Indexing Moving Object Trajectories With Hilbert Curves

Reaz Uddin, Chinya V. Ravishankar, Vassilis J. Tsotras

University of California, Riverside, CA, USA
{uddinm,ravi,tsotras}@cs.ucr.edu

ABSTRACT

Efficiently querying large trajectory datasets is a challenge of growing importance. Abstracting trajectory segments with minimum bounding boxes and indexing them in R-Trees results in a high false positive rate due to high dead space. Space filling curves (SFCs), which have excellent locality preserving and dimensionality reduction properties, have been shown to be effective for indexing points in space. However, they can yield a high false positive count and slow query times if used to index trajectory segments. Our work shows how to use SFCs to index trajectory polylines. In our experiments, the proposed method runs 2–15 times faster than other state-of-the-art approaches.

CCS CONCEPTS

• **Information systems → Database query processing**;

ACM Reference Format:
Reaz Uddin, Chinya V. Ravishankar, Vassilis J. Tsotras , *University of California, Riverside, CA, USA,* {uddinm,ravi,tsotras}@cs.ucr.edu . 2018. Indexing Moving Object Trajectories With Hilbert Curves. In *Proceedings of ACM SIGSPATIAL conference (SIGSPATIAL '18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3274895.3274912

1 INTRODUCTION

The problem of efficiently querying large trajectory datasets has grown in importance in recent years. Large trajectory repositories are now common, thanks to the widespread use of mobile technologies, the ubiquity of cellular networks, and the increased accuracy of GPS devices. Location-based services offer applications that need fast query support over such datasets. Given the data volumes, efficient indexing methods are required. While this problem has already received considerable attention, in this paper we describe a new approach for indexing spatiotemporal data using traditional technologies such as R-trees and Hilbert curves. Our experiments show that the proposed method significantly outperform existing approaches.

Trajectory Model: For simplicity, we assume that trajectories correspond to objects moving in a 2-dimensional space. For our purposes, a trajectory consists of a unique identifier for the object, and a *polyline* whose endpoints are tuples of the form (x, y, t), denoting the spatial coordinates of the object at time t. Such tuples may come from sensor readings, GPS updates, social network check-ins, and so on. We assume that objects travel in straight lines between consecutive observations (x_i, y_i, t_i) and $(x_{i+1}, y_{i+1}, t_{i+1})$, as long as $t_{i+1} - t_i < \delta$, for some threshold δ. Whenever $t_{i+1} - t_i \geq \delta$, we assume that the object has started a new trip or trajectory. This allows us to create meaningful application-specific trajectories, using different δ thresholds. Multiple trajectories may exist for the same object, representing different trips during different non-overlapping time intervals. Modeling an object's movement as a sequence of line segments rather than as a collection of endpoints allows us to answer queries about the object's intermediate positions.

Space filling curves (SFCs) such as the Hilbert curve (HC) and the Z-curve reduce the dimensionality of the native space, and have been shown to be effective for indexing spatial objects. SFCs have hitherto been mainly used to index multidimensional points [1–3] or order arbitrary objects [4, 5]. Orenstein [6, 7] provided a general approach by which any spatial object can be represented as a collection of ranges, where a range starts when the SFC enters the spatial object's area and ends when the SFC exits it. Each such range corresponds to a continuous SFC range that overlaps the spatial object. A spatial range query is translated into a collection of range queries in the SFC space.

An important property of an SFC is how well it preserves clustering or proximity, so that points that are close to each other in the original space remain close to each other in the transformed space. This important property reduces the disk I/O for hierarchical indexing methods. Another important property is the average number of SFC ranges per object. Better proximity preservation and a lower number of ranges result in better performance. Among the SFCs that have been studied, the Hilbert curve has been shown to have better proximity preservation [8–10], and lower average number of ranges per object [11].

The contributions of this paper are as follows:

- We show how to address some problems of representing trajectories using Hilbert curves. We separate the temporal and spatial dimensions because of their inherent semantic differences, and index spatio-temporal information with a new index structure (the HT-index) based on 2D R-Trees.
- Through an extensive experimentation we validate the model and show that our proposed method outperforms current state of the art approaches. It is usable in any database system supporting R-Trees.

