Chapter 9
Multi-attribute Trajectory Data Management

Jianqiu Xu

9.1 Introduction

Trajectory data, keeping track of historical movements of moving objects such as vehicles and ships, is becoming ubiquitous due to the widespread use of GPS devices. Such data that records geographical locations changing over time is of crucial importance for emerging applications, e.g., route recommendation (Chen et al. 2010; Tong et al. 2017, 2018), tracking (Lange et al. 2011), monitoring (Yao et al. 2014), to name but a few.

Despite tremendous efforts made on studying trajectory databases, proposals in the literature mainly deal with *standard trajectories* (Tzoumas et al. 2009; Long et al. 2013; Zheng et al. 2013b), i.e., a sequence of time-stamped geo-locations. The majority of queries are limited to the spatio-temporal evaluation such as range queries (Wang and Zimmermann 2011), nearest neighbors (Güting et al. 2010b) and convoys (Jeung et al. 2008). In the real world, typical moving objects such as vehicles and persons are associated with pieces of descriptive information. The database system should represent moving objects by considering several aspects and allow users to query objects with extensive knowledge to better understand the movement and users' behavior. As a fundamental step towards that, a new form of trajectories is investigated called *multi-attribute trajectories*, each of which consists of a standard trajectory and a set of attribute values. Modeling and representing standard trajectories has been well established (Güting and Schneider 2005), while attributes have various semantics according to applications. The combination allows users to issue queries with both spatio-temporal and attribute predicates.

J. Xu (✉)
Nanjing University of Aeronautics and Astronautics, Nanjing, China
e-mail: jianqiu@nuaa.edu.cn

M. Werner, Y.-Y. Chiang (eds.), *Handbook of Big Geospatial Data*,
https://doi.org/10.1007/978-3-030-55462-0_9

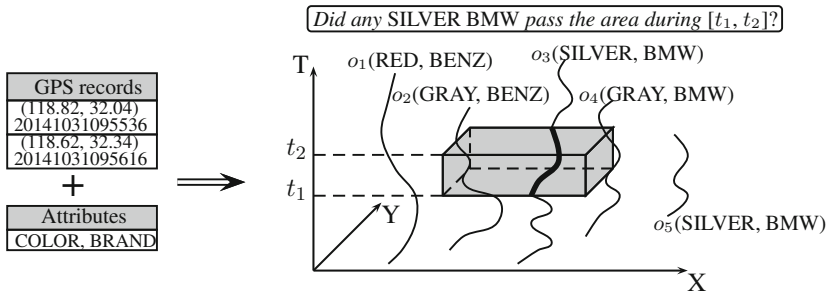


Fig. 9.1 Querying multi-attribute trajectories

Consider a database storing vehicle trips in a city. Each trip contains a standard trajectory and two attribute values over domains COLOR = {RED, SILVER, GRAY} and BRAND = {BENZ, BMW}, respectively, as illustrated in Fig. 9.1. A query that contains a tuple of attribute values and a spatio-temporal box is issued, that is, "*Did any* SILVER BMW *pass the area during* [t_1, t_2]?". Boolean range queries are studied to report objects containing query attribute values and fulfilling the spatio-temporal condition. In the example, o_3 is returned. Although o_2 intersects the query window, it is not a SILVER BMW.

Recently, researchers have started to investigate spatio-temporal trajectories annotated with additional information, e.g., semantic trajectories (Yan et al. 2011; Parent et al. 2013; Zhang et al. 2014; Zheng et al. 2015), activities trajectories (Zheng et al. 2013a), and transportation modes (Xu and Güting 2013). In particular, a semantic trajectory is essentially an enriched version of a standard trajectory in terms of locations. Labels are attached to geo-locations to describe places that users have visited or performed activities at, e.g., *hotel*, *sport*, *restaurant*. However, semantic data is restricted to locations. This is orthogonal to multi-attribute trajectories that consider location-independent attributes. The major differences include three aspects.

- Attributes represent a range of aspects and aim to provide a full picture of moving objects, as opposed to semantics limited to locations. This will support a different (even broader) range of applications.
- Semantic locations are sparsely defined because among a person's trajectory a few locations have semantics. Attributes are location-independent and associated with the complete trajectory. They are not derived from time-stamped locations or the geographical environment. For example, a semantic trajectory is of the form $o = \langle (loc_1, t_1, coffee), (loc_2, t_2, pizza) \rangle$, where *coffee* and *pizza* are defined at two locations. That is, there is no semantic at locations between loc_1 and loc_2. A multi-attribute trajectory is of the form $o = \langle (loc_1, t_1), (loc_2, t_2) \rangle$, (RED, BENZ), where (RED, BENZ) is associated with the complete trajectory.
- Semantic trajectories cope with similarity search or ranking queries rather than the exact match on attribute values with spatio-temporal predicts, leading to

different tasks when developing the index. Semantic trajectories are grouped in terms of locations and semantics, but attributes are not related to locations.

To efficiently process multi-attribute trajectories, an index is essentially required because a sequential scan over the database is prohibitively expensive for large datasets. Standard trajectory indexes such as TB-tree (Pfoser and Jensen 2000), SETI (Chakka et al. 2003) and TrajStore (Mauroux et al. 2010) only deal with the spatio-temporal data without managing attributes. Such a method is suboptimal because one cannot use the index to prune the search space at the attribute level. As a result, objects after performing the spatio-temporal evaluation are sequentially processed, significantly inhibiting the performance. Furthermore, the pruning technique of *min* and *max* distances[1] cannot be applied for nearest neighbor queries if attribute values are not determined. This is because objects that are close to the query may not fulfill the attribute condition and cannot be used for pruning further objects. The trajectory subset containing query attributes changes according to the query setting and cannot be pre-computed. False dismissals will occur if one performs the pruning without the awareness of attribute values.

One can employ two individual indexes (e.g., a 3-D R-tree and a B-tree) on standard trajectories and attributes, respectively. The problem is, when the query evaluates the selective predicate on both parts, an intersection will be performed on two candidate sets that are separately retrieved, which is suboptimal. Another solution is to employ an attribute index. The method first receives trajectories containing query attributes and then proceeds to processing standard trajectories. However, this method is limited in scope and inherently suffers from the performance issue. Standard trajectories will be processed by either performing a sequential scan or accessing an on-line built index. If the attribute predicate is selective, the query cost may be acceptable because a small dataset is processed. If the attribute predicate has a poor selectivity, a large number of trajectories will be returned. Both the sequential scan and building an on-line index incur high CPU and I/O costs. Furthermore, creating an index for each query at runtime causes extra storage space. This calls for a structure that is able to simultaneously manage both standard trajectories and attributes. Meanwhile, the structure should be general and flexible in order to answer queries on standard trajectories and support update-intensive applications. From a system point of view, existing techniques need to be extended or adapted to deal with coming issues rather than developing individual structures each of which only applies to one problem.

The rest of the chapter is organized as follows. Related work is analyzed in Sect. 9.2. Multi-attribute trajectories and queries are defined in Sect. 9.3. Indexing and querying multi-attribute trajectories are introduced in Sects. 9.4 and 9.5,

[1]Given three rectangles a, b, c, each contains a set of points inside. We aim to find the nearest point to a given point inside a. Let $Max(b, a)$ and $Min(c, a)$ denote the maximum and minimum distances between two rectangles. If $Max(b, a) \leq Min(c, a)$, then no point inside the rectangle c can be closer than a point in the rectangle b to a. As a consequence, we can omit c when searching for the nearest neighbor to a.

respectively. The system development is presented in Sect. 9.6 and the performance evaluation is reported in Sect. 9.7. Future directions are pointed out in Sect. 9.8, followed by conclusions in Sect. 9.9.

9.2 Related Work

The current state-of-the-art is classified into two parts: (i) extending the representation of standard trajectories by incorporating semantics, and (ii) indexing standard trajectories with additional data.

9.2.1 Enriching Spatio-Temporal Trajectories

Semantic trajectories Emerging applications require extensive information about trajectories such as quality and semantics (Zheng and Su 2015). *Semantic trajectories* are based on discovering meaningful knowledge from locations (Alvares et al. 2007; Yan et al. 2011; Zheng et al. 2015). Formally,

Definition 1 (Semantic trajectory) A semantic trajectory is represented by a sequence of time-stamped positions complemented with annotations, that is, $o_{\text{sem}} = \langle (loc_1, t_1, \mathcal{A}_1), \ldots, (loc_n, t_n, \mathcal{A}_n) \rangle$ in which $loc \in \mathcal{R}^2$, $t \in T$, and $\mathcal{A}$ is a set of labels (strings) describing locations.

Interesting patterns can be properly defined and extracted. For example, a so-called *fine-grained sequential pattern* reports trajectories that satisfy spatial compactness, semantic consistency and temporal continuity simultaneously (Zhang et al. 2014). Consider actions that users can take at particular places such as *sport*, *dining* and *entertaining*. *Activity trajectories* are defined by associating geo-spatial points with activities. A similarity search returns k trajectories whose semantics contain the query and have the shortest minimum match distance (Zheng et al. 2013a). Motivated by the fact that standard trajectories do not make much sense for humans, a *partition-and-summarization* approach is proposed to automatically generate texts to highlight the significant semantic behavior (Su et al. 2015). A good survey of semantic trajectories refers to Parent et al. (2013).

Motion modes Moving objects with transportation modes are investigated in Xu and Güting (2013) and Xu et al. (2015a,b) . A trajectory over diverse geographical spaces includes time-stamped locations and a sequence of transportation modes such as *Indoor* $\rightarrow$ *Walk* $\rightarrow$ *Car*. Queries containing transportation modes can be answered, e.g., "*who arrived at the university by taxi*".

Definition 2 (Trajectories with transportation modes) A trip with transportation modes is represented by a sequence of units, each of which defines the movement over a time interval and a certain mode. That is, each unit is of the form $u_{\text{tm}} = (loc_1,$

loc_2, t_1, t_2, m) in which $loc_1, loc_2 = (oid, loc')$, $oid \in int$, $loc' \in \mathcal{R}^2$, $t \in T$, and $m \in$ {Indoor, Walk, Car, Bus, Metro, Bike, Taxi}.

The location representation employs a reference system in which *oid* points to a geographical object such as a road, a walking area or a bus. Then, the relative location in the geographical object is recorded. The transportation mode does not change for each piece of movements.

Symbolic trajectories The task is to deal with generic semantic information including transportation modes and users' activities (Valdés and Güting 2014; Güting et al. 2015). A generic model is proposed to capture a wide range of meanings derived from a standard trajectory. The symbolic information is computed from the movement itself or obtained from the geographical environment, and a symbolic trajectory is represented by a time-dependent label. Typical examples include names of roads, activities and transportation modes. The goal is to provide a simple and flexible model for any kind of semantic information, while geometric locations are not defined.

Definition 3 (Symbolic trajectory) A symbolic trajectory is represented as a sequence of pairs (t, l), in which t is a time interval and l is a label (short character string) describing certain aspects of a trajectory.

If transportation modes are considered, a symbolic trajectory is denoted by $o_{\text{sym}} = \langle([t_1, t_2], \textit{Walk}), ([t_2, t_3], \textit{Bus}), ([t_3, t_4], \textit{Metro}), ([t_4, t_5], \textit{Walk}), ([t_5, t_6], \textit{Indoor})\rangle$.

There are fundamental differences between those works and multi-attribute trajectories. First, multi-attribute trajectories consider attributes that are location-independent, differing from attaching location labels in semantic trajectories. Symbolic trajectories do not contain geo-locations, while multi-attribute trajectories do. Multi-attribute trajectories are defined in a broad context by annotating trajectories with domain-specific attributes such that users can issue queries combining different aspects of moving objects. Second, different queries are evaluated. Multi-attribute trajectories incorporate attributes into the evaluation for Boolean queries and search for the objects fulfilling the spatio-temporal condition during a time interval or at each time point. Previous queries deal with spatial closeness and attributes similarity instead of time-dependent distances and exact matches on attributes. Labels are sparsely defined in semantic trajectories because a few locations may contain semantics. As a result, ranking queries are primarily dealt with rather than the spatio-temporal evaluation at each time point with attributes.

Heterogeneous k-nearest neighbor queries are studied in Su et al. (2007). A moving object is represented by a location-independent attribute and a set of coordinates. By defining a function that combines the costs of distances and the location-independent attribute, the query returns objects having the k-th smallest value. Although the work considers the location-independent attribute, there are three major differences in comparison with ours. First, the data representation is limited in scope because each moving object is associated with only one attribute.

Second, they query objects based on a ranking function on distance and attribute, but our queries require exact matches on attribute, leading to different results. Third, their distance function is not time-dependent, while queries of multi-attribute trajectories support distances changing over time.

Spatial keywords Queries of spatial keywords have been extensively studied in the literature (Chen et al. 2013; Lee et al. 2015; Wang et al. 2016). The task is to support queries that take a geo-location and a set of text descriptions called *keywords* as augments and return (i) objects that are close to the query location and contain the keywords called Boolean *k*NN query (De Felipe et al. 2008), or (ii) objects with the highest ranking scores measured by a combination of distances to the query location and the text relevance to the keywords called Top-*k* NN query (Cong et al. 2009). To efficiently answer the query, a spatial index such as 2-D R-tree and a text index structure are combined. For example, the IR-tree (Cong et al. 2009) augments each node of the R-tree with a pointer to an inverted file that contains a summary of the text content of the objects in the corresponding subtree. During the query procedure, one uses the combined structure to estimate both the spatial distance and the text relevancy to prune the objects that cannot contribute to the result. However, spatial keywords focus on static geo-locations and location-dependent text descriptions, leading to different queries. Text descriptions and attributes will make different tasks when designing the index structure. The index groups close spatial objects in terms of spatial distances and location-related text relevances. It is possible to attach attributes to time-stamped locations, but each piece of trajectories will have all attributes along with the trajectory, resulting in an extremely large amount of redundant data. In fact, the key issue of boosting the index for multi-attribute trajectories is to know which objects contain particular attribute values and where the objects are located in the spatio-temporal index. Therefore, a different criterion is used to design the index.

