Range Queries on Multi-Attribute Trajectories

Jianqiu Xu, Hua Lu, *Senior Member, IEEE*, and Ralf Hartmut Güting

Abstract—Motivated by the trend of providing comprehensive knowledge about trajectory data, we study multi-attribute trajectories each of which contains a sequence of time-stamped locations and a set of characteristic attributes. This enriches the data representation by providing a comprehensive description of moving objects and thus enables new types of queries on moving object trajectories. In this paper, we consider answering range queries that return trajectories (i) containing particular attribute values and (ii) passing a certain area during the query time. We integrate standard trajectories and attributes into one unified framework and propose an index structure as well as the query algorithm. The structure is general and flexible in terms of handling both multi-attribute trajectories and standard trajectories, answering a range of queries and supporting update-intensive applications. The evaluation is conducted in a prototype database system and experimental results demonstrate that our method outperforms alternative methods by a factor of 3-10 on a data set of one million real trajectories and synthetic attribute values.

Index Terms—Trajectories, range queries, multiple attributes, index structure

1 Introduction

TRAJECTORY databases are used to manage the historical data of moving objects, allowing for complex queries and analysis of movements in the past. Although there is an extensive literature on modeling and querying trajectory data, existing research works mainly focus on standard trajectories, i.e., a sequence of time-stamped locations. In the real world, typical moving objects such as vehicles and persons are associated with pieces of descriptive information in addition to location and time. Moving objects databases should expand the capability to have a full representation for handling both standard trajectories and attributes. This allows users to query objects with extensive knowledge and to find particular trajectories by integrating the attribute into the query. Consequently, users can fully understand the behavior of moving objects, i.e., knowing not only when and where but also which and what. The multi-attribute data representation provides an overview of different aspects of real-world objects and has been recently involved in a number of new applications [1].

In this paper, we focus on multi-attribute trajectories in which the content of each object consists of time-stamped locations and descriptive attributes. GPS data contains location and time but is not sufficient. We need to combine different data sources. Queries are extended as users are allowed to include spatio-temporal parameters and attribute values to find interesting trips.

Consider a database storing vehicle trips in a city. Each of the trips contains a standard trajectory and two attribute values over domains COLOR = {RED, SILVER, GRAY} and BRAND = {BENZ, BMW}, as illustrated in Fig. 1. We answer the query that contains a tuple of attribute values and a spatio-temporal box, for example, "*Did any* SILVER BMW *pass the area during* $[t_1, t_2]$?". We study Boolean range queries that will report objects containing query attribute values and fulfilling the spatio-temporal condition. In the example, o_3 is returned. To the best of our knowledge, the problem has not been studied in the literature.

Recently, researchers have started to investigate trajectories annotated with semantic data [2]. Labels are attached to geo-locations to describe places that the user has visited or performed activities at. However, the current semantic data is restricted to locations. This is orthogonal to our work that considers location-independent attributes. First, multi-attribute trajectories represent different aspects of the objects such as categories and roles, and combine the attributes with standard trajectories to have a comprehensive picture of moving objects. This can support a different (even broader) range of applications. Semantic locations are sparsely defined because a few locations may have semantics among all locations of a traveler, but attributes are not. The similarity-based search studied in [3] defines a score function to rank objects by combining spatial closeness and keywords similarity. However, that is essentially a spatial query without considering the time and semantics is also location-dependent.

Second, different tasks will be performed when designing the index. The indexes for semantic trajectories group objects in terms of locations and semantics, but attributes in multi-attribute trajectories are not related to locations. We could employ the index of semantic trajectories by attaching attributes to locations but this does not suit queries only involving attributes, e.g., "find all SILVER BMWs". To answer the query, we have to employ the traditional index such as B-tree. Therefore, semantic trajectory indexes are mainly used to answer queries involving both locations and semantics but their performance generally is not optimal.

To efficiently process multi-attribute trajectories, an index is essentially required because a sequential scan over the database is prohibitively expensive for large datasets. One can employ two individual indexes (e.g., 3D R-tree and 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. The challenge is how to design an integrated structure that simultaneously manages both parts and allows evaluating the AND predicate. Meanwhile, the structure should be general and flexible in order to support a range of queries for multi-attribute trajectories and standard trajectories, and update-intensive applications. We make the following contributions in the paper:

(i) We introduce multi-attribute trajectories, give the data representation and define a new query.
(ii) We propose a hybrid index structure and an efficient algorithm to answer range queries on multi-attribute trajectories together with an efficient updating method for the index.
(iii) Using large real datasets, we conduct an extensive experimental study to demonstrate the superior performance of our method over five alternative methods in terms of efficiency and scalability.