The rest of the paper is organized as follows: Section 2 describes related work while Section 3 provides background on Hilbert curves. Section 4 presents the proposed HT-Index. Section 5 presents the experimental results and Section 6 concludes the paper.

SIGSPATIAL '18, November 6–9, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274912

2 RELATED WORK

Spatial Range Queries: In [1–3] the Hilbert curve is used to represent locations of moving objects. The B^x-Tree [1] indexes current and future moving object positions. It partitions the time axis into fixed intervals according to the maximum duration between two updates from a moving object. The value indexed for each object update is the concatenation of the index partition and the Hilbert representation of the object's position at that time. As time passes the earliest interval (and its index) expires and a new interval is added. The ST^2B-Tree [2] also considers predictive queries and uses a similar approach to the B^x-Tree but allows for different HC resolution based on the moving object density. In contrast, the BB^x-Tree [3] keeps one B^+-Tree for each time interval and thus preserves the past positions of the moving objects; it can thus answer 'historical' queries as well. One key property of these B-tree based methods is that moving object updates need to occur within a maximum interval; that is, a moving object has to report its position within that interval. The methods we propose in this paper, do not have this limitation.

[4, 5] use Hilbert numbers not to represent but instead to sort spatial objects. The Hilbert R-tree [4] uses a HC to decide object insertion order. The idea is to improve R-tree performance by inserting objects together that are close to each other in the actual space. [5] focuses on indexing trajectories of objects moving on some constrained network (road network, etc.) Edges of the road network are sorted according to the Hilbert number representing their midpoint. Then edges are assigned 1D non-overlapping adjacent intervals of length proportional to the length of the edge. The spatial coordinates of trajectory segments are mapped to 1D intervals by corresponding them to network edges. Network edges and trajectories are indexed separately and evaluating a range query requires searching both indices.

The trajectory indexing methods that do not use SFCs can be categorized by the index they use (an R-tree [12] or cell-based partitioning). Among the R-tree methods, the STR-Tree and TB-Tree [13] focus on clustering segments of the same trajectory close by in the index leaves, while the TPR-Tree [14] is for predictive queries; as all these approaches use MBRs to index segments, they have the 'dead' space disadvantage. TrajStore [15], facilitates a quad tree to store cells that are created by partitioning the spatial domain based on the trajectory density. The time dimension is not used explicitly in the index (conceptually, all trajectories are projected on their spatial coordinates) but as an interval that is assigned to each data page within a cell. At query time, cells are picked based on the query spatial range; out of the pages that a cell contains, the method considers only those pages with appropriate time interval. Having a good estimate of a cell's density is important for the method's efficiency. However, the cell density is estimated by using all the endpoints of the trajectories crossing the cell, which implies that the location update frequency should be high (so that that there is an endpoint in the cell for a fast object that passes through the cell) and the same for all moving objects (so that fast and slow objects contribute the same in the density estimation). Such requirements may be limiting for many location-based applications.

All the above methods consider time as a separate dimension that is explicitly or implicitly added to the index. Work on indexing temporal data [16] has proposed yet another approach for indexing time, that of partial persistence [17]. Consider an object that was at spatial position e_1 at time t_1 and moved to e_2 at time t_2. This move can be approximated as a segment (e_1, e_2) that first appeared at time t_1 and 'lived' until time t_2. As time proceeds, new segments are created and 'live' until they expire. The problem is then translated into maintaining a data structure that can maintain such segment evolution. This is achieved by taking an 'ephemeral' data structure that can solve the problem for a single time instant and making it partially-persistent [17] so as to also maintain the temporal evolution. This is the approach taken by the multi-version R-tree (MVR-tree) [18] and the MV3R-tree [19] for storing spatial objects that change over time. The multi-version approach typically provides very fast query times for single time instant (or short time interval) queries. Nevertheless, it introduces additional space through a controlled duplication (so as objects are temporally clustered). Moreover, this approach still uses some 'dead space', since in the above example the whole segment (e_1, e_2) is approximated from time t_1 even though the object was only at e_1 at that instant.

