A New Proposed Algorithm for OBBx-index Structure

K. Appathurai
Asst.Prof. And Head
Department of Information Technology
Karpagam University

S. Karthikeyan
Director, School of computer Science
Karpagam University
Coimbatore – 21

ABSTRACT
Even though lot of spatio-temporal indexing techniques for moving objects are availed, some more intelligence has been given to the advance of techniques that professionally support queries about the past, present, and future positions of moving objects. 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 method and minimized time complexity. The simulation results shows that the proposed algorithm provides enhanced performance than OBB^x index structure.

Keywords
Moving Objects, $POBB^x$ index, OBB^x index, Migration and BB^x-trees.

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]. 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]. This paper is based on the source paper [21].

2. RELATEDWORK
Some modern appraisals of moving-object indexing techniques exist that focus on different views [1, 6, 7]. The first variant of indices includes the TPRtree (Time-Parameterized R-tree) family of indexes [2, 5]. One of the initial works is the Historical R-tree (HR-tree) [18], which logically constructs a "new" R-tree each time an update occurs. Repetition of object is the major negative aspect of R-tree. After R-tree Pfoser et al. propose the Spatio-Temporal R-tree (STR-tree) and the Trajectory-Bundle tree (TB-tree). Yongquan Xia, Weili Li , and Shaohui Ning, Moving Object Detection Algorithm Based on Variance Analysis [16] is derived. Besides Muiti-Version 3D R-tree (MV3R-tree) [19] is proposed by Tao and Papadias. Then, B. Liu. Querying about the Past, the Present, and the Future in Spatio-Temporal Databases [20]. Besides Dan Lin, Christian S. Jensen, and Ooi proposed [10] algorithm to supports queries about the past, present, and future A recent algorithm is proposed by K.Appathurai , S.Karthikeyan [21,22] to supports queries about the past, present, and future .

3. OBBx INDEX Structure

The main aim of the OBB^x (Optimal Broad B^x) algorithm is to decreases the complexity of BB^x index structure. Besides the overall performance of the OBB^x algorithm is good than BB^x index about 40%. The scalability is considered as twice for the better result. The OBBx-index the nodes consist of the form (x _rep; t_{start}; t_{end}; 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 OBB^x-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 B^x-tree and is obtained by partitioning the time axis in the same way as for the B^x-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 BB^x-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.

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.

Fig 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 OBB^x. 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

t'index ⟵ time En will be indexed

pos'index ⟵ position of En at t'index

keyn ⟵ x-value of the pos'index

insert En into the latest tree according to keyn

Fig 2: Algorithm for Update

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 any object is not reached then that object is identified and then migrated to next tree. Next that objects are removed or deleted from the existing old tree because to avoid duplication of index. The following algorithm shows how the migration process takes place in OBB^x. In this algorithm first identify the tree where the migrate object is located and then find out the position of the object in that tree and then the object is removed and migrated in new tree from old tree.

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

Algorithm Migrate(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

Fig 3: Algorithm for Migrate

4. STATEMENT OF PROBLEM

In OBB^x index structure the searching process is one of the major crisis, during updation and migration process the searching took more time in OBB^x index. and took more efffort 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 OBB^x index structure. While in case of OBB^x 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 OBB^x index 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 OBB^x index. All these are clearly mentioned in performance studies section.

The proposed algorithm of $POBB^x$ 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.
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.

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

Algorithm Update(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.

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.

Fig 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 6 moving objects are consider for indexing. The starting time is 45 ms and the ending time is 205.98214875 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).

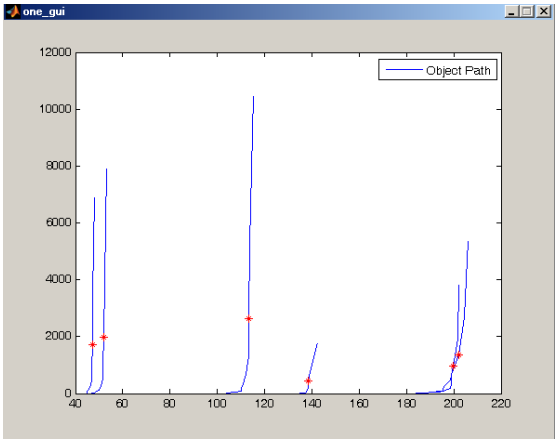

Fig 5 : This figure shows how the objects moving randomly in un specified path. And It describes the clear path of the every moving objects.

The below figure 6 shows how the processing speed are vary for all the three cases. The OBB^x index performance is better than other methods.