The rest of the paper is organized as follows. Section 2 surveys the related work. Section 3 defines multi-attribute trajectories and the query. Sections 4 and 5 present the index and query algorithm, respectively. Section 6 provides the updating method. Section 7 reports on the experiment evaluation, followed by conclusion in Section 8.

2 Related Work

In the literature, much research has focused on querying standard trajectories, e.g., nearest neighbor, similarity queries and prediction

- *J. Xu is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 211106, China. E-mail: jianqiu@nuaa.edu.cn.*
- *H. Lu is with the Department of Computer Science, Aalborg University, Aalborg East, Nordjylland DK-9220, Denmark. E-mail: luhua@cs.aau.dk.*
- *R. H. Güting is with the Fakultät für Mathematik und Informatik, LG Datenbanksysteme für neue Anwendungen, Hagen, NRW 58084, Germany. E-mail: rhg@fernuni-hagen.de.*

Manuscript received 22 May 2017; revised 16 Nov. 2017; accepted 23 Dec. 2017. Date of publication 27 Dec. 2017; date of current version 27 Apr. 2018.
(Corresponding author: Jianqiu Xu.)
Recommended for acceptance by R. Cheng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2787711

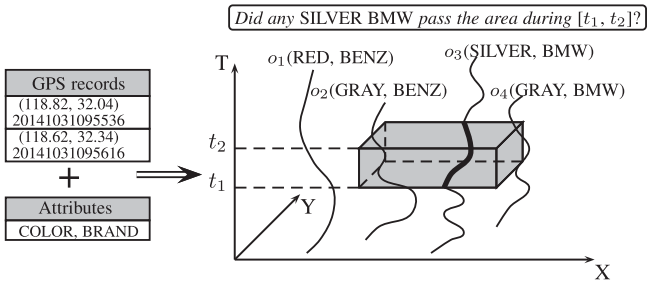


Fig. 1. An illustrative example.

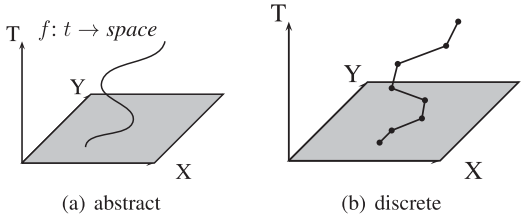


Fig. 2. Standard trajectory representation.

[4]. A large number of indexes have been proposed [5], [6]. However, these techniques only deal with spatial and temporal perspectives without attributes. 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. Existing methods cannot be directly applied because either false results will be yielded if the attributes are not evaluated or the performance is not good because spatio-temporal indexes cannot be used to evaluate the attributes.

2.1 Semantic Trajectories

Recently, trajectories with semantic data have received increasing attention [7]. A semantic-enriched trajectory is typically defined to be a sequence of time-stamped places, where each place is represented by a location with a semantic label, e.g., *hotel*, *restaurant*. Frequent patterns can be discovered by grouping trajectories on spatial, temporal and semantic aspects. Similarly, an activity trajectory is defined as a sequence of geo-spatial points associated with activities. An activity represents a type of actions such as sport, dining and entertaining that a user can take at certain places [2]. A query returns k trajectories whose semantics contain certain activities and have the shortest match distance. Efficient similarity search is also studied on activity trajectory databases [3]. In [8], a trajectory over diverse geographical spaces includes time-stamped locations and a sequence of transportation modes such as *Indoor*→ *Walk* →*Car*, and queries containing transportation modes can be answered.

Those works deal with trajectories with supplementary information, but the extended data is limited to locations such as places of interest and activities. We intend to represent location-independent attributes that cannot be derived from geometric trajectories and the geographic environment. Multi-attribute trajectories will lead to new queries that consider the attribute match and the intersecting condition on spatial and temporal parameters.

Heterogeneous k-nearest neighbor queries are studied in [9]. A moving object is represented by a location-independent attribute and a set of coordinates. By defining a function that combines the cost of distances and the location-independent attribute, the query returns objects having the kth smallest value. Although that work considers the location-independent attribute, there are three major differences. First, the data representation is limited in scope because each moving object is associated with only one attribute. We consider multiple attributes to have a generic solution. Second, they evaluate objects based on a ranking function on distance and attribute, but we require the exact match on attribute. This leads to different query results. Third, their distance function is not time-dependent, but we deal with spatio-temporal queries.