9.2.2 Indexing Spatio-Temporal Trajectories

In the last decade, a substantial number of spatio-temporal index structures have been proposed to efficiently access trajectories. A good survey on trajectory indexing and retrieved is given in Dinh et al. (2010) and Zheng and Zhou (2011). Indices can be classified into three categories according to the environment: (1) free space (Pfoser and Jensen 2000; Tao and Papadias 2001; Chakka et al. 2003; Pelanis et al. 2006); (2) road network (Frentzos 2003; Pfoser and Jensen 2003; de Almeida and Güting 2005; Popa et al. 2011); and (3) indoor (Jensen et al. 2009; Lu et al. 2012). Several algorithms are proposed to minimize the total volume of trajectory approximations given a user-specified number of splits (Hadjieleftheriou et al. 2002). Rasetic et al. (2005) provide a better solution that splits trajectories into a number of sub-trajectories and builds an index on them to minimize the number of expected disk I/Os with respect to an average size of spatio-temporal range queries.

The method expects the query window as the input but in real applications the size of the window varies and the assumption leads to inaccurate estimations.

Indexing semantic and symbolic trajectories Recently, traditional spatio-temporal indexes have been studied to incorporate semantic information. A grid index is established to organize spatio-temporal trajectories with activities in a hierarchical manner (Zheng et al. 2013a). A similar structure is developed to incorporate both spatial and semantic information for approximate keyword search (Zheng et al. 2015). The grid is in fact a spatial index and is extended to maintain objects based on spatial and activity proximities for ranking queries. This line of work is not applicable to our problem. On the one hand, our attributes are not related to locations and therefore it does not make sense to group objects by considering both spatio-temporal data and attributes. On the other hand, our query reports trajectory objects rather than individual locations. A framework of analyzing large sets of movement data having time-dependent attributes is developed (Valdés and Güting 2017, 2019). They aim to support pattern matching queries on tuples of time-dependent values, e.g., "*return all tuples that include either a flight on Tuesday or a work in Dortmund with a later bus trip*". A new pattern language is proposed and the superiority is thoroughly analyzed in terms of flexibility and expressiveness. The corresponding matching algorithm uses a collection of different indexes and is divided into a filtering and an exact matching phase. A composite index structure for sets of tuples of time-dependent value is proposed in which a single index of a suitable type is created for each time-dependent attribute.

Indexing trajectories with keywords An index structure called IOC-Tree is proposed to answer spatial keyword range queries on trajectories (Han et al. 2015). The structure consists of an inverted index and a set of 3-D quadtrees termed *octrees*. The inverted index has two components: a search structure for all keywords and lists of references to documents containing words. One is called a *dictionary* and the other is called *inverted lists*. Each keyword is combined with one reference, that is an octree built on the keyword in the dictionary to organize relevant trajectory points. In an octree, each leaf node maintains a signature represented by a bit vector to summarize the identifications of a set of trajectories intersecting the node. The signature of a non-leaf node is achieved by performing the union on the signatures of its child nodes.

The IOC-Tree can be extended to solve our problem by setting attribute values as keywords associated with trajectory points. One can implement the inverted index as an array of attribute values and each value contains a pointer to an octree. Certain parameters are defined: the maximal depth $h = 5$ and the split threshold $\varphi = 80$. Leaf nodes that do not contain enough trajectories are merged as one node (still a leaf node). A 64-bit integer is used for the signature in each node. Each bit corresponds to a range of trajectory ids. Each octree leaf node is assigned a morton code and empty nodes (no trajectory intersects) are not materialized. Since each attribute value corresponds to an octree, we will have a set of octrees and combine the attribute value and the morton code as the key for each node. A relation stores

all leaf nodes and tuples are increasingly sorted on keys in order to maintain the locality of nodes in terms of the spatio-temporal proximity. A B-tree is built on the relation.

The main difference between trajectories with keywords and multi-attribute trajectories is that keywords are location-dependent texts, but attribute values are location-independent. A keyword is relevant to one or a few location points of the trajectory, while all location points of the trajectory have the same attribute values. This results in two major changes when maintaining the IOC-Tree and performing the query, in particular, inserting trajectory points into the index. A thorough analysis and comparison is provided in the following.

(i) In the context of keywords, location points will be distributed into octrees each of which corresponds to a keyword that the trajectory point contains. Each octree stores one or a few relevant location points of the trajectory. However, for multi-attribute trajectories each octree contains all location points of the trajectory because they all have the attribute value. Consider the following two trajectories.

- given a trajectory with keywords $o_1 = \langle (loc_1, t_1, coffee), (loc_2, t_2, pizza) \rangle$, we will store (loc_1, t_1) and (loc_2, t_2) in two octrees for coffee and pizza, respectively;
- given a multi-attribute trajectory $o_2 = (\langle (loc_1, t_1), (loc_2, t_2) \rangle$, (GRAY, BENZ)), we will store both (loc_1, t_1) and (loc_2, t_2) in two octrees for GRAY and BENZ, respectively.

The IOC-Tree is efficient for processing trajectories with keywords because only relevant trajectory points are indexed. However, attribute values are not related to locations but associated with the complete trajectory. That means, each attribute value is relevant to all points of the trajectory. Then, the number of trajectory points in each octree for multi-attribute trajectories is larger than that for trajectories with keywords, as demonstrated in Table 9.1. To gain trajectories with keywords, we randomly assign two attributes as keywords to each trajectory point using the dataset BTaxi in the experiment (Sect. 9.7). During the query procedure, the numbers of processed octree leaf nodes and trajectories increase, leading to more CPU and I/O costs. The values in Table 9.1 are calculated based on the condition that the number of trajectory points is the same in both cases. In fact, such a value for trajectories with keywords is much smaller than that for multi-attribute trajectories. We will explain this at point (ii) below.

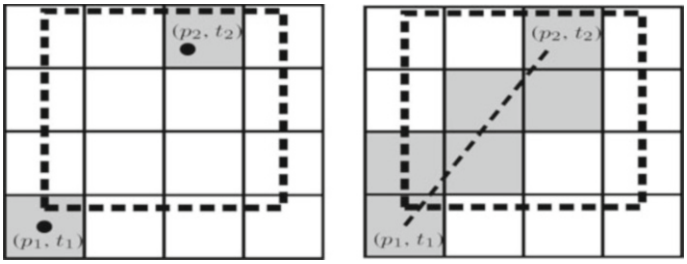
The variation in processed trajectory points also makes the signature in IOC-Tree less effective when we perform the intersection on trajectories containing different keywords. Each node in the octree maintains a signature represented by a bit vector to summarize the identifications of trajectories passing through the node. Table 9.2

Table 9.1 The average number of relevant trajectory points in an octree (BTaxi, d = 10, dom(Att) = [1, 151])

Multi-attribute trajectory	Trajectory with keywords
83,974	75,198

Table 9.2 The percentage of defined bits (64 in total) in the signature at each level in IOC-Tree (d = 10, [1, 151])

Att	Leaf nodes	$H = 4$	$H = 3$	$H = 2$	$H = 1$
1	45%	98%	100%	100%	100%
20	49%	96%	95%	100%	100%
50	47%	97%	100%	100%	100%
100	46%	98%	100%	100%	100%
Avg	46%	97%	99%	100%	100%



(a) (b)

Fig. 9.2 Cells intersecting the trajectory. (**a**) Trajectories with keywords. (**b**) Multi-attribute trajectories

reports the percentage of defined bits in the vector at each level of the IOC-Tree. We can see that almost all bits are defined for signatures in non-leaf nodes, weakening the pruning ability.

(ii) Trajectory points with keywords are sparsely defined because only a few locations of the trajectory may have semantics such as *coffee* and *mall*, but attributes are associated with all locations of the trajectory. This results in different numbers of octree leaf nodes intersecting the trajectory. Still using o_1 and o_2, we assume that the space is partitioned into 2×2 cells. Figure 9.2a and b show the cells intersecting o_1 and o_2, respectively. For trajectories with keywords, each point is assigned to the cell intersecting the trajectory. Locations between two sampled points will not be addressed because keywords are not defined. For multi-attribute trajectories, attribute values are associated with the overall movement and all cells intersecting the trajectory are included. Consequently, the number of maintained trajectory IDs in the IOC-Tree is much larger than that of trajectories with keywords. Given a query window, multi-attribute trajectories process more nodes and trajectories than trajectories with keywords, increasing the query cost.

Indexing spatial objects with keywords In the field of spatial keywords search, geo-textual indexes combine spatial and text aspects such that both types of information can be utilized to prune the search space. To answer Boolean kNN queries (Wu et al. 2012), the data is partitioned into multiple indexing groups each of which shares as few attributes as possible. A hierarchical aggregate grid

index called *HAGI* is developed to support heterogeneous kNN queries (Su et al. 2007). The method can be adapted to answer our queries, but it is limited as only one attribute is considered. A function is defined to combine the cost of distances and location-independent attributes, and the query returns objects having the k-th smallest function value. Each node in *HAGI* maintains *min* and *max* attribute values of all objects stored in the subtree. Although *min* and *max* values may work well for one attribute, they fail to guarantee good pruning ability for multiple attributes as *min* and *max* values are likely from different attributes. Also, the query evaluates objects based on a ranking function, whereas we require the exact match on attributes. Furthermore, the distance function is not time-dependent, whereas we deal with distances changing over time.

9.3 Problem Definition

9.3.1 Data Representation

A composite data model $\mathcal{O}(Trip; Att)$ is used to represent a multi-attribute trajectory database, in which *Trip* denotes standard trajectories and *Att* denotes multi-attributes. A standard trajectory is typically modeled by a function from time to 2D space. In a database system, a discrete model is implemented and the continuously changing locations are represented by linear functions of time, as illustrated in Fig. 9.3. That is, a trajectory is represented by a sequence of so-called *temporal units*, each of which records start and end locations during a time interval. Locations between start and end locations are estimated by interpolation. A data type called *mpoint* is defined (Forlizzi et al. 2000; Güting et al. 2000).

Definition 4 $D_{mpoint} = \{< u_1, \ldots, u_n > | n \geq 1$, and $u = (loc_1, loc_2, t_1, t_2)$ where $loc_1, loc_2 \in \mathcal{R}^2, t_1, t_2 \in T\}$

Let A be the set of multiple attributes. The ith attribute and its domain are denoted by $A[i]$ and $dom(A[i])$ $(i \in 1,\ldots, |A|)$, respectively. Assume that each $dom(A[i])$ is represented by a set of positive integers and a data type called D_{att} is defined for the set of attributes. For the sake of readability, the enum data type is used for attributes in the following.

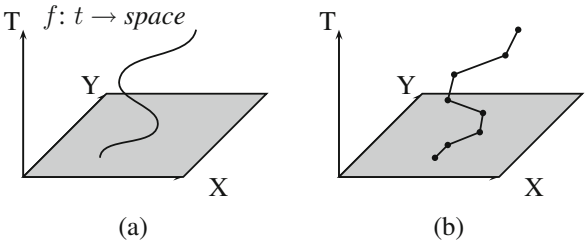


Fig. 9.3 Standard trajectory representation. (**a**) Abstract. (**b**) Discrete

Table 9.3 An integration of standard trajectories and attributes

Id: int	*Trip: mpoint*	*Att: att*
o_1	location + time	(RED, BENZ)
o_2	location + time	(GRAY, BENZ)
o_3	location + time	(SILVER, BMW)
o_4	location + time	(GRAY, BMW)
o_5	location + time	(SILVER, BMW)

Table 9.4 Summary of notations

Notation	Description
O	The set of multi-attribute trajectories
o	A multi-attribute trajectory
$\|A\|$	The number of attributes
$dom(A_i)$, $dom(A)$	The domain of A_i, the overall domain
Q_a	Query attribute expression
o_q, d	A query trajectory, the query distance
k	The number of nearest neighbors
t	A time point or interval
$T(o)$	The time period of a trajectory

Definition 5 (Multi-attribute representation) $D_{\underline{att}} = \{(a_1, \ldots, a_{|A|}) \mid a_i \in \text{dom}(A[i]), i \in \{1, \ldots, |A|\}\}$ such that

(i) $\forall i \in \{1,\ldots, |A|\}$: $dom(A[i]) \subset N^+$;
(ii) $\forall i, j \in \{1,\ldots, |A|\}$: $i \neq j \Rightarrow dom(A[i]) \cap dom(A[j]) = \varnothing$.

The data model is translated to a relation with the schema (*Id*: *int*, *Trip*: *mpoint*, *Att*: *att*) by embedding *mpoint* and *att* as relational attributes, as shown in Table 9.3. To be more specific, a relation is used to store multi-attribute trajectories by defining two attributes *Trip* and *Att*. The system manipulates multi-attribute trajectories via a relation. The advantage of using the relational interface is that (i) it allows combining heterogeneous data models, i.e., spatio-temporal and attribute; and (ii) existing operators on standard trajectories can be leveraged, benefiting the system development.

Table 9.4 summarizes notations frequently used in the chapter.

9.3.2 Queries

Attribute values are incorporated into the evaluation and the attribute expression is defined in the following.

Definition 6 (Attribute query expression) $Q_a = (a_1, \ldots, a_{|A|})$, $a_i \in dom(A_i)$ or $a_i = \perp$

Given a tuple of attribute values $Q_a = (a_1, \ldots, a_{|A|})$ and a multi-attribute trajectory $o \in \mathcal{O}$, an operator **contain**($o.Att$, Q_a) returns *True* if $\forall\ a \in Q_a$: $a \in o.Att$ or $a = \perp$.

Three types of queries are supported: (i) range queries, (ii) continuous range queries and (iii) continuous nearest neighbor queries. The range query is called RQMAT (Range Queries on Multi-attribute Trajectories) (Xu et al. 2018b). Formally,

Definition 7 (Range queries on multi-attribute trajectories (RQMAT)) Given a spatio-temporal window Q_{box} and attribute values Q_a, RQMAT returns a set of trajectories $\mathcal{O}' \subseteq \mathcal{O}$ such that $\forall\ o \in \mathcal{O}'$: (i) $o.Att$ **contains** Q_a; and (ii) $o.Trip$ **intersects** Q_{box}.

