Geoinformatica (2019) 23: 1–36
https://doi.org/10.1007/s10707-018-0329-2

Spatio-temporal access methods: a survey (2010 - 2017)

Ahmed R. Mahmood[1] · **Sri Punni**[1] · **Walid G. Aref**[1]

Received: 11 January 2018 / Revised: 10 September 2018 / Accepted: 25 September 2018 /
Published online: 9 October 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
The volume of spatio-temporal data is growing at a rapid pace due to advances in location-aware devices, e.g., smartphones, and the popularity of location-based services, e.g., navigation services. A number of spatio-temporal access methods have been proposed to support efficient processing of queries over the spatio-temporal data. Spatio-temporal access methods can be classified according to the type of data being indexed into the following categories: (1) indexes for historical spatio-temporal data, (2) indexes for current and recent spatio-temporal data, (3) indexes for future spatio-temporal data, (4) indexes for past, present, and future spatio-temporal data, (5) indexes for spatio-temporal data with associated textual data, and (6) parallel and distributed spatio-temporal systems and indexes. This survey is Part 3 of our previous surveys on the same subject (Mokbel et al. IEEE Data Eng Bull 26(2):40–49, 2003; Nguyen-Dinh et al. IEEE Data Eng Bull 33(2):46–55, 2010). In this survey, we present an overview and a broad classification of the spatio-temporal access methods published between 2010 and 2017.

Keywords Spatio-temporal data · Indexing · Databases

1 Introduction

With the popularity of location-aware devices, e.g., smartphones and GPS devices, many spatio-temporal applications have emerged, e.g., traffic analysis, and navigation applications. In these applications, moving objects periodically report their timestamped geo-locations to a spatio-temporal database server. Then, the server stores these timestamped location-updates for further processing. Some applications require the retention of the entire history of the spatio-temporal data, e.g., security and surveillance applications. In contrast,

✉ Ahmed R. Mahmood
amahmoo@cs.purdue.edu

Sri Punni
svelagap@purdue.edu

Walid G. Aref
aref@purdue.edu

[1] Purdue University, West Lafayette, IN, USA

other applications keep only the most recent history of the spatio-temporal data due to privacy agreements, e.g., cell phone companies cannot keep location data longer than specific durations. An alternative class of applications needs to maintain only the current locations of the moving objects, e.g., taxi and ride-sharing mobile applications. Applications, e.g., traffic analysis, may store additional information, e.g., the objects' velocities and directions in the spatio-temporal database server. This additional data is useful for predicting future positions of moving objects. Several spatio-temporal access methods have been proposed to speed-up query processing over spatio-temporal data. Nowadays, spatio-temporal data can also be associated with textual content, e.g., tweets. New access methods have been developed to index spatio-temporal data with associated textual content.

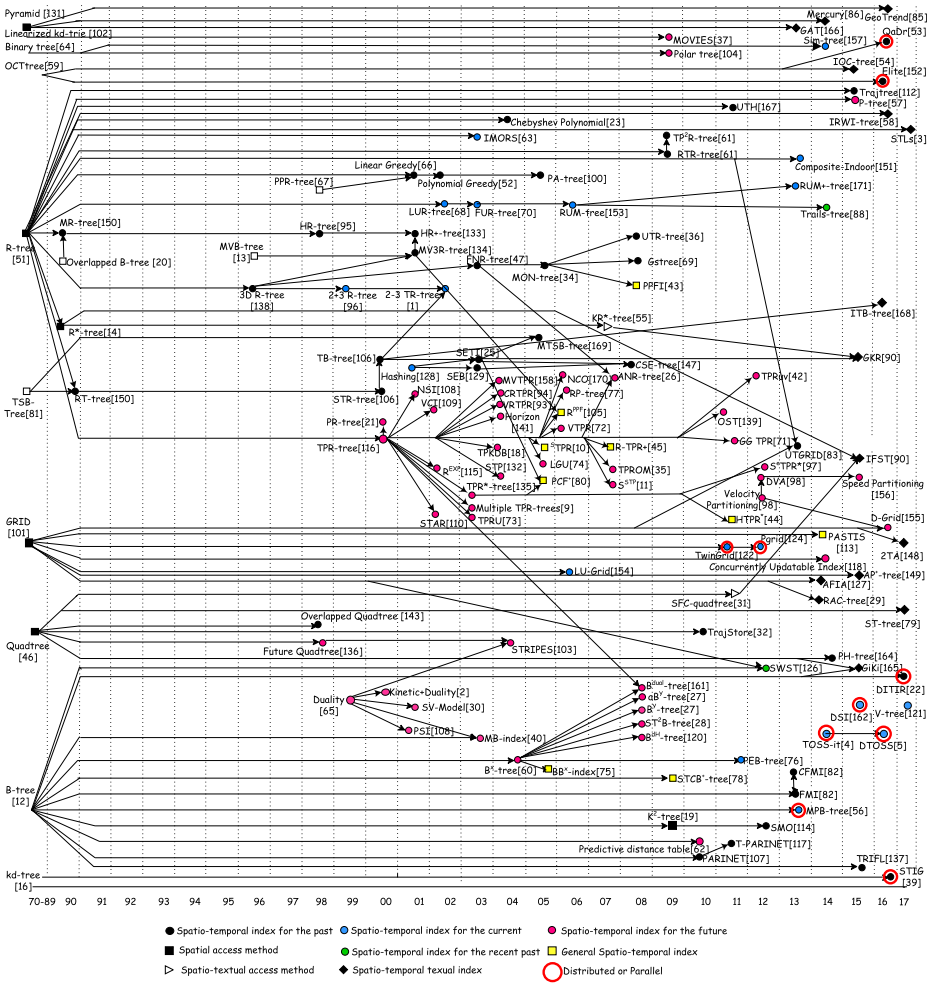

Fig. 1 The evolution of spatio-temporal access methods

In the past, we have surveyed spatio-temporal access methods that have been published on or before 2003 [91], and then from 2003 to 2010 [99]. This survey is Part 3 of this series of surveys that covers and classifies the new spatio-temporal access methods published between the years 2010 and 2017. Spatio-temporal access methods that only apply existing data structures in the context of spatio-temporal data are not covered in the survey. For example, spatio-temporal access methods that simply use a loose quadtree [89, 144] are not covered as they simply use an existing access method, in this case, a loose quadtree.

Figure 1 illustrates the spatio-temporal access methods developed between 2010 and 2017. Lines in the Figure indicate the relationship between a new spatio-temporal index structure and the original index structure it has evolved from.

The rest of this paper proceeds as follows: Sections 2, 3, and 4 present an overview of the spatio-temporal access methods for historical, current (and recent historical), and future data, respectively. Section 5 surveys spatio-temporal access methods for indexing data at all points in time. Section 6 overviews access methods for spatio-temporal and textual data. Section 7 overviews parallel and distributed spatio-temporal indexes. Finally, Section 8 concludes the survey.

2 Indexing the past

In this section, we present an overview of the access methods for indexing historical spatio-temporal data. Storing the entire historical data of moving objects is not feasible when moving objects update their locations frequently due to the massive volume of this data. To address this issue, moving objects may report location updates only when there is a significant change in their location. Alternatively, sampling can be used to shrink the volume of indexed spatio-temporal data. Linear or nonlinear interpolation can be used to reconstruct trajectories from the sampled location data. We classify spatio-temporal access methods that index past spatio-temporal data based on the approach adopted for handling the time and the space dimensions, and whether or not the index accounts for the sequential nature of the historical trajectories of the moving objects.

2.1 Multi-dimensional structures

In the category, space and time are handled as dimensions in the multi-dimensional space. All dimensions of the indexed spatio-temporal data are treated similarly. In other words, for the indexed spatio-temporal data, there is no distinction in the handling of the spatial and the temporal dimensions.

PH-tree [164] The PH-tree (PATRICIA-hypercube-tree) is a multi-dimensional data structure that extends both the Quad-tree and the PATRICIA-trie to optimize the search performance and the space utilization when indexing large amounts of multi-dimensional data. A k-dimensional object has k attributes that represent the object's position in the k-dimensional space. The PH-tree follows the Quad-tree in that the PH-tree partitions the space across all dimensions at any given node. However, instead of using an integer or a floating point representation of the attributes of the indexed objects, the PH-tree serializes the attributes of the indexed objects using binary representation. Objects are indexed in a PATRICIA-trie-like approach that uses the binary bit-representation of the attributes of the

indexed objects. The PH-tree uses common prefixes among the bit representation of the attributes to reduce the space required by the index. The PH-tree can be seen as a hyper-cube of size 2^k when indexing data with k dimensions. However, this hyper-cube is typically sparse and the PH-tree automatically switches to a linear representation of the hyper-cube that stores bit prefixes and pointers to children nodes when sparsity in the data is detected.

2.2 Overlapping and multi-version structures

In this category of spatio-temporal indexes, the treatment of the spatial dimensions is different from that of the temporal dimension. The idea behind this category of indexes is to build a separate index for each time instance.

SMO-index [114] The focus of the *S*uccinct *M*oving *O*bject Index (SMO-index) is the efficient processing of timestamp and interval spatio-temporal queries while reducing the storage requirements of the index. In the SMO-index, both the data and the index are stored compactly in the same structure without using external memory. The SMO-index has two main components: (1) a time-ordered sequence of snapshots of the objects' locations indexed by K^2-trees [19], and (2) a sequence of movement logs, where each log is the time-ordered sequence of changes in the objects' locations between consecutive snapshots. A Snapshot S is of the form $< Tree, Leaves, Labels >$, where $Tree$ and $Leaves$ are bitmaps that represent the internal nodes and the leaves of the K^2-tree, respectively. $Labels$ is an array of object identifiers. Instead of storing the absolute position of an object at each time instance in the movement logs, movement in the horizontal and vertical axes with respect to the previous location of the object is stored. The reason is to optimize the storage needed by the movement logs. Timestamp and interval queries are processed by accessing the snapshots and movement logs. Accessing a snapshot is similar to processing a range query on a K^2-tree.

2.3 Trajectory-oriented access methods

Trajectory-oriented access methods focus on answering topological and navigational queries over trajectories. Topological queries focus on the locations visited during the movements of the objects. In contrast, navigational queries focus on the information that can be inferred from the movements of objects, e.g., the speed and the direction of an object.

TrajStore [32] The main idea behind TrajStore is to partition the trajectories (that result from the movements of the objects) into sub-trajectories, and then cluster into disk pages the trajectory segments that are spatially and temporally close to each other. TrajStore targets to minimize the number of disk reads required for queries over specific spatial regions with large time-intervals. An adaptive quadtree is used as a spatial index, where each cell in the tree corresponds to multiple disk pages that contain the sub-trajectories in the area covered by this cell. A sparse temporal index is associated with each cell. This temporal index contains the least start-timestamp and the highest end-timestamp for each page within a specific cell. These cells are recursively split and merged according to a cost function that ensures optimal cell size and minimizes disk I/Os. Each cell is further compressed to avoid redundancy in cases where many trajectories have approximately the same path in a cell. Spatio-temporal range query processing is executed by first filtering the cells using the spatial index, and then by retrieving the pages that overlap the temporal range of the query.

PARINET [107, 117] PARINET (short for *PAR*titioned *I*ndex for in-*NE*twork *T*rajectories) is an access method for retrieving the historical trajectories of moving objects over road networks. PARINET partitions the road network and trajectory data based on the data distribution and the network topology. The time intervals for the trajectory data within each partition are indexed using a B^+-tree. A Road-Partitioning table (RP, for short) is maintained to store information about the partitions of both the spatio-temporal data and the road network. An entry of a partition, say P, within RP contains the list of road identifiers covered by P and a pointer to the B^+-tree of the time intervals in Partition P. A network-constrained spatio-temporal range query Q is represented in the form (Q_s, Q_t), where Q_s is a list of road identifiers and $Q_t = [t_s, t_e]$, t_s is the start-timestamp, and t_e is the end-timestamp of the query. The spatio-temporal range query Q is processed by first finding the set of partitions that contain the road identifiers in the query. Then, a range scan is performed on the B^+-tree index of the selected partitions to identify the candidate moving objects that overlap Q_t. Each candidate is further filtered to check if it satisfies both spatial and temporal conditions of Q.

T-PARINET [117] Temporal PARINET indexes trajectories by periodically creating new PARINET indexes for specific time windows. A new PARINET is created when the performance degradation and lifespan of the current index exceed specific thresholds. The structure of the new PARINET is based on the expected data distribution and the expected query load. A time-partitioning table is used to maintain the time window of each PARINET and a pointer to the PARINET. A spatio-temporal range query Q is of the form (Q_s, Q_t), where Q_s is the spatial range of Q and Q_t is the temporal range of the Q. If the temporal range $Q_t = [t_s, t_e]$ of the query exceeds the time window of the current PARINET, Q_t is split into k intervals using the time-partitioning table. Then, the original query is converted into a set of k queries that are executed on their corresponding PARINET indexes.

UTH [167] Trajectories may have uncertain portions. When moving objects report discrete location updates to the spatial database server, there is no information about the object's movement between consecutive location updates, and hence the uncertainty in the trajectory. A spatio-temporal index that accounts for this uncertainty assumes a specific model of movement between location updates for the uncertain trajectories. The *U*ncertain *T*rajectory *H*ierarchy indexes uncertain trajectories on road networks in the following way. UTH assumes a time-dependent probability distribution for the uncertain movement between discrete location updates. UTH consists of three main components: (1) an edge hash table, (2) a movement R-tree (mR), and (3) a trajectory list. An edge in the road network can be retrieved effectively from the edge hash table using the edge ID. For every edge, a 1D movement R-tree (mR) is maintained. Movement R-trees index the time periods in which the objects are moving on an edge. An entry in mR for an edge $e(v_i, v_j)$ and an object a moving on the edge is of the form $[t_{ea}(v_i), t_{ld}(v_j)]$, where t_{ea} and t_{ld} are the earliest arrival-time and latest departure-time of the associated vertex, respectively. This entry represents the *maximum time interval* (MTI) for Object a while being on e. There is one entry for each possible path of an object moving on Edge e. The trajectory list contains actual trajectory data. In the trajectory list, trajectory samples are sorted by their timestamps per moving object. Spatio-temporal range queries are processed in a filter-refine approach. First, edges that overlap the spatial range of the query are identified, and the movement trees of edges are queried to find candidate entries. Then, candidate entries are further refined by calculating a qualification probability per entry. The qualification probability of an entry with respect

to a query represents the likelihood of the object to belong to the resultset of the query. The qualification probability $QP^r_{a,q}$ quantifies the probability of Object a being within network distance r from Query q. Candidate entries with qualification probability below a specific threshold are removed from the query's resultset. To find candidate edges, an expansion tree $ET(q, r)$ for Query q is created. This tree is rooted at q, and contains all the positions along the edges of the road network that are within Distance r from q, where r represents the spatial range of the query. Then, using UTH, the mR tree of every candidate edge within the expansion tree is queried to find the entries with $t_q \in MTI$. The candidate paths that are the paths pointed at by these entries are further refined by calculating a qualification probability and checking if the path qualification probability meets a specific threshold.