A generic model is proposed to capture a wide range of meanings derived from a standard trajectory, called *symbolic trajectory* [10]. Such a trajectory is represented by a time-dependent label. For example, labels can be names of roads, modes and speed profile. Nevertheless, symbolic trajectories focus on semantic labels without considering geo-locations.

2.2 Spatial Keywords Search

Spatial keyword queries (SKQ for short) have attracted widespread attention in the literature [11]. A typical SKQ takes a geo-location and a set of text descriptions called keywords as arguments and returns objects that satisfy other query conditions. A Boolean kNN query [12] returns objects that are close to the query location and contain the query keywords. A top-k NN query [13] returns objects with the highest ranking scores measured by a combination of distances to the query location and the text relevance to the query keywords. To efficiently answer a query, a spatial index like 2D R-tree and a text index structure are combined. For example, the IR-tree arguments 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 processing, the combined structure is used to estimate both the spatial distance and the text relevancy, and to prune the spatial objects that cannot contribute to the result.

Compared to SKQ that studies geo-locations and text descriptions, multi-attribute trajectories are in principle defined by the combination of standard trajectories and attributes. Both problems extend the traditional spatial and moving objects to enrich the data representation. However, there are some significant differences. In the aspect of data representation, SKQ focuses on static geo-locations and location-dependent text descriptions. Multi-attribute trajectories, on the other hand, deal with moving objects with location-independent attributes. Text descriptions and attributes entail different designs of the index structures. In SKQ, the index groups objects in terms of spatial distance and location text relevance. In our problem, attributes do not depend on locations but are associated with the complete trajectory. Furthermore, Boolean range queries have been studied in spatial keywords search [14], but not for multi-attribute trajectories.

3 Problem Definition

3.1 Standard Trajectories

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. We define each trajectory by a sequence of so-called *temporal units*, as illustrated in Fig. 2. Each unit records the start and end locations during a time interval, and locations between start and end points are estimated by interpolation. A data type called *mpoint* is defined for standard trajectories. The comprehensive framework for trajectories is presented in [15].

3.2 Multi-Attribute Trajectories

A multi-attribute trajectory consists of a sequence of time-stamped locations and a set of attributes, represented by o(*Trip*, *Att*), where *Trip* is a standard trajectory and *Att* defines a set of d attributes. The domain of each attribute is denoted by $dom(A_i)$ ($i \in [1, d]$). We define the data type for multi-attributes as follows.

Definition 3.1. *Multi-attributes*

$$D_{att} = \{(a_1, \ldots, a_d) | a_i \in dom(A_i), i \in \{1, \ldots, d\}\}.$$

TABLE 1
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)

Using the example in Fig. 1, we have two attributes A_1 = COLOR and A_2 = BRAND with domains $dom(A_1)$ = {RED, SILVER, GRAY} and $dom(A_2)$ = {BENZ, BMW}. We use a relational interface to integrate standard trajectories and attributes into one framework. Table 1 gives the representation of multi-attribute trajectories in Fig. 1.

Employing a relational interface, a multi-attribute trajectory is represented by a tuple consisting of an *mpoint* and a d-value attribute. The integration method has the advantage that one can (i) leverage existing trajectory operators and relational operators to formulate queries, and (ii) extract standard trajectories from multi-attribute trajectories, resulting in a flexible way to manipulate the data. This benefits the system development.

The range query is one of the fundamental operations in trajectory databases. We incorporate the attribute into queries and evaluate the exact match of attribute values. Let O denote a set of multi-attribute trajectories. Given a tuple of attribute values $Q_a = (a_1, \ldots, a_d)(a_i \in dom(A_i)$ or $a_i = \perp)$ and a multi-attribute trajectory $o \in O$, we define that $o.Att$ *contains* Q_a if $\forall\, a_i \in Q_a$: $a_i \in o.Att$ or $a_i = \perp$. The query called RQMAT (Range Queries on Multi-attribute Trajectories) is defined as follows.

Definition 3.2 (RQMAT). *Given a spatio-temporal window Q_{box} and attribute values Q_a, the query returns a set of trajectories $O' \subseteq O$ such that $\forall\, o \in O' : (i)$ $o.Att$ contains Q_a; and (ii) o.*Trip *intersects Q_{box}.*

Using the query in Fig. 1, attribute values are formated by Q_a = (SILVER, BMW). We can use the SQL-like language to express the query.