There is a variation of RQMAT that returns objects containing query attributes and keeping within a spatial range to a moving target at each query time point. The query is called CRQMAT, Continuous Range Queries on Multi-attribute Trajectories. Let $T(o)$ return the time period of an object. The function in Frentzos et al. (2007) is employed to return the time-dependent distance between two trajectories $o_1, o_2 \in \mathcal{O}$, denoted by $dist\ (o_1, o_2, T(o_1) \cap T(o_2))$.

Definition 8 (Continuous range queries on multi-attribute trajectories (CRQMAT)) Given a query trajectory o_q, a distance threshold d and an attribute predicate Q_a, CRQMAT aims to identify the result set $\mathcal{O}' \subseteq \mathcal{O}$ such that $\forall\ o' \in \mathcal{O}'$: (i) **contain**($o'.Att$, Q_a) ; (ii) $\forall\ t \in T(o_q) \cap T(o')$, $dist(\ o_q, o', t\ ) \leq d$.

Consider an example in Fig. 9.4. Assume that o_3 is a special object that carries VIP passengers or sensitive materials. For security reasons, one detects whether the special object is stalked. To this end, one makes use of multiple attributes to form a semantic-richer query, e.g., *Did any GRAY BENZ always keep 50 meters to* o_3. The returned objects must satisfy the criteria: (i) time-dependent distance constraint and (ii) attribute consistency. Although o_1 is within 50 meters to o_3, it is not a GRAY BENZ and should not be returned. Note that o_4 and o_2 fulfill the condition during $[t_1, t_2]$ and $[t_2, t_3]$, respectively, but they do not fulfill the condition during the overall query time. As a result, the query reports $\{([t_1, t_2], o_4), ([t_2, t_3], o_2)\}$.

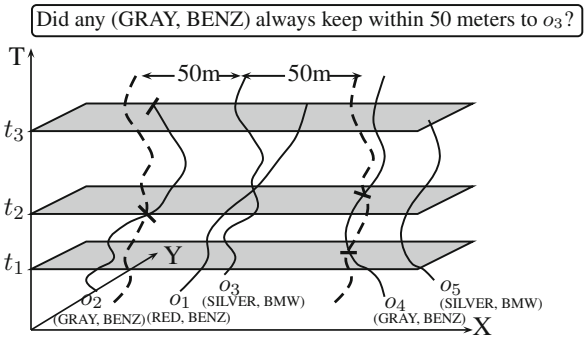


Fig. 9.4 Example of CRQMAT

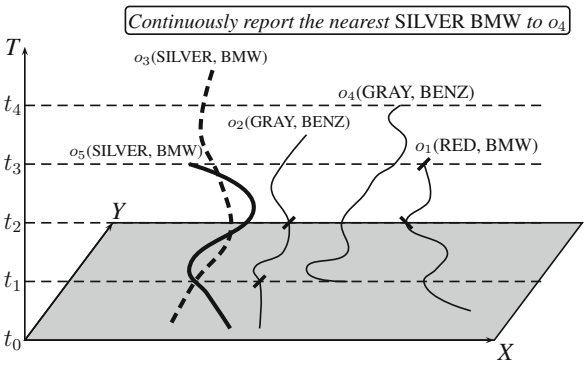


Fig. 9.5 Example of CkNN_MAT

The third type of queries is called CkNN_MAT (Continuous k Nearest Neighbor queries over Multi-attribute Trajectories) (Xu et al. 2018a). Such a query returns the objects fulfilling the condition: (i) *attribute consistency* and (ii) *time-dependent distance closeness*.

Definition 9 (Continuous k nearest neighbor queries over multi-attribute trajectories (CkNN_MAT)) Given a query standard trajectory o_q, an integer k and a set of query attributes Q_a, CkNN_MAT receives k trajectories denoted by $\mathcal{O}' \subseteq \mathcal{O}$ at each query time such that (i) $\forall\, o \in \mathcal{O}'$: **contain**($o.Att$, Q_a) returns *True*; (ii) $\nexists$ $o' \in \mathcal{O} \setminus \mathcal{O}'$: **contain**($o'.Att$, Q_a) $\wedge$ o' is closer than $\forall\, o \in O'$ to o_q.

An example is illustrated in Fig. 9.5. CkNN_MAT returns ($[t_1$, $t_2]$, o_3), ($[t_2$, $t_3]$, o_5), ($[t_3$, $t_4]$, o_3) indicating the key aspect that only objects fulfilling the attribute condition will be evaluated on the time-dependent closeness. Although o_1 and o_2 are closer than o_3 and o_5 to the query trajectory, they do not contain (SILVER BMW) and will not be included. Since distances between moving objects vary over time, results change at certain time points.

Generalizing query attribute expression Up to now, one assumes that the query defines a single value for each attribute. It is possible that multiple values are defined, e.g., *Continuously report the nearest* SILVER BMW or VW *to* o_4. The query expression is extended to support multiple values.

Definition 10 (An extension of query attributes) $Q_a = (X_1, \ldots, X_{|A|})$, $X_i \subseteq dom(A_i)$ or $X_i = \varnothing$

At the concept level, $Q_a = (X_1 , \ldots, X_{|A|})$ defines the component for each attribute over $\{A_1, \ldots, A_{|A|}\}$, in which X_i is a set of attribute values. The multi-value query SILVER BMW or VW is formed by $Q_a = (\{\text{SILVER}\}, \{\text{BMW, VW}\})$. At the implementation level, the query is defined by a relation in which a tuple supports multi-valued attributes. The operator **contain** is extended accordingly: **contain**($o.Att$, Q_a) returns *True* if $\forall\, X_i \in Q_a$: $o.Att[i] \in X_i$ or $X_i = \varnothing$.

Attribute queries with negative values are also supported, that is, $o.Att[j] \neq Q_a[j]$. Negative queries can be transformed into set queries by setting $Q_a[j]$ the values that are not equal to the query, e.g., $o.Att[j] \neq \text{RED} \Rightarrow o.Att[j] = \text{GRAY}$ or SILVER.

9.4 Indexing Multi-attribute Trajectories

9.4.1 An Overview of the Structure

To efficiently answer queries, the index should manage both spatio-temporal trajectories and attributes in order to prune the search space on both predicates. A hybrid structure is developed that consists of a 3-D R-tree and a composite structure named BAR, as shown in Fig. 9.6. The 3-D R-tree that serves as indexing standard trajectories is a height balanced tree. Each node contains an array of entries, each of which couples (i) a pointer to a subtree or an object with (ii) a rectangle that bounds data objects in the subtree. BAR is a composite structure that includes a $\underline{B}$-tree, a relation $\underline{A}$tt_Rel and a record file $\underline{R}$F.

The system builds BAR on top of the 3-D R-tree by extracting attribute values from multi-attribute trajectories. The structure builds the connection between attribute values and R-tree nodes and enables us to know attribute values in a sub-tree. For a leaf node, each entry stores a pointer to a tuple in the trajectory relation and the tuple is accessed to obtain the attribute value. For a non-leaf node, attribute values are collected by performing the union on values from child nodes. BAR maintains attribute values in an efficient way such that one is able to fast settle the R-tree nodes that (i) contain query attributes and (ii) fall into the range of the query time. Before elaborating the index structure, we first introduce pre-processing trajectories in order to have a compact dataset for building a good shape of the 3-D R-tree (nodes have similar sizes in spatio-temporal dimensions). Sections 9.4.2 and 9.4.3 present grouping small units and partitioning trips according to spatio-temporal distributions, respectively.

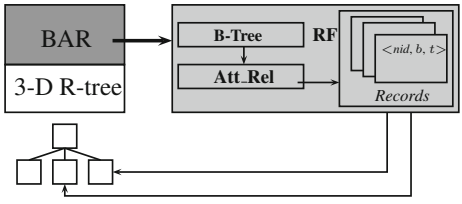


Fig. 9.6 Index architecture

9.4.2 Packing Trajectories

The R-tree is supposed to be built on sorted minimum bounding rectangles (MBRs) that approximate trajectories. By observation, raw trajectories from GPS records contain a large number of small units due to short time intervals or slow movement. In order to reduce the size of the dataset, small pieces of movements are packed to have fewer but larger units. Let u_i denote the trajectory extent in the ith dimension. The average extent over all units in the ith dimension is denoted by $\triangle_i$. Then, the deviation of a unit is given as:

$$f(u) = \sum \frac{u_i}{\Delta_i}, i \in \{d_x, d_y, d_t\} \tag{9.1}$$

A threshold *Bound* is defined to select small units. Duplicate values are removed to overcome the impact of the number of small units. The lower bound is analytically estimated.

$$Bound = \text{Avg}(\text{Unique}(\lfloor f(u) \rfloor)) \geq \text{Avg}(\lfloor f(u) \rfloor) \approx 3 \tag{9.2}$$

Let U be the set of all temporal units and the unit with the maximum deviation is

$$u^* = \arg\max_{u \in U} Unique(\lfloor f(u) \rfloor) \tag{9.3}$$

Not all values in $\{0, 1, \ldots, \lfloor f(u^*) \rfloor\}$ may be defined and thus the upper bound is

$$Bound \approx \text{Avg}(0 + \ldots + f(u^*)) \leq \frac{f(u^*)}{2} \tag{9.4}$$

The packing can be treated as building the R-tree in a different way, as demonstrated in Fig. 9.7. Small pieces of trajectories are packed to obtain large units which are taken as the input for a leaf node. The index is built by bulk load (Bercken and Seeger 2001; Bercken et al. 1997) which uses the same threshold as the standard value to group units into one leaf node, guaranteeing the spatio-temporal locality. The *Bound* is the average value over Unique($f(u)$) and thus will not result in grouping units into a large extent. During the packing procedure, neither raw units are modified/simplified nor data is lost. One does not need extra storage space and the same number of original units is maintained.

Demonstrate packing trajectories Using 500 GPS records of taxis, we calculate the unit deviation and report their values as well as *Bound* in Fig. 9.8. One can see that the majority of units have the derivation smaller than *Bound*. We pack successive small units of the trajectory as one unit such that the deviation of the unit is larger than *Bound*. The overall number of trajectory approximations (MBRs) is greatly reduced, leading to a compact dataset to build the index.

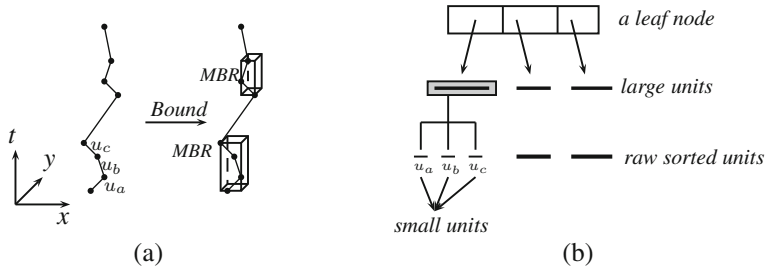
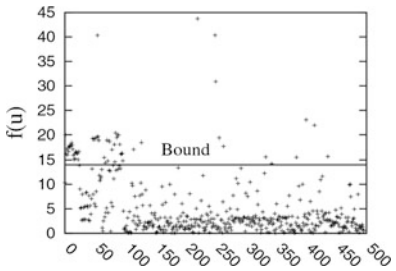


Fig. 9.7 The packing procedure. (**a**) Pack small units. (**b**) Build the index

Fig. 9.8 Effect of packing trajectories



9.4.3 Partitioning Trajectories

Trajectories have different distributions over time and space. We would like to decompose them into pieces which have similar sizes in terms of spatial and temporal dimensions. This will benefit the index structure because spatio-temporal extents of nodes are similar, derivations among nodes are small and the area of *inactive space*[2] is reduced. The time dimension is partitioned into a set of equal-sized intervals $\{T_1,\ldots,T_K\}$ $(K > 1)$ and the 2-D space is partitioned into a set of equal-sized cells. Given a multi-attribute trajectory, its spatio-temporal trajectory is split into a set of so-called *cell trajectories*, each of which represents the movement within a cell during an interval $T_k \in \{T_1 ,\ldots,T_K\}$.

Definition 11 (Cell trajectory) Let $Cell(o, t)$ return the cell where o is located at a time point $t \in T(o)$. A cell trajectory $o[i]$ is a subset of $o.Trip$ such that (i) $\forall\, t_1, t_2 \in T(o[i]) : Cell(\, o[i], t_1\,) = Cell(\, o[i], t_2\,)$; (ii) $\exists\, T_k \in \{T_1,\ldots, T_K\}$: $T(o[i]) \subseteq T_k$.

We partition each $o \in \mathcal{O}$ into a set of cell trajectories in three steps: (1) *o.Trip* is decomposed into a sequence of sub trajectories such that the time of each sub trajectory is contained by T_k; (2) For each sub trajectory, a set of cells intersecting the 2-D bounding box of the trajectory is identified, which is efficiently determined

[2]The space is contained by the node but there are few or no data objects. One can also call this *dead space* (Tao and Papadias 2001), meaning that the area will be evaluated but few objects are there or even no object exists.

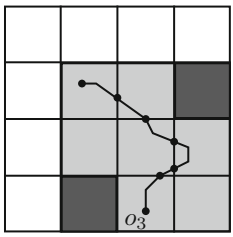


Fig. 9.9 Partitioning o_3 into cell trajectories

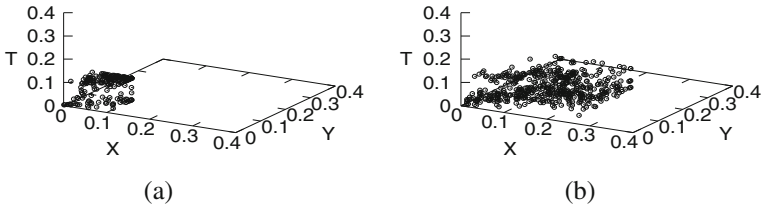


Fig. 9.10 Leaf node extents affected by partition. (**a**) Partition. (**b**) Without the partition

by finding the left-bottom and right-top cells; (3) Each sub trajectory is split into a set of cell trajectories. We may encounter the case that the object enters the cell more than once. As a consequence, there are several cell trajectories from one object located in the same cell. Assume that the 2-D space is partitioned into 4×4 cells and o_3 is contained by a time interval. The cells intersecting o_3 and o_3's cell trajectories are reported in Fig. 9.9. The index is built on cell trajectories sorted by time, cell id and 3-D bounding box following a bulk loading approach (Bercken et al. 1997).

