Spatio-temporal Access Methods[*]

Mohamed F. Mokbel Thanaa M. Ghanem Walid G. Aref

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398
{mokbel,ghanemtm,aref}@cs.purdue.edu

Abstract

The rapid increase in spatio-temporal applications calls for new auxiliary indexing structures. A typical spatio-temporal application is one that tracks the behavior of moving objects through location-aware devices (e.g., GPS). Through the last decade, many spatio-temporal access methods are developed. Spatio-temporal access methods focus on two orthogonal directions: (1) Indexing the past, (2) Indexing the current and predicted future positions. In this short survey, we classify spatio-temporal access methods for each direction based on their underlying structure with a brief discussion of future research directions.

1 Introduction

Spatio-temporal databases deal with objects that change their location and/or shape over time. A typical example of spatio-temporal databases is moving objects in the D-dimensional space. Moving objects learn about their own location via location detection devices, e.g., GPS devices. Then, the objects report their locations to the server using the underlying communication network, e.g., via wireless networks. The server stores the updates from the moving objects and keeps a history of the spatio-temporal coordinates of each moving object. In addition, the server stores additional information to help predict the future positions of moving objects. Typical queries that are supported by such a server include time slice queries e.g., "*Find all objects that cross a certain area at time t*" and window queries "*Find all objects that cross a certain area in the time interval* $[t_1, t_2]$". Time slice queries and window queries may ask about the past, current, or future times. Some queries are concerned only with the past, e.g., trajectory queries "*What is the maximum speed of a certain object in the last hour?*" Other queries are concerned only with the future, e.g., moving window queries "*Find the objects that intersect a moving area in a certain time interval*".

Numerous research have been done in developing spatio-temporal access methods as an auxiliary structure to support spatio-temporal queries. Figure 1 gives the evolution of spatio-temporal access methods with the underlying spatial and temporal structures. Lines in the Figure indicate the relation between a new proposed spatio-temporal index structure and the original structure that is based upon.

The rest of this paper is organized as follow: Section 2 surveys spatio-temporal indexing methods that index the past (i.e., index historical spatio-temporal data). In Section 3, we survey spatio-temporal indexing methods that keep track of the current status of spatio-temporal data. Section 4 surveys the spatio-temporal indexing methods that help answer queries related to the future. In Section 5, we give an overview of available indexing toolkits that can help in implementing spatio-temporal access methods. Finally, Section 6 concludes the paper.

[*]This work was supported in part by the National Science Foundation under Grants IIS-0093116, EIA-9972883, IIS-0209120, and by Purdue Research Foundation.