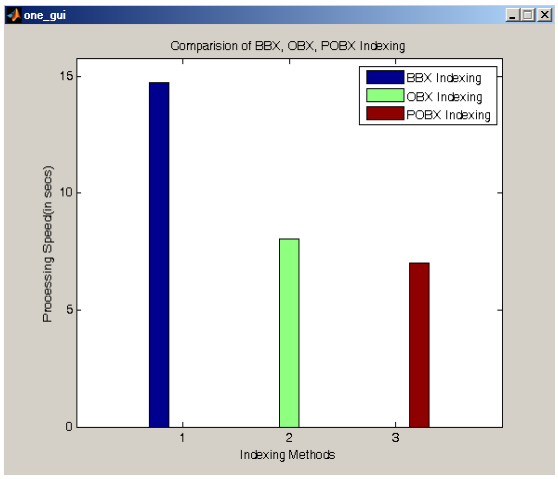

Fig 6 : This figure shows the comparison of processing speed of BB^x,OBB^x and $POBB^x$

Actually in this strategy the number of created tree is same in OBB^x index method and $POBB^x$ Index techniques. But vary in BB^x index method because of scalability. This is clearly shown in the below figure 7.

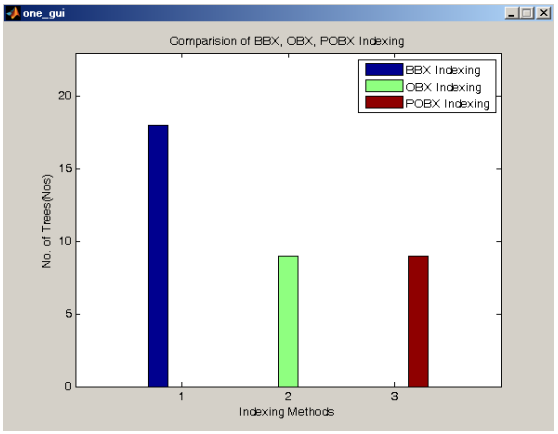

Fig 7 : This figure shows the comparison of creation of number of trees in BB^x,OBB^x and POBB^x

The below figure 8 shows the number of migration hits in all the three cases. The number of migration hits are same in OBB^x index method and POBB^x Index techniques. But vary in BB^x index method because of scalability.

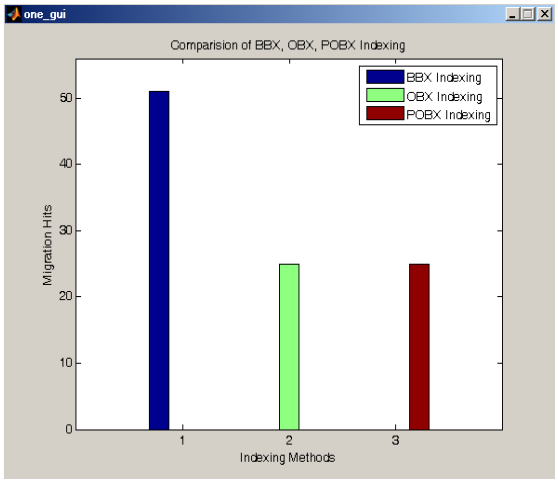

Fig 8 : This figure shows the comparison of number of migration hits of BB^x,OBB^x and POBB^x

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

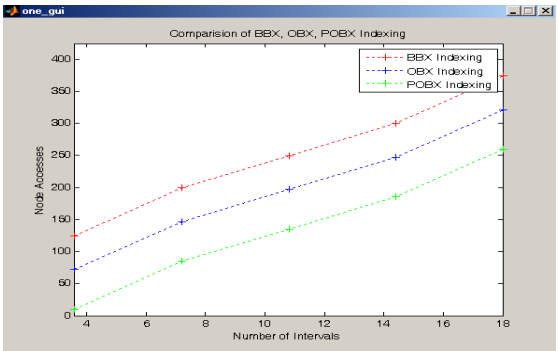

Fig 9 : This figure shows the comparison of number of node access of BB^x,OBB^x and POBB^x

7. RESULTS

Using MATLAB the following results are produced.