Demonstrate partitioning the trajectories We use part of real trajectories in the experimental evaluation (66,000 taxi trajectories in Beijing) and build two indexes on trajectories with and without performing the partition, respectively. The extents on all dimensions are reported by randomly selecting 500 leaf nodes, as illustrated in Fig. 9.10. Clearly, the deviation among different dimensions after the partition is much smaller than that without the partition. This contributes to create a good R-tree.

Grid granularity Grid granularity plays a pivotal role in the index design as an arbitrary value cannot guarantee an optimal query performance. Assume the 2-D space is partitioned into $\delta \times \delta$ equal-sized cells. If we set a coarse granularity, e.g., $\delta = 1$, all trajectories are located in one cell. The index does not exhibit the spatio-temporal proximity, increasing false positives in query processing. At the opposite end, a fine granularity leads to small cells and each cell contains fewer trajectories having small extent in x and y dimensions. This is good for preserving locality. However, the finer the granularity is, more nodes are maintained. This is because the number of cell trajectories grows proportionally as a spatio-temporal trajectory is partitioned into all intersecting cells. As shown in Fig. 9.11, we will visit all cells under the setting $\delta = 2$ because they are within the d-distance to o_3. However, in

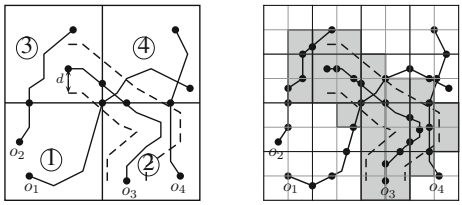


Fig. 9.11 Coarse and fine grid granularities. (**a**) $\delta = 2$. (**b**) $\delta = 8$

cells ③ and ④, cell trajectories of o_2 and o_4 do not fulfill the distance condition. Considering $\delta = 8$, although we can greatly reduce the search space (gray area), more cells are accessed and some of them do not contain trajectories.

9.4.4 BAR

The relation Att_Rel The key component in BAR is the relation Att_Rel that builds the connection between attribute values and R-tree nodes. The relation schema is defined as

Att_Rel: (A_VAL:*int*, H: *int*, RecId: *int*).

For each attribute value, the system maintains a tuple for all nodes containing the value at the same height. The nodes are stored in a record. A tuple stores the attribute value, the height, and the record identifier. The relation is created as follows. Step 1, for each $a \in dom$(A) the approach traverses the R-tree in depth-first search to collect all nodes containing a and creates an intermediate tuple for each node. One sets a as the key and records the node height. Step 2, the intermediate tuples are grouped according to the height and a record stores all nodes containing a at each height. A tuple is created to store the record id. Steps 1–2 are repeated for all $a \in dom$(A). One creates a B-tree on Att_Rel by making a key combining A_VAL and H.

A unique key is required for each attribute value. The ideal case is that attribute domains do not overlap. In practice, it may be not possible to have non-intersecting domains, but this problem can be solved. One can use a composite number to represent the attribute value. This is achieved by combining the attribute id and the value. In turn, a two-dimensional point (i, a) $(i \in [1, |A|], a \in dom(A_i))$ is formed. Then, a space-filling curve Z-order is used to map points of a two-dimensional space to one-dimensional values. This is done by interleaving the binary coordinate values, which guarantees that attribute domains do not overlap.

Record Storage The system maintains a list of items in each record. Each item is represented by a three-tuple: (nid, b, t), in which nid is the node id, b is a bitmap and t is a time interval. The bitmap represents the entries containing the attribute value in a node and t is the overall time of entries. The design is made based on the observation that the number of entries containing an attribute value cannot be

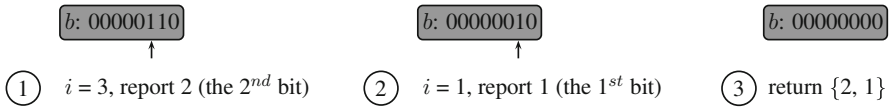


Fig. 9.12 Report defined bits

larger than the total number of entries, usually much smaller. Also, the number of entries containing an attribute value increases from leaf level to root level. This is because if a node contains the value, all its ancestor nodes will contain the value. To efficiently settle the entries fulfilling the attribute condition, the bitmap is accessed at first instead of performing a linear scan over all entries.

Let m denote the length of a collection of bit-vectors and E be the entry count of a node. A mapping between m and E is performed. There are two cases. Case (i): $m \geq E$, each bit maps to an entry. If the ith ($i \in [0, m)$) entry contains the attribute, one has $b[i] = 1$. Otherwise, $b[i] = 0$. Case (ii): $m < E$, each bit maps to a sequence of entries. The corresponding entries for the ith bit are calculated by $[i \cdot \lceil \frac{E}{m} \rceil, (i + 1) \cdot \lceil \frac{E}{m} \rceil)$. The system defines $b[i] = 1$ if one of the entries contains the attribute. The bitmap index incurs little storage overhead and is efficient for processing data in small quantity due to the speed of bit-wise operations. The length m depends on the implementation, e.g., a 32-bit integer. The bitmap fast determines qualified entries for the intersection condition of several attributes. A data type is embedded into an relation to represent the records.

Querying the bitmap In order to know the entries containing the query attribute, the method accesses the bitmap to report defined bits. Let $\mathcal{B} = < 2^0, 2^1, \ldots, 2^{m-1} >$ be a sequence of integers. Given a bitmap b, its defined bits are reported as follows: Step 1, by performing a binary search one finds the smallest $2^i \in B$ such that $2^i \geq b$. Step 2, if $2^i = b$, i is reported and the searching is terminated because the bit is already found. If $2^i > b$, the procedure updates i - 1 and $b = b - 2^{i-1}$. Then, steps 1–2 are repeated until b is equal to 0, during which bits are progressively reported from high to low positions. Figure 9.12 depicts the procedure of reporting $b = 00000110$ in Record 3.

Let P denote the set of defined bit positions, initially empty. Two indexes s and e are used to define the sth and eth integers in $\mathcal{B}$. To find the smallest i such that $2^i \geq b$, the procedure performs a binary search and terminates when either b is equal to an integer in $\mathcal{B}$ or $e = s + 1$. In the former case, all bits are found already. In the latter case, the position of the highest bit is found and put into P. To continue searching the bits, the approach updates b as well as s and e by setting $e \leftarrow s$ and $s \leftarrow 0$.

Time complexity One needs $O(\log m)$ to report the highest bit and the position is $p \in [0, m)$. To find the second highest bit, a binary search is performed, leading to $O(\log p)$. The iteration time depends on p. The smaller p is, the fewer iterations are needed. If $p \in [m/2, m)$, $\log p = \log m$ iterations are required. If $p \in [0, m/2)$,

$\log m - 1$ iterations are achieved. To report the ith highest bit, the iteration time is between $[\log m - (i-1), \log m]$, depending on where the bit is located. To sum up,

Theorem 1 (Upper Bound)

$$O(|b| \log m) \tag{9.5}$$

Theorem 2 (Lower Bound)

$$O\left(\sum_{i=1}^{|b|}(\log m - (i-1))\right) = O(|b| \log m - (|b|-1)|b|/2) \tag{9.6}$$

Proof One performs a binary search to look for $|b|$ defined bits in $O(m)$. In the worst case, they are the $|b|$ highest bits and each iteration needs the time $O(\log m)$, leading to $O(|b| \log m)$. An optimal case is that one needs $O(\log m - 1)$ for the second iteration when the position of the second highest bit is smaller than $\frac{m}{2}$. If the position of the second highest bit is $\leq \frac{m}{4}$, one needs $O(\log m - 2)$ for the third iteration and so on. □

9.4.5 Updating the Index

The database needs to keep track of the incoming data and allow querying both the historical and new data. An important task is to synchronize index structures in order to be consistent with the underlying data space. Given a set of incoming multi-attribute trajectories, inserting them into the index incurs updating two structures: (i) 3-D R-tree and (ii) BAR. In general, a new R-tree named $\mathcal{R}_u$ is created on new trajectories and BAR is built on $\mathcal{R}_u$. To distinguish between historical and new structures, the new structure is termed BAR_u. New created structures $\mathcal{R}_u$ and BAR_u are inserted into historical structures $\mathcal{R}$ and BAR, as illustrated in Fig. 9.13.

Updating 3-D R-tree The incoming trajectories are packed and a new R-tree is created by bulk load. The new R-tree is maintained by the same storage file as

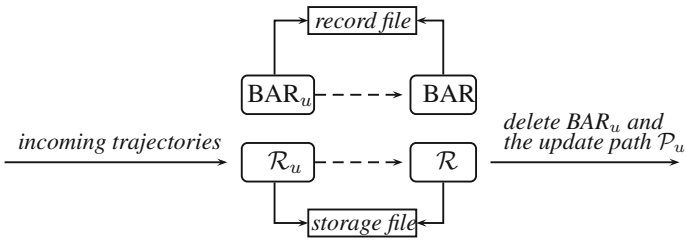


Fig. 9.13 An outline for updating

the historical R-tree in order to simplify the procedure of accessing the structure. Otherwise, one has to detect whether the accessed node belongs to the new R-tree or the historical R-tree and select the corresponding file. It is a rather complex task to maintain many storage files for frequent updates. $\mathcal{R}_u$ is inserted into R as follows. Let H_u and H denote the heights of the two R-trees, respectively. Assume that $H_u \leq H$. This is because the number of incoming trajectories for one update is usually much smaller than that of the historical data. If $H_u = H$, a new root node is created to hold root nodes of R and R_u as two entries. If $H_u < H$, the root node of R_u is inserted as an entry to an appropriate node in the target tree R whose height is equal to H_u. This is achieved by performing a top-down traversal in the target tree until a node whose height is equivalent to the new R-tree. During the traversal, the last entry of each accessed node is always chosen as the node to be processed at the next level. This is because entries are increasingly sorted by time and incoming trajectories are certainly located after existing trajectories. If the node is not full, the root of the new R-tree is inserted as an entry into the node. Otherwise, a new node is created for the R-tree.

Updating BAR We insert BAR_u into BAR: step 1, for each tuple in BAR_u.Att_Rel, the procedure searches for the matching tuple in BAR.Att Rel and appends record items for the nodes in $\mathcal{R}_u$; step 2, record items are updated for each node appearing in $\mathcal{P}_u$.

$\mathcal{R}_u$ is inserted into $\mathcal{R}$ as a sub-tree and the nodes in $\mathcal{P}_u$ are updated in terms of (i) spatio-temporal boxes; and (ii) attribute values. For each attribute value in new trajectories, the method looks for tuples in BAR.Att_Rel having the value and the appropriate height according to $\mathcal{P}_u$. Note that the height is increasingly numbered from leaf to root level, guaranteeing that the heights of $\mathcal{R}$ and $\mathcal{R}_u$ are consistent. If the tuple is found, the record is accessed to update the item for the node. Precisely, the bitmap and the time box are updated. Later, the record is refreshed to synchronize the data.

New arrival trajectories incur an ongoing expansion of the time. The time range of the nodes in $\mathcal{P}_u$ overlaps with that of new trajectories. To enhance the update performance, record items are increasingly sorted on time and updated from the end of the list.

Definition 12 (Sorted records) $D_{\underline{rec}} = \{<\text{nid}_1, b_1, t_1 >,\ldots,(\text{nid}_n, b_n, t_n > \mid t_1 < \ldots < t_n, \text{nid}_i \in \underline{int}, b_i \in \underline{int}, t_i \in \underline{interval}\}$

If a new root node is created, there is no matching tuple in BAR.Att_Rel. Therefore, the tuple as well as the record are created and inserted into BAR. Afterwards, BAR_u and $\mathcal{P}_u$ are dropped. Let $\mathcal{O}_u$ be the set of new arrival trajectories. In order to achieve good performance for updating, a light-weight BAR named lw-BAR is proposed to reduce the I/O cost. The idea is to buffer record items in lw-BAR rather than updating BAR for each new trajectory. A relation and a B-tree make up lw-BAR and there is no record file. The relation schema is of the form

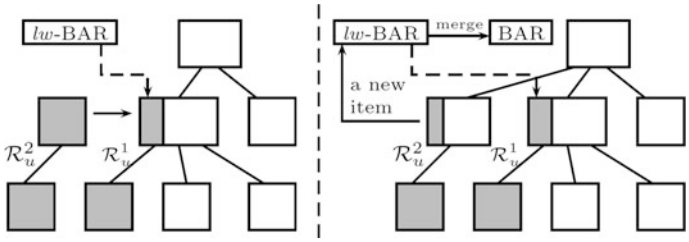


Fig. 9.14 Overflow and Update lw-BAR, BAR

lw-Att Rel:(*A_VAL*: *int*, *H*: *int*, *RecItem*: *rec*).

For each attribute value appearing in BAR_u, a tuple in lw-Att_Rel is maintained for the node in $\mathcal{P}_u$. In the update path, there is only one node at each height from H_u to H, and therefore the number of updated items in a record is one. The lw-BAR only stores one item in a record. To have a compact structure, the record component (managed in a record file in BAR) is merged into the relation by replacing the record id by the record item. Employing the lw-BAR, the number of I/O accesses for updating will be considerably reduced because only one tuple is processed.

To accommodate frequent updates, the record items for $\mathcal{P}_u$ have to be updated whenever $\mathcal{R}_u$ is inserted into $\mathcal{R}$. If entries in the inserted node do not overflow, only record items for historical nodes are updated. However, under a continuous updating load, frequent insertions will cause the node overflow and lead to new nodes, as illustrated in Fig. 9.14. In this scenario, the record item in lw-BAR is merged into the one in BAR and another record item in lw-BAR is created for the new node.

9.4.6 The Generality

The proposed index structure is general from three aspects: (i) packing standard trajectories, (ii) managing attribute values, and (iii) supporting a range of queries on multi-attribute trajectories and also queries on standard trajectories.