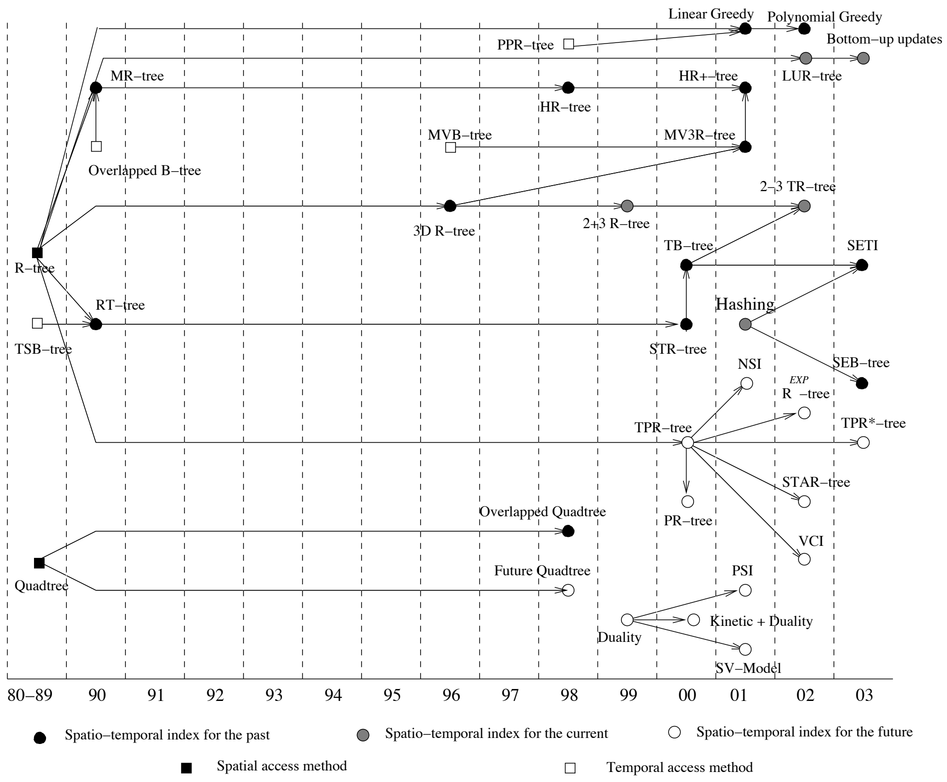

Figure 1: Survey of Spatio-temporal Access Methods.

2 Indexing the Past

In this section, we are interested in indexing methods for historical spatio-temporal data. The size of the history is continuously increasing over time. Consider moving objects that continuously send their positions. Keeping track of all updates is almost infeasible. Two approaches are used to minimize the history size: (1) Sampling. The stream of data is sampled at certain time positions. Linear interpolation may be used between sample points to form trajectory lines. (2) Update on change only. Moving objects send information only when their data is changed (e.g., change in speed or direction). We categorize the spatio-temporal indexing schemes for historical data into three categories. The first category augments the temporal aspect into already existing spatial access methods. The second category manages both the spatial and temporal aspects into one structure. The third category takes a more radical step by indexing mainly the temporal dimension, while treating the spatial dimension as second priority. The following sections overview these three categories and their corresponding access methods.

2.1 Dealing with the Temporal Dimension

In this category of spatio-temporal indexing methods, the main concern is to handle the spatial domain. Dealing with temporal queries is considered as a secondary issue.

RT-tree [45]: The RT-tree combines the foundation of the R-tree [14] as a spatial access method and the TSB-tree [24] as a temporal access method. In the RT-tree, a new entry is added to the regular R-tree that indicates the start and end times of the current object. An RT-tree entry is of the form (id, MBR, t_s, t_e), where id is the object identifier, MBR is the objects minimum bounding rectangle, and t_s and t_e give the time interval in which this object is valid. The RT-tree supports spatial queries as efficient as the regular R-tree. However time slice queries and interval queries may span the whole tree.

3D R-tree [41]: The 3D R-tree treats time as yet another dimension in addition to the spatial dimensions. The main idea is to avoid discrimination between spatial and temporal queries. The 3D R-Tree supports both the temporal and spatial queries, although with performance drawbacks. A main drawback is that timeslice queries are no longer dependent on the live entries at the query time, but on the total number of entries in the history.

STR-tree [28]: The STR-Tree is an extension of the R-Tree, with a different insert/split algorithm. Leaf nodes are in the form (id, t_{id}, MBR, o) where t_{id} is the trajectory identifier and o is the orientation of this trajectory in the MBR. The main idea of the STR-Tree is to keep spatial closeness and partial trajectory preservation by trying to keep line segments belonging to the same trajectory together while keeping spatial closeness as the R-Tree. A parameter p is introduced to balance between spatial properties and trajectory preservation. p indicates the number of levels reserved for trajectory preservation. When inserting a new line segment the goal is to insert it as close as possible to its predecessor in the trajectory within p levels. A smaller p decreases the trajectory preservation, while increasing the spatial closeness.

2.2 Overlapping and Multi-version Structures

In the second category of spatio-temporal indexes, the temporal dimension is discriminated from the spatial dimensions. The goal is to keep all spatial data that are alive at one time instance together in one index structure (e.g., the R-tree). The ultimate goal is to build a separate R-tree for each time instance. This approach requires excessive storage.