UT$_{GRID}$ [83] The *U*ncertain *T*rajectories *GRID* is designed to answer top-k similarity queries over uncertain trajectories. The top-k similarity query identifies the k most-similar trajectories to a query trajectory. UT$_{GRID}$ is designed as a spatial-first index that partitions the indexed space into uniform non-overlapping cells. Within a grid cell, say C, a 1D R-tree is used to index the trajectories that overlap Cell C according to the probability of overlap between the trajectories and C. Trajectories are partitioned into segments to be indexed in UT$_{GRID}$. Trajectory segments that span multiple grid cells are split at grid cell boundaries. An indexed trajectory segment within a grid cell C consists of the trajectory identifier, the time span of the trajectory segment, and the probability of overlap between the trajectory and C. To avoid using complex probability functions to represent the probability of overlap between trajectory segments and cells, the entire temporal range is split into small time-intervals. Then, the probability of overlap between a trajectory and a grid cell is represented as a sequence of pairs (ϕ, ϵ), where ϕ is the average probability of overlap between a trajectory and the grid cell in this small time-interval, and ϵ is the maximum deviation between the exact probability of overlap and ϕ.

FMI [82] *F*oot*M*ark *I*ndex is used for the efficient processing of the *time-period most-frequent path query (TPMFP, for short)*. The TPMFP query identifies the most-frequent path between a specific source location v_s and a specific destination location v_d during a certain time period T. This query is processed by creating a footmark graph G_f for v_d during T. The footmark graph G_f is a sub-graph of the entire road-network graph G. The edge weights of G_f represent the number of trajectories that pass through an edge in G and reach v_d during T. Then, TPMFP from v_s to v_d is found from G_f by using a dynamic programming algorithm. The purpose of FMI is to filter the indexed trajectories according to v_d and T to construct the footmark graph. FMI consists of a B^+-tree BT_{v_i} for each vertex, say v_i, of the road-network graph G. The B^+-tree for Vertex v_i indexes the time at which the trajectories reach v_i. The leaf entries of the B^+-trees are of the form $< tid, t_a >$, where tid is the trajectory identifier, and t_a is the time at which the trajectory tid reaches the vertex v_i. A hashmap from the vertices to their corresponding B^+-trees is maintained within FMI. **CFMI** (Containment-Based Footmark Index) is an improved version of FMI that stores only the dominant trajectories, i.e., the longest trajectories that share the same path with the other contained trajectories and pass through v_d during T. The footmarks of the contained trajectories are calculated by storing their starting locations with respect to their corresponding dominant trajectory. This requires fetching only the dominant trajectories from disk. The leaf entries of the B^+-tree in CFMI are of the form $< tid, t_s, t_a, did, sloc >$, where tid is the trajectory identifier, t_s is the starting time of the trajectory tid, did is the identifier of the dominant trajectory of tid, and $sloc$ is the starting location of tid within the dominant trajectory did.

TrajTree [112] The TrajTree index is developed to efficiently answer k-NN queries in large trajectory databases. In the TrajTree, trajectories are represented as a sequence of trajectory bounding-boxes by partitioning each trajectory into a large number of segments, where sub-trajectories that are close to each other are grouped. The root of the TrajTree represents a *trajectory box sequence* that is a sequence of bounding boxes constructed over the entire spatial range covered by the indexed trajectories. The trajectories at the root are recursively partitioned into different groups until a node is reached that contains less than n trajectories. Leaf nodes in the TrajTree contain the trajectories (or the sub-trajectories) to be indexed while non-leaf nodes contain trajectory box-sequences. A new trajectory is inserted into the index by adding it to the trajectory box sequence that undergoes the minimum expansion in volume. To increase the pruning power of the TrajTree, a set of d spatial points, termed the vantage points, are distributed in the trajectory space. For every trajectory, a vantage descriptor is maintained to store the distance between a trajectory and vantage points. These vantage descriptors are later used to give an upper bound on the distance between a query and the indexed trajectories.

TRIFL [137] TRIFL is an access method that is optimized for indexing trajectories in flash storage. TRIFL is optimized for the specific nature of trajectory data that involves insertions at high rates with infrequent deletions. To better fit the nature of flash storage, TRIFL favors large-granularity I/Os over page-granularity I/Os. First, TRIFL performs spatial partitioning on the trajectory data. Then, the trajectory data that lies within a spatial partition is indexed temporally. TRIFL uses Grid-based spatial partitioning that is based on the spatial distribution of the trajectory data. For temporal indexing, TRIFL uses the following two indexes: (1) the *A*ppend *O*nly B+-tree (B^{AO}+-tree) for indexing timely updates, i.e., updates that come with a timestamp that is greater than all the indexed timestamps, and (2) the *T*ime *I*nterval *I*ndex (TII) for indexing deferred insertions, i.e., updates that have a timestamp that is earlier than the current timestamp. The B^{AO}+-tree is a variation of the B+-tree that supports append-only operations and that has a page fill-factor of 100%. The TII index splits the time domain into a set of time intervals. For every time interval within the TII index, a list of trajectories that overlap this interval is maintained. Periodically, the B^{AO}+-tree and the TII index for a spatial partition are merged into a new B^{AO}+-tree.

The trajectory-oriented access methods, discussed above, have been designed to handle several important aspects of trajectory indexing, including (1) the efficiency of the index, e.g., PARITNET and T-PARITNET, (2) the uncertainty in the objects' locations, e.g., UTH, and (3) the scalability in indexing the trajectory data, e.g., TrajStore.

3 Indexing the current and recent past data

In this section, we present an overview of the spatio-temporal access methods that deal with queries about the current location or the recent history of the moving objects. We classify these spatio-temporal access methods based on whether or not a recent history of an object is maintained or only the current location of the moving object.

3.1 Indexing the current locations of moving objects

Many applications require online access to the current locations of moving objects, e.g., traffic analysis. Online access can be achieved by indexing the current locations of the moving objects. Indexes that store the current location of a moving object often requires the

removal of the previous locations of the moving object every time the current location of the moving object changes. The access methods that are presented in this section have been proposed to index the current locations of moving objects.

PEB-tree [76] The *P*olicy *E*mbedded B^x-tree is designed to index the current locations of the moving objects while preserving the privacy of the users' locations. To efficiently achieve peer-wise privacy, i.e., protect the location of the user from unauthorized peers, the PEB-tree accounts for both the location proximity of users and the rules that define privacy of users. The main idea of the PEB-tree is to generate an indexing key for each object. The indexing key encodes both the location and the privacy policy information. In the PEB-tree, users that are allowed to see each others' locations and that are spatially close to each other are stored adjacently in the index. The PEB-tree is based on the B^x-tree [60] index. Leaf nodes of the PEB-tree are of the form $< PEB\_key, user\_id, x, y, v_x, v_y, t, Pnt_p >$, where PEB_key is the indexing key, (x, y) and (v_x, v_y) are the user's location and velocity, respectively, at Time t and Pnt_p points to the user's privacy policy. The PEB_key is calculated based on: (1) the timestamp of the user's location, (2) a sequence value that is calculated based on compatibilities among privacy policies of different users, and (3) the z-curve value of the user's location. This calculation of the PEB_key is a one-time process and is performed offline when users are first registered. Insertion and deletion of objects in the PEB-tree are similar to those for the B^x-tree. To answer privacy-aware range queries, the PEB_key of the range is computed by combining both the spatial range and privacy policy constraints. To calculate the search range of the policy constraints, a list is maintained per user to keep track of other users that have policies that are compatible with the list owner.

DIME [33] The *D*isposable *I*ndex for *M*oving obj*E*cts (DIME, for short) maintains the current locations of the moving objects. DIME has been designed to answer snapshot and continuous spatial range queries on moving objects. The purpose of the spatial range query is to identify moving objects inside a specific spatial range. This spatial range is defined using a minimum bounding-rectangle (MBR). The main objective of DIME is to efficiently support frequent updates of object locations as objects move. Processing location updates in spatio-temporal indexes that have been developed prior to DIME requires expensive delete operations. The delete operation is needed to remove the obsolete location after the object moves to a new location. In DIME, instead of performing separate delete operations per location update, large portions of the expired index are removed. DIME maintains multiple instances of a spatial index, e.g., the R*-tree. One instance is stored in main memory to consume the incoming updates. The remaining instances are maintained on disk. The main-memory instance is periodically flushed to disk to create a new empty main-memory instance, and the oldest disk-instance is disposed of, i.e., is deleted. The deleted disk instance can never contribute to the resultset of any query because it gets removed after the maximum duration between two consecutive updates is reached. The main advantage of this index is that updates happen in main memory and no disk-based index-update operations are required. Also, expensive delete operations are grouped when disposing of the oldest instance. One implication of DIME is that querying takes place over two data structures, the one in main-memory and the one in disk and results from both structures need to be reconciliated.

RUM+-tree [171] The RUM+-tree extends the RUM-tree [125, 153] with an additional data structure. The RUM-tree builds on the R-tree index, where it augments the R-tree with a main-memory update memo for indexing the current locations of moving objects. The

additional structure in the RUM+-tree is a hash table on object identifiers apart from the update memo. During an update, the leaf node of the object having the update is directly located through the hash table. The main idea is to deal with updates locally in a bottom-up manner, i.e., if the new location of the moving object still falls in the same MBR as its old location, only the location of the object is updated within the leaf node. If the new location of the moving object falls outside the old MBR, then a new version of this object is inserted into the RUM+-tree. The old version of the object is removed lazily with the help of the update memo. This addresses the limitation of the RUM-tree that location updates are always inserted as new versions of the objects even if these updates occur in the same MBR. One disadvantage of the RUM+-tree over the original RUM-tree is the maintenance of the additional auxiliary hash table during object updates, and whose size is proportional to the number of distinct object identifiers. The RUM-tree does not use the extra hash table at the potential extra cost during update time.

Composite index for indoor moving objects [151] This index supports temporal changes in the indoor space, and requires pre-computing the shortest distances in the indoor space. The composite index has the following three layers: (1) the geometric layer that is composed of a tree tier and a skeleton tier, (2) the topological layer, and (3) the object layer. The tree tier indexes the indoor partitions, e.g., the rooms, the staircases, and the hallways, using an R-tree-like index, where a leaf node represents a partition, say P, and contains a pointer to a bucket containing the moving objects located in P. The skeleton tier is a graph containing information about the staircases. Graph nodes in the skeleton-tier represent entrances. Edges connect entrances that belong to the same staircase or that are on the same floor. The topological layer captures the connectivity between partitions. The object layer contains all the moving objects' buckets. A hash table, *o-table*, is used to map each moving object to all the partitions it overlaps with. Notice that for indoor moving objects, there is uncertainty in the locations of the moving objects. A moving object may be estimated to overlap multiple partitions. To process range and k-NN queries, the geometric layer is searched to identify the candidate objects and partitions. Candidate moving objects are pruned by computing the upper and lower distance-bounds between the moving objects and the query. Exact indoor distance is computed only for the unpruned moving objects.

Sim-tree [157] The Sim-tree is a two-dimensional access method used for traffic simulations over road networks. Spatial indexes are used in traffic simulations to store the locations of moving objects. Traditional R-Tree-based indexes suffer from poor performance when indexing objects that frequently update their locations as they move. The Sim-tree is a balanced binary-tree that uses average object-densities over the road network to build the index. This index studies the expected road-densities across the day, e.g., morning {pre-peak, peak, and post-peak}, and then builds an index for every period by recursively decomposing the space into two sections with almost equal densities. Building the Sim-tree using the expected densities eliminates the need to re-balance the index and significantly improves the simulation performance.

V-tree [121] This access method is designed to answer the snapshot kNN queries on the current locations of objects. The distance metric used is the road-network distance. The V-tree is a balanced f-ary tree that recursively partitions the graph of the road network, where f is the fanout of a tree node. Leaf nodes in the V-tree contain subgraphs of the road network and maintain the shortest-path distance between every pair of vertices in the subgraph. Boundary vertices have edges between two sibling subgraphs. Sibling subgraphs

share the same parent node. Non-leaf nodes in the V-tree maintain the distances between boundary nodes in their child subgraphs. Moving objects are indexed at the nearest leaf-level vertex by storing the distance between the moving object and its nearest vertex. Updates to the locations of moving objects either change the distance between the object and the vertex or change the vertex to which the moving object is attached to. Leaf-level vertices that have moving objects attached to it are called active vertices. kNN processing is performed by finding the nearest active vertices to the location of the query in order to generate the kNN list of moving objects.

The access methods, discussed above, that index the current locations of the moving objects have been designed to handle various aspects in the maintenance of the current location of moving objects including (1) efficiency, e.g., DIME and the RUM+-tree, (2) handling of the movements of objects indoors, e.g., the composite index for indoor moving objects, and (3) constraining the movement of objects over road-networks, e.g., the V-tree.

3.2 Indexing the recent past

In this category, we survey access methods for indexing the recent past for spatio-temporal data. Many applications do not retain the entire historical spatio-temporal data but keep track of only the recent history of the moving objects, e.g., due to privacy requirements. This is achieved by maintaining a sliding window and deleting the expired entries periodically.

SWST [126] The *S*liding *W*indow *S*patio-*T*emporal index supports indexing of limited-history spatio-temporal data under a time-sliding window. SWST is designed as a two-layered index that consists of a spatial grid index and a temporal index within every spatial grid cell. The temporal index is based on the B^+-tree index. Keys used in the B^+-trees embed both temporal and spatial information to improve the pruning power of SWST. Additionally, an *isPresent* memo structure is maintained per spatial cell. This memo stores a histogram that identifies which temporal intervals have spatio-temporal data. Every spatial grid cell maintains two B^+-trees. Each B^+-tree is responsible for indexing data for an entire time-sliding window. One B^+-tree indexes incoming location-updates and the other holds older updates. Sliding-window maintenance is performed as follows: when an entire time-sliding window expires, the B^+-tree that holds old data is entirely removed, and a new B^+-tree is created to hold the newly incoming data. Query evaluation is performed by identifying spatial grid cells that overlap the spatial range of the query. Then, the B^+-trees within the candidate grid cells are accessed to identify the temporally overlapping objects. The purpose of the *isPresent* memo is to reduce the number of accesses to the B^+-trees during query processing.