3 BACKGROUND

A ***trajectory*** is a polyline with endpoints $T = (x_0, y_0, t_0), \ldots, (x_n, y_n, t_n)$, $t_i < t_{i+1}$ for $i = 0, 1, \ldots, n-1$. A ***trajectory segment*** is a straight line between two consecutive location updates $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})$ of the same moving object, where $i \in \mathbb{N}_0$. A ***subtrajectory*** of length m of the trajectory T, is a subsequence $T' = (x_i, y_i, t_i), \ldots, (x_{m+i}, y_{m+i}, t_{m+i})$, of m contiguous trajectory segments, where $i \geq 0, m < n$.

A Hilbert curve [20] is a self avoiding, continuous curve that goes through every point of a discretized multidimensional space exactly once. We thus assume that the spatial domain is represented by a $N \times N$-**grid** where $N = 2^l, l = 0, 1, 2, \ldots$. Each cell in this grid corresponds to a point; i.e., there are a total of N^2 possible locations (points) in the spatial domain all of which are visited once by the HC. Clearly, the higher the l (also called the *order* of the Hilbert Curve) the higher the space resolution.

4 THE HT-INDEX, A HYBRID STRUCTURE

The key to our method is a hybrid strategy. We assign Hilbert numbers to the spatial coordinates, and use them as one of the dimensions in a $2D$ R-Tree, making time the second dimension. A trajectory segment (whether an original segment or one resulting from a split) with endpoints $(x_i, y_i, t_i), (x_k, y_k, t_k)$ is represented by the MBR with corners $(h_i, t_i), (h_k, t_k)$, where h_i and h_k are the Hilbert number assigned to points (x_i, y_i) and (x_k, y_k). Such MBRs are then indexed using a $2D$ R-tree. We call this approach the Hilbert Trajectory Index or *HT-Index*.

Consider a Hilbert curve $\mathcal{H}$ and a spatial object S, which may either be a query rectangle or a line segment. Let h^S_{min} and h^S_{max} be the minimum and the maximum Hilbert number on the sections of $\mathcal{H}$ that overlap S. The range of Hilbert numbers $[h^S_{min}, h^S_{max}]$ is called a *run.* This run will cover the points of S as well as many points not overlapping S, since the Hilbert curve may leave and enter S many times during the course of this run. Each continuous part of the run $[h^S_{min}, h^S_{max}]$ lying outside the object S is called a *jump.*

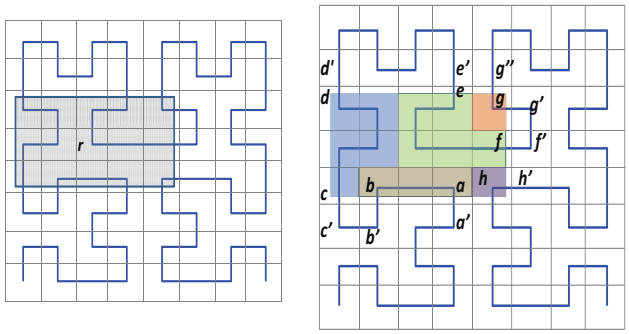


Figure 1: Runs of a range query.

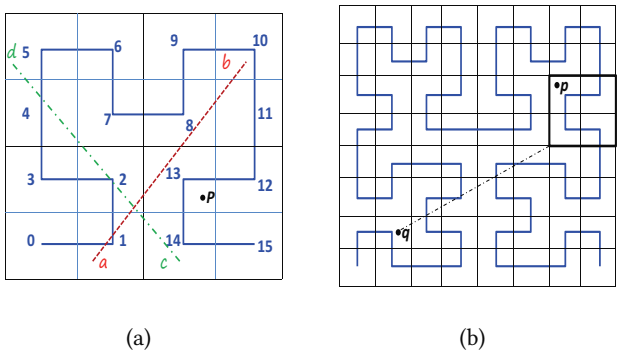


Figure 2: (a) False negatives when using Hilbert range. (b) Distance between a query point and a sub-grid.

For example, Figure 1(a) shows a 3rd-order Hilbert curve used to cover the space, and a spatial object, which is a query range rectangle R. As in Orenstein's approach [7], to avoid all points in the run $[h^R_{min}, h^R_{max}]$ that are not in the rectangle, the run is split into smaller runs, namely, $[a, b]$, $[c, d]$, $[e, f]$, $[g, g]$ and $[h, h]$. We now have the following jumps: $[b', c']$, $[d', e']$, $[f', g']$ and $[g'', h']$, as seen in Figure 1(b). The number of cells in a jump is the length of the jump. For instance, the length of $[d', e']$ is $e' - d' + 1$.