MR-tree [45]: The MR-tree employs the idea of overlapping B-trees [6] in the context of the R-tree. The main idea is to avoid the storage overhead of having separate R-trees for each timestamp. The saving in storage is achieved by not storing the common objects among consecutive R-trees. Instead, links from different roots point to the same nodes where all the node entries keep their values over the different timestamps. This idea is perfect in the case of a time slice query. The search is directed to the appropriate root, and then a spatial search is performed using the R-tree. However, the performance of time window queries is not efficient. Also, one major drawback is that many entries can be replicated. Consider the case that only one node entry is changed over two consecutive timestamps, then all other node entries need to be replicated in two consecutive R-trees.

HR-tree [25]: The Historical R-tree (HR-tree) is very similar to the MR-tree. The HR-tree has a concrete algorithm and implementation details of using the overlapping B-tree [6] in the context of the R-tree. The same idea of overlapping trees is applied in the context of quadtrees, where it results in *overlapping quadtrees* [43].

HR+-tree [37]: The HR+-tree is designed mainly to avoid the replication of some entries in the HR-tree. The main reason for having duplicate entries in the HR-tree is that the HR-tree has a condition that any node can contain only entries that belong to the same root, i.e., ones that have the same timestamp. The HR+-tree relaxes this condition by allowing entries from different timestamps to reside in the same node. However, the parent of this node in each R-tree has only access to the entries that belong to the parent's timestamp. In other words, a node may have multiple parents, where each parent has access only to a different part of the node.

MV3R-tree [38]: The MV3R-tree is based mainly on the multi-version B-tree (MVB-tree) [5]. The main idea is to build two trees, an MVR-tree to process timestamp queries, and a 3D R-tree to process long interval queries. Short interval queries are optimized to check which tree is to be used based on a threshold value.

Greedy algorithms in the PPR-tree [20]: The partially-persistent R-tree (PPR-tree) [21], designed mainly for bi-temporal databases, is extended to support spatio-temporal applications [20]. However, this would result in highly dead space. To overcome the dead space, artificial object updates are introduced. An optimal greedy algorithm [20] is used to find the optimal locations for the artificial updates in linearly moving objects. This work is extended in [15] to support objects moving using a combination of polynomial functions.

2.3 Trajectory-Oriented Access Methods

The third category of spatio-temporal access methods focus on trajectory-oriented queries. Dealing with spatial queries and gathering spatially closed objects together is of second concern.

TB-tree [28]: The Trajectory-bundle tree (TB-tree) is an R-tree-like structure that strictly preserves trajectories. A leaf node can only contain segments belonging to the same trajectory. As a drawback, line segments of different trajectories that lie spatially close will be stored in different nodes. The TB-tree grows from left to right. The left-most leaf node is the first inserted node and the right-most leaf node is the last inserted one. The TB-tree is an extension of the STR-tree to handle only trajectories.

SETI [8]: The Scalable and Efficient Trajectory Index (SETI) partitions the spatial dimension into static, non-overlapping partitions. The main observation is that the change of the spatial dimension is limited while the temporal dimension is continuously evolving. Thus, the spatial dimensions are partitioned statically. Within each partition the trajectory segments are indexed using an R-tree. Using a good partitioning function results in having line segments of the same trajectory stored in the same partition. Thus, trajectory preservation is achieved by minimizing the effect of the spatial dimensions in the R-tree. A segment that crosses the boundary of two spatial partitions is clipped and is stored twice in both partitions. This may lead to duplicates in the query result.

The SEB-tree [35]: The Start/End timestamp B-tree (SEB-tree) has an idea similar to SETI, where the space is partitioned into zones that may be overlapped. Each zone is indexed using the SEB-tree that considers only the start and end timestamps of the moving objects. Each moving object is hashed to its zone. A key difference over SETI is that there are no trajectories. Instead only two-dimensional points are indexed. By having the spatial zoning partitioning, two-dimensional points that belong to similar trajectories are kept together.

3 Indexing the Current Positions (NOW)

All previous spatio-temporal indexing techniques assume that all movements are known a priori. Thus, only closed trajectories are stored. Current positions of moving objects are neither stored nor queried. The issue of the current positions, or the NOW positions is challenging [10]. In the following, we give an overview of spatio-temporal index structures that help answer queries about NOW.