Trails-tree [87, 88] The trails-tree is a disk-based structure for indexing recent trajectories of moving objects. The trails-tree maintains a temporal sliding-window over indexed data. The trails-tree addresses the performance limitations of SWST [126], and requires a lower number of disk I/Os during update and query processing. The trails-tree extends the RUM-tree [125, 153], and uses a 3D R-tree with an additional data structure named the *current memo*. Indexed entries within the trails-tree are of the form (oid, x, y, t_s, t_e), where oid is the identifier of the moving object, (x, y) is the location of Object oid during Time Period (t_s, t_e). The trails-tree uses the current memo to maintain information about the most-recent update of an object. Initially, all incoming updates are stored in the trail-tree as $(oid, x, y, t_s, NOWTIME)$, where $NOWTIME$ is a constant indicating that the exact

value of t_e is not known. The trails-tree employs a lazy-cleaning mechanism that maintains the time-sliding window by removing expired objects and sets the proper t_e for old entries with the help of the current memo. Query processing is performed using a filter-and-refine approach. First, an initial resultset is retrieved by traversing the nodes of the trails-tree. Then, this initial resultset is refined by removing the expired and non-current entries with the help of the current memo.

4 Indexing the future

In this section, we discuss spatio-temporal access methods that predict the future positions of moving objects. Various approaches are adopted to predict the future positions, e.g., using historical data, using the objects' reference locations and their velocities, or using probability distribution models. In this section, we classify the indexes based on the approach adopted to predict the future locations of moving objects.

4.1 Indexing the future based on the underlying road-network

This category of spatio-temporal access methods predicts the future locations of moving objects based on the underlying road network.

P-tree [57] The *P*redictive *Tree* supports predictive queries over moving objects without the knowledge of the objects' historical trajectories. The main idea behind the P-tree depends on the connectivity of the underlying road-network. The P-tree assumes that moving objects travel through the shortest path to reach their destination. A P-tree is built per moving object as it starts a trip on the road network. The starting node of an object on the road-network graph is deemed to be the root of the P-tree of the moving object. The P-tree of an object consists of all the nodes that are reachable from the root node within a certain time frame T following the shortest route. The probability of reaching a node is predicted based on a *probability assignment model*. A node of the road-network graph will be added to a P-tree only if the object's probability exceeds a specific threshold P. For every node in the road-network graph, a list of the objects predicted to be present at this node is maintained with their probabilities and travel time cost, i.e., the estimated travel time, to reach this node from the current location. When the location of an object is updated, the new location becomes the new root of the P-tree. Nodes that are not reachable from the new root are pruned from the P-tree. The P-tree will be extended to add new nodes that are reachable from the new root within a specific probability threshold. A new P-tree will be created if the new location of the object deviates from the estimated shortest-path route. With every modification of the P-tree, the probabilities of nodes are re-computed and the list of predicted objects associated with each node in the road-network graph is updated accordingly. Spatial range and kNN queries are processed using the list of predicted objects at the qualifying nodes, and this list is manipulated according to the type of query issued.

4.2 Indexing the future based on historical data

This category of spatio-temporal access methods predicts the future location of a moving object based on the history of movements of the object.

Prediction distance table [62] This indexing technique supports server-side processing of predictive range queries based on the mobility statistics extracted from historical trajectories. The main idea behind this technique is to use the turning patterns at the level of the individual moving objects. Initially, each moving object shares the most probable turning pattern for each vertex containing mobility statistics with the server. A moving object sends an update to the server only if there is a change in the most probable turning pattern for any of its vertices. The region covered by the road network is partitioned into a grid containing $m \times n$ cells. For each cell, say c, the set of cells, say S, that intersect the predicted paths are identified along with the estimated travel times from c to every cell in S. This is performed for every object as the mobility statistics differ for the different objects. The minimum travel times among all objects between two cells, say c_i and c_j, i.e., $D_P(c_i, c_j)$ is stored in the *prediction distance table*. An entry in the *prediction distance table* is of the form $[c_i, c_j, h']$, where h' is the minimum travel-time between c_i and c_j. A hash table for destination cells is also maintained, where the key to the hash table is the destination cell identifier, i.e., c_j. This hash table stores a pointer to a B^+-tree-like sorted container with key h' and with value being the origin cell i.e., c_i. When there is a new turning pattern for an object, say o, there will be an update to the *prediction distance table* only if any $D_p(c_i, c_j)$ computed for Object o is less than the existing value for $[c_i, c_j]$. This indexing approach supports predictive range queries that are processed using the hash table by first identifying the candidate origin cells whose destination cells fall in the query range. Candidate entries from the hash table are refined by pruning the cells whose Prediction Distance h' is lower than the predictive distance range specified in the query.

Concurrently updatable index [118] This index structure uses separate indexes for the spatial and temporal domains. The spatial domain is indexed using a grid-based index [48], where each cell of the grid has pointers to the moving objects or queries that intersect the cell. There are also pointers to objects that might intersect the cell with a probability that exceeds a specific threshold. Operations on various cells can be performed concurrently without depending on the other cells. The temporal index partitions the objects into buckets, where each bucket represents a time interval, and stores objects or queries that belong to this time interval. A Bucket is deleted when the lifespan of this bucket exceeds a specific duration. When a bucket gets deleted, objects belonging to that bucket are also deleted from the spatial index. Updates from moving objects are classified into two types: general updates and temporal updates. A general update happens when there is a change in the path adopted by the moving object, and this takes place by deleting the old trajectory and inserting the new trajectory. A temporal update does not involve a change in the spatial location, and hence does not affect the spatial index. However, a temporal update means that there is a change in the time of events associated with the object. If a temporal update results in a change of a temporal bucket, then a new pointer is added to the new bucket, and the pointer pointing to the old bucket is deleted.

4.3 Indexing the future based on velocity partitioning

When dealing with predictive queries, the velocities of the moving objects significantly affect query performance. This indexing approach accounts for the skew in velocity distribution of the moving objects to improve query performance.

DVA [98] This velocity partitioning technique is based on partitioning the velocity domain according to the *D*ominant *V*elocity *A*xes (DVAs, for short). The dominant velocity axes are

the ones in which most of the objects' velocities are parallel to. DVAs are computed using a combination of Principal Components Analysis (PCA, for short) and k-means clustering [84]. The coordinate space is transformed according to DVAs. A moving object index, e.g., the TPR*-tree [135], or the B^x-tree [60], is created based on each DVA. A moving object is inserted into the index with the closest DVA. A threshold is defined per DVA to determine whether an object belongs to it or not. If the threshold is not met, the object is inserted into an outlier index that uses the regular coordinate system. Before processing a spatio-temporal range query, the query range is transformed into the coordinate space of all DVAs. Query results are obtained by querying all indexes of all DVAs and combining the results.

Speed partitioning using dynamic programming [156] The main idea in this partitioning technique is to partition the index based on the velocities of the moving objects such that the expansion of the query search space is minimized thereby improving the query performance. Unlike other speed-partitioning techniques that rely on heuristics, this technique computes an optimal partitioning using a dynamic programming algorithm. First, the speed domain is partitioned into k optimal-parts. Then, objects are also partitioned into k index structures according to their speed values. Each partition can be further decomposed into four quadrants based on the directions of the moving objects. This indexing technique is structured into three components: (1) the *speed analyzer* that computes the optimal speed-partitioning by analyzing the data from the moving objects, (2) the *index controller* that partitions a user's query, forwards the query partitions into their corresponding index partitions, and combines the partial query results received from each index partition, and (3) the *index partitions*, where the actual processing of queries takes place. When a moving object updates its location or velocity, the index controller determines whether the object should be inserted into a different partition or not. This decision is based on the current speed of the object. Partitions are updated periodically because locations and speed distributions of the continuously moving objects keep changing over time.

D-Grid [155] The *D*ual space *Grid* is an in-memory data structure that indexes the moving objects based on both velocity and location. The D-Grid prunes the queries' search space using the associated velocity information. Answering predictive spatio-temporal range queries using a query window enlargement rectangle (, *QwER*, for short) suffers from the drawback of missing some slow-moving candidate objects that may not be part of the enlarged query window. The reason is that the maximum velocity of the moving objects is used to compute *QwER*. The D-Grid addresses this issue by partitioning the objects based on their velocity information, and representing the velocity space as a uniform grid termed *v-grid*. Each *v-grid* cell is associated with a location grid termed the *l-grid*. Each *l-grid* cell points to a doubly-linked list of buckets. These buckets store the data of an object, i.e, the object's location and velocity. Similar to the U-Grid [123], D-Grid contains a hash-based secondary index on object identifiers that provides direct access to the objects to handle object updates. Local updates are performed by just updating the current cell. In contrast, non-local updates, i.e., the ones that span multiple cells, are performed by marking the object in its previous location as invalid and inserting the new location update into a new grid cell. All invalid objects are deleted using a lazy garbage-cleaning mechanism when the number of invalid objects within an *l-grid* cell exceeds a predefined threshold. Predictive range and kNN queries are processed by first computing the *QwER* using the velocity histogram. Then, for each *v-grid* cell that intersects the *QwER*, a range search is performed using the enlarged query window of the corresponding *l-grid* cell.

4.4 Time-parameterized future indexing

This category of access methods depends on indexing the objects based on parametric rectangles. The boundaries of a parametric rectangle at a specific timestamp, say t, are defined using a function of the current location of the moving object, t, and the velocity of the moving object.

OST-tree [139] The *O*bfuscating *S*patio-*T*emporal data tree (OST-tree, for short) has been designed to preserve the privacy of spatio-temporal data. The OST-tree extends the TPR-tree [116] with spatial and temporal obfuscation. Spatial obfuscation is achieved by enlarging the spatial range in which a moving object is projected to be located inside. Temporal obfuscation enlarges the temporal range a specific object can be located inside. For example, the obfuscated location of an object located in (x, y) at timestamp t is the rectangle $(x_{min}, y_{min}, x_{max}, y_{max})$ within the temporal range (t_{min}, t_{max}). The OST-tree accounts for the spatial and temporal obfuscation parameters into the function defining the parametric rectangles. The spatial obfuscation parameter defines the enlargement of the projected spatial location of the user within the parametric rectangles. Similarly, the temporal obfuscation parameter defines the enlargement in the temporal range within the parametric rectangles.

GG TPR-tree [71] The *G*rid-based *G*rouping time parametrized tree (GG TPR-tree, for short) has been designed to optimize the performance of the TPR-tree [116] for objects moving over specific paths, e.g., road networks. The GG TPR-tree predicts the future locations of the moving objects. The key idea is to group compatible objects into clusters and treat the objects as groups not individually. Compatible objects have similar velocities, move on the same network edge, and are located close to each other. The GG TPR-tree makes use of the fact that compatible objects tend to have similar behavior with respect to their projected future locations. This behavior changes at the intersection points of the underlying network. Initially, the GG TPR-tree handles objects individually. Then, once compatible objects are detected, they are grouped together. To prevent the deterioration in performance of the GG TPR-tree, the indexed objects get regrouped when the objects are no longer compatible.

HTPR*-tree [44] The *H*istory TPR*-tree (HTPR*-tree, for short) has been designed to extend the abilities of the TPR*-tree [135] to support historical queries. The TPR*-tree [135] supports queries on current and future locations of moving objects. The main difference between the HTPR*-tree and the TPR*-tree is that the HTPR*-tree keeps track of the creation and update times of the moving objects within the leaf nodes of the HTPR*-tree. This allows the HTPR*-tree to support historical queries over the indexed objects. The HTPR*-tree is able to efficiently support frequent updates of moving objects by using a *bottom-up update approach*. The bottom-up update approach uses auxiliary memory structures, i.e., a hash table, a main memory bit vector, and a direct access table. The hash table locates the leaf nodes of the HTPR*-tree that contain the most-recent update of the moving object. The direct access table identifies parent nodes within the HTPR*-tree. The bit vector indicates whether or not the leaf nodes are full. During an update within the HTPR*-tree, instead of searching the index in an expensive top-down approach, the HTPR*-tree uses the auxiliary memory structure to reduce the number of updates needed for the update based on the following rules: (1) if the incoming update lies outside the boundaries of the root node of the HTPR*-tree, the top-down approach is used, and (2) the hash table and the bit vector are used to identify the leaf node containing the previous location information of the moving object. If the incoming update is located inside the previous leaf node, the direct access

table is used to update and tighten the boundaries of the parent nodes all the way to the root of the tree. Otherwise, the top-town insertion is used.

TPRuv-tree [42] The *T*ime *P*arametric *R*-tree with *U*ncertain *V*elocities (TPRuv-tree, for short) has been designed to answer the *C*ontinuous *k N*earest *N*eighbor query for objects moving on road networks (CkNN, for short). In this query, it is required to report the k objects that have the highest likelihood to be close to the query's focal point in an upcoming temporal range, e.g., within the next [1-5] minutes. The likelihood of closeness is used instead of the exact distance because TPRuv-tree does not assume that moving objects have fixed velocities. Both the data objects and the query point are continuously moving over the road network. An uncertain velocity is represented by a velocity range $[v_{min}, v_{max}]$, where v_{min} and v_{max} are the minimum and maximum velocities of a moving object, respectively. The distance interval between a moving object and a query is calculated based on the object's minimum velocity, the object's maximum velocity, the object's direction, and the direction of the query. The distance interval $[d_{min}, d_{max}]$ represents how close the object to the query point, where d_{min} is the minimum possible distance and d_{max} is the maximum possible distance. Objects are assigned closeness likelihood scores based on their distance intervals. TPRuv-tree is structured as a two-layered index. The top layer of TPRuv-tree is composed of an R-tree index that is used to store the spatial information of the underlying road network. Leaf nodes of the R-tree point to the lower layer of the index. Each leaf node contains a direct access table that contains the information of edges contained in the leaf node's MBR. An entry in the direct access table contains the edge identifier, the edge's speed limit, a list of neighboring edges, and a list of objects moving on that edge. TPRuv-tree is only updated when objects move from one edge to another. In TPRuv-tree, distance intervals of moving objects change as objects move across edges because the direction and the speed of the objects change according to the edge. Hence, the overall temporal range of the query is split into *temporal subintervals*. Within each temporal subinterval, objects do not change the edges they move on. The result of the CkNN query is reported per subinterval based on the closeness likelihood scores of objects. These scores are calculated using the objects' distance intervals per temporal subinterval.