Packing. The established method produces a compact data set by reducing the number of approximations. There is no information loss and no extra storage cost. The procedure can be applied for other trajectory queries to enhance the performance.

BAR. The system is able to flexibly build the traditional trajectory index or the hybrid index, depending on whether standard trajectories or multi-attribute trajectories are processed. BAR is not tightly integrated into the spatio-temporal index and therefore can be combined with other traditional trajectory indexes, categorized into (i) R-tree based indexes, e.g., TB-tree (Pfoser and Jensen 2000), MV3R-Tree (Tao and Papadias 2001), and (ii) grid based indexes, e.g., SETI (Chakka et al. 2003). The well-established structures do not have to be modified,

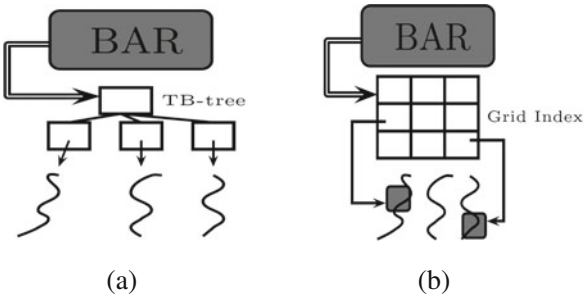


Fig. 9.15 Popularizing BAR. (**a**) TB-tree. (**b**) Grid index

benefiting the system development. Figure 9.15 reports BAR built on top of TB-tree and Grid index. One instantiates into the 3-D R-tree due to the advantage of preserving the spatio-temporal proximity and the efficiency of answering nearest neighbor queries (Güting et al. 2010b). The comparison with other trajectory indexes is as follows.

- TB-tree. The structure has the trajectory preservation property that only stores units of the same trajectory within a leaf node, resulting in a large spatial extent of leaf nodes. The spatial proximity is not preserved because segments of different trajectories that lie spatially close will be stored in different nodes. One cannot effectively prune the search space by *min* and *max* distances, resulting in poor performance for nearest neighbor queries. The STR-tree (Pfoser and Jensen 2000) introduces a parameter to balance between spatial properties and trajectory preservation, but the main concern is to handle the spatial domain and treating the temporal as a secondary issue.
- SETI. The space is divided into disjoint cells, each of which contains trajectory segments that are completely within the cell and has a temporal index (an R-tree) for objects' time intervals. The number of spatial partitions plays a crucial role in index design, but setting an optimal value is not trivial. Trajectories can be uniformly and uniformly distributed, making the performance unstable. The method focuses on the spatial proximity and has the limitation that the boundaries of the spatial dimension remain constant. The SEB-tree (Song and Roussopoulos 2003) is similar to SETI where the space is partitioned into zones, but the difference is that only the zone information is stored in the database without knowing the exact location.
- MV3R-tree. The index combines a multi-version R-tree (MVR-tree) and a small auxiliary 3-D R-tree built on the leaf nodes of the MVR-tree. The former is to process time-stamp queries and the latter is to process long interval queries. Short interval queries are processed by selecting the appropriate tree. Multi-attribute trajectories deal with interval queries and thus the structure is essentially a 3-D R-tree.

Queries. RQMAT, CRQMAT and C*k*NN_MAT can all be answered by employing the proposed index structure. One accesses BAR to find the subtrees in the

spatio-temporal index fulfilling the attribute condition and then explores the spatio-temporal index. If attributes are not considered, the algorithm directly searches the spatio-temporal index without accessing BAR.

9.5 Query Algorithms

9.5.1 An Outline

The query procedure follows the *filter-and-refine* strategy. In general, one performs a traversal on the index during which objects are pruned on both spatio-temporal and attribute conditions. When the leaf level is reached, we open the node to retrieve objects from the relation. The filter step returns a set of candidate trajectories, each of which is likely to be in the result and will be iteratively evaluated, called *refinement*.

9.5.2 Processing RQMAT

The query processing runs in two steps. Step 1 accesses BAR to determine the R-tree nodes that contain query attribute values and overlap the query time. Step 2 takes the nodes returned from Step 1 as well as the spatio-temporal box to perform a breadth-first search on the R-tree, as illustrated in Fig. 9.16.

To determine the objects containing query values, the procedure accesses BAR to look for the nodes fulfilling the attribute condition. That is, the method searches for the nodes in which there are trajectories containing Q_a. For each attribute value, BAR is accessed to find the tuple and get the record. Each item in the record stores a node id and a bitmap marking the entries containing the value. The item is obtained for each attribute value and the intersection operation is performed on bitmaps to find the entries containing the query. The time dimension of the node is checked to determine whether the item (a candidate node) exists in the returned node set, denoted by N_a. If not, an item (*nid*, *b*, *t*) is inserted by adding a counter, initialized by 1. If yes, the counter is increased and the bitmap is updated by performing the operation AND. In the end, items in N_a that cannot contribute to the result are removed.

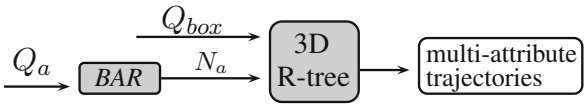


Fig. 9.16 Procedure of RQMAT

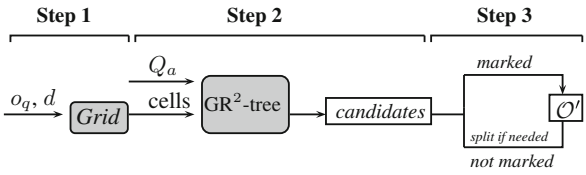


Fig. 9.17 Procedure of CRQMAT

9.5.3 Processing CRQMAT

The query CRQMAT is answered in three steps, as illustrated in Fig. 9.17. The index structure *GR*^{2}*-tree* includes a Grid R-tree and an attribute relation.

In Step 1, the spatio-temporal area restricted by o_q and d is established. Based on the grid partition one quickly determines the cells within the d-distance to the query. This is achieved by computing the distance between the 2-D bounding box of the query trajectory and the cell. The nodes that do not intersect the cells can be safely pruned. Usually, a cell is not always within the d-distance to the query as the location of the trajectory changes over time. Time-dependent cells are reported and maintained by a composite structure including three components: *cell tree*, *cell set* and *cell list*. The cell tree is a binary tree that records a time interval and the cells intersecting the query trajectory. The structure reports all cells within the d-distance to the query during a time interval. A cell may be valid at different time intervals. The method maintains a cell set by removing duplicate results. The cell list determines whether all trajectories in a leaf node are within the d-distance to the query. If yes, the exact distance computation can be avoided as a leaf node stores trajectories whose movements are restricted in a cell.

An example By referring to Fig. 9.18a, we enlarge the bounding box of o_3 in both x and y dimensions to find all cells within the d-distance to the query (depicted in gray). Two dashed lines are depicted to help figure out the cells. The time interval $T(o_3)$ intersects $\{T_1, T_2, T_3\}$. The cells $\{c_{5,1}, c_{5,2}, c_{6,1}, c_{6,2}, c_{7,2}\}$ are within the d-distance to the query at T_1, but they should not be considered at T_3. There are three *marked* cells $\{c_{5,5}, c_{6,3}, c_{7,4}\}$ at $T_2 \cup T_3$. Thus, the cell trajectory of o_1 in $c_{5,5}$ and the cell trajectory of o_4 in $c_{7,4}$ can be directly returned without performing the accurate distance computation (the attribute condition is not considered here). The structure of the time-dependent cells is reported in Fig. 9.18b. The cell set consists of three parts C_1, C_2 and C_3, partitioned by time intervals. The cell tree is built on cells with corresponding time intervals. Since cells $\{c_{5,5}, c_{6,3}, c_{7,4}\}$ are *marked cells*, they are put into the cell list with time intervals.

In Step 2, the procedure traverses the R-tree to return a set of candidates, each of which contains Q_a and has the distance to o_q less than d. Given an R-tree node, the algorithm determines the cells intersecting the node. The cells reported in step 1 are used to prune the node if there is no overlap between the cells intersecting the node and the cells within the d-distance to the query. When a leaf node is accessed, the

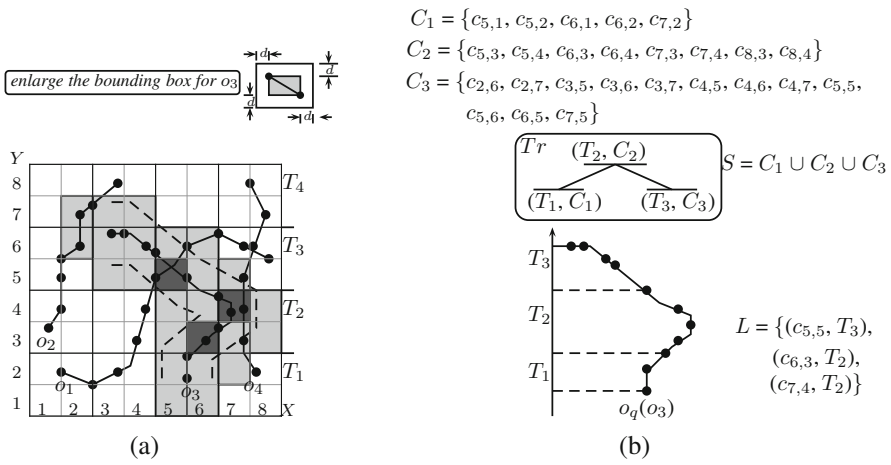
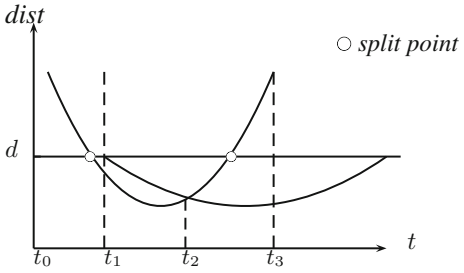


Fig. 9.18 An example of establishing qualified cells. (**a**) Cells within d to o_q. (**b**) Cell tree, cell set and cell list

Fig. 9.19 Distance curve



algorithm iteratively computes the exact distance between each trajectory and the query. The distance is an approximate value calculated by using minimum bounding boxes of trajectories. A candidate is marked if its maximum distance to o_q is less than d.

Step 3 iteratively checks the accurate distance. If the candidate is marked, it will be directly put into the result set. Otherwise, the actual distance is computed. A trajectory may be split because only the piece of movements fulfilling the distance condition is reported. Two trajectories are mapped into pieces with the same time. The task is to compute the intersections among a set of distance curves to determine the curves whose values to o_q are smaller than d, and return the parts corresponding to these pieces. The time-dependent distance is represented by a square root of a quadratic polynomial, as demonstrated in Fig. 9.19.

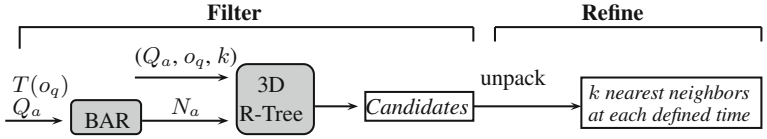


Fig. 9.20 Procedure of C*k*NN_MAT

9.5.4 Processing CkNN_MAT

In the filter step, the structure BAR is accessed to determine the R-tree nodes including query attribute values and intersecting the query time, denoted by N_a. Next, the procedure takes N_a coupled with o_q and k as input to traverse the R-Tree in breadth-first order, during which the search space is pruned by taking into account spatial and temporal parameters as well as attributes. The filter returns a set of candidates, each of which fulfills the attribute condition and approximately belongs to k nearest neighbors. In the refinement step, each candidate trajectory is unpacked to get the original temporal units and perform the exact distance computation to return k nearest neighbors at each query time. The query procedure is shown in Fig. 9.20.

Collecting R-tree nodes The R-tree nodes containing Q_a are collected level by level. For each $a \in Q_a$, the procedure starts from $h = 1$ and accesses BAR to find the records. For each item (*nid*, b, t) in the record, one checks whether the item identified by *nid* is already in N_a. If not, the method inserts the item into N_a by attaching a counter, initialized by 1. Such a value represents the number of query attribute values contained in the node. The extended record item is denoted by λ. If the item already exists in N_a, the counter is increased and the bitmap is updated by performing the bitwise AND. This is because a node fulfilling the condition must contain all values in Q_a.

Lemma 1 *Given an item $\lambda \in N_a$, the item is pruned if $\lambda.counter \neq |Q_a|$ or $\lambda.b = 0$.*

Proof (i) $\lambda.counter \neq |Q_a|$: Obviously, it is impossible that $\lambda.counter > |Q_a|$ as distinct values are counted. If $\lambda.counter < |Q_a|$, this means that the number of attributes contained by λ is less than $|Q_a|$ and therefore λ can be safely pruned. (ii) $\lambda.b = 0$: There is no entry in the node containing all $a \in Q_a$ and λ can be safely pruned. □

An extension: multiple values An extension is made to allow a query attribute with multiple values. A node is satisfied if it contains one of the values. The aforementioned $\lambda \in N_a$ is extended to (*nid*, b, t, *aid*) by adding the attribute id. The bitwise OR is performed on bitmaps for values from the same attribute.