- *Did any* SILVER BMW *pass the area during* $[t_1, t_2]$?

```
LET Qbox be the spatio-temporal box;
SELECT o.Id
FROM O as o
WHERE o.Att contains (SILVER, BMW)
      and o.Trip intersects Qbox
```

4 Index

Standard trajectory indexes are not capable of managing attributes and will inhibit the performance because such indexes do not allow us to perform the selection on attribute values. An alternative choice is to build a separate attribute index and evaluate the attribute condition first to obtain trajectories containing query attribute values. If the attribute predicate is selective, the query cost is acceptable for a sequential scan because only a small dataset is processed. If the attribute predicate has a low selectivity, a large number of trajectories will be returned. Both the sequential scan and on-line index construction incur high computation cost.

To efficiently answer RQMAT, we develop an index structure to manage multi-attribute trajectories, as shown in Fig. 3. The idea is

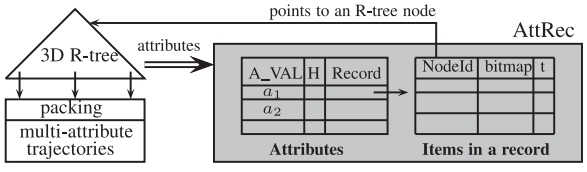


Fig. 3. The index architecture.

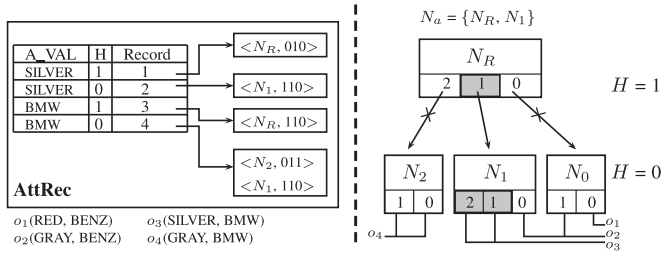


Fig. 4. *AttRec* on 3D R-tree.

to let the index preserve the spatio-temporal proximity and also maintain attribute values. The structure includes a 3D R-tree for indexing standard trajectories and a structure named *AttRec* (Attributes and Records) for maintaining attribute values contained in R-tree nodes. *AttRec* includes a relation for storing attribute values and a set of records for storing R-tree nodes.

Raw standard trajectory data contains many small temporal units in terms of a short time interval or slow movement. We process standard trajectories by packing small pieces of movements in order to reduce the number of objects for building the index. Afterwards, we build the 3D R-tree by bulk load on packed trajectories. Each piece of trajectories is approximated by a 3D bounding box. An R-tree node contains a list of entries, each of which stores a pair of items: a bounding box that is the union information of spatial and temporal data and a pointer references to a subtree or trajectory. The structure *AttRec* is created on top of the R-tree to build the connection between attribute values and R-tree nodes. To index attribute values in a unified way, we map each value to an integer and there is no overlap between different domains. This is done by using a two-dimensional point (x, y) $(x, y \in \mathbb{Z}^+)$ for each attribute value in which x is the attribute id and y is the value. Then, we interleave the binary representations of x and y to achieve the combined value.

Let $dom(Att) = \bigcup_{i=1}^{d} dom(A_i)$ denote the overall domain. For each $a \in dom(Att)$, we create a relation to record the nodes containing a at each level of the R-tree in which *A_VAL* is the attribute, H is the level and *Record* is a record storing nodes containing the attribute value. At the root level, only one node contains the value. At the leaf level, a large number of nodes may contain the value.

Each record maintains a list of items: each item defines a bitmap to mark the entries in a node containing the attribute value and the union time of those entries. During the query processing, we employ the bitmap to quickly determine the entries (subtrees) containing the query rather than perform a sequential scan over all entries. Usually a few entries fulfill the condition, especially when the traversal goes to the low levels of the R-tree. The method guarantees that each visited node contains the query (at least one of attribute values) except the root node and prunes child nodes without loading them from disk into main-memory, reducing both CPU and I/O costs. Based on the running example, we give a part of the structure *AttRec* in Fig. 4.

5 Query Processing

The query processing runs in two steps. Step 1 accesses *AttRec* 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, during which the search space is pruned based on spatio-temporal parameters and attribute values. Fig. 5 outlines the procedure.

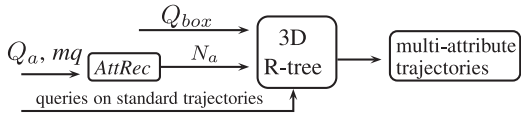


Fig. 5. An outline of the query procedure.

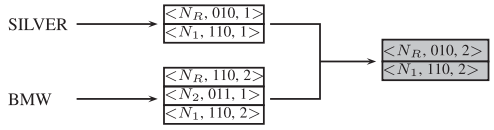


Fig. 6. Select nodes according to Q_a.