S^{e}TPR*-tree [97] The *S*hared *E*xecution TPR*-tree (S^{e}TPR*-tree, for short) is a disk-based index that is designed to optimize the performance of both the range and kNN queries over moving objects. The main observation behind the S^{e}TPR*-tree is that the moving objects tend to have frequent updates and indexes that are optimized for handling frequent updates tend to have poor query performance. To address this issue, the S^{e}TPR*-tree uses shared query execution along with lazy insertions and deletions to maintain efficient query performance while supporting frequent updates. The S^{e}TPR*-tree uses a main-memory buffer that contains a main-memory TPR*-tree [135] to receive incoming updates. The S^{e}TPR*-tree uses a disk-based TPR*-tree to persist the updates of moving objects. Also, the S^{e}TPR*-tree uses a main-memory deletion hash-table to store all the delete operations that are reported by the moving objects. Batched insertion and deletions are subsequently reflected in the disk-based TPR*-tree. Only one main-memory page is allocated for batch query-processing. The S^{e}TPR*-tree uses a shared query-execution algorithm that ensures that any disk page that is relevant to a batch of queries is loaded to the buffer only once. The shared query-execution algorithm rearranges the steps for processing the queries into group queries that read a disk page only once. All the incoming insertions are held in the main-memory TPR*-tree. When the main-memory buffer is full, updates in the main-memory TPR*-tree are inserted into the disk-based TPR*-tree. To reduce the number of disk I/Os for the disk-based insertions,

the S^{e}TPR*-tree adopts a *proximity-ordered insertion* approach. In this approach, the main-memory TPR*-tree is traversed in depth-first order. During this traversal, the encountered objects are added into an insertion list. The depth-first ordered traversal in the main-memory TPR*-tree ensures a proximity-ordered insertion into the disk-based TPR*tree, and reduces the overall number of disk I/Os needed to merge the main-memory TPR* tree with the disk-based TPR*tree.

The access methods above use the time-parametrized approach to index the moving objects under various scenarios including (1) maintaining user privacy, e.g., the OST-tree, (2) indexing the moving objects in road networks, e.g., the TPRuv-tree, and (3) supporting the frequent updates of moving objects, e.g., the S^{e}TPR*-tree.

5 Indexing the past, the present, and the future

In this section, we study a class of spatio-temporal indexes that index the spatio-temporal data at all points in time, i.e., the past, current, and future times.

PASTIS [113] The *P*Arallel *S*patio-*T*emporal *I*ndexing *S*ystem is an in-memory index that supports past-, present-, and future-time queries. The history of the moving objects is maintained in a table termed *Location* that stores the location, velocity, and timestamp of the moving objects. The spatial domain is partitioned into uniform grid cells, and is ordered using the Z-order space-filling curve. Each grid cell has a partial temporal-index that consists of a lookup table containing entries for time intervals over the past N days. Data older than N days is stored on disk. Each entry in the lookup table contains a compressed bitmap (*CBmap*, for short) that identifies the moving objects that have been in a specific grid cell at a given time interval, and a hashmap termed *Hm-RIDList*. *Hm-RIDList* associates each moving object with a list of record identifiers that locate the actual movement records of the moving object in the *Location* table. For an update of an object with Timestamp ds_t, the interval lookup table is checked to determine if ds_t maps to an existing interval, and if so, then the corresponding bitmap and hashmap entries are updated. If ds_t maps to a non-existing interval, then a new interval is initialized. The predicted locations are computed for a location update according to the projected velocity and the current location of the moving object. The predicted locations are stored in a new hashmap, termed PHm, that is maintained to answer predictive queries. The processing of range queries is performed by bitwise ORing of the temporal bitmaps of the objects in cells that are fully covered by the spatial range of the query. For partially-covered cells, the *RIDList*s of the moving objects are traversed to determined if the objects are located inside the query range.

6 Spatio-temporal and spatio-textual indexing

Recently, many applications have emerged that deal with text data, where the text data is associated with spatial and temporal attributes. Examples of these applications include the analysis of microblogs and the processing of activity trajectories. Microblogs, e.g., tweets, contain a set of keywords, a timestamp, and a spatial attribute that represents the location of the user. Activity trajectories associate keywords to the spatio-temporal trajectories of users. These keywords represent the activities performed by users at specific locations. These applications may require the filtering or ranking of objects based on their spatial, temporal, or textual properties. One approach to index spatio-temporal text data is to use an

existing spatio-temporal index, and extend it to include text data. Alternatively, one can use an existing spatio-textual index, and extend it to include the temporal dimension. One of the earliest spatio-textual indexes is the structure proposed by Aref and Samet [8]. This index combines the spatial pyramid [131] with the bitmap index to answer queries about the features of map data, e.g., Where are the "Corn fields" located? Spatio-temporal and textual data can be modeled as individual objects, e.g., tweets, or a related sequence of objects, e.g., activity (textual) trajectories. In this section, we survey the indexing techniques proposed to answer spatio-temporal and textual queries. We classify spatio-temporal and textual indexes based on their adopted data model, i.e., individual objects or textual trajectories.

6.1 Indexing individual objects

This category of spatio-temporal and textual indexes handles objects individually. Most of the access methods in this category are designed to answer aggregate analytical queries over spatio-temporal and textual data, e.g., identifying the most-frequent keywords in specific locations. Other types of spatio-temporal and textual indexes that handle individual objects address continuous queries over streamed spatio-temporal and textual data.

AFIA [127] The *A*daptive *F*requent *I*tem *A*ggregator (*AFIA*, for short) is designed to identify the top-k frequent terms, i.e., keywords, in a specific spatio-temporal range. This index uses a grid-based approach [6] with uniform and fixed cell-sizes. Spatial grids of multiple granularities are used to partition the indexed space, where each grid cell per granularity stores a summary of the most-frequent terms in that cell. For temporal support, new instances of grid cells are created periodically. Also, temporal cells are also created at multiple time-granularities, and each spatial grid-cell maintains frequencies of terms for all supported temporal granularities, e.g., hour, day, week, and month. To process a query with a specific spatio-temporal range, the query range is partitioned into several coarser regions, and the aggregates from these regions are combined to get the final top-k result. Also, this index changes the size of the summaries dynamically to adapt to changes in the number of frequent terms within grid cells.

Mercury [86] Mercury uses a partial in-memory pyramid [8] to support top-k spatio-temporal-textual queries over microblogs under constrained memory. The pyramid structure is a multi-level partitioning of the indexed space. In the spatial pyramid, each cell at Level i is partitioned into four equal cells in the subsequent level, i.e., Level $i+1$. Each pyramid cell maintains a list of the microblogs that have arrived in the spatial range of the cell during the past T time units. Microblogs within a cell are ordered according to their arrival timestamps. To reduce the insertion overhead, microblogs are periodically bulk-inserted into Mercury using a main-memory buffer. To avoid an extremely deep pyramid, a pyramid cell is split only if its content spans at least two quadrants. This check is performed by maintaining per pyramid cell a 4-bit variable, termed SplitBits. Cells are merged only when three of the four sibling cells are empty to avoid having redundant split and merge operations. Deletion of the expired microblogs is performed either during the insertion of new microblogs or during a periodic deletion. Top-k microblogs are identified by computing the scores of the indexed microblogs according to a ranking function of the spatial proximity of the microblog to the location of the query and the time recency of the microblog. The query-processing algorithm uses a priority queue of the pyramid cells being searched to visit the pyramid cells according to a ranking function that depends on the spatial proximity between the cell and the location of the query and the timestamp of the most recent microblog in every pyramid

cell. Also, during query processing, a list of the top-k microblogs is maintained. This list is sorted in the order of the scores of the microblogs, and gets updated as the pyramid cells are being visited.

AP$^+$-tree [149] The AP$^+$-tree (*A*daptive spatial-textual *P*artition tree) indexes the queries and not the spatio-textual data objects. It is designed to index and answer a multiplicity of continuous moving spatio-textual filter queries at the same time. A spatio-textual filter query is defined using a spatial range and an associated set of keywords. A moving spatio-textual filter query continuously changes its location over time because the query issuer may be moving in space and needs to continously retrieve the updated query answer as it changes its location. In this type of query, it is required to identify the spatio-textual data objects that are located inside the spatial range of the query and that contain all the query keywords. A spatio-textual object is defined using a spatial point location and an associated set of keywords. A continuous spatio-textual filter query progressively runs over streams of spatio-textual data objects. The AP$^+$-tree is an f-ary in-memory index, where continuous queries are recursively partitioned. Partitioning the indexed queries in an AP$^+$-tree node is either spatial or textual. The type of partitioning is based on a cost function that chooses the best partitioning approach. Nodes partitioned spatially are termed s-nodes while those that are partitioned textually are termed k-nodes. The AP$^+$-tree is adaptive to the movement of queries, and adds extra cost in the direction of movement of the queries to reflect the query movement-patterns. For efficient insertion and deletion, a list of queries in each leaf node is maintained as a hashmap structure, termed the s-list. Each incoming data object has an expiration time. Active objects are augmented to the leaf nodes that contain the relevant queries in a list, termed the m-list. One disadvantage of storing the data objects along with the continuous queries is the duplication of thr stored data objects. The reason is that data objects will be stored in all the leaf nodes that have relevant queries. For a query, say Q, a list of the leaf nodes that overlap Q is maintained to handle efficiently the location updates of the continuously moving queries. When queries move, they need to be re-evaluated. This re-evaluation is performed incrementally by reporting either positive updates (i.e., the addition of new output data objects), or negative updates (i.e., the deletion of the expired output data objects).

GeoTrend [85] GeoTrend is an access method for identifying the trending keywords within recent microblogs in a specific spatial region. GeoTrend adopts a hybrid spatio-textual and temporal data structure that builds on the incomplete pyramid structure [8] for spatial indexing. In every cell in the pyramid, a textual index is maintained. This textual index is a hash table that stores aggregate statistics of the keywords in the microblogs over the past time-period, say T. The length of the time duration T depends on the availability of main memory. The aggregate statistics of a keyword, say k, is a set of N counters. Each counter stores the number of microblogs containing Keyword k for a partial time-interval of length $\frac{T}{N}$. GeoTrend uses an expiration technique to evict the obsolete aggregates. When the index is under high workload, GeoTrend adopts a load-shedding technique that evicts the less-important aggregates that are less likely to contribute to any query answer. The main difference between GeoTrend and Mercury [86] is that Mercury searches for individual microblogs while GeoTrend uses aggregates over microblogs to identify the trending keywords.

R-trees with STLs [3] This disk-based index provides exact answers to the top-k *F*requent *S*patio-*T*emporal query (the kFST query, for short). This query identifies the most-frequent

terms in a specific spatio-temporal range. This index extends the nodes of a multi-dimensional R-tree with sorted terms lists (STL, for short). An STL of an R-tree node, say N, is a list that contains the frequencies of terms of the objects covered by Node N. This list is sorted based on the frequencies of terms. To improve the query performance, STLs are added to both leaf and internal nodes of the underlying R-tree. To reduce the memory overhead of the index, STLs store the frequencies of the most frequent λ terms, where λ is estimated analytically.

6.2 Indexing textual trajectories

This category of spatio-temporal and textual indexes addresses the sequential nature of textual (also termed, activity) trajectories. These indexes answer several interesting similarity queries over textual trajectories.

GAT [166] The *G*rid index for *A*ctivity *T*rajectories (GAT) has been designed to address the *A*ctivity *T*rajectory *S*imilarity *Q*uery (ATSQ, for short). ATSQ is represented by a set, say S, of location points, where each point has an associated set of activities. The answer to this query is the k most-similar activity trajectories to S. Activity trajectories are represented as an ordered sequence of spatio-temporal location updates, where each location update is associated with a (possibly empty) set of activities. A matching activity trajectory contains all the activity keywords of the query at close proximity to the query's location points. GAT is composed of a *multi-level grid*, i.e., a pyramid, where every grid cell at Level i covers four cells at Level $i+1$. A *hierarchal inverted cell list* (HICL) that maintains an inverted list of activity keywords for every level in the multi-level grid. This inverted list maintains, for every activity keyword α, a list of grid cells that contain trajectory updates involving α. The size of HICL can grow extensively due to the large number of indexed activity keywords and their posting lists. Hence, HICL may not fit entirely in the main memory. To address this issue, parts of HICL that represent the top levels of the multi-level grid are kept in main memory, and parts of HICL that represent the lower levels of the multi-level grid are stored on disk. Within every cell in the multi-level grid, and for each activity, an *inverted trajectory list* (ITL, for short) is maintained to keep track of trajectory identifiers (IDs, for short) that contain that activity. Also, GAT maintains a *trajectory activity sketch* (TAS, for short) to summarize the activities per trajectory to efficiently prune trajectories that do not match the required activities of the query. ATSQ is processed by first using HICL to identify candidate leaf-level grid cells that are closest to the location of the query. Then, candidate cells are checked using ITL to validate candidate trajectories. TAS is used to efficiently ensure that the trajectories contain the query keywords.

RAC-tree [29] RAC-tree has been proposed to answer the *R*anking-based Activity *T*rajectory *S*earch query (RTS, for short). RTS is composed of a spatial location, a set of keywords that represents the activities specified by the query, and a threshold on the travel distance of the retrieved trajectories. Similar to the *A*ctivity *T*rajectory *S*imilarity *Q*uery (ATSQ, for short) [166], RTS retrieves the k-most relevant activity trajectories, and needs to cover all query keywords. In RTS, if a user is searching for activity trajectories that involve the keyword *"restaurant"*, the ranking of the restaurant is considered alongside with the spatial proximity between the location of the query and the locations of the trajectory. Hence, the main difference between ATSQ and RTS is that RTS takes into account the ranking of the activities. The RAC-tree is based on a quad-tree partitioning of the spatial locations of the indexed trajectories. Leaf nodes of the RAC-tree contain the locations

of the trajectories, the activity keywords, and the rankings of the activities. A non-leaf node within the RAC-tree contains a summary of the activity keywords covered with the non-leaf node. The RAC-tree is traversed to identify the candidate trajectories while using the keyword summary-information to efficiently prune the non-relevant index nodes. Candidate trajectories are subsequently refined to identify the final resultset of the query.

GKR [90] The Grid and KR*-tree index is a hybrid index structure that combines SETI [25] for indexing the trajectories of moving objects and the KR*-tree [55] for indexing spatio-textual objects. Similar to SETI, GKR uses a grid to partition the space into uniform disjoint cells and to store the content of these cells into separate disk pages. Each disk page is associated with a set of keywords from the trajectory segments stored in it. Disk pages of a grid cell are organized using the KR*-tree. A KR*-tree contains a structure that associates index nodes with keywords, and organizes disk pages according to their temporal properties. Spatio-temporal and textual queries of the form $Q = (R, T, \psi)$, where R specifies the spatial range, T is the time interval, and ψ is a set of keywords, are processed by first finding candidate grid-cells that overlap R. Then, the corresponding KR*-trees of the candidate grid-cells are traversed to find nodes whose timestamp overlaps T and contain a keyword from Set ψ. The corresponding disk-pages of these nodes are further filtered in two steps to discard false-positive trajectory segments in the spatio-temporal dimensions and to remove trajectory segments that do not fully cover the set of query keywords ψ.