Lemma 2 *Let AttCount(Q_a) ($\leq |Q_a|$) return the number of query attributes. An item $\lambda \in N_a$ is pruned if $\lambda.counter \neq AttCount(Q_a)$ or $\lambda.b = 0$.*

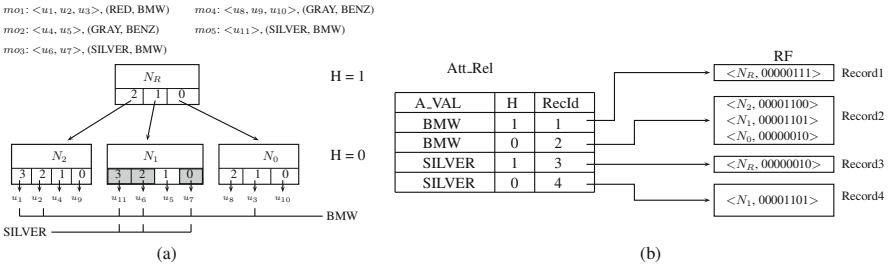


Fig. 9.21 3D R-Tree

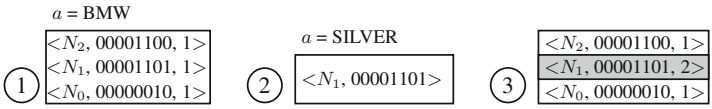


Fig. 9.22 Reporting nodes at $H = 0$

An example Using trajectories in Fig. 9.5, we report the 3D R-Tree and the structure BAR for attributes SILVER and BMW in Fig. 9.21. Consider Q_a = (SILVER, BMW). At $H = 1$, we access records 1 and 3, and have $b = 00000111$ for BMW and $b = 0000010$ for SILVER. By performing the bitwise "AND" operation, the 0th and 2nd entries are not defined and therefore N_0 and N_2 are pruned. Figure 9.22 depicts the procedure of determining nodes for Q_a = <SILVER, BMW> at $H = 0$ (t is omitted).

Reporting candidates The approach traverses the R-tree from root to leaf level and uses a list to maintain accessed nodes. For each visited node, the algorithm determines whether (i) the node fulfills the attribute condition; and (ii) objects in the subtree will contribute to the result. The query needs only k neighbors. If there are k candidates at each defined time, objects that are further than current candidates to o_q can be safely pruned. To determine whether there are enough candidates, a segment tree is maintained by storing the time interval, the distance and the number of trajectories in a node. One can do the pre-computation for each attribute value and use it for $|Q_a| = 1$. However, such a value has to be calculated on-the-fly for $|Q_a| > 1$, which is a costly procedure. The sub-tree will be traversed to count the number of trajectories containing query attributes as one needs to determine the intersection set of different query attributes.

The refinement This step includes two phases: (i) unpack each candidate to obtain temporal units; (ii) apply the slightly modified plane-sweep algorithm (Bentley and Ottmann 1979) to determine the k lowest time-dependent distance curves to report the result. Phase (ii) takes in a sequence of candidate trajectories ordered on time. The time-dependent distances to the query trajectory are computed to find k nearest

objects at each time point. To achieve this, one determines pieces of movements with overlapping time and applies the distance function by employing the linear interpolation on each piece (Forlizzi et al. 2000; Frentzos et al. 2005). The method manipulates temporal units to calculate the distance. Split points between curves are found to determine the k lowest pieces of curves.

9.6 The System Development

9.6.1 The Architecture

A prototype database system is developed to efficiently manage multi-attribute trajectories including data representation, index structures, query algorithms and optimizations (Wang and Xu 2017; Xu and Güting 2017; Wei and Xu 2018). Since standard trajectories have been supported in a database system SECONDO (Güting et al. 2010a), the task is to develop modules for multi-attribute trajectories and seamlessly integrate them into the system. Key system components are shown in Fig. 9.23.

The query interface animates standard trajectories and displays multiple attributes. Not only objects whose locations changing over time are visualized, but also their time-dependent attribute values are displayed. Queries on multi-attribute trajectories include several predicates, leading to different query plans. The optimization selects the best plan according to the analytical model. Then, the corresponding algorithm is executed and the index structure is accessed. The index component is in principle made up of a 3-D R-tree and a composite structure BAR. The 3-D R-tree preserves the spatio-temporal proximity, and BAR manages attribute values. The data storage component includes several modules such as spatial and temporal data, standard trajectories, relational tables, and attributes.

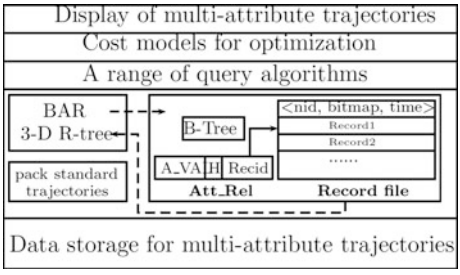


Fig. 9.23 The system architecture

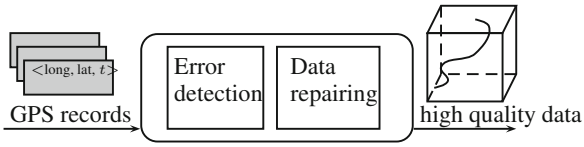


Fig. 9.24 GPS-clean workflow

9.6.2 A Tool for GPS Data Clean

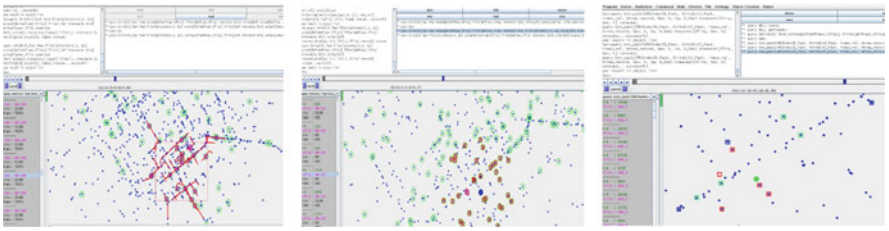
Real-life data is far from being reliable enough for applications predominantly due to a large number of errors such as inaccurate measurement, noisy, distortion and outliers, mainly caused by the limitations of devices or signal loss. A data cleaning tool will serve as the pre-processing module to bring data from a messy to a neat state. The primary task is the minimization or the total removal, if possible, of GPS errors, and the repairing of trajectories after removing some sample points. The tool can be treated as an objective function *Clean*: *RawData* → *HighQualityData* and must be an integral part of a moving objects database.

The procedure consists of two steps: error detection and data repairing, as illustrated in Fig. 9.24. *Error detection* identifies incorrect data values, which can be classified into two categories: *point error* and *trajectory error*. The first is identified by checking the data item in each individual record such as time-stamp and long/lat, and the second is established by evaluating a sequence of records of the same object such as *distortion* and long time still. *Data repairing* involves updating the available data to remove any detected errors, and derives and fills in missing data from the existing data.

Refining a good clean function requires a rich set of detection rules, filtering operations, statistical analysis and missing value imputation methods. Prediction models can be built by learning features from historical data with different characters. To evaluate the data quality, we define data metrics to measure the quality and employ machine learning techniques to classify the raw mobility data.

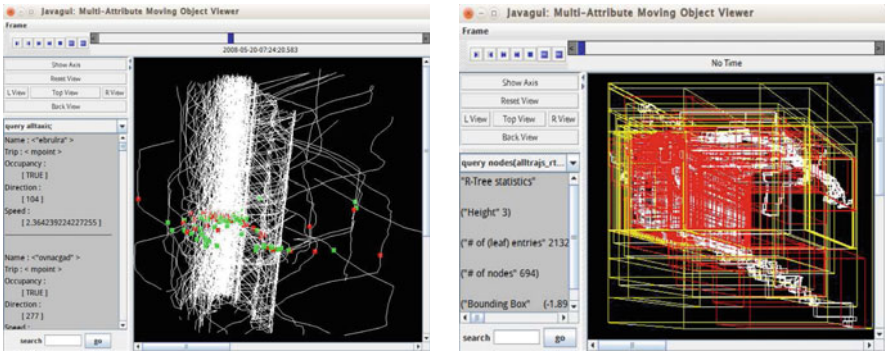
9.6.3 The Generation of Multi-attribute Values and Query Interface

There is a number of public trajectory data but attribute values are not easy to collect. A tool is developed to generate attributes. One can flexibly scale the number of attributes and the domain of each attribute. For each attribute, the value is randomly and uniformly selected from its domain. By making use of real trajectories from a company DataTang (2018) (http://factory.datatang.com/en/) and synthetic attribute values, the chapter demonstrates queries RQMAT, CRQMAT, C*k*NN_MAT, as illustrated in Fig. 9.25. Queries are "*Find all* SILVER VWs *intersecting the query window*", "*Keep reporting all* BENZs *within 5km to the target*" and "*Continuously report the nearest* RED BMW (or SILVER VM, BLACK BENZ) *to the query*



(a) (b) (c)

Fig. 9.25 Query and visualize multi-attribute trajectories. (**a**) RQMAT. (**b**) CRQMAT. (**c**) C*k*NN_MAT



(a) (b)

Fig. 9.26 The 3-D query interface. (**a**) Dynamic attributes. (**b**) Visualizing 3-D R-tree

trajectory". Objects in the figure are results at a time point and the interface provides the animation. Users can also define multiple values for each attribute. The query interface provides zoom in/out to let users get a closer/further view.

By making use of cab mobility traces from Piorkowski et al. (2009), the system demonstrates querying time-dependent attributes. Each taxi is associated with a flag marking whether the taxi is free or occupied. To make a good judgment about the shape of the R-tree, a tool graphically viewing the structure is needed. The developed query interface supports displaying dynamic attributes and visualizing the R-tree in a 3-D view, as illustrated in Fig. 9.26.

9.6.4 MDBF: A Tool for Monitoring Database Files

File monitoring plays an essential role in operating system that constantly watches folders and files. Actions are triggered when files are created or accessed (read-/write). A tool called MDBF (Monitoring Database Files) is developed to monitor

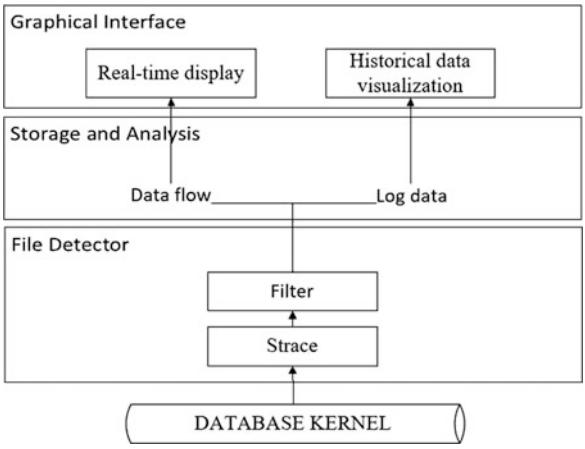


Fig. 9.27 The framework of MDBF

database files during the query execution (Wei and Xu 2018). The tool consists of three components: file detector, storage and analysis system, and graphical interface, as illustrated in Fig. 9.27. The file detector, serving as the key component, makes use of a tool Strace, which is a diagnostic, debugging and instructional userspace utility. Strace monitors and tampers with interactions between processes and Linux kernel, including system calling, signal delivering, process state changing, and read and write blocks of data.

When a query is executed, the tool detects database file operations and produces a monitoring log. The log data is automatically reported to the filter and formatted in a relational table in the system. The filtering is performed by extracting the data related to database file operations and storing action statistics. Strace captures all system calls in the query evaluation, but file operations are the main tasks. The data flow received from Strace is transformed to a certain format transferred between system modules. The information of accessing files is displayed when the querying is running and also recorded as historical data. A thorough analysis is performed on log data. MDBF monitors queries incurring database file operations. A graphical interface visualizes the access content to help understanding the query progress. Users can compare the access information of different files.

9.7 Performance Evaluation

The proposal is implemented in C/C++ and the evaluation is performed in an extensible database system SECONDO (Güting et al. 2010a). The system is a freely open source software and has an extensible architecture well-supported for spatial and spatio-temporal data management. A standard PC (Intel(R) Core(TM) i7-4770CPU, 3.4GHz, 4GB memory, 2TB hard disk) running Suse Linux 13.1 (32 bits, kernel version 3.11.6) is used.

Name	#Trips	#GPS Records
BTaxi	992,997	55,950,357

(a)

$X(Y)$	$30\% \cdot \Delta X\ (\Delta Y)$
T	$40\% \cdot \Delta T$
$\lvert Q_a \rvert$	$\{1, 2, \mathbf{3}, 4, 5\}$

(b)

Fig. 9.28 Datasets and parameters. (**a**) Standard trajectories. (**b**) Query parameters

9.7.1 Evaluation of RQMAT

We use real taxi trajectories in Beijing from DataTang (2018) (http://factory.datatang.com/en/). The statistics of standard trajectories and query parameters are reported in Fig. 9.28. Q_{box} is randomly generated with sizes $X = 30\% \cdot \Delta X$ and $Y = 30\% \cdot \Delta Y$, in which ΔX and ΔY are lengths of x and y dimensions, respectively. The time interval is 40% of the overall time. One can arbitrarily enlarge or shrink the spatio-temporal window. We did some preliminary tests and found that smaller windows may not receive any result. We focus on evaluating the performance affected by attributes and hence keep the same size for Q_{box}.

In the evaluation, CPU time and I/O accesses are used as performance metrics and the results are averaged over 20 runs. Five alternative methods are included: (i) **3D R-tree**, (ii) 3D R-tree + Attribute Set (**RAttSet** for short). The idea is similar to IR-tree (Cong et al. 2009) employed in spatial keyword querying that augments each R-tree node with a summary of keywords in the subtree, (iii) **4D R-tree**, for each multi-attribute trajectory, we distribute $(a_1, \ldots, a_{|A|})$ into $|A|$ trajectories, each of which defines the 4-D data: *location*, *time* and *a single-attribute value*. (iv) **IOC-Tree** (Han et al. 2015), the structure consists of an inverted index and a set of three-dimensional quadtrees, each of which corresponds to an attribute and stores relevant trajectory points and (v) **HAGI** (Su et al. 2007), the method employs a hierarchical aggregate grid index. The evaluation demonstrates the impact of $|Q_a|$ on the performance. Figure 9.29 shows that our method is an order of magnitude faster than other methods in most settings.