We observe that most of the jump lengths are very small, except a few that are quite high. Merging two runs with a small jump between them will add a few redundant points, possibly resulting in a few false positives. However, merging runs with big jumps between them will result in many false positives and increase the query time. It may be more efficient to run another range query instead.

4.1 Range Query Evaluation

We consider spatiotemporal range queries that include a spatial range and a time interval (3D queries, in effect). The query's spatial range is first mapped to the overlapping cells of the underlying grid, enlarging the query rectangle, as needed, to align with the nearest cell boundaries. We call this *query enlargement*. The spatial range is then split into a number of *query runs* as already described. Runs with small jumps are now merged to form *merged runs*. Finally, the query temporal interval is combined with each of them to obtain the *ht*-MBRs.

Effectively, the original query is divided into many smaller range queries, resulting in a set of *ht*-MBRs. The HT-index is queried for these *ht*-MBRs, obtaining a set of *ht*-MBRs representing spatio-temporal trajectory segments. However, we need to validate these results, since they may contain false positives. A direct way to detect false positives would be to take the *result runs* in the search result from the *ht*-MBRs, recover their spatial (x, y) coordinates and check if they are within the spatial range specified by the query.

Instead, we use the query runs and result runs to avoid converting Hilbert numbers to spatial coordinates. False positives may occur because of (1) query enlargement, or (2) merging of query runs, resulting in jumps. Let R_q be the query rectangle. We verify the query result as follows. Let R_{out} be the smallest rectangle aligned with the grid boundaries, with $R_q \subseteq R_{out}$. Similarly, let R_{in} be the largest rectangle aligned with grid cell boundaries, with $R_{in} \subseteq R_q$. Let R_{mid} be the region between R_{out} and R_{in}.

If a result run r_h does not overlap with any of the original query runs (before merging) then the corresponding trajectory segment is outside R_{out} and resulted from merging query runs. Then we check if r_h overlaps with the runs for R_{in}. If yes, then it belongs to the query result. We only need to convert the Hilbert numbers of r_h and the cells in R_{mid} to spatial coordinates when r_h is neither inside R_{in} nor outside R_{out}. Thus by using R_{out} and R_{in} we can verify most of the results efficiently.

5 EXPERIMENTAL EVALUATION

Our experiments were all run on an Intel Xeon 3.0GHz processor running Linux 2.6.18 with 8GB of main memory. C++ was used for implementation. We use the GeoLife dataset [21], which contains public activity data (shopping, dining, sightseeing, hiking, and cycling) in Beijing, China.

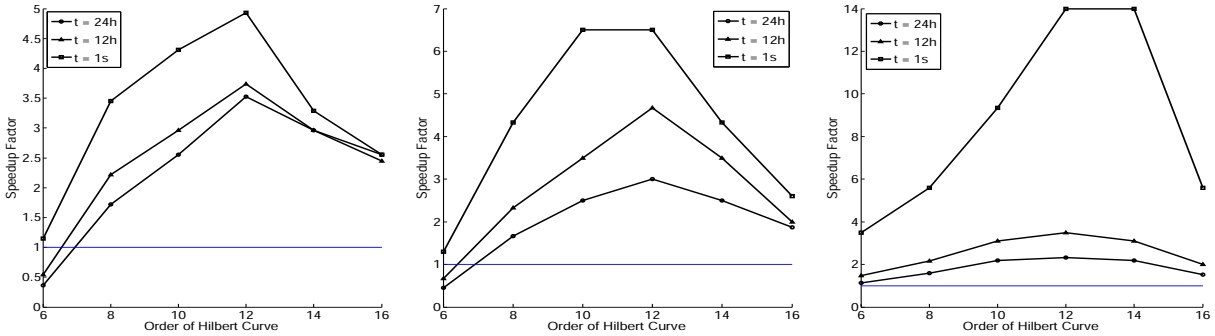
5.1 Parameters Influencing Performance

In our first set of experiments, we evaluated query performance for queries with spatial extent of 0.1%, 1% and 10% of the total area. For temporal extent we use random one second (timestamp), 12 hours and 24 hours of time interval. We also experimented with different threshold values for merging the runs of a range query. To evaluate query performance for various cell sizes we vary Hilbert order from 6 to 16. Our region of interest is a 4° latitude by 4° longitude area. An order-16 Hilbert curve in this region yields grid cells that are approximately 7m by 7m.