The 2+3 R-tree [26]: The 2+3 R-tree aims to index both the current and past information of moving objects. The main idea is to have two separate R-trees; one for the current two-dimensional points, and the second for the historical three-dimensional trajectories (two spatial dimensions and one temporal dimension). This idea is similar to the one proposed in the context of the bi-temporal indexes [21]. Once the current object is updated, the object trajectory is constructed with its three-dimensional MBR, and is inserted into the three-dimensional R-tree while being deleted from the two-dimensional R-tree. Depending on the query time, both trees may need to be searched.

The 2-3 TR-tree [1]: The 2-3 TR-tree has the same idea as the 2+3 R-tree where the 2-3 TR-trees also keeps two separate R-trees; a two-dimensional R-tree for the current objects, and a three-dimensional R-tree for the historical data. However, two differences can be distinguished: (1) In the 2-3 TR-tree, the three-dimensional R-tree keeps track of only the multi-dimensional points but not the trajectories; thus avoids the problem of high dead space. (2) The 2-3 TR-tree uses the underlying structure of the TB-tree to allow answers for trajectory-oriented queries.

The LUR-tree [22]: The Lazy Update R-tree (LUR-tree) is concerned only with the current positions of spatio-temporal objects. No historical data is stored into the LUR-tree. Once an object updates its location, the object's old entry is deleted and the new entry is inserted. The LUR-tree aims to handle the frequent updates of moving objects without degrading the performance of the R-tree index structure. The main idea is that as long as the new position of the moving object lies inside its MBR, there is no action taken other than updating the position. Once an object moves out from its MBR, two approaches are proposed: (1) The object is deleted and to be investigated in spatio-temporal indexing. Most of the work so far supports selection operators and range queries. More research is needed to support other kinds of operators (e.g., spatio-temporal join) and queries (e.g., nearest-neighbor queries).

References

[1] M. Abdelguerfi , J. Givaudan, K. Shaw, and R. Ladner. The 2-3 TR-tree, A Trajectory-Oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets. In *Proc. of the ACM workshop on Adv. in Geographic Info. Sys., ACM GIS*, pages 29–34, Nov. 2002.

[2] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. of the ACM Symp. on Principles of Database Systems, PODS*, pages 175–186, May 2000.

[3] W. G. Aref and I. F. Ilyas. SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees. *Journal of Intelligent Information Systems , JIIS*, 17(2-3):215–240, 2001.

[4] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. of the ACM-SIAM symposium on Discrete algorithms, SODA*, pages 747–756, 1997.

[5] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An Asymptotically Optimal Multiversion B-Tree. *VLDB Journal*, 5(4):264–275, 1996.

[6] F. W. Burton, J. G. Kollias, D. G. Matsakis, and V. G. Kollias. Implementation of Overlapping B-trees for Time and Space Effi cient Representation of Collections of Similar Files. *The Computer Journal*, 33(3):279–280, 1990.

[7] M. Cai and P. Revesz. Parametric R-Tree: An Index Structure for Moving Objects. In *Proc. of the Intl. Conf. on Management of Data, COMAD*, Dec. 2000.

[8] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets with SETI. In *Proc. of the Conf. on Innovative Data Systems Research, CIDR*, Asilomar, CA, Jan. 2003.

[9] H. D. Chon, D. Agrawal, and A. E. Abbadi. Storage and Retrieval of Moving Objects. In *Mobile Data Management*, pages 173–184, Jan. 2001.

[10] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Trans. on Database Systems , TODS*, 22(2), 1997.

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

[12] GiST: http://gist.cs.berkeley.edu/.

[13] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.-B. Yu. Processing Queries By Linear Constraints. In *Proc. of the ACM Symp. on Principles of Database Systems, PODS*, pages 257–267, May 1997.

[14] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD*, pages 47–57, June 1984.

[15] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Effi cient Indexing of Spatiotemporal Objects. In *Proc. of the Intl. Conf. on Extending Database Technology, EDBT*, pages 251–268, Czech Republic, Mar. 2002.

[16] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, pages 562–573, Sept. 1995.