9.7.2 Evaluation of CRQMAT

We use real GPS records of Beijing taxis (DataTang (2018) http://factory.datatang.com/en/). The dataset statistics and the settings of query parameters ($|Q_a|$ and d) are reported in Table 9.5. We perform the evaluation by comparing our method named GR^2-tree (Grid R-tree with an attribute Relation) with five baseline methods in terms of scalability and efficiency: (i) **3-D R-tree**; (ii) **RIB**, we adapt the method in Wu et al. (2012). Multi-attribute trajectories are grouped on attribute values by applying Z-order to map the $|A|$-dimensional value to one-dimensional. Each R-tree node contains a pointer to an inverted bitmap that records the positions of entries

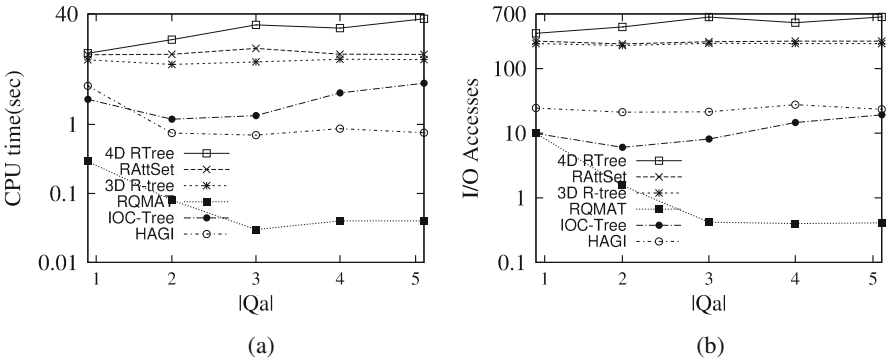


Fig. 9.29 Effect of $|Q_a|$ ($|A| = 10$). (**a**) CPU Time (sec). (**b**) I/O Accesses($\times 10^3$)

Table 9.5 Datasets and parameter settings

Name	#GPS Records	$\|\mathcal{O}\|$	$\|A\|$	$dom(A)$	X and Y ranges
BTaxi	235634511	4220435	10	[1, 151]	[21, 119958], [0, 119653]
Query settings					
$\|Q_a\|$: {1, 2, 3, 4, 5}			d (km): {1, 5, 10, 20, 50}		

Table 9.6 Datasets for scaling $|\mathcal{O}|$

Name	$\|\mathcal{O}\|$
BT1	533635
BT2	1009579
BT3	1424273
BT4	2757312
BT5	4220435

defining the attribute value. A relation stores the bitmaps by setting the fanout as the bit length; (iii) **4-D R-tree**; (iv) **IOC-Tree** (Han et al. 2015); (v) **HAGI** (Su et al. 2007).

Scalability. To vary the data size, different subsets of BTAXI are selected, as summarized in Table 9.6. The performance result is reported in Fig. 9.30. When the data size grows, the costs of all methods rise proportionally, but our method outperforms baseline methods by a factor of 5-50x on the largest dataset.

Varying $|Q_a|$. We perform the evaluation by varying the number of query attributes. The results, as reported in Fig. 9.31, demonstrate that our method substantially outperforms baseline methods in all settings. When $|Q_a|$ increases, the performance becomes better as the attribute predicate is more selective.

Varying the distance d. We evaluate the performance affected by d, as reported in Fig. 9.32. When d increases the performance degrades as expected due to more objects being processed. The advantage of our method is significant. When d increases, more cells will be included in the search space.

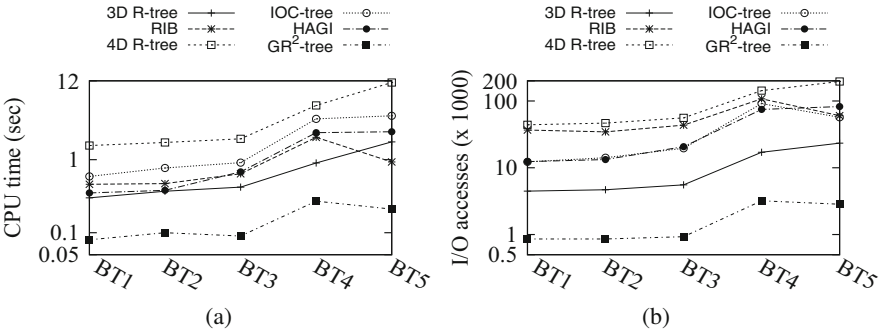


Fig. 9.30 Scaling $|\mathcal{O}|$. (**a**) CPU time. (**b**) I/O accesses

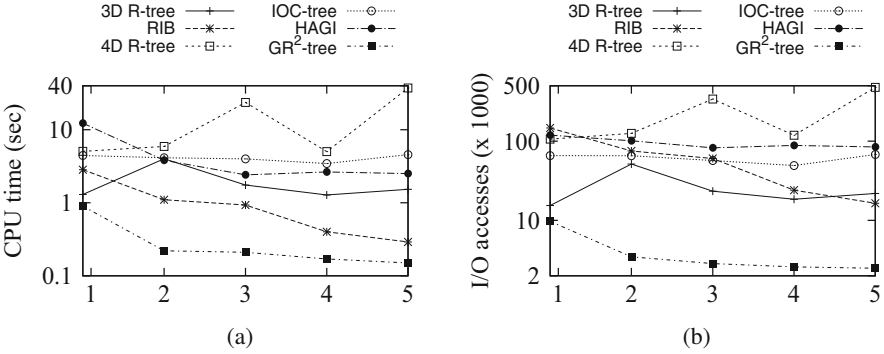


Fig. 9.31 The effect of $|Q_a|$. (**a**) CPU time. (**b**) I/O accesses

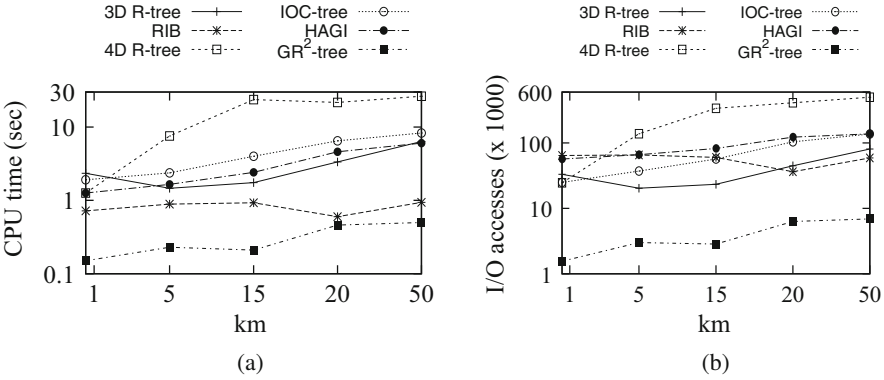


Fig. 9.32 The effect of d. (**a**) CPU time. (**b**) I/O accesses

Table 9.7 Datasets statistics and parameters

Name	$\|\mathcal{O}\|$ (million)	Temporal units # (million)	$\|A\|$	$dom(A)$
TAXI	3.0	32.5	1	[1, 4]
BUS	1.1	8.3	1	[1, 384]
MAT5	1.8	110.88	10	[1, 127]

$\|Q_a\|$	{1, 2, **3**, 4, 5}
k	{1, **5**, 10, 20, 50, 100}

9.7.3 Evaluation of CkNN_MAT

Both real and synthetic datasets are used, shown in Table 9.7. Real datasets are from a company called Datatang (2018) (http://factory.datatang.com/en/): Shanghai taxis (TAXI) and Beijing buses (BUS). TAXI includes GPS records from four taxi companies in 2014. The company id is defined as an attribute. BUS contains bus card records in 2014. Each record stores the time and bus stops where passengers gets on and off the bus. Each bus is identified by its id and bus stops are identified by the order in the route and long/lat. We build bus trips from these records. This is done by grouping records on bus id and then sorting them on time. There are 384 bus routes in total and the route id is set as an attribute. Part of the data can be found at http://dbgroup.nuaa.edu.cn/jianqiu/. Synthetic datasets are generated by utilizing a tool MWGen (Xu and Güting 2012). For each query, o_q is randomly selected over the dataset. The settings for Q_a and k are listed, in which default values are in bold. Each $a \in Q_a$ is a stochastic value from the domain.

We develop three baseline methods for performance comparison. (i) **4D R-tree**. (ii) **3D R-tree + Attribute Relation** (3D RAR). The method in Güting et al. (2010b) is extended to support proposed queries by recording the set of attributes contained by each node. During the query procedure, we first determine whether the accessed node contains Q_a. If yes, we open the node and move forward to spatial and temporal examinations. Otherwise, we prune the node. However, the R-tree does not know the number of trajectories containing Q_a for each node because queries issue different attributes. Consequently, the criterion of pruning trajectories based on distance and the number of trajectories can not be used. (iii) **3D R-tree + Inverted Bitmap** (RIB) (Wu et al. 2012). Our method is named BAR.

Scaling the number of attributes and the domain. We evaluate the scalability affected by attributes: $|A|$ and $dom(A)$. Figure 9.33 reports the settings. Figure 9.34 reports the result on scaling $|A|$. Our method achieves the best performance in all settings and RIB performs competitively to our method when $|A| = 3$. RIB manages attributes and the attribute predicate has a good selectivity when $|Q_a| = |A|$. However, when $|A|$ increases, $dom(A)$ rises proportionally, and the RIB performance degrades significantly. We analyze that Z-order values cannot well preserve the locality when the dimension becomes large, and the linear scanning method of determining entries is inferior to our bitmap querying approach. When

Fig. 9.33 Datasets for Scalability. (**a**) $|A|$. (**b**) $dom(A)$

$\|A\|$	$dom(A)$
1	[1, 5]
3	[1, 28]
6	[1, 76]
10	[1, 127]
20	[1, 247]

(a)

$\|A\|$	$dom(A)$
10	[1, 79]
	[1, 127]
	[1, 247]
	[1, 427]
	[1, 857]

(b)

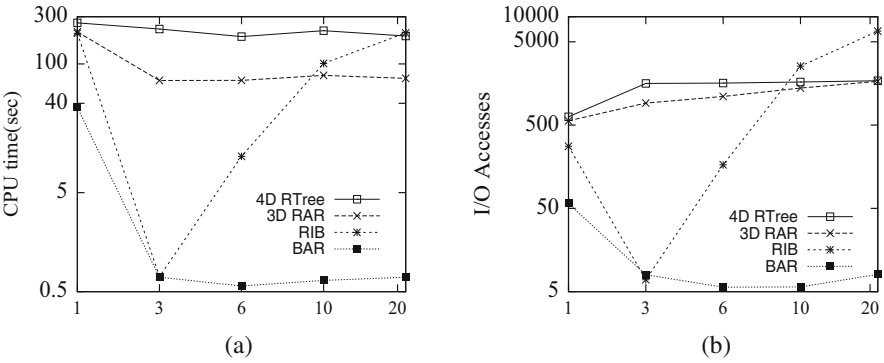


Fig. 9.34 Scaling $|A|$. (**a**) CPU Time (s). (**b**) I/O Accesses($\times 10^3$)

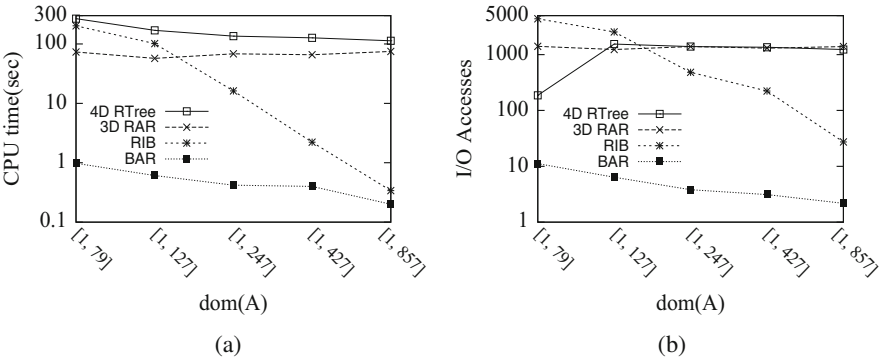


Fig. 9.35 Scaling $dom(A)$, $|A| = 10$. (**a**) CPU Time (s). (**b**) I/O Accesses($\times 10^3$)

scaling $dom(A)$, our method also outperforms baseline methods. One can see the trend that the performance increases when $dom(A)$ becomes large as the attribute predicate is more selective. The 4D R-tree has poor performance because the dataset is enlarged when $dom(A)$ increases (Fig. 9.35).

Effect by k. We evaluate the performance effected by k and report the results in Figs. 9.36 and 9.37. TAXI and BUS contain only one attribute and therefore we

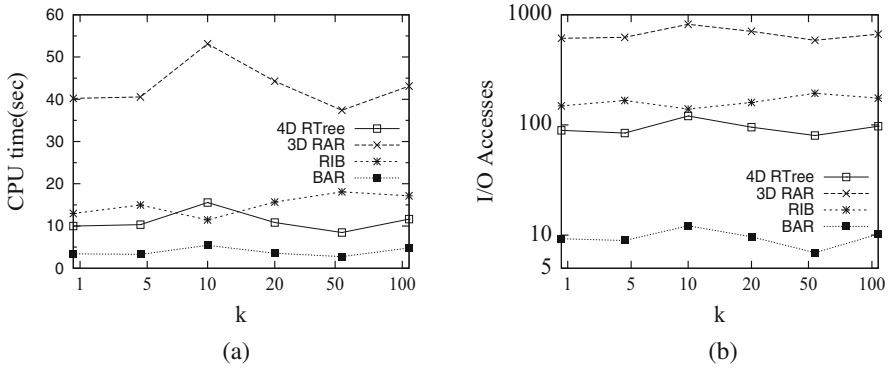


Fig. 9.36 Scaling k using TAXI. (**a**) CPU Time (s). (**b**) I/O Accesses($\times 10^3$)

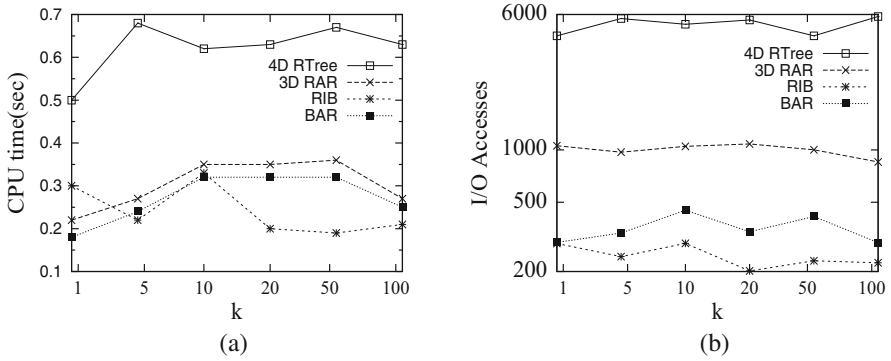


Fig. 9.37 Scaling k using BUS. (**a**) CPU Time (s). (**b**) I/O Accesses($\times 10^3$)

set $|Q_a| = 1$ for them. All methods are not sensitive to k. Our method significantly outperforms alternative methods for TAXI. For BUS, the RIB performance is close to ours due to good attribute selectivity. BUS has $|A| = 1$ and $dom(A)$ is [1, 384]. In contrast, we have $|A| = 1$, $dom(A) = [1, 4]$ for TAXI.