Our method combines 3D R-tree and *AttRec*, and is able to flexibly choose to build the traditional trajectory index or the hybrid index depending on whether standard trajectories or multi-attribute trajectories are processed. *AttRec* is not tightly integrated into the 3D R-tree and thereby can be widely built on many other trajectory indexes, e.g., TB-tree [5], SETI [16]. Furthermore, *AttRec* is only used for the attribute evaluation and spatio-temporal parameters will be evaluated when accessing the trajectory index. As a result, nearest neighbor and similarity queries with attributes can also be answered using the proposed structure, making our approach general.

5.1 Select R-Tree Nodes

The 3D R-tree only manages standard trajectories without knowing attribute values. To determine the objects containing query values, we access *AttRec* to look for the nodes fulfilling the attribute condition. That is, we search the nodes in which there are trajectories containing Q_a. For each attribute value, we search *AttRec* to find the tuple and get the record. Each item in the record is represented by (*NodeId*, b), in which *NodeId* is the node id and b is the bitmap marking entries containing the value. We obtain such an item for each attribute value and perform the intersection on bitmaps to find the entries containing the query.

Let T be a function returning the defined time of an R-tree node and the query trajectory. We evaluate the time dimension of the node and check whether the item (a candidate node) exists in the returned node set, denoted by N_a. If not, we insert the item by adding a *counter*, initialized by 1. If yes, we increase the *counter* and update the bitmap by performing the operation AND. In the end, we remove items in N_a that cannot contribute to the result. The algorithm is presented in Algorithm 1.

Algorithm 1. *CollectNodes*(Q_a, *AttRec*, *mq*)

```
1: N_a ← ∅;
2: for all a ∈ Q_a do
3:     RD ← GetRecord(AttRec, a);
4:     for all (NodeId, b) ∈ RD do
5:         if ∃ δ ∈ N_a : δ.NodeId = NodeId ∧
6:         T(δ) overlaps T(mq) then
7:             δ.counter++;
8:             δ.b ← δ.b AND b;
9:         else
10:            δ ← (NodeId, b, 1);
11:            N_a ← δ ;
12: Prune nodes in N_a according to Lemma 1;
13: return N_a;
```

Lemma 1. *We prune each* $\delta \in N_a$ *if* δ.counter $\neq |Q_a|$ *or* $\delta.b = 0$.

Proof. (i) $\delta.counter \neq |Q_a|$: It is impossible that $\delta.counter > |Q_a|$ because distinct attribute values are counted. If $\delta.counter < |Q_a|$, the number of attributes in the node is less than $|Q_a|$ and δ can be safely pruned. (ii) $\delta.b = 0$: There is no entry in the node containing all query values and thus we can safely prune δ. □

Based on Fig. 4, we illustrate the selecting procedure for Q_a = (SILVER, BMW) in Fig. 6. We first find the nodes containing SILVER and then evaluate whether they contain BMW. N_2 will be removed because the counter is only 1.

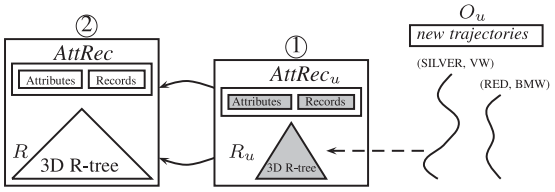


Fig. 7. Update the index.

Algorithm 2. *RQMAT*(Q_a, Q_{box}, *AttRec*, R, O)

```
1: N_a ← CollectNodes(Q_a, AttRec);
2: O' ← ∅, L ← R.RootId;
3: while L is not empty do
4:     N ← GetNode(R, L.top());
5:     for all entry N[i] do
6:         if N[i] is a leaf node then
7:             o ← GetTuple(O, N[i].tid);
8:             if o.Att contains Q_a ∧ o.Trip intersects Q_box then
9:                 O' ← o;
10:        else
11:            if N[i] ∈ N_a ∧ box(N) intersects Q_box then
12:                L ← N[i];
13: return O';
```

5.2 The Algorithm

We traverse the R-tree denoted by R in a top-down fashion, during which we evaluate the spatio-temporal condition on the nodes from Algorithm 1. The algorithm is given in Algorithm 2.

Fig. 4 exemplifies the procedure. By accessing *AttRec*, we find N_R and N_1, and then traverse the R-tree level by level. N_2 and N_0 are pruned because they do not contain (SILVER, BMW). We access the node N_1, test entries, and report o_3.