IFST [90] The *I*nverted *F*ile with *S*patio-*T*emporal order index (IFST, for short) is based on SFCQuad [31], and extends SFCQuad to support the temporal dimension. IFST consists of two main structures: (1) an inverted file that contains the trajectory's segment-identifiers per keyword, and (2) a spatio-temporal structure to index the segments according to their spatio-temporal properties. The spatio-temporal index is composed of a quad-tree that indexes the trajectory segments according to their spatial locations using the Z-curve ordering. A leaf node in the quad-tree contains an R*-tree to index the timespan of trajectory segments. The inverted lists are split into blocks and are compressed before storing them on disk. To process a spatio-temporal and textual query of the form $Q = (R, T, \psi)$, the underlying quad-tree is traversed to identify the nodes that overlap R. For a qualified quad-tree leaf node, the corresponding R*-tree is traversed to identify segment identifiers that have a timespan that overlaps T. The inverted index is used to find the segment identifiers for the keywords contained in ψ. Then, similar to GKR [90], a two-step filter is used to obtain the final result.

IOC-tree [54] The *I*nverted *OC*-tree (the IOC-tree, for short) answers spatio-temporal and textual filter queries on trajectories. The IOC-tree is based on an inverted index, where query processing is performed by filtering the indexed data using a keyword-first strategy. In the IOC-Tree, each keyword has an octree [59] that is built by recursively dividing the spatio-temporal space into eight nodes. Leaf nodes are encoded using the 3D Morton code [92], where non-empty leaf nodes are stored on disk in a one-dimensional structure ordered by Morton codes. A signature is also maintained per octree node that contains a summary of the trajectory information within that node. This signature is used to filter out the non-qualifying nodes, and the signature gets updated during the insertion/deletion of trajectories. Exact trajectory information per non-leaf nodes is stored on disk. Query processing is performed by dividing the nodes into three types: (1) nodes that do not satisfy the spatio-temporal constraint, (2) nodes that are partially covered by the spatio-temporal range of the query, and (3) nodes that are fully-covered by the spatio-temporal range of the query. After the

signature test is performed to filter out the non-qualifying nodes, candidate trajectories are loaded from disk, and are validated to get the final result.

GiKi [165] The *G*rid *i*ndex *K*eyword *i*ndex (GiKi, for short) has been designed to answer the *Approximate Keyword Query of Semantic Trajectories*(AKQST, for short). The input to AKQST is a set of keywords, where it is required to retrieve the k most-relevant semantic trajectories or sub-trajectories. A semantic trajectory (or sub-trajectory), i.e., an activity trajectory, needs to cover all the query keywords while having the shortest travel-distance. Coverage of keywords in AKQST is based on approximate keyword-matching, e.g., to tolerate any misspelled keywords. The relevance of trajectories is defined as a function of (1) the aggregate travel-distance of the trajectory, and (2) the similarity between the trajectory keywords and the query keywords. GiKi consists of a *S*emantic *Q*aud-tree (SQ-tree, for short) and a *K*eyword-*Ref*erence Index (K-Ref, for short). The SQ-tree is constructed based on a multi-level grid-partitioning of the indexed activity-trajectories. Grid cells that overlap the trajectories are used to build the spatial quad-tree within the SQ-tree. A non-leaf node in the SQ-tree contains (1) an identifier of the corresponding grid cell in the multi-level grid, (2) pointers to children nodes of the quad-tree, and (3) a signature of all the keywords covered within the corresponding grid cell in the multi-level grid. The signature of keywords is a *MinHash* [160] of all the keywords covered by the quad-tree node. A MinHash signature of the keywords is calculated by generating multiple hash-functions over all the n-grams [50] of all the keywords covered by a quad-tree node. The n-gram of a string, say S, is a set of strings similar to S that is calculated by introducing wild-card characters in S. For example, the 3-gram of "Box" is {*"##B, #Bo, Box, ox$, x$$"*}. A leaf node in the SQ-tree contains the keyword signature and pointers to the indexed trajectories. K-Ref is a textual index that is maintained per trajectory to speed up the computation of *the string edit-distance*. In K-Ref, *K-means* clustering is used to identify the keyword clusters per trajectory. For every cluster, a *reference keyword* is chosen. These reference keywords are considered as representatives of the clusters of keywords. Then, keywords of a trajectory are indexed based on their distance to the reference keywords using a B^+-tree. AKQST is processed by using the SQ-tree and K-Ref to identify candidate trajectories that have keywords similar to the query keywords. Then, the relevance of the candidate trajectories is calculated to identify the k most-relevant trajectories.

IRWI-tree [58] The IRWI-tree is designed for indexing spatio-textual trajectories. The IRWI-tree is able to efficiently answer the sequenced spatio-temporal and textual query over trajectories. A sequenced spatio-temporal and textual query, say q, searches for trajectories that satisfy a sequence of spatio-temporal and textual range queries $q_1, q_2, \cdots, q_n$, e.g., to retrieve objects that take the bus in the morning and the train in the afternoon. The IRWI-tree is a hybrid index that combines an R-tree index with an inverted list. Leaf entries in the IRWI-tree are of the form of trajectory units $[I, L, Seg]$, where I is the interval of the entry, L is the textual content of the entry, and Seg is the spatial location of the indexed trajectory unit. Internal nodes of the IRWI-tree contain summaries of the trajectory units in the leaf level. Sequenced spatio-temporal and textual queries are answered by splitting the sequenced query into multiple simple queries that are processed in parallel starting at the root of the IRWI-tree. Only trajectories that satisfy all simple queries in the proper order are reported in the resultset of the query.

ITB-tree [168] The *I*nverted *T*rajectory *B*undle-tree (ITB-tree, for short) is designed to answer a variation of the top-k spatial-keyword similarity query on activity trajectories. The

parameters of this query are a spatial location and a set of keywords. In this query, it is required to retrieve the k most-relevant activity trajectories. The relevance of activity trajectories is defined as a function of (1) the spatial proximity between the activity trajectories and the location of the query, (2) the number of query keywords contained in the activity trajectory, and (3) the popularity of activity points in the trajectory. The popularity of an activity trajectory point, say P, is measured based on the number of other trajectories visiting P. In other words, an activity trajectory is popular when it contains points that are frequently visited by other trajectories. The ITB-tree extends the *T*rajectory *B*undle-tree (the TB-tree, for short) [106] for indexing spatio-temporal trajectories. The TB-tree is a hierarchical spatio-temporal structure that ensures that all trajectory points in a leaf node belong to the same trajectory. When a trajectory spans multiple leaf nodes, these leaf nodes get connected by a doubly-linked list. The ITB-tree adds an inverted list to the nodes of the TB-tree. A leaf node in the ITB-tree contains trajectory points and an inverted list for all keywords indexed in this node. Trajectory points in the posting lists are sorted based on their popularity. Also, for each keyword, say w, two flags are maintained to indicate whether or not w appears in the previous or the subsequent leaf nodes in the connected doubly-linked list. In the ITB-tree, a non-leaf node, say N, maintains an inverted list for all the keywords covered by N. This inverted list maintains the maximum popularity for any keyword covered by N.

Multi-index [145] The multi-index is a combination of heterogeneous traditional access methods for indexing symbolic trajectories of moving objects. A symbolic trajectory is a sequence of units. A unit consists of a time interval and a label. A label is a symbolic textual description of the location visited or the action performed during the time interval of the unit. For example, the sequence of street names visited by a moving object constitute the labels of a symbolic trajectory. More than one label can exist for the same moving object to describe its movement, e.g., the transportation mode, the districts, and the points of interest visited. Fabio et al. [145] adopt an expressive pattern-matching-based query language to query symbolic trajectories. One example query is to identify the moving objects that have visited specific points of interest at a specific sequence or at specific time intervals. To efficiently support pattern-matching-based queries, the multi-index is adopted. The multi-index combines the following structures to index labels based on the labels' data types: (1) A trie to index strings, (2) a 2D R-tree to index points and rectangles, (3) a 1D R-tree to index time intervals, and (4) a B^+-tree to index numeric data.

2TA [148] Wang et al. have introduced the 2TA algorithm to answer the *E*xemplar *T*extual *Q*uery (ETQ, for short). Similar to the *A*ctivity *T*rajectory *S*imilarity *Q*uery (ATSQ) [166], ETQ retrieves the k most-relevant activity trajectories. However, ETQ does not require the retrieved trajectories to cover all the query keywords. The relevance of the retrieved trajectories depends on a function of (1) the spatial proximity between the locations of the query points and the locations of the trajectory points, and (2) the number of shared keywords between the query and the trajectory. 2TA uses the spatial grid and the inverted list to answer ETQ. The spatial grid is used to identify the candidate trajectory-points based on spatial proximity. The inverted list is used to keep track of trajectory points per keyword. 2TA uses the spatial grid and the inverted list data structures to search activity trajectories and rank them to answer ETQ.

ST-tree [79] The ST-tree is designed to answer semantic-aware similarity queries on activity trajectories. Instead of adopting exact or approximate keyword matching, the semantic-aware similarity query considers the semantic similarity between the keywords

representing the activities of the trajectories. For example, Keywords "Gym" and "Exercise" have high semantic-similarity. This query attempts to identify the k most-relevant activity-trajectories to a specific set of keywords and spatial locations. Relevance is defined as a function of (1) the spatial proximity between the locations of the trajectories and the locations of the query, and (2) the semantic similarity between the keywords representing the trajectories' activities and the keywords of the query. To measure the semantic similarity of the keywords, *Latent Dirichlet Allocation* (LDA, for short) [17, 159] is used to map the keywords of activities into a high-dimensional vector that represents the semantics of the keywords. The ST-tree integrates the quad-tree with *Locality Sensitive Hashing* (LSH, for short) [49]. LSH is used to reduce the dimensions of the LDA representation and to ensure that relevant activity trajectories are assigned to the same bucket with high probability. In the ST-tree, activity trajectories are first indexed using a quad-tree. Every leaf node in the quad-tree points to an LSH structure for each trajectory point indexed by this leaf node. Query processing in the ST-tree uses the quad-tree to find the candidate trajectories that are close to the location of the query. Then, LSH is probed to identify the semantically-similar activity-trajectories.

The indexes discussed above support various important trajectory similarity queries over textual trajectories. These trajectory queries employ multiple cost functions that measure the relevance of the textual trajectories to the query.

7 Parallel and distributed spatio-temporal access methods

The current scale of spatio-temporal data being generated has made centralized indexes less suited to support the needs of spatio-temporal applications. The performance of centralized indexes is restricted by the resources of a single machine. This has led to the development of parallel and distributed spatio-temporal access methods to scale up the processing of spatio-temporal queries. There are two main approaches for designing scalable spatio-temporal access methods; (1) As extensions to general-purpose scalable systems, and (2) As standalone indexes. An Index that extends a general-purpose system often integrates a traditional spatio-temporal index into the general-purpose scalable system. In contrast, a standalone index does not build on an existing system. In this section, we highlight the main parallel and distributed spatio-temporal access methods.

7.1 Indexes that extend general-purpose scalable systems

This category of spatio-temporal access methods builds on an existing general-purpose scalable system. The main advantage of this approach is to inherit the scalability and fault-tolerance features of the underlying general-purpose scalable system. General-purpose scalable systems can be classified into batch and streaming systems. Batch systems, e.g., Hadoop [38] and Spark [163] require minutes or even hours to process large amounts of data. Streaming systems, e.g., Storm [140] process data in real-time with minimal latency. In this section, we highlight the main spatio-temporal indexes that extend general-purpose scalable systems.

DSI [162] The *D*ynamic *S*trip *I*ndex (DSI, for short) is a distributed structure to support the processing of kNN queries over moving objects. DSI partitions the two-dimensional space into two sets of non-overlapping strips (vertical and horizontal strips). DSI is realized on top of the Storm streaming system [140]. DSI maintains the current locations of moving objects

within the strips. Strips have a lower and an upper bound on the number of objects they contain. When a strip has more objects than the upper bound, the strip splits. Conversely, when a strip has fewer objects than the lower bound, the strip attempts to merge with neighbor strips. Using horizontal and vertical strips simplifies the splitting and merging operations. Strips are assigned to distributed processes, and a single worker process can have more than one strip. Splitting and merging of strips guarantee that there will be no overloaded or under-utilized processes. To answer kNN queries using DSI, candidate strips are identified to calculate local kNN resultsets. Then, the global kNN resultset is aggregated over all the local kNN resultsets.

QaDR-tree [53] The QaDR-tree is a distributed index designed to support spatio-temporal range queries over spatio-temporal data inside Hadoop [38]. Hadoop is a distributed cluster-based big data processing system. The QaDR-tree belongs to spatio-temporal access methods for the past. The QaDR-tree is a two-layered index that is composed of a *global-index* layer and a *local-index* layer. The global-index is based on a 3D quad-tree, i.e., an octree [59, 89]. The dimensions of the 3D quad-tree are the space and time dimensions. The 3D quad-tree partitions the spatio-temporal data into blocks. The size of a data block is set to 60MB that is smaller than the total size of each Hadoop data block that is 64MB. This extra space is allocated for the local index to be stored with the spatio-temporal in each of the Hadoop data blocks. The local index used is a 3DR-tree. To answer a spatio-temporal range query, the global 3D quad-tree is consulted to identify the relevant Hadoop data blocks. Then, the local 3DR-tree within each of these blocks is used to identify the final query results.

ST-hadoop [7] *S*patio-*T*emporal Hadoop is a distributed framework for storing, indexing, and querying spatio-temporal data. ST-Hadoop builds on the SpatialHadoop [41] system. SpatialHadoop extends the Hadoop MapReduce [38] system with spatial constructs, i.e., a spatial query language and spatial indexes. Indexing of spatio-temporal data in ST-Hadoop is performed using the following phases: (1) sampling and (2) bulk-loading. In the sampling phase, a MapReduce [38] job scans the spatio-temporal data, and keeps a sample of the data in main memory. This sample guides the indexing of all data in the bulk-loading phase. Spatio-temporal indexing in ST-Hadoop is a temporal-first index, where data is first partitioned into temporal slices, then a spatial index is built for every temporal slice. The boundaries of the temporal slices are estimated from the sample, and can be either time-driven or data-driven. In time-driven slicing, the temporal ranges of the slices are fixed, e.g., each slice spans one month. In data-driven slicing, slices hold the same amount of data, and the temporal ranges of slices may not be the same. To further improve the performance of ST-Hadoop, a hierarchical temporal index is built on top of the temporal ranges of the slices. Spatial indexing within a slice uses traditional spatial indexes that already exist in Spatial-Hadoop, i.e., the Grid, the R-tree, and the KD-tree. Bulk-loading uses a MapReduce job to scan the spatio-temporal data and indexes the data according to the temporal slices.

