node C inserted on left else inserted on right. If already the nodes are there the same way created and stored. The insertion time for each object is stored in the variable Arr and total object is inserted is stored in the variable Tot.
7. For each move from one tree to another, While Arr not equal to Null, it is checked whether all the moving objects are reached to the new tree or not, if it is reached call the function update or else all the function migration.

Remove the object Eo from the tree Thisarr of the position Oodum.
Locate object En in the tree Curarr of the position Curpos.
Oodum ← □ indexed position of the object En.
Thisarr ← □ last tree value of the object En.
Migrate Node[i] to ts[Pos-1]
Algorithm Migrate(Eo; En)
Input: Eo and En are old and new objects respectively
Oodum ← □ indexed position of the object Eo.
Thisarr ← □ last tree value of the object Eo.
Curpos ← □ current position of the object En. □
Curtim ← □ current time of the object En
Curarr ← □ current tree of the object En which lifespan contains Curtim.

Figure 4: Algorithm to Tree Construction, Object Insertion, Updation and Migration

6. Performance Studies

The below figure 5 shows how the objects moving randomly in un specified path and it describes the clear path of the every moving objects. In this example 4 moving objects are consider for indexing. The starting time is 34 ms and the ending time is 212.882137762 ms, this is clearly shown in the figure 5. In figure 5 the x axis is time and y axis is points i.e. by Hilbert curve the multidimensional data is converted as points (single dimensional data).

Figure 5: This figure shows how the objects moving randomly in un specified path. And It describes the clear path of the every moving objects.
The below figure 6 shows how the processing speed are vary for all the three cases. The POBBx index performance is better than other methods.

![Figure 6: Comparison of processing speed of BBx, OBBx and POBBx](image)

Figure 6: Comparison of processing speed of BBx, OBBx and POBBx

Actually in this strategy the number of created tree is same in OBBx index method and POBBx Index techniques. But vary in BBx index method because of scalability. This is clearly shown in the below figure 7.

![Figure 7: Comparison of creation of number of trees in BBx, OBBx and POBBx](image)

Figure 7: Comparison of creation of number of trees in BBx, OBBx and POBBx

The below figure 8 shows the number of migration hits in all the three cases. The number of migration hits are same in OBBx index method and POBBx Index techniques. But vary in BBx index method because of scalability.

![Figure 8: Comparison of number of migration hits of BBx, OBBx and POBBx](image)

Figure 8: Comparison of number of migration hits of BBx, OBBx and POBBx

The below figure 9 shows the number of node access in all the three cases. In POBBx index method the searching process is less when compared with OBBx index method, so automatically the number of node access is very less than other methods.

![Figure 9: Comparison of number of node access of BBx, OBBx and POBBx](image)

Figure 9: Comparison of number of node access of BBx, OBBx and POBBx

The below figure 10 shows the storage requirement of all the three cases. In POBBx index method the searching process is less when compared with other two methods, so automatically the storage requirement also very
less than other methods. In BBx, OBBx during searching all the intermediate values are stored but no need in case of POBBx index structure.

Using MATLAB the following results are produced. The number of Moving Objects consider is 4

![Comparison of storage requirement of BBx, OBBx and POBBx](image)

Figure 10: Comparison of storage requirement of BBx, OBBx and POBBx

### 7. Results

<table>
<thead>
<tr>
<th>Aspects</th>
<th>BBx</th>
<th>OBBx</th>
<th>POBBx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time</td>
<td>1.359196e+001</td>
<td>6.663245e+000</td>
<td>5.936776e+000</td>
</tr>
<tr>
<td>No. of Trees</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Migration Hits</td>
<td>54</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Node Accesses</td>
<td>3.777000e+002</td>
<td>3.193000e+002</td>
<td>2.667000e+002</td>
</tr>
<tr>
<td>Storage Requirement</td>
<td>4.560000e+001</td>
<td>1.660000e+001</td>
<td>1.380000e+001</td>
</tr>
</tbody>
</table>

Table 1: comparison of BBx, OBBx and POBBx

### 8. Conclusion

This paper presents a new advanced indexing technique, the POBBx-index (Parameterized Optimal BBx-index), which can answer queries about the past, the present and the future. The POBBx -index is based on the concepts underlying the OBBx-index. Besides the POBBx -index is compared with BBx and OBBx index methods under 5 different aspects that is mentioned in table 1. The POBBx -index is better performance than other two methods. Moreover there is no change in number of trees created in both OBBx-index and POBBx-index. But lot of searching time is greatly reduced in POBBx-index than OBBx-index. So the processing speed is increased than OBBx-index structure. Wide performance studies were conducted that indicate that the POBBx-index outperforms the existing method, with respect of historical, present and predictive queries. In both the cases there is no change in number of migration hits. The Future work is planed to further reducing of migration hit and perk up the performance.

### References


Author’s Bibliography

K. Appathurai was born on 12th May 1974. He received his Master degree in Computer Applications from University of Bharathidasn in 1998. He completed his M.Phil from Manonmaniam Sundaranar University in 2003. He is working as an Asst. Professor and Head of the Department of Information Technology at Karpagam University, Coimbatore. Currently He is pursuing Ph.D. His fields of interest are Spatial Database.

Dr. S. Karthikeyan received the Ph.D. Degree in Computer Science and Engineering from Alagappa University, Karaikudi in 2008. He is working as a Professor and Director in School of Computer Science and Applications, Karpagam University, Coimbatore. At present he is in deputation and working as Assistant Professor in Information Technology, College of Applied Sciences, Sohar, Sultanate of Oman. He has published more than 14 papers in National/International Journals. His research interests include Cryptography and Network Security.