6 Update

Update-intensive applications require database systems to support updates efficiently. A significant task is to synchronize index structures. Given a set of new multi-attribute trajectories, we need to update two structures: (i) 3D R-tree and (ii) *AttRec*. In general, a new R-tree named R_u is created on the new data set O_u and the structure $AttRec_u$ is built on R_u. Then, R_u and $AttRec_u$ are inserted into existing ones to let the structures grow and be synchronized with the underlying data, as illustrated in Fig. 7.

The update comprises two phases: updating 3D R-tree and AttRec. We focus on the second phase as the first is straightforward. We create $AttRec_u$ on R_u and insert both R_u and $AttRec_u$ into existing structures. R_u is inserted into R by recording the path P_u from the root node to the node where R_u is located. Node heights are increasingly numbered from the leaf to root level, guaranteeing that R and R_u are consistent. $AttRec_u$ is inserted into *AttRec* by appending tuples. In the end, we update the corresponding record for each node appearing in P_u. Precisely, the bitmap and time interval are updated according to attribute values contained in R_u. The update algorithm is given in Algorithm 3.

Algorithm 3. *Update*(*AttRec*, R, O_u)

```
1: create R_u on O_u and AttRec_u on R_u;
2: R ← R_u and get the update path P_u;
3: AttRec ← AttRec_u;
4: for all node in P_u do update the record in AttRec;
5: delete AttRec_u and P_u;
```

Our method has low latency which is important for on-line applications. The evaluation can be executed when R_u and $AttRec_u$ are created but not yet inserted into R and *AttRec*. When the system

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

(a) Standard trajectories

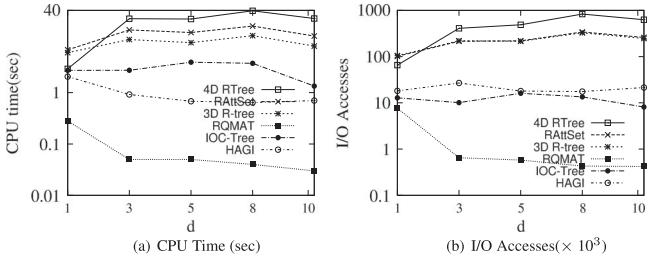
$X(Y)$	$30\% \cdot \Delta X\ (\Delta Y)$
T	$40\% \cdot \Delta T$
$\lvert Q_a \rvert$	$\{1, 2, \mathbf{3}, 4, 5\}$

(b) Query parameters

d	1	2	5	8	**10**
dom (Att)	[1, 20]	[1, 43]	[1, 74]	[1, 118]	[**1**, **151**]
Index storage (Mb)	2.64	3.17	3.59	3.59	3.3
dom (Att) (d = 1)	[1, 10]	[1, 20]	[1, 50]	[1,100]	[1, 500]

(c) Attribute settings

Fig. 8. Datasets and parameters.



(a) CPU Time (sec) (b) I/O Accesses($\times 10^3$)

Fig. 9. Effect of d ($|Q_a| = 3$ for $d > 1$).

detects that the insertion is not complete, we can separately search R, AttRec and R_u, $AttRec_u$ and then perform the union on the two parts to obtain the final answer.

7 Experimental Evaluation

The experimental evaluation is performed in an extensible database system SECONDO [17]. The implementation is programmed in C/C++. A standard PC (Intel(R) Core(TM) i7-4770 CPU, 3.4 GHz, 4 GB memory, 2 TB hard disk) running Suse Linux 13.1 (32 bits, kernel version 3.11.6) is used.

We use real taxi trajectories in Beijing, bought from a company DataTang [18]. Fig. 8a reports the statistics of standard trajectories. Due to the lack of real attributes, we generate the data by defining the number of attributes and the domain. The value of each attribute is randomly selected from the domain. Query parameters are reported in Fig. 8b. Q_{box} is randomly generated with the size X = 30 percent $\cdot$ ΔX and Y = 30 percent $\cdot$ ΔY, in which ΔX and ΔY are lengths of x and y dimensions, respectively. The time interval is 40 percent 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}. The number of query attributes $|Q_a| \in [1, d]$ is varied in which we set $|Q_a| = 3$ by default. Fig. 8c reports the setting for attributes, where the index storage cost includes 3D R-tree and *AttRec*.

7.1 Alternative Methods for RQMAT

(i) *3D R-tree*. This method uses the index of standard trajectories to find the objects intersecting Q_{box} and then performs the attribute evaluation sequentially.