[17] A. Henrich, H.-W. Six, and P. Widmayer. The lsd tree: Spatial access to multidimensional point and nonpoint objects. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, pages 45–53, Aug. 1989.

[18] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD*, pages 369–380, May 1997.

[19] G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. In *Proc. of the ACM Symp. on Principles of Database Systems, PODS*, pages 261–272, June 1999.

[20] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing Animated Objects Using Spatiotemporal Access Methods. *IEEE Trans. on Knowledge and Data Engineering, TKDE*, 13(5):758–777, 2001.

[21] A. Kumar, V. J. Tsotras, and C. Faloutsos. Designing Access Methods for Bitemporal Databases. *IEEE Trans. on Knowledge and Data Engineering, TKDE*, 10(1):1–20, 1998.

[22] D. Kwon, S. Lee, and S. Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In *Mobile Data Management, MDM*, pages 113–120, Jan. 2002.

[23] M. Lee, W. Hsu, C. Jensen, B. Cui, and K. Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, Sept. 2003.

[24] D. B. Lomet and B. Salzberg. Access Methods for Multiversion Data. In *Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD*, pages 315–324, May 1989.

[25] M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proc. of the ACM Symp. on Applied Computing, SAC*, pages 235–240, Feb. 1998.

[26] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis. Evaluation of Access Structures for Discretely Moving Points. In *Proc. of the Intl. Workshop on Spatio-Temporal Database Management, STDBM*, pages 171–188, Sept. 1999.

[27] R. C. Nelson and H. Samet. A Consistent Hierarchical Representation for Vector Data. In *Proc. of the ACM SIGGRAPH*, pages 197–206, Aug. 1986.

[28] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, pages 395–406, Sept. 2000.

[29] K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In *Proc. of the Intl. Symp. on Advances in Spatial and Temporal Databases, SSTD*, pages 59–78, Redondo Beach, CA, July 2001.

[30] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.

[31] C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled. STAR-Tree: An Effi cient Self-Adjusting Index for Moving Objects. In *Proc. of the Workshop on Alg. Eng. and Experimentation, ALENEX*, pages 178–193, Jan. 2002.

[32] S. Saltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. of the Intl. Conf. on Data Engineering, ICDE*, Feb. 2002.

[33] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD*, pages 331–342, May 2000.

[34] Z. Song and N. Roussopoulos. Hashing Moving Objects. In *Mobile Data Management*, pages 161–172, Jan. 2001.

[35] Z. Song and N. Roussopoulos. SEB-tree: An Approach to Index Continuously Moving Objects. In *Mobile Data Management, MDM*, pages 340–344, Jan. 2003.

[36] SP-GiST: http://www.cs.purdue.edu/homes/aref/dbsystems_fi les/SP-GiST/index.html.

[37] Y. Tao and D. Papadias. Effi cient Historical R-trees. In *Proc. of the Intl. Conf. on Scientific and Statistical Database Management, SSDBM*, pages 223–232, July 2001.

[38] Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, pages 431–440, Sept. 2001.

[39] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In *Proc. of the Intl. Conf. on Very Large Data Bases, VLDB*, Sept. 2003.

[40] J. Tayeb, Ö. Ulusoy, and O. Wolfson. A Quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

[41] Y. Theodoridis, M. Vazirgiannis, and T. Sellis. Spatio-Temporal Indexing for Large Multimedia Applications. In *Proc. of the IEEE Conference on Multimedia Computing and Systems, ICMCS*, June 1996.

[42] TPR-tree: http://www.cs.auc.dk/TimeCenter/software.htm.

[43] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. Overlapping Linear Quadtrees: A Spatio-Temporal Access Method. In *Proc. of the ACM workshop on Adv. in Geographic Info. Sys., ACM GIS*, pages 1–7, Nov. 1998.

[44] D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *Proc. of the Intl. Conf. on Data Engineering, ICDE*, pages 516–523, Feb. 1996.

[45] X. Xu, J. Han, and W. Lu. RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In *Proc. of the Intl. Symp. on Spatial Data Handling, SDH*, pages 1040–1049, July 1990.