The number of Moving Objects consider is : 6

Starting Time : 45.00000000

Ending Time : 205.98214875

For BBX, Maximum Anticipated Time Interval : 9.22356835
For OBX, Maximum Anticipated Time Interval : 18.44713669
For POBX, Maximum Anticipated Time Interval : 18.44713669
Processing Time for BB^X Indexing : 1.473893e+001
Processing Time for OBB^X Indexing : 8.053991e+000
Processing Time for POB^X Indexing : 7.022134e+000
Migration Hits for BBX Indexing : 51
Migration Hits for OBB^X Indexing : 25
Migration Hits for POB^X Indexing : 25
Node Accesses for BB^X Indexing : 3.747000e+002
Node Accesses for OBB^X Indexing : 3.217000e+002
Node Accesses for POB^X Indexing : 2.601000e+002

8. CONCLUSION

This paper presents a advanced indexing technique, the POBBx-index (Parameterized Optimal BBx-index), which can answer queries about the past, present and the future positions of moving objects. The POBBx -index is based on the concepts underlying the OBBx-index. It avoids duplicating objects like the OBBx-index and thus achieves significant space saving and efficient query processing. Moreover there is no change in number of trees created in both OBBx-index and POBBx-index. But lot of searching time is vastly reduced in POBBx-index than OBBx-index. So the processing speed is increased than OBBx-index structure. Extensive 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 improve the performance.

8. REFERENCES

[1] Long-Van Nguyen-Dinh, Walid G. Aref, Mohamed F. Mokbel 2010. Spatio-Temporal Access Methods: Part 2 (2003 - 2010). Bulletin of the IEEE Computer SocietyTechnical Committee on Data Engineering

[2] M. Pelanis, S. ˇ Saltenis, and C. Jensen. Indexing the past, present, and anticipated future positions of moving objects.TODS, 31(1):255–298, 2006.

[3] Z.-H. Liu, X.-L. Liu, J.-W. Ge, and H.-Y. Bae. Indexing large moving objects from past to future with PCFI+-index. In COMAD, pages 131–137, 2005.

[4] V. Chakka, A. Everspaugh, and J. Patel. Indexing large trajectory data sets with SETI. In CIDR, 2003

[5] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In VLDB, 2003.

[6] C. Jensen, D. Lin, and B. Ooi. Query and update efficient B+-tree based indexing of moving objects. In VLDB, 2004.

[7] M. Mokbel, T. Ghanem, andW. G. Aref. Spatio-temporal access methods. IEEE Data Eng. Bull., 26(2):40–49, 2003.

[8] J. Ni and C. V. Ravishankar. PA-tree: A parametric indexing scheme for spatio-temporal trajectories. In SSTD, 2005.

[9] P. Zhou, D. Zhang, B. Salzberg, G. Cooperman, and G. Kollios. Close pair queries in moving object databases. In GIS, pages 2–11, 2005.

[10] Dan Lin, Christian S. Jensen, Beng Chin Ooi, Simonas Sˇ altenis, BBx index :Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects, MDM 2005 Ayia Napa Cyprus

[11] P. K. Agarwal and C. M. Procopiuc. Advances in Indexing for Mobile Objects. IEEE Data Eng. Bull., 25(2): 25–34, 2002.

[12] G. Kollios, D. Gunopulos, V. J. Tsotras. On Indexing Mobile Objects. In Proc. PODS, pp. 261–272, 1999.

[13] K.Appathurai, Dr. S. Karthikeyan. A Survey on Spatiotemporal Access Methods.International Journal of Computer Appliations. Volume 18, No 4, 2011.

[14] [14] Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid G. Aref, Continuous Query Processing of Spatio-temporal Data Streams in PLACE, 2004 Kluwer Academic Publishers. Printed in the Netherlands

[15] Su Chen · Beng Chin Ooi · Zhenjie Zhang, An Adaptive Updating Protocol for Reducing Moving Object Database Workload.

[16] Yongquan Xia, Weili Li , and Shaohui Ning, Moving Object Detection Algorithm Based on Variance Analysis, 2009, Second International Workshop on Computer Science and Engineering Qingdao, China

[17] Arash Gholami Rad, Abbas Dehghani and Mohamed Rehan Karim, Vehicle speed detection in video image sequences using CVS method, 2010, International Journal of the Physical Sciences Vol. 5(17), pp. 2555-2563.

[18] M. A. Nascimento and J. R. O. Silva. Towards Historical R-trees. In Proc. ACM Symposium on Applied Computing, pp. 235–240, 1998.

[19] Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In Proc. VLDB, pp. 431–440, 2001.

[20] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the Past, the Present, and the Future in Spatio-Temporal Databases. In Proc. ICDE, pp. 202–213, 2004.

[21] K. Appathurai, Dr. S. Karthikeyan (2012), "A New Proposed Algorithm for BBx-Index Structure", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 3, No 1, May 2012

[22] K. Appathurai, Dr. S. Karthikeyan (2012), " A Novel Indexing Method for BBx- Index structure", Int.J.Computer Technology & Applications,Vol 3 (2), 779-784

13. AUTHORS PROFILE

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, Sulatanate of Oman. He has published more than 14 papers in Natrional/International Journals. His research interests include Cryptography and Network Security