(ii) 3D R-tree + Attribute Set (*RAttSet* for short). The idea is similar to IR-tree [13] employed in spatial keyword querying that arguments each R-tree node with a summary of keywords of spatial objects in the subtree. In each node, *RAttSet* stores the set of attribute values defined by trajectories in the subtree. We traverse the tree in a top-down fashion. For each accessed node, we make use of the attribute value set to prune the search space. The method is efficient when attribute values are location-dependent. One can group close objects in terms of locations and attribute values. However, attributes in our problem are not related to locations and grouping d value attributes according to the closeness is not a trivial task. Furthermore, the summarization information is able to tell whether a particular value exists or not, but cannot determine which entries (subtrees) contain the value. A sequential scan will be performed over all entries to load child nodes into main-memory for the evaluation, incurring both CPU and I/O costs. If several values are considered, we need to evaluate the nodes based on the intersection condition. However, the stored summarization information cannot make the decision.

(iii) *4D R-tree*. For each multi-attribute trajectory, we distribute attribute values $(a_1, \ldots, a_d)$ into d trajectories by decomposing $(a_1, \ldots, a_d)$ into d individual values and combining each value with the standard trajectory. This leads to d trajectories each of which contains only one attribute. For example, o_1(RED, BENZ) will become o_1'(RED) and o_1''(BENZ). Consequently, we will have four-dimensional data: location (x + y), time and a single value attribute. To answer the query, one combines the spatio-temporal window and the attribute value to form a high dimensional query box. Let $|Q_a| \leq d$ be the number of query attribute values. We need $|Q_a|$ query boxes. The trajectory is returned only if among its d trajectories there are $|Q_a|$ trajectories intersecting query boxes. The method enlarges the data set d times and does not achieve a good locality when grouping objects for a large d.

(iv) *IOC-Tree*. The paper [19] proposes a hybrid index consisting of an inverted index and a set of three-dimensional quadtrees termed *octrees* to answer spatial keyword range queries on trajectories. The method is extended to solve our problem by defining attributes as keywords. An inverted index for attribute values is developed in which each value is associated with an octree storing relevant trajectory points. Since our attributes are location-independent, all location points of the trajectory will be put into the octree. The significant increase of maintained trajectory points incurs the index overhead and deteriorates query performance.

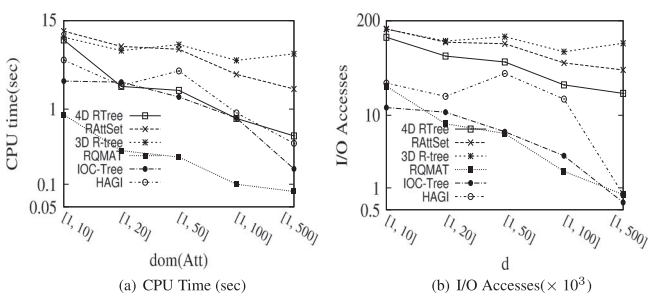
(v) *HAGI*. We also implement the hierarchical aggregate grid index (HAGI for short) in [9] and the corresponding query algorithm to solve our problem as that paper deals with nearest neighbor queries. Each node in HAGI maintains *min* and *max*attribute values of all objects stored in the subtree.

7.2 Performance

In the evaluation, CPU time and I/O accesses are used as performance metrics and the results are averaged over 20 runs.

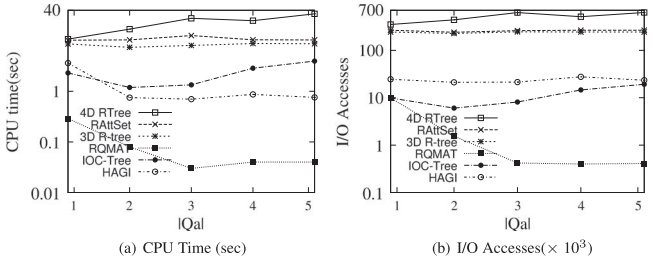
Scale d. The results are reported in Fig. 9. Our method significantly outperforms all alternative methods. When $d > 1$, our solution consistently achieves at least 8 times faster than alternative methods. Actually, the performance increases when d becomes larger for the reason that the attribute predicate has a good selectivity, leading to less nodes and trajectories being accessed for the spatio-temporal evaluation.