DTR-tree [146] The *D*istributed *T*rajectory *R-tree* (DTR-tree, for short) is a realization of the R-tree index on top of Apache Spark [163]. Apache Spark is a general-purpose distributed big-data system. The DTR-tree indexes trajectories and activity trajectories using distributed R-trees based on the trajectories' spatial attributes. The DTR-tree is organized in a global-local setup, where a global R-tree is maintained to provide a partitioning over the indexed trajectory-data. Leaf nodes of the global R-tree represent children R-trees that

are stored in distributed machines. In the DTR-tree, trajectories are indexed based on their spatial locations using two-dimensional R-trees.

DMTR-tree [15] The DMTR-tree index is designed to support the *skyline trajectory query* over activity trajectories. The parameters of the skyline trajectory query are a trajectory start location, a trajectory end location, a set of keywords, and a distance threshold. In the skyline trajectory query, it is required to identify the set of skyline trajectories that are not dominated by other trajectories in any of the following dimensions: (1) the spatial proximity, and (2) the query keywords contained in the trajectory. The DMTR-tree uses the DTR-tree [146] with a separate inverted list to keep track of the trajectories that contain the popular keywords.

DITIR [22] DITIR is a distributed index for indexing and querying trajectory data in real-time. It supports the ingestion and indexing of trajectory data at high rates. DITIR is built on top of Apache Storm [140]; a distributed data streaming system. DITIR uses an insertion server to index the incoming trajectory-data, and a query server to handle the incoming queries. DITIR stores data in a distributed file system in temporal chunks. The insertion server builds data chunks as in-memory B+-trees. The key of a B+-tree entry is the Z-value of the geo-location of an incoming data-entry. The B+-trees are periodically flushed into a distributed file system. To avoid spending time on node splits during B+-tree insertions, DITIR uses template-based B+-tree indexing. In template-based indexing, it is assumed that the spatial distribution of data (and the distribution of the Z-values) does not change significantly between consecutive data chunks. DITIR uses the structure of the B+-tree from a previous chunk as a template to index a subsequent chunk. The query server maintains metadata about chunks in the distributed file system to improve search performance in DITIR. The metadata includes an R-tree that stores the spatial ranges of the various chunks.

The parallel and distributed access methods, discussed above, extend general-purpose scalable systems with spatio-temporal indexing abilities. These indexes often inherit the performance, scalability, and fault-tolerance properties of the underlying general-purpose scalable system.

7.2 Standalone parallel and distributed spatio-temporal indexes

This category of spatio-temporal access methods does not depend on an existing general-purpose scalable system. In this section, we highlight the main standalone parallel and distributed spatio-temporal access methods.

TwinGrid [122] The TwinGrid index maintains the current locations of update-intensive moving objects. It uses two separate grid-based memory-resident indexes for handling queries and updates so that both can be processed in parallel without interference. The updates index, also termed the writer store, is a memory-resident write-only structure that contains the most up-to-date copy of the data. The reader-store is a read-only index that contains a near up-to-date snapshot of the earlier index. The entire memory-resident writer-store, i.e., the updates index, is copied periodically. The duration between two successive copies is called the cloning period. Copying is performed using the memcpy system call from the write-store to the reader-store while pausing updates. Therefore, the query results are not up-to-date with the maximum staleness of the cloning period. The structure of TwinGrid extends the uniform grid-based structure [123]. Each grid cell does not store data but contains a pointer to a linked list of buckets where data is actually stored. Also, TwinGrid uses a secondary-index structure that indexes objects based on their identifiers,

and has direct access to the data of an object based on a bucket pointer, and a pointer to the object within the bucket. These pointers prevent the need to scan an entire cell or bucket during updates. TwinGrid supports multi-threading by maintaining multiple queues, one queue per thread. These queues contain the incoming updates as well as the queries that are being processed.

PGrid [124] The *P*arallel *Grid* (PGrid, for short) indexes the current locations of moving objects. PGrid uses a locking mechanism that handles both the queries and the updates concurrently, thereby providing up-to-date query results. PGrid has a structure that is similar to that of TwinGrid [122], where queries are served by a primary uniform grid [6] and a secondary index that handles updates in a bottom-up fashion [70]. Each object, say o, in PGrid can have up to two versions at a time; one representing o's previous location and the other for o's current location. The previous version of an object is maintained to ensure correct query answers even during an update of a moving object. The indexed entry of a moving object contains the update timestamp to identify the latest version of the object's location, the object identifier, and the location of the moving object. A new location update logically deletes the old version. The actual deletion of the old version happens with the subsequent update to the object. Both the primary and the secondary indexes are modified concurrently. Locking is used at both the object and the grid-cell levels. A latch-based optimistic lock-free index traversal (OLFIT, for short) [24] and a single-instruction multiple-data (SIMD, for short) technologies are used for parallel and atomic object reads and writes within PGrid.

MPB-tree [56] The *M*ulti-dimensional *P*arallel *B*inary *T*ree (the MPB-tree, for short) indexes four-dimensional spatio-temporal data. The four dimensions are x, y, z, and time. The MPB-tree consists of four binary trees, one binary tree per dimension, and a shared *memory-pool* that stores the intervals of the four dimensions. Each binary tree is a *T*riangular *B*inary *T*ree (the TB-tree, for short) that uses a triangular decomposition strategy similar to that in the *T*riangular *D*ecomposition *T*ree (the TD-tree, for short) [130]. The TD-tree is a temporal index that uses a two-dimensional representation for temporal intervals, and performs triangular decomposition of the indexed intervals. A TB-tree node stores the triangle covered by the node, pointers to the left and right children nodes, and an interval pointer-array that contains pointers to intervals in the shared memory-pool. A shared memory-pool entry contains the following items: (1) the identifier of the spatio-temporal object, (2) an offset from which the object is stored in the file, and (3) an interval array that stores the four intervals of the 4D MBR of the object. Each interval is pointed to by a leaf node in the corresponding TB-tree. An interval is inserted into the MPB-tree through four parallel TB-tree insertions. The four insertions correspond to the indexed four dimensions, where every dimension has its own TB-tree. Leaf nodes in the TB-tree have a maximum capacity threshold. If the leaf node chosen for the insertion of an interval pointer exceeds the maximum capacity, the leaf node is split by recursively decomposing the node's triangle into smaller sub-triangles using a triangular-decomposition strategy. A spatio-temporal range-query is divided into four parallel interval-queries that are executed on their corresponding TB-trees. Each interval query is transformed into a 2D rectangular region, and the query result consists of all the intervals that occur within this rectangle. Thus, a 4D MBR is transformed into four 2D rectangles and each of the four rectangular regions is an input to the corresponding TB-tree in each dimension.

ToSS-it [4] The Throwaway Spatial Index Structure (ToSS-it, for short) indexes the current locations of moving objects on a distributed server. The main idea behind ToSS-it is to

exploit the underlying parallel and distributed architecture by building a new index whenever there is a location change instead of updating the old index. This eliminates the need to maintain a centralized update-tracking buffer to maintain updates that are not yet reflected into the index. This improves the scalability of the system. ToSS-it uses a Voronoi diagram [119] that is distributed over the multiple nodes. The Voronoi diagram is constructed in a distribute-first-build-later fashion by first distributing all the objects across the cloud servers while maintaining their spatial locality. The initial distribution of the data is performed using a centralized server. Then, local Voronoi diagrams (LVD, for short) are built at each server. LVDs decompose the space into disjoint polygons. The generation of the LVDs utilizes all the available cores of the CPUs to further partition the objects at each node. A hierarchical Voronoi index structure is also built at each server to speed up query processing. A query, say q, can by submitted to one node, say N_q, that will then find the nodes IN_q that intersect the query region and forward the query to these nodes. The query is run in parallel in the IN_q nodes using LVDs. Partial results of the query at each node are sent back to N_q for aggregation. D-ToSS [5] is an enhanced version of ToSS-it, where D-ToSS does not require a centralized server when partitioning data across the nodes.

STIG [39] The goal of *Sp*atio-*T*emporal *I*ndexing using *G*PUs (STIG, for short), is to support the processing of interactive spatio-temporal range queries that require multiple point-in-polygon (PIP, for short) tests. STIG belongs to spatio-temporal access methods that index the past/historical spatio-temporal data. A single index is used to simultaneously filter spatio-temporal data over multiple dimensions to reduce the number of costly PIP operations. STIG leverages the parallelism provided by the parallel processors in GPUs to concurrently execute multiple PIP tests that are independent of each other. STIG is a generalization of a *kd*-tree with $k = 2 \times s + m$, where s represents the spatial dimension, and m represents other attributes, e.g., the temporal dimension. This index is designed for data with multiple spatial and temporal attributes, e.g., taxi logs with pick-up and drop-off locations and times. STIG consists of two parts: internal nodes of the *kd*-tree, and leaf nodes. A leaf node stores a pointer to a leaf disk-block and a k-dimensional box that bounds all the records in that block. STIG clusters the points along the k dimensions to speed up query processing and to maximize the utilization of the underlying GPUs. STIG does not support updates, and has to be rebuilt periodically when new records are added to the database.

Elite [152] Elite is an access method for spatio-temporal trajectories. Elite supports parallel updates of moving objects and query processing over multiple compute nodes. Elite addresses both range and nearest-neighbor queries. Spatio-temporal data is distributed across multiple nodes based on the spatio-temporal locality of the data. Data is indexed at the local and global levels. Data in each node is indexed using a local index. A global index coordinates the communication among the multiple local indexes. Elite consists of the following three layers: (1) the skip-list layer, (2) the torus layer, and (3) the oct-tree layer. The skip-list and torus layers constitute the global index while the oct-tree layer constitutes the local indexes. The oct-tree [59] in each local index stores the trajectory locations and maintains a hash table. The hash-table maps the identifier of a trajectory to the oldest and the most-recent locations of the trajectory. The most-recent location of the trajectory is maintained to efficiently insert the incoming trajectory-updates. Every trajectory location has a pointer to the successive trajectory-location. Both the oldest location of the trajectory and the successor pointers are used for traversing the entire trajectory. The torus layer consists of chained tori, where each torus is a cluster of nodes. Each node in a torus maintains a routing table that contains the IP addresses and the data ranges of the neighboring nodes.

The information in the routing table is used for communication among nodes in the torus. The skip-list layer contains a doubly-linked skip-list [111], where each node in the skip-list corresponds to a torus cluster. The key of a skip-list node consists of the temporal interval of the torus cluster and a pre-assigned consecutive IP-address segment. The IP-address segment serves as a pointer to the torus cluster. For communication between two tori, one torus node picks a random IP-address from the IP-address segment of its linked torus. Then, the picked random node performs intra-torus communication to find the destination node. This communication is needed to pass query information. Spatio-temporal range queries are evaluated by first identifying the torus nodes that overlap the query region. All candidate torus-nodes run the corresponding sub-queries in parallel. The local indexes are traversed to identify trajectories that overlap the query region. Idle torus-nodes get allocated to each of the candidate nodes to perform further resultset refinement. Query results from all these nodes are merged to compose the final result.

The spatio-temporal indexes, discussed above, use several techniques to improve the scalability of spatio-temporal data processing without depending on existing general-purpose scalable systems. These indexes implement their own scalability and fault-tolerance methodologies.

8 Conclusion

This survey is Part 3 of a series of two other surveys [91, 99] that collectively cover the spatio-temporal access methods developed up to 2017. In the years 2010 to 2017, new categories of spatio-temporal access methods have been developed, namely, (1) spatio-temporal access methods for indexing the recent past, (2) spatio-temporal access methods with text support, and (3) parallel and distributed spatio-temporal access methods. Spatio-temporal indexes for the recent past supports the limited retention of the data. Spatio-temporal and textual indexes have been developed due to the ubiquity of GPS-enabled smartphones and their applications. Social networks, e.g., as in Twitter [142], generate text data that is associated with the location where text is produced. The concept of activity (or textual) trajectories has been introduced to represent trajectories that have a textual description associated with the trajectory points. Also, several interesting spatio-temporal and textual similarity queries on textual trajectories have been proposed. Spatio-temporal and textual indexes integrate a spatial or a spatio-temporal index with a textual index. Due to the massive scale of spatio-temporal data, there has been a large body of research that targets the development of parallel and distributed spatio-temporal indexes. Many parallel and distributed spatio-temporal access methods have been realized inside a general-purpose big-data processing system.

Acknowledgements This work has been partially supported by the National Science Foundation under Grant Number III-1815796.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

1. Abdelguerfi M, Givaudan J, Shaw K, Ladner R (2002) The 2-3TR-tree, a trajectory-oriented index structure for fully evolving valid-time spatio-temporal datasets. In: ACM-GIS, pp 29–34