5.2 Comparisons with Competing Methods

In our second series of experiments, we compared our method with 3D R-tree for range queries. We ran 100K, 10K, and 1K instances of queries with spatial ranges covering 0.1%, 1%, 10% of the whole region, respectively.

We compared the speedup obtained by using our method, defined as the ratio of the running time for the 3D R-tree to that of our method. Figure 3 shows the speedup factor. For each spatial extent we use three temporal extents, of 1 sec, 12 hrs and 24 hrs, respectively. A higher speedup of up to 14 times is obtained for



(a)0.1% Queries (b)1% Queries (c)10% Queries

Figure 3: Speedup Factor compared to 3D R-Tree.

timestamp queries, as the R-tree performs poorly for timestamp queries. As we increase the order of the Hilbert curve, the speedup factor increases and then it starts to decrease, because with lower-order Hilbert curves, the bigger grid cells result in greater query enlargements. Besides, merging runs with bigger cell size result in more false positives and higher search time. The performance deterioration at higher order will be explained with the cost model later in this section.

6 CONCLUSION

In this paper we propose techniques to improve query performance on trajectory data by utilizing the Hilbert curve to represent trajectory polylines. Our proposed method outperforms the traditional 3*D* R-tree and can be easily integrated with any RDBMS that supports R-Trees. As future work we will devise an accurate cost model that will enable the user to identify the appropriate Hilbert order that optimizes performance for a given dataset.

Acknowledgement. This work was partially supported by NSF grant IIS-1527984.

REFERENCES

[1] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and update efficient b$^+$-tree based indexing of moving objects," in *VLDB*, 2004.
[2] S. Chen, B. C. Ooi, K.-L. Tan, and M. A. Nascimento, "St^{2}b-tree: A self-tunable spatio-temporal b$^+$-tree index for moving objects," in *SIGMOD*, June 2008.
[3] D. Lin, C. S. Jensen, B. C. Ooi, and S. Saltenis, "Efficient indexing of the historical, present, and future positions of moving objects," in *MDM*, 2005.
[4] I. Kamel and C. Faloutsos, "Hilbert r-tree: An improved r-tree using fractals," in *VLDB*, 1994, pp. 500–509.
[5] D. Pfoser and C. S. Jensen, "Indexing of network constrained moving objects," in *GIS*, November 2003.
[6] J. Orenstein, "A class of data structures for associative searching," in *SIGACT*, 1984.
[7] J. A. Orenstein, "Spatial query processing in an object-oriented database system," in *SIGMOD*, 1986.
[8] P. Xu and Tirthapura, "A lower bound on proximity preservation by space filling curves," in *IPDPS*, 2012.
[9] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," in *TKDE*, 2001.
[10] P. Xu and S. Tirthapura, "On optimality of clustering through a space filling curve," in *PODS*, 2012.
[11] H. V. Jagadish, "Analysis of the hilbert curve for representing two-dimensional space." in *Information Processing Letters*, 1997.
[12] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.
[13] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object trajectories," in *VLDB*, 2000.
[14] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing positions of continuously moving objects," in *SIGMOD*, vol. 29, no. 2, 2000, pp. 331–342.
[15] P. Cudre-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," in *ICDE*, 2010.
[16] B. Salzberg and V. J. Tsotras, "Comparison of access methods for time-evolving data," in *ACM Computating Survey*, vol. 31, no. 2, 1999, pp. 158–221.
[17] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, "Making data structures persistent," in *Journal of Computer and System Sciences*, vol. 38, no. 1, 1989, pp. 85–124.
[18] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos, "Indexing spatiotemporal archives," *VLDB J.*, vol. 15, no. 2, pp. 143–164, 2006.
[19] Y. Tao and D. Papadias, "The mv3r-tree : A spatio-temporal access method for timestamp and interval queries," in *VLDB*, 2001.
[20] D. Hilbert, "Uber die stetige abbildung einer linie auf ein flachenstuck." in *Mathematische Annalen*, vol. 38, 1891, pp. 459–460.
[21] http://research.microsoft.com/en-us/projects/geolife/.