Scale dom(Att), d = 1. One can see from Fig. 10 that the performance of all methods improves in general when $|dom(Att)|$



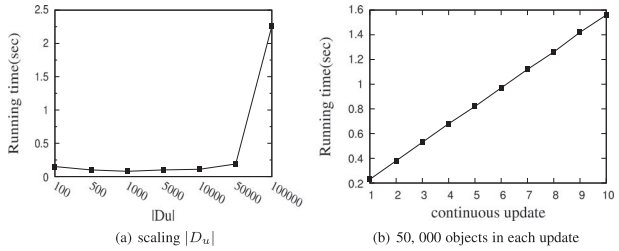
(a) CPU Time (sec) (b) I/O Accesses($\times 10^3$)

Fig. 10. Effect of dom(*Att*) (d = 1).



(a) CPU Time (sec) (b) I/O Accesses($\times 10^3$)

Fig. 11. Effect of $|Q_a|$ ($d = 10$).



(a) scaling $|D_u|$ (b) 50, 000 objects in each update

Fig. 12. The update cost.

becomes larger. The reason is, when $|dom(Att)|$ grows, the number of trajectories for each attribute value decreases and the attribute selectivity increases. Our method substantially outperforms alternative methods in all cases except that the I/O cost of IOC-Tree is close to ours in a few settings. This is because the IOC-Tree maintains a *octree* for each attribute value and the keyword-first-pruning strategy is efficient when $d = 1$.

Vary $|Q_a|$. This part demonstrates the impact of $|Q_a|$ on the performance. Fig. 11 shows that our method is an order of magnitude faster than other methods in most settings.

Update performance. We carry out two experiments: scaling $|D_u|$ and performing a series of updates. The results, as reported in Fig. 12, demonstrate that the time cost increases marginally when scaling $|D_u|$ except a sharp curve from 50,000 to 100,000. To evaluate the performance of continuous updates, we set $|D_u|$ = 50,000 for each update and continuously perform 10 times updates. The time cost is reported in an accumulated way.

8 Conclusions

We propose an efficient method to answer range queries on multi-attribute trajectories and demonstrate the method's performance advantage over alternative methods on large datasets.

Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities, NO. NS2017073.

References

[1] J. Xu, D. V. Kalashnikov, and S. Mehrotra, "Efficient summarization framework for multi-attribute uncertain data," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2014, pp. 421–432.
[2] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. W. Sadiq, and X. Zhou, "Approximate keyword search in semantic trajectory database," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2015, pp. 975–986.
[3] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang, "Towards efficient search for activity trajectories," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2013, pp. 230–241.
[4] Y. Tong, et al., "The simpler the better: A unified approach to predicting original taxi demands based on large-scale online platforms," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1653–1662.
[5] D. Pfoser and C. Jensen, "Novel approaches in query processing for moving object trajectories," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2000, pp. 395–406.
[6] P. C. Mauroux, E. Wu, and S. Madden, "TrajStore: An adaptive storage system for very large trajectory data sets," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2010, pp. 109–120.
[7] C. Zhang, J. Han, L. Shou, J. Lu, and T. F. L. Porta, "Splitter: Mining fine-grained sequential patterns in semantic trajectories," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 769–780, 2014.
[8] J. Xu, R. Güting, and Y. Zheng, "The TM-RTree: An index on generic moving objects for range queries," *GeoInformatica*, vol. 19, no. 3, pp. 487–524, 2015.
[9] Y. Su, Y. Wu, and A. L. P. Chen, "Monitoring heterogeneous nearest neighbors for moving objects considering location-independent attributes," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2007, pp. 300–312.
[10] R. H. Güting, F. Valdés, and M. Damiani, "Symbolic Trajectories," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 2, 2015, Art. no. 7.
[11] D. Zhang, Y. M. Chee, A. M., A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 688–699.
[12] I. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2008, pp. 656–665.
[13] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009.
[14] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel, "Text versus space: Efficient geo-search query processing," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, 2011, pp. 423–432.
[15] R. H. Güting, et al., "A foundation for representing and querying moving objects," *ACM Trans. Database Syst.*, vol. 25, no. 1, pp. 1–42, 2000.
[16] V. P. Chakka, A. Everspaugh, and J. M. Patel, "Indexing large trajectory data sets with SETI," in *Proc. Conf. Innovative Data Syst. Res.*, 2003.
[17] R. H. Güting, T. Behr, and C. Düntgen, "SECONDO: A platform for moving objects database research and for publishing and integrating research implementations," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 56–63, Jun. 2010.
[18] 2016. [Online]. Available: http://factory.datatang.com/en/
[19] Y. Han, L. Wang, Y. Zhang, W. Zhang, and X. Lin, "Spatial keyword range search on trajectories," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2015, pp. 223–240.