2. Agarwal PK, Arge L, Erickson J (2000) Indexing moving points. In: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS), pp 175–186. ACM
3. Ahmed P, Hasan M, Kashyap A, Hristidis V, Tsotras VJ (2017) Efficient computation of top-k frequent terms over spatio-temporal ranges. In: The international conference on management of data (SIGMOD'17), pp 1227–1241
4. Akdogan A, Shahabi C, Demiryurek U (2014) ToSS-it: A cloud-based throwaway spatial index structure for dynamic location data. In: The IEEE international conference on mobile data management (MDM'14), pp 249–258
5. Akdogan A, Shahabi C, Demiryurek U (2016) D-toSS: A distributed throwaway spatial index structure for dynamic location data. IEEE Trans Knowl Data Eng (TKDE) 28(9):2334–2348
6. Akman V, Franklin WR, Kankanhalli M, Narayanaswami C (1989) Geometric computing and uniform grid technique. Comput Aided Des 21(7):410–420
7. Alarabi L, Mokbel MF (2017) A demonstration of ST-hadoop: A mapreduce framework for big spatio-temporal data. The Proceedings of the VLDB Endowment (PVLDB'17) 10(12):1961–1964
8. Aref WG, Samet H (1990) Efficient processing of window queries in the pyramid data structure. In: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp 265–272
9. Atluri V, Adam NR, Youssef M (2003) Towards a unified index scheme for mobile data and customer profiles in a location-based service environment. In: Workshop on next generation geospatial information (NG2i'03). Citeseer
10. Atluri V, Guo Q (2005) Unified index for mobile object data and authorizations. In: European symposium on research in computer security, pp 80–97. Springer
11. Atluri V, Shin H (2007) Efficient security policy enforcement in a location based service environment. In: IFIP Annual conference on data and applications security and privacy, pp 61–76. Springer
12. Bayer R, MCCReight E (1972) Organization and maintenance of large ordered indexes. Acta Informatica 1:173–189
13. Becker B, Gschwind S, Ohler T, Seeger B, Widmayer P (1996) An asymptotically optimal multiversion B-tree. Intern J Very Large Data Bases (VLDB Journal) 5(4):264–275
14. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-tree: An efficient and robust access method for points and rectangles. SIGMOD Rec 19(2):322–331
15. Belhassena A, HongZhi W (2017) Distributed skyline trajectory query processing. In: Proceedings of the ACM Turing 50th Celebration Conference-China, p 19. ACM
16. Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
17. Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Machine Learn Res 3(Jan):993–1022
18. Bok KS, Seo DM, Shin SS, Yoo JS (2004) TPKDB-Tree: An index structure for efficient retrieval of future positions of moving objects. In: International conference on conceptual modeling, pp 67–78. Springer
19. Brisaboa NR, Ladra S (2009) Navarro, g.: k2-trees for compact web graph representation. In: The international symposium on string processing and information retrieval, vol 9, pp 18–30
20. Burton FW, Kollias JG, Matsakis D, Kollias V (1990) Implementation of overlapping B-trees for time and space efficient representation of collections of similar files. Comput J 33(3):279–280
21. Cai M, Revesz P (2000) Parametric R-tree: An index structure for moving objects. In: International conference on management of data and advances in data management (COMAD'00)
22. Cai R, Lu Z, Wang L, Zhang Z, Fu TZ, Winslett M (2017) DITIR: Distributed Index for high throughput trajectory insertion and real-time temporal range query. The Proceedings of the VLDB Endowment (PVLDB'17) 10(12):1865–1868
23. Cai Y, Ng R (2004) Indexing spatio-temporal trajectories with chebyshev polynomials. In: International conference on management of data (SIGMOD'04), pp 599–610. ACM
24. Cha SK, Hwang S, Kim K, Kwon K (2001) Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems. In: The Proceedings of the VLDB Endowment (PVLDB'01), vol 1, pp 181–190
25. Chakka VP, Everspaugh A, Patel JM (2003) Indexing large trajectory data sets with SETI. In: The biennial conference on innovative data systems research (CIDR'03)
26. Chen JD, Meng XF (2007) Indexing future trajectories of moving objects in a constrained network. J Comput Sci Technol 22(2):245–251
27. Chen N, Shou LD, Chen G, Dong JX (2008) Adaptive indexing of moving objects with highly variable update frequencies. J Comput Sci Technol 23(6):998–1014
28. Chen S, Ooi BC, Tan KL, Nascimento MA (2008) ST2B-tree: A self-tunable spatio-temporal b+-tree index for moving objects. In: International conference on management of data (SIGMOD'11), pp 29–42. ACM

29. Chen W, Zhao L, Jiajie X, Zheng K, Zhou X (2014) Ranking based activity trajectory search. In: International conference on web information systems engineering, pp 170–185. Springer
30. Chon HD, Agrawal D, El Abbadi A (2001) Storage and retrieval of moving objects. In: International conference on mobile data management (MDM'01), pp 173–184. Springer
31. Christoforaki M, He J, Dimopoulos C, Markowetz A, Suel T (2011) Text vs. space: efficient geo-search query processing. In: The ACM international conference on information and knowledge management (CIKM'11), pp 423–432
32. Cudre-Mauroux P, Wu E, Madden S (2010) Trajstore: an adaptive storage system for very large trajectory data sets. In: The international conference on data engineering (ICDE'10), pp 109–120. IEEE
33. Dai J, Lu CT (2011) DIME: Disposable Index for moving objects. In: The IEEE international conference on mobile data management (MDM'11), vol 1, pp 68–77
34. De Almeida VT, Güting RH (2005) Indexing the trajectories of moving objects in networks. Geoinformatica 9(1):33–60
35. Ding X, Lu Y, Ding X, Zhao N, Wei Q (2007) An efficient index for moving objects with frequent updates. In: International conference on wireless communications, networking and mobile computing (wicom'07), pp 5951–5954. IEEE
36. Ding Z (2008) UTR-Tree: An index structure for the full uncertain trajectories of network-constrained moving objects. In: International conference on mobile data management (MDM'08), pp 33–40. IEEE
37. Dittrich J, Blunschi L, Salles MAV (2009) Indexing moving objects using short-lived throwaway indexes. In: International symposium on spatial and temporal databases, pp 189–207. Springer
38. Dittrich J, Quiané-Ruiz JA (2012) Efficient big data processing in hadoop mapreduce. Proceedings of the VLDB Endowment (PVLD'12) 5(12):2014–2015
39. Doraiswamy H, Vo HT, Silva CT, Freire J (2016) A GPU-based index to support interactive spatio-temporal queries over historical data. In: The IEEE international conference on data engineering (ICDE'16), pp 1086–1097
40. Elbassioni K, Elmasry A, Kamel I (2003) An efficient indexing scheme for multi-dimensional moving objects. In: International conference on database theory, pp 425–439. Springer
41. Eldawy A, Mokbel MF (2015) Spatialhadoop: a mapreduce framework for spatial data. In: The IEEE international conference on data engineering (ICDE'15), pp 1352–1363
42. Fan P, Li G, Yuan L, Li Y (2012) Vague continuous k-nearest neighbor queries over moving objects with uncertain velocity in road networks. Inf Syst 37(1):13–32
43. Fang Y, Cao J, Peng Y, Wang L (2008) Indexing the past, present and future positions of moving objects on fixed networks. In: International conference on computer science and software engineering, vol 4, pp 524–527. IEEE
44. Fang Y, Cao J, Wang J, Peng Y, Song W (2011) HTPR*-Tree: An efficient index for moving objects to support predictive query and partial history query. In: International conference on web-age information management (WAIM'11), pp 26–39. Springer
45. Feng J, Lu J, Zhu Y, Mukai N, Watanabe T (2007) Indexing of moving objects on road network using composite structure. In: International conference on knowledge-based and intelligent information and engineering systems, pp 1097–1104. Springer
46. Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta informatica 4(1):1–9
47. Frentzos E (2003) Indexing objects moving on fixed networks. In: International symposium on spatial and temporal databases, pp 289–305. Springer
48. Ghanem TM, Hammad MA, Mokbel MF, Aref WG, Elmagarmid AK (2007) Incremental evaluation of sliding-window queries over data streams. IEEE Trans Knowl Data Eng (TKDE) 19(1):57–72
49. Gionis A, Indyk P, Motwani R et al (1999) Similarity search in high dimensions via hashing. In: The Proceedings of the VLDB Endowment (PVLDB'99), vol 99, pp 518–529
50. Gravano L, Ipeirotis PG, Jagadish HV, Koudas N, Muthukrishnan S, Srivastava D et al (2001) Approximate string joins in a database (almost) for free. In: The Proceedings of the VLDB Endowment (PVLDB'01), vol 1, pp 491–500
51. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. SIGMOD Rec 14:47–57
52. Hadjieleftheriou M, Kollios G, Tsotras VJ, Gunopulos D (2002) Efficient indexing of spatiotemporal objects. In: International conference on extending database technology, pp 251–268. Springer
53. Han L, Huang L, Yang X, Pang W, Wang K (2016) A novel spatio-temporal data storage and index method for ARM-based hadoop server. In: International conference on cloud computing and security, pp 206–216. Springer
54. Han Y, Wang L, Zhang Y, Zhang W, Lin X (2015) Spatial keyword range search on trajectories. In: The international conference on database systems for advanced applications (DASFAA'15), pp 223–240

55. Hariharan R, Hore B, Li C, Mehrotra S (2007) Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In: The international conference on scientific and statistical database management (SSDBM'07), pp 16–16
56. He Z, Kraak MJ, Huisman O, Ma X, Xiao J (2013) Parallel indexing technique for spatio-temporal data. ISPRS J Photogramm Remote Sens 78:116–128
57. Hendawi AM, Bao J, Mokbel MF, Ali M (2015) Predictive tree: an efficient index for predictive queries on road networks. In: The IEEE international conference on data engineering (ICDE'15), pp 1215–1226
58. Issa H, Damiani ML (2016) Efficient access to temporally overlaying spatial and textual trajectories. In: The IEEE international conference on mobile data management (MDM'16), vol 1, pp 262–271
59. Jackins CL, Tanimoto SL (1980) OCT-Trees and their use in representing three-dimensional objects. Comput Graphics and Image Process 14(3):249–270
60. Jensen CS, Lin D, Ooi BC (2004) Query and update efficient b+-tree based indexing of moving objects. In: The Proceedings of the VLDB Endowment (PVLDB'04), pp 768–779
61. Jensen CS, Lu H, Yang B (2009) Indexing the trajectories of moving objects in symbolic indoor space. In: International symposium on spatial and temporal databases, pp 208–227. Springer
62. Jeung H, Yiu ML, Zhou X, Jensen CS (2010) Path prediction and predictive range querying in road network databases. Intern J Very Large Data Bases (VLDB J) 19(4):585–602
63. Kim KS, Kim SW, Kim TW, Li KJ (2003) Fast indexing and updating method for moving objects on road networks. In: International conference on web information systems engineering workshops, pp 34–42. IEEE
64. Knuth D (1973) The art of computer programming
65. Kollios G, Gunopulos D, Tsotras VJ (1999) On indexing mobile objects. In: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS), pp 261–272. ACM
66. Kollios G, Tsotras VJ, Gunopulos D, Delis A, Hadjieleftheriou M (2001) Indexing animated objects using spatiotemporal access methods. IEEE Trans Knowl Data Eng (TKDE) 13(5):758–777
67. Kumar A, Tsotras VJ, Faloutsos C (1998) Designing access methods for bitemporal databases. IEEE Trans Knowl Data Eng (TKDE) 10(1):1–20
68. Kwon D, Lee S, Lee S (2002) Indexing the current positions of moving objects using the lazy update R-tree. In: International conference on mobile data management (MDM'03), pp 113–120. IEEE
69. Le TTT, Nickerson BG (2008) Efficient search of moving objects on a planar graph. In: International conference on advances in geographic information systems (SIGSPATIAL'08), p 41. ACM
70. Lee ML, Hsu W, Jensen CS, Cui B, Teo KL (2003) Supporting frequent updates in R-trees: a bottom-up approach. In: The Proceedings of the VLDB Endowment (PVLDB'03), pp 608–619
71. Liang Y (2011) A efficient indexing maintenance method for grouping moving objects with grid. pp 486–492 Elsevier
72. Liao W, Tang G, Jing N, Zhong Z (2006) VTPR-Tree: An efficient indexing method for moving objects with frequent updates. In: International conference on conceptual modeling, pp 120–129. Springer
73. Lin B, Mokhtar H, Pelaez-Aguilera R, Su J (2003) Querying moving objects with uncertainty. In: Vehicular technology conference (VTC'03), vol 4, pp 2783–2787. IEEE
74. Lin B, Su J (2005) Handling frequent updates of moving objects. In: International conference on information and knowledge management, pp 493–500. ACM
75. Lin D, Jensen CS, Ooi BC, Šaltenis S (2005) Efficient indexing of the historical, present, and future positions of moving objects. In: International conference on mobile data management (MDM'05), pp 59–66. ACM
76. Lin D, Jensen CS, Zhang R, Xiao L, Lu J (2011) A moving-object index for efficient query processing with peer-wise location privacy. The Proceedings of the VLDB Endowment (PVLDB'11) 5(1):37–48
77. Lin D, Zhang R, Zhou A (2006) Indexing fast moving objects for kNN queries based on nearest landmarks. Geoinformatica 10(4):423–445
78. Lin HY (2009) Indexing the trajectories of moving objects. International multi-conference of engineers and computer scientists
79. Liu H, Xu J, Zheng K, Liu C, Du L, Wu X (2017) Semantic-aware query processing for activity trajectories. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp 283–292. ACM
80. Liu Z, Liu X, Ge J, Bae H (2005) Indexing large moving objects from past to future with PCFI+-index. In: International conference on management of data and advances in data management (COMAD'05), pp 131–137
81. Lomet D, Salzberg B (1989) Access methods for multiversion data, vol 18. ACM
82. Luo W, Tan H, Chen L, Ni LM (2013) Finding time period-based most frequent path in big trajectory data. In: The international conference on management of data (SIGMOD'13), pp 713–724

83. Ma C, Lu H, Shou L, Chen G (2013) KSQ: Top-K similarity query on uncertain trajectories. IEEE Trans Knowl Data Eng (TKDE) 25(9):2049–2062
84. MacQueen J et al (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1, pp 281–297
85. Magdy A, Aly AM, Mokbel MF, Elnikety S, He Y, Nath S, Aref WG (2016) GeoTrend: Spatial trending queries on real-time microblogs. In: The ACM international conference on advances in geographic information systems (SIGSPATIAL'16), p 7
86. Magdy A, Mokbel MF, Elnikety S, Nath S, He Y (2014) Mercury: a memory-constrained spatio-temporal real-time search on microblogs. In: The IEEE international conference on data engineering (ICDE'14), pp 172–183
87. Mahmood AR, Aly AM, Kuznetsova T, Basalamah S, Aref WG (2018) Disk-based indexing of recent trajectories. ACM Transactions on Spatial Algorithms and Systems (TSAS) 4(3):7.1–7.27
88. Mahmood AR, Aref WG, Aly AM, Basalamah S (2014) Indexing recent trajectories of moving objects. In: The ACM international conference on advances in geographic information systems (SIGSPATIAL'14), pp 393–396
89. Meagher DJ (1980) OCTRee encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory
90. Mehta P, Skoutas D, Voisard A (2015) Spatio-temporal keyword queries for moving objects. In: The ACM international conference on advances in geographic information systems (SIGSPATIAL'15), p 55
91. Mokbel MF, Ghanem TM, Aref WG (2003) Spatio-temporal access methods. IEEE Data Eng Bull 26(2):40–49
92. Morton GM (1966) A computer oriented geodetic data base and a new technique in file sequencing. International Business Machines Company, New York
93. Mukai N, Feng J, Watanabe T (2004) Heuristic approach based on lambda-interchange for VRTPR-tree on specific vehicle routing problem with time windows. In: International conference on industrial, engineering and other applications of applied intelligent systems, pp 229–238. Springer
94. Mukai N, Feng J, Watanabe T (2004) Indexing approach for delivery demands with time constraints. In: Pacific rim international conference on artificial intelligence, pp 95–103. Springer
95. Nascimento MA, Silva JR (1998) Towards historical R-trees. In: Symposium on applied computing, pp 235–240. ACM
96. Nascimento MA, Silva JR, Theodoridis Y (1999) Evaluation of access structures for discretely moving points. In: Spatio-temporal database management, pp 171–189. Springer
97. Nguyen T, He Z, Chen YPP (2012) SeTPR*-tree: Efficient buffering for spatiotemporal indexes via shared execution. Comput J 56(1):115–137
98. Nguyen T, He Z, Zhang R, Ward P (2012) Boosting moving object indexing through velocity partitioning. The Proceedings of the VLDB Endowment (PVLDB'12) 5(9):860–871
99. Nguyen-Dinh LV, Aref WG, Mokbel MF (2010) Spatio-temporal access methods: Part 2 (2003-2010). IEEE Data Eng Bull 33(2):46–55
100. Ni J, Ravishankar CV (2005) PA-Tree: A parametric indexing scheme for spatio-temporal trajectories. In: International symposium on spatial and temporal databases, pp 254–272. Springer
101. Nievergelt J, Hinterberger H, Sevcik KC (1984) The grid file: an adaptable, symmetric multikey file structure. ACM Trans Database Syst (TODS) 9(1):38–71
102. Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS), pp 181–190. ACM
103. Patel JM, Chen Y, Chakka VP (2004) STRIPES: An efficient index for predicted trajectories. In: The international conference on management of data (SIGMOD'04), pp 637–646
104. Patroumpas K, Sellis T (2009) Monitoring orientation of moving objects around focal points. In: International symposium on spatial and temporal databases, pp 228–246. Springer
105. Pelanis M, Šaltenis S, Jensen CS (2006) Indexing the past, present, and anticipated future positions of moving objects. ACM Trans Database Syst (TODS) 31(1):255–298
106. Pfoser D, Jensen CS, Theodoridis Y et al (2000) Novel approaches to the indexing of moving object trajectories. In: The Proceedings of the VLDB Endowment (PVLDB'00), pp 395–406
107. Popa IS, Zeitouni K, Oria V, Barth D, Vial S (2010) PARINET: A tunable access method for in-network trajectories. In: The IEEE international conference on data engineering (ICDE'10), pp 177–188. IEEE
108. Porkaew K, Lazaridis I, Mehrotra S (2001) Querying mobile objects in spatio-temporal databases. In: International symposium on spatial and temporal databases (SSTD'01), pp 59–78. Springer