The analysis. Consider the 4D R-tree. To build the index, each multi-attribute trajectory is decomposed into $|A|$ trajectories, each of which contains a single attribute. This enlarges the dataset by $|A|$ times. Also, the attribute is approximately evaluated when traversing the index. The method 3D RAR is able to select R-tree nodes containing individual attributes, but cannot determine whether data objects contain all query attributes if $|Q_a| > 1$. RIB achieves good performance when the attribute predicate is quite selective, for example, (i) $|Q_a| = |A|$, or (ii) $dom(A)$ is large and $|Q_a| = 1$. Thus, this method is limited in scope. Our method achieves the stable performance and generalizes to queries on standard trajectories, achieved by skipping the step of accessing BAR.

9.8 Future Directions

9.8.1 Data Analytics

By making use of multi-attribute trajectories, one is able to provide a fine-grained analysis on the traffic condition by discovering dense areas such as taxi pick-ups and drop-offs and crowded bus/metro stations. An optimal deployment can be made by recommending the ride-sharing and redesigning the scheduling/route. Based on the number of passengers on the bus/metro at populated places, the system can recommend people to advance or postpone their trips in a small derivation in order to avoid the peak time and reduce the waiting time.

Mining and analyzing multi-attribute trajectories is also an interesting topic. One can utilize the rich contexts from attributes to fully understand spatio-temporal trajectories and discover potential relationships and behavior. For example, in order to know whether large vehicles such as buses and trucks have a great negative impact on traffic flow, the system should consider not only the number of vehicles in crowded places but also the percentage of large vehicles. Based on that, policies can be made to improve the traffic. In some applications, attribute values change over time, e.g., taxi status (free or occupied), fuel consumption and the number of passengers in a bus. The data representation needs to be extended to support dynamic attributes. One solution is to define a moving integer/real to represent the time-dependent value. The question is how to efficiently manage several dynamic attributes in the framework and adjust the index structure correspondingly. Application queries and analysis can be performed by considering spatio-temporal parameters and dynamic attribute values to find some interesting behavior.

9.8.2 Intelligent Trajectory Data Management

The artificial intelligence community has accomplished many promising results, while the specialization to moving objects will offer new opportunities because solutions are fitted to particular properties of mobile data. Due to the increasing number of models and structures, a number of parameters are required to control the system. An automatic approach of recommending system settings is preferred that leverages past experiences of workloads. Machine learning models can be trained by transferring previous experiences to apply for a new application.

The query interface should be powerful enough to support arbitrary queries and simple enough to let users express their questions in natural language. User queries are expressed by natural language but will be translated into an executable language in the system. However, employing natural language to query the database is a non-trivial task. Most database systems are queried by structured query language which is often difficult to write for non-experts. Moving objects databases should be capable of providing a communication model that translates natural language

questions into well-formed queries such that the user and the system can understand each other. There are hundreds of published papers about querying moving objects in which example queries are expressed in natural languages, but an interface translating natural languages into an SQL or SQL-like language is not available.

9.9 Conclusions

This chapter introduces multi-attribute trajectories that enrich the spatio-temporal trajectory representation. A range of new queries is studied that search for the target fulfilling both spatio-temporal and attribute conditions. A hybrid index structure is designed and updating the index is supported. Efficient query algorithms are developed with optimization strageties. A systematic design is made such that the proposed structure is also able to process standard trajectories with little effort. We develope a prototype database system for multi-attribute trajectories including data storage, access methods, index structures, data generators, monitoring tools and the query interface. The performance evaluation is conducted by using real and synthetic datasets.

Acknowledgments This work is supported by NSFC under grants 61972198, Natural Science Foundation of Jiangsu Province of China under grants BK20191273 and National Key Research and Development Plan of China (2018YFB1003902). Thanks Weiwei Wang for developing the 3-D visualization tool and Xiangyu Wei for developing the monitoring tool.

References

Alvares LO, Bogorny V, Kuijpers B, Moelans B, Fern JA, Macedo ED, Palma AT (2007) Towards semantic trajectory knowledge discovery. Data Mining and Knowl Discov

Bentley JL, Ottmann T (1979) Algorithms for reporting and counting geometric intersections. IEEE Trans Comput 28(9):643–647

Bercken J, Seeger B (2001) An evaluation of generic bulk loading techniques. In: VLDB, pp 461–470

Bercken J, Seeger B, Widmayer P (1997) A generic approach to bulk loading multidimensional index structures. In: VLDB, pp 406–415

Chakka VP, Everspaugh A, Patel JM (2003) Indexing large trajectory data sets with seti. In: CIDR

Chen Z, Tao Shen H, Zhou X, Zheng Y, Xie X (2010) Searching trajectories by locations: an efficiency study. In: SIGMOD, pp 255–266

Chen L, Cong G, Jensen CS, Wu D (2013) Spatial keyword query processing: an experimental evaluation. PVLDB 6(3):217–228

Cong G, Jensen CS, Wu D (2009) Efficient retrieval of the top-k most relevant spatial web objects. PVLDB 2(1):337–348

de Almeida VT, Güting RH (2005) Indexing the trajectories of moving objects in networks. GeoInformatica 9(1):33–60

De Felipe I, Hristidis V, Rishe N (2008) Keyword search on spatial databases. In: ICDE, pp 656–665

Dinh L, Aref WG, Mokbel MF (2010) Spatio-temporal access methods: part 2 (2003–2010). IEEE Data Eng Bull 33(2):46–55

Forlizzi L, Güting RH, Nardelli E, Schneider M (2000) A data model and data structures for moving objects databases. In: SIGMOD, pp 319–330

Frentzos E (2003) Indexing objects moving on fixed networks. In: SSTD, pp 289–305

Frentzos E, Gratsias K, Pelekis N, Theodoridis Y (2005) Nearest neighbor search on moving object trajectories. In: SSTD, pp 328–345

Frentzos E, Gratsias K, Pelekis N, Theodoridis Y (2007) Algorithms for nearest neighbor search on moving object trajectories. GeoInformatica 11(2):159–193

Güting RH, Schneider M (2005) Moving objects databases. Morgan Kaufmann, Amsterdam

Güting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, Vazirgiannis M (2000) A foundation for representing and querying moving objects. ACM TODS 25(1):1–42

Güting RH, Behr T, Düntgen C (2010a) SECONDO: a platform for moving objects database research and for publishing and integrating research implementations. IEEE Data Eng Bull 33(2):56–63

Güting RH, Behr T, Xu J (2010b) Efficient k-nearest neighbor search on moving object trajectories. VLDB J 19(5):687–714

Güting RH, Valdés F, Damiani ML (2015) Symbolic trajectories. ACM Trans Spat Algorithms Syst 1(2):Article 7

Hadjieleftheriou M, Kollios G, Tsotras VJ, Gunopulos D (2002) Efficient indexing of spatiotemporal objects. In: EDBT, pp 251–268

Han Y, Wang L, Zhang Y, Zhang W, Lin X (2015) Spatial keyword range search on trajectories. In: DASFAA, pp 223–240

Jensen CS, Lu H, Yang B (2009) Indexing the trajectories of moving objects in symbolic indoor space. In: SSTD, pp 208–227

Jeung H, Yiu ML, Zhou X, Jensen CS, Shen HT (2008) Discovery of convoys in trajectory databases. PVLDB 1(1):1068–1080

Lange R, Dürr F, Rothermel K (2011) Efficient real-time trajectory tracking. VLDB J 20(5):671–694

Lee T, Park J, Lee S, et al (2015) Processing and optimizing main memory spatial-keyword queries. PVLDB 9(3):132–143

Long C, Wong RCW, Jagadish HV (2013) Direction-preserving trajectory simplification. PVLDB 6(10):949–960

Lu H, Cao X, Jensen CS (2012) A foundation for efficient indoor distance-aware query processing. In: ICDE, pp 438–449

Mauroux PC, Wu E, Madden S (2010) Trajstore: an adaptive storage system for very large trajectory data sets. In: ICDE, pp 109–120

Parent C, Spaccapietra S, Renso C, Andrienko GL, Andrienko NV, Bogorny V, Damiani ML, Gkoulalas-Divanis A, de Macêdo JAF, Pelekis N, Theodoridis Y, Yan Z (2013) Semantic trajectories modeling and analysis. ACM Comput Surv 45(4):42

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

Pfoser D, Jensen CS (2003) Indexing of network constrained moving objects. In: GIS, pp 25–32

Pfoser D, Jensen CS (2000) Novel approaches in query processing for moving object trajectories. In: VLDB, pp 395–406

Piorkowski M, Sarafijanovic-Djukic N, Grossglauser M CRAWDAD dataset epfl/mobility (v. 24 Feb 2009). http://crawdad.org/epfl/mobility/20090224

Popa IS, Zeitouni K, Oria V, Barth D, Vial S Indexing in-network trajectory flows. VLDB J 20(5):643–669 (2011)

Rasetic S, Sander J, Elding J, Nascimento MA (2005) A trajectory splitting model for efficient spatio-temporal indexing. In: VLDB, pp 934–945

Song Z, Roussopoulos N (2003) Seb-tree: an approach to index continuously moving objects. In: MDM, pp 340–344

Su Y, Wu Y, Chen ALP (2007) Monitoring heterogeneous nearest neighbors for moving objects considering location-independent attributes. In: DASFAA, pp 300–312
Su H, Zheng K, Zeng K, Huang J, Sadiq SW, Yuan NJ, Zhou X (2015) Making sense of trajectory data: a partition-and-summarization approach. In: ICDE, pp 963–974
Tao Y, Papadias D (2001) Mv3r-tree: a spatio-temporal access method for timestamp and interval queries. In: VLDB, pp 431–440
Tong Y, Chen Y, Zhou Z, Chen L, Wang J, Yang Q, Ye J, Lv W (2017) The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms. In: ACM SIGKDD, pp 1653–1662
Tong Y, Zeng Y, Zhou Z, Chen L, Ye J, Xu K (2018) A unified approach to route planning for shared mobility. PVLDB 11(11):1633–1646
Tzoumas K, Yiu ML, Jensen CS (2009) Workload-aware indexing of continuously moving objects. PVLDB 2(1):1186–1197
Valdés F, Güting RH (2014) Index-supported pattern matching on symbolic trajectories. In: ACM SIGSPATIAL, pp 53–62
Valdés F, Güting RH (2017) Index-supported pattern matching on tuples of time-dependent values. GeoInformatica 21(3):429–458
Valdés F, Güting RH (2019) A framework for efficient multi-attribute movement data analysis. VLDB J 28(4):427–449
Wang H, Zimmermann R (2011) Processing of continuous location-based range queries on moving objects in road networks. IEEE Trans Knowl Data Eng 23(7):1065–1078
Wang W, Xu J (2017) A tool for 3d visualizing moving objects. In: APWeb-WAIM, pp 353–357
Wang X, Zhang Y, Zhang W, Lin X, Huang Z (2016) SKYPE: top-k spatial-keyword publish/subscribe over sliding window. PVLDB 9(7):588–599
Wei X, Xu J (2018) MDBF: a tool for monitoring database files. In: ER Workshops, pp 54–58
Wu D, Yiu ML, Cong G, Jensen CS (2012) Joint top-k spatial keyword query processing. IEEE Trans Knowl Data Eng 24(10):1889–1903
Xu J, Güting RH (2012) Mwgen: a mini world generator. In: IEEE MDM, pp 258–267
Xu J, Güting RH (2013) A generic data model for moving objects. GeoInformatica 17(1):125–172
Xu J, Güting RH (2017) Query and animate multi-attribute trajectory data. In: ACM CIKM, pp 2551–2554
Xu J, Güting RH, Qin X (2015a) Gmobench: benchmarking generic moving objects. GeoInformatica 19(2):227–276
Xu J, Güting RH, Zheng Y (2015b) The TM-RTree: an index on generic moving objects for range queries. GeoInformatica 19(3):487–524
Xu J, Güting RH, Gao Y (2018a) Continuous k nearest neighbor queries over large multi-attribute trajectories: a systematic approach. GeoInformatica 22(4):723–766
Xu J, Lu H, Güting RH (2018b) Range queries on multi-attribute trajectories. IEEE Trans Knowl Data Eng 30(6):1206–1211
Yan Z, Chakraborty D, Parent C, Spaccapietra S, Aberer K (2011) Semitri: a framework for semantic annotation of heterogeneous trajectories. In: EDBT, pp 259–270
Yao B, Xiao X, Li F, Wu Y (2014) Dynamic monitoring of optimal locations in road network databases. VLDB J 23(5):697–720
Zhang C, Han J, Shou L, Lu J, La Porta TF (2014) Splitter: mining fine-grained sequential patterns in semantic trajectories. PVLDB 7(9):769–780
Zheng K, Su H (2015) Go beyond raw trajectory data: quality and semantics. IEEE Data Eng Bull 38(2):27–34
Zheng Y, Zhou X (2011) Computing with spatial trajectories. Springer, New York
Zheng K, Shang S, Yuan NJ, Yang Y (2013a) Towards efficient search for activity trajectories. In: ICDE, pp 230–241
Zheng K, Zheng Y, Yuan NJ, Shang S (2013b) On discovery of gathering patterns from trajectories. In: ICDE, pp 242–253
Zheng B, Yuan NJ, Zheng K, Xie X, Sadiq SW, Zhou X (2015) Approximate keyword search in semantic trajectory database. In: ICDE, pp 975–986