109. Prabhakar S, Xia Y, Kalashnikov DV, Aref WG, Hambrusch SE (2002) Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans Comput 51(10):1124–1140
110. Procopiuc CM, Agarwal PK, Har-Peled S (2002) Star-tree: an efficient self-adjusting index for moving objects. In: Workshop on algorithm engineering and experimentation, pp 178–193. Springer
111. Pugh W (1990) Concurrent maintenance of lists. In: Dept. of computer science, university of maryland, college park
112. Ranu S, Deepak P, Telang AD, Deshpande P, Raghavan S (2015) Indexing and matching trajectories under inconsistent sampling rates. In: The IEEE international conference on data engineering (ICDE'15), pp 999–1010
113. Ray S (2014) Towards high performance spatio-temporal data management systems. In: The IEEE international conference on mobile data management (MDM'14), vol 2, pp 19–22
114. Romero M, Brisaboa N, Rodríguez MA (2012) The SMO-index: A succinct moving object structure for timestamp and interval queries. In: Advances in geographic information systems, pp 498–501
115. Saltenis S, Jensen CS (2002) Indexing of moving objects for location-based services. In: International conference on data engineering (ICDE'02), pp 463–472. IEEE
116. Šaltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. In: International conference on management of data (SIGMOD'00), vol 29, pp 331–342. ACM
117. Sandu Popa I, Zeitouni K, Oria V, Barth D, Vial S (2011) Indexing in-network trajectory flows. Intern J Very Large Data Bases (VLDB J) 20(5):643–669
118. Schmiegelt P, Behrend A, Seeger B, Koch W (2014) A concurrently updatable index structure for predicted paths of moving objects. Data Knowl Eng 93:80–96
119. Senechal M (1993) Spatial tessellations: Concepts and applications of voronoi diagrams. Science 260(5111):1170–1173
120. Seo DM, Song SI, Park YH, Yoo JS, Kim MH (2008) Bdh-tree: A B+-tree based indexing method for very frequent updates of moving objects. In: International symposium on computer science and its applications (CSA'08), pp 314–319. IEEE
121. Shen B, Zhao Y, Li G, Zheng W, Qin Y, Yuan B, Rao Y (2017) V-Tree: Efficient kNN search on moving objects with road-network constraints. In: The IEEE international conference on data engineering (ICDE'17), pp 609–620
122. Šidlauskas D, Ross K, Jensen C, Šaltenis S (2011) Thread-level parallel indexing of update intensive moving-object workloads. Adv Spatial Temporal Database 6849:186–204
123. Šidlauskas D, Šaltenis S, Christiansen CW, Johansen JM, Šaulys D (2009) Trees or grids?: indexing moving objects in main memory. In: The ACM international conference on advances in geographic information systems (SIGSPATIAL'09), pp 236–245
124. Šidlauskas D, Šaltenis S, Jensen CS (2012) Parallel main-memory indexing for moving-object query and update workloads. In: The international conference on management of data (SIGMOD'12), pp 37–48
125. Silva YN, Xiong X, Aref WG (2009) The RUM-tree: supporting frequent updates in R-trees using memos. Intern J Very Large Data Bases (VLDB J) 18(3):719–738
126. Singh M, Zhu Q, Jagadish H (2012) SWST: A disk based index for sliding window spatio-temporal data. In: The IEEE international conference on data engineering (ICDE'12), pp 342–353
127. Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: The IEEE international conference on data engineering (ICDE'14), pp 148–159
128. Song Z, Roussopoulos N (2001) Hashing moving objects. In: International conference on mobile data management (MDM'01), pp 161–172. Springer
129. Song Z, Roussopoulos N (2003) SEB-Tree: An approach to index continuously moving objects. In: International conference on mobile data management (MDM'03), pp 340–344. Springer
130. Stantic B, Topor R, Terry J, Sattar A (2010) Advanced indexing technique for temporal data. Computer Science and Information Systems 7(4):679–703
131. Tanimoto S, Pavlidis T (1975) A hierarchical data structure for picture processing. Comput Graphics Image Process 4(2):104–119
132. Tao Y, Faloutsos C, Papadias D, Liu B (2004) Prediction and indexing of moving objects with unknown motion patterns. In: International conference on management of data (SIGMOD'04), pp 611–622. ACM
133. Tao Y, Papadias D (2001) Efficient historical R-trees. In: The international conference on scientific and statistical database management (SSDBM'01), p 0223. IEEE
134. Tao Y, Papadias D (2001) MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In: The Proceedings of the VLDB Endowment (PVLDB'01), pp 431–440

135. Tao Y, Papadias D, Sun J (2003) The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In: International conference on very large data bases (PVLDB'03), pp 790–801. VLDB endowment
136. Tayeb J, Ulusoy Ö, Wolfson O (1998) A quadtree-based dynamic attribute indexing method. Comput J 41(3):185–200
137. That DHT, Popa IS, Zeitouni K (2015) TRIFL: A generic trajectory index for flash storage. ACM Trans Spatial Algorithm Syst 1(2):6
138. Theodoridis Y, Vazirgiannis M, Sellis T (1996) Spatio-temporal indexing for large multimedia applications. In: International conference on multimedia computing and systems, pp 441–448. IEEE
139. To QC, Dang TK, Kung J (2011) OST-Tree: An access method for obfuscating spatio-temporal data in location based services. In: International conference on new technologies, mobility and security (NTMS'11), pp 1–5. IEEE
140. Toshniwal A, Taneja S et al (2014) Storm@ twitter. In: The international conference on management of data (SIGMOD'14), pp 147–156
141. Tung HDT, Jung YJ, Lee EJ, Ryu KH (2004) Moving point indexing for future location query. In: International conference on conceptual modeling, pp 79–90. Springer
142. (2018) Twitter. https://twitter.com
143. Tzouramanis T, Vassilakopoulos M, Manolopoulos Y (1998) Overlapping linear quadtrees: a spatio-temporal access method. In: International symposium on advances in geographic information systems, pp 1–7. ACM
144. Ulrich T (2000) Loose octrees. Game Programming Gems 1:434–442
145. Valdés F, Güting RH (2017) Index-supported pattern matching on tuples of time-dependent values. GeoInformatica 21(3):429–458
146. Wang H, Belhassena A (2017) Parallel trajectory search based on distributed index. Inf Sci 388:62–83
147. Wang L, Zheng Y, Xie X, Ma WY (2008) A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval. In: International conference on mobile data management (MDM'08), pp 1–8. IEEE
148. Wang S, Bao Z, Culpepper JS, Sellis T, Sanderson M, Qin X (2017) Answering top-k exemplar trajectory queries. In: The IEEE international conference on data engineering (ICDE'17), pp 597–608. IEEE
149. Wang X, Zhang Y, Zhang W, Lin X, Wang W (2015) AP-Tree: Efficiently support location-aware publish/subscribe. Intern J Very Large Data Bases (VLDB J.) 24(6):823–848
150. Xu X, Lu JHW (1990) RT-tree: An improved R-tree indexing structure for temporal spatial databases. In: The international symposium on spatial data handling, pp 1040–1049
151. Xie X, Lu H, Pedersen TB (2013) Efficient distance-aware query evaluation on indoor moving objects. In: The IEEE international conference on data engineering (ICDE'13), pp 434–445. IEEE
152. Xie X, Mei B, Chen J, Du X, Jensen CS (2016) Elite: an elastic infrastructure for big spatiotemporal trajectories. Intern J Very Large Data Bases (VLDB J) 25(4):473–493
153. Xiong X, Aref WG (2006) R-trees with update memos. In: The IEEE international conference on data engineering (ICDE'06), pp 22–22
154. Xiong X, Mokbel MF, Aref WG (2006) LUGRid: Update-tolerant grid-based indexing for moving objects. In: International conference on mobile data management (MDM'13), p 13
155. Xu X, Xiong L, Sunderam V (2016) D-grid: an in-memory dual space grid index for moving object databases. In: The IEEE international conference on mobile data management (MDM'16), pp 252–261
156. Xu X, Xiong L, Sunderam V, Liu J, Luo J (2015) Speed partitioning for indexing moving objects. In: The international symposium on spatial and temporal databases (SSTD'15), pp 216–234
157. Xu Y, Tan G (2014) Sim-Tree: indexing moving objects in large-scale parallel microscopic traffic simulation. In: ACM Conference on principles of advanced discrete simulation (PADS) (SIGSIM'14), pp 51–62
158. YAN Qy, MENG Fr (2004) Multiple version TPR-tree. Comput Eng Design 10:057
159. Yan X, Guo J, Lan Y, Cheng X (2013) A biterm topic model for short texts. In: Proceedings of the 22nd international conference on World Wide Web, pp 1445–1456. ACM
160. Yao B, Li F, Hadjieleftheriou M, Hou K (2010) Approximate string search in spatial databases. In: The IEEE international conference on data engineering (ICDE'10), pp 545–556. IEEE
161. Yiu ML, Tao Y, Mamoulis N (2008) The Bdual-tree: Indexing moving objects by space filling curves in the dual space. Intern J Very Large Data Bases (VLDB J) 17(3):379–400
162. Yu Z, Liu Y, Yu X, Pu KQ (2015) Scalable distributed processing of k nearest neighbor queries over moving objects. IEEE Trans Knowl Data Eng (TKDE) 27(5):1383–1396
163. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ et al (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65

164. Zäschke T, Zimmerli C, Norrie MC (2014) The PH-tree: A space-efficient storage structure and multi-dimensional index. In: The international conference on management of data (SIGMOD'14), pp 397–408
165. Zheng B, Yuan NJ, Zheng K, Xie X, Sadiq S, Zhou X (2015) Approximate keyword search in semantic trajectory database. In: The IEEE international conference on data engineering (ICDE'15), pp 975–986. IEEE
166. Zheng K, Shang S, Yuan NJ, Yang Y (2013) Towards efficient search for activity trajectories. In: The IEEE international conference on data engineering (ICDE'13), pp 230–241. IEEE
167. Zheng K, Trajcevski G, Zhou X, Scheuermann P (2011) Probabilistic range queries for uncertain trajectories on road networks. In: The international conference on extending database technology (EDBT'11), pp 283–294
168. Zheng K, Zheng B, Xu J, Liu G, Liu A, Li Z (2016) Popularity-aware spatial keyword search on activity trajectories. World Wide Web 4(20):749–773
169. Zhou P, Zhang D, Salzberg B, Cooperman G, Kollios G (2005) Close pair queries in moving object databases. In: Proceedings of the 13th annual ACM international workshop on Geographic information systems, pp 2–11. ACM
170. Zhu Y, Ren X, Feng J (2006) NCO-Tree: A spatio-temporal access method for segment-based tracking of moving objects. In: International conference on knowledge-based and intelligent information and engineering systems, pp 1191–1198. Springer
171. Zhu Y, Wang S, Zhou X, Zhang Y (2013) RUM+-Tree: A new multidimensional index supporting frequent updates. In: The international conference on web-age information management (WAIM'13), pp 235–240

Ahmed R. Mahmood is a Ph.D. candidate at the Department of Computer Science, Purdue University. His research interests are spatial, spatial-keyword, and distributed stream processing. He is the first place winner of the 2017 ACM SIGSPATIAL student research competition. He has been awarded the Purdue CS Teaching Fellowship, the Teaching Academy Graduate Teaching Award, and the Raymond Boyce Graduate Teacher Award. Ahmed is the main designer and developer of Tornado; the first distributed spatial-keyword stream processing system. For more information, please visit: http://www.cs.purdue.edu/homes/amahmoo.

Sri Punni is currently a software engineer at Amazon Inc. She received her master's degree from the Computer Science Department, Purdue University. She received her Bachelor degree in computer science from Vellore Institute of Technology, India. Her research interests are in the area of spatial and spatio-temporal data indexing techniques.

Walid G. Aref is a professor of computer science at Purdue. His research interests are in extending the functionality of database systems in support of emerging applications, e.g., spatial, spatio-temporal, graph, biological, and sensor databases. He is also interested in query processing, indexing, data streaming, and geographic information systems (GIS). Walid's research has been supported by the National Science Foundation, the National Institute of Health, Purdue Research Foundation, Qatar National Research Foundation, CERIAS, Panasonic, and Microsoft Corp. In 2001, he received the CAREER Award from the National Science Foundation and in 2004, he received a Purdue University Faculty Scholar award. Walid is a member of Purdue's CERIAS. He is the Editor-in-Chief of the ACM Transactions of Spatial Algorithms and Systems (ACM TSAS), and an editorial board member of the Journal of Spatial Information Science (JOSIS), and has served as an editor of the VLDB Journal and the ACM Transactions of Database Systems (ACM TODS). Walid has won several best paper awards including the 2016 VLDB ten-year best paper award. He is a Fellow of the IEEE, and a member of the ACM. Between 2011 and 2014, Walid has served as the chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL).