The TM-RTree: an index on generic moving objects for range queries

Jianqiu Xu · Ralf Hartmut Güting · Yu Zheng

Abstract Existing works on moving objects mainly focus on a single environment such as free space and road network, and do not investigate the complete trip for humans who can pass several environments, e.g., road network, pavement areas, indoor. In this paper, we consider multiple environments and study moving objects with different transportation modes, also called generic moving objects. We aim to answer a new class of queries supporting three kinds of conditions: temporal, spatial, and transportation modes. To efficiently provide the result, we propose an index structure called TM-RTree, which takes into account the feature of moving objects in different environments and has the capability of managing objects on not only temporal and spatial data but also transportation modes. This property is not maintained by existing indices for moving objects. Different cases on transportation modes are supported. Correspondingly, several algorithms are developed. The TM-RTree and related algorithms are developed in a real DBMS to have a practical and solid result for applications. In the experiment, we conduct the performance evaluation using extensive datasets and compare the proposed technique with the other two competitors, demonstrating the efficiency and significant superiority of our solution in various settings.

Keywords Transportation modes · Range queries · Index structure

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

R. H. Güting
FernUniversität in Hagen, Hagen, Germany
e-mail: rhg@fernuni-hagen.de

Y. Zheng
Microsoft Research Asia, Beijing, China
e-mail: yuzheng@microsoft.com

1 Introduction

Moving objects databases have been extensively studied in the last decade due to their wide applications such as location-based services and transportation networks. Recently, the topic of moving objects with multiple transportation modes receives increasing attention in the literature [33, 35]. Some researchers build an inference model to discover outdoor transportation modes from raw GPS data (inactive for indoor) such as walking, cycling and bus, e.g., Microsoft's project Geolife [46, 47]. In a transportation system, advanced trip planning [8] is provided so that the user can specify different modes and constraints for the path. For example, find a trip by bus and walk fulfilling the condition: (i) two bus transfers in maximum, and (ii) the walking distance is less than 300m.

In this context, different environments are considered for moving objects such as road network, public transportation system and pavement areas. We target a complete trajectory that contains not only the location data but also motion modes. In the meantime, pieces of movements with different modes are distinguished and precisely represented. Consider two example trips:

(1)   Bobby moved from his office room to the building exit, then walked to the parking place, and from there he went home by car.
(2)   Bobby moved from his office room to the building exit, then walked to a fast food shop for lunch, and came back to the office room.

Throughout the paper, we also use the term generic moving objects to refer to those moving objects. Figure 1 depicts the two trips. Different line types are utilized to distinguish movements with different transportation modes. The two generic moving objects can be abstractly described as follows:

(1) mo_1: <*Indoor*, *Walk*, *Car*>;
(2) mo_2: <*Indoor*, *Walk*, *Indoor*>.

Instead of traveling by car, people can also choose the public transportation system and then we can have a trip like <*Indoor*, *Walk*, *Bus*> or <*Indoor*, *Walk*, *Taxi*>. Most prior research concentrated on a single environment such as free space [20, 34], road network
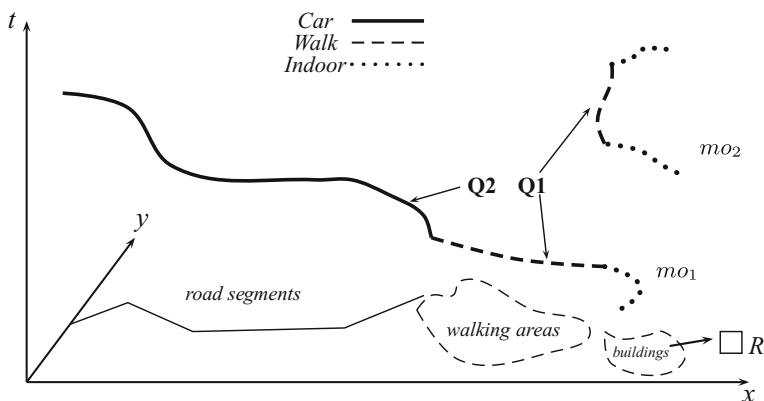

Fig. 1 Moving objects with multiple transportation modes

[19, 21] and indoor [22], in which different data types and operators are proposed. But in the real world, the trajectories for humans can cover multiple environments. The database system should manage the complete trip instead of a subtrip limited to one environment. Complete movements have to be maintained in a consistent framework in terms of accurate locations and transportation modes in various environments. Fully recognizing motion modes with location data can enrich the mobility with informative and context knowledge. This is substantially meaningful to analyze travel patterns and behavior of mobile users. In particular, by analyzing the historical movement, routes with high passenger flow can be discovered. Therefore, the schedule in a public transportation system can be adjusted to improve the traffic. In a recommendation or advanced service system, one could discover friends based on location history [48]. This is to understand the correlation between people and locations, and recommendation potential friends in a GIS community by measuring the similarity of trajectories. The result will benefit from knowing precise transportation modes.

This imposes new challenges on database systems regarding the data management as well as efficient query processing. The work of this paper does not start from scratch but is based on [43]. There, a data model is presented to represent moving objects in different environments, in which both outdoor and indoor movements are uniformly treated.

In this paper, we focus on moving objects with multiple transportation modes and provide an efficient method to answer a novel category of queries, called RQTM (range queries with transportation modes). Standard range query is one of the most studied problems in moving objects databases, e.g., [28, 32]. The query contains a time window $Q.t$ and a spatial window (a bounding box) $Q.s$, and returns all objects that pass through $Q.s$ during $Q.t$. However, existing techniques only handle a single environment without considering the complete trip such as mo_1 and mo_2. As a result, transportation modes are not processed. In contrast, we manage moving objects in different environments and aim to retrieve a list of trajectories that fulfill not only temporal and spatial conditions but also transportation modes. Consider two example queries below.

- **Q1:** Find all people walking through the specified area during [10am, 2pm] on Saturday.
- **Q2:** Given a university area R, find all people who pass through R on Friday with the following sequence of transportation modes: $<Indoor, Walk, Car>$.

Besides spatial and temporal parameters, one can issue queries on a particular transportation mode (**Q1**) and even imposes a particular order on a set of modes (**Q2**). In the example, **Q1** gets mo_1 and mo_2, assuming that the query region covers the walking area shown in the figure. Suppose that buildings in Fig. 1 include R, **Q2** only retrieves mo_2 because mo_1 does not fulfill the condition on transportation modes. In comparison with previous work, RQTM has two noticeable differences. First, a complete trip including several environments is represented and managed in a database system. Existing works focus on one environment and only deal with a subtrip (e.g., the movement in a road network). Consequently, RQTM is not supported. Second, besides time and spatial parameters, RQTM contains an additional condition on transportation modes, which have different cases. **Q1** includes one mode and **Q2** specifies a sequence of transportation modes. RQTM is useful and important to understand users' mobility, and the query result can be used to discover classical trajectories and provide advanced trip plannings such as multiple transportation modes.

The brute-force approach to answering RQTM is to access each trajectory and check spatial, temporal and transportation modes to see if they fulfill the query condition. Obviously, the performance is not acceptable for large datasets. To efficiently answer RQTM,

the key is to reduce the number of objects to be accessed, i.e., minimize the portion of the dataset processed. Thus, an index structure for generic moving objects is essentially needed. Despite the fact that there is a large amount of indices proposed for trajectories in the literature [9, 10, 23, 31], they only tackle the problem in one environment without addressing transportation modes. However, the property of managing movements with transportation modes is highly desired because the query condition in RQTM contains these pieces of information. This calls for the development of a novel index that is equipped with the capability of distinguishing movements in different environments and accessing them efficiently when a complete trip is handled. To solve the problem, we develop an index structure called TM-RTree (Transportation Mode RTree), which is able to group pieces of movements on temporal, spatial and transportation modes. In other words, the three kinds of data are managed in an integrated way by the TM-RTree.

Based on the proposed index structure, several algorithms are developed for different query conditions, e.g., a single mode (**Q1**) and a sequence of modes (**Q2**). In general, the query procedure is composed of two steps: *filter* and *refinement*. The former traverses the TM-RTree to find candidate objects and the latter performs the searching step to get the precise value. More importantly, we propose a method to let the index be capable of maintaining the movement with not only a single mode but also a pair of modes following the time order, leading to a more efficient method for answering the query for a sequence of modes. The TM-RTree and related algorithms are implemented in an extensible and open source database system SECONDO to have a practical result for real applications. We perform the experimental evaluation using extensive datasets to demonstrate the efficiency and effectiveness of the proposed technique. The contribution of this paper is summarized as follows:

- We formulate the problem of RQTM, supporting three kinds of queries on transportation modes: a single mode, multiple modes and a sequence of modes.
- An index structure called TM-RTree is developed in a system to efficiently manage generic moving objects. We present how to derive the input data for a TM-RTree and the procedure of building such an index.
- Employing the TM-RTree, several algorithms are proposed to answer different query conditions on transportation modes. In particular, we propose a method that significantly improves the performance for answering a sequence of modes. On average, our method is 6 times faster than the other methods.
- We carry out extensive experiments to confirm the efficiency and effectiveness of the proposed techniques. The results demonstrate that our method substantially outperforms the other three access methods, four or five times faster on average.

The rest of the paper is organized as follows: The related work is reviewed in Section 2. We define the problem in Section 3. The TM-RTree is introduced in Section 4. Query algorithms are presented in Sections 5, 6 and 7. Experimental evaluation is performed in Sections 8 and 9 concludes the paper.

2 Related work

2.1 Transportation modes

Recently, researchers started to investigate the detection of transportation modes from GPS data based on mobile phone devices. In Microsoft's project GeoLife, the works [46, 47] aim

to discover and infer transportation modes from raw GPS trajectory data. By mining multiple users' location histories, one can discover the most interesting locations, classical travel sequences and travel experts in a given geo-spatial region. The work in [33] is to determine the transportation modes of an individual outside where five transportation modes are considered: stationary, walking, running, biking and in motorized transport (car, bus, train). Only using GPS information might reduce the detection accuracy, thus the underlying transportation networks are utilized to improve the result. An approach to inferring a user's mode is proposed in [35] where the method is based on the GPS sensor from the mobile device as well as the knowledge of the underlying transportation network, e.g., bus stops and spatial rails. A trip analysis system is proposed in [26] to support traffic operation planning and transit system design. Such a system, consisting of mobile apps and a centralized analyzer, automatically identifies the travel mode and purpose of the trips sensed by mobile devices. Travel modes including driving, taking bus or on foot are detected by sensing the users' movement using embedded GPS and accelerometer.

Advanced trip plannings with multiple transportation modes are studied. A data model is proposed in [8] to provide a trip consisting of several transportation modes, e.g., <*Walk*, *Bus*, *Train*>. The user can also specify the constraint on a specific mode such as less than two bus transfers or the length of the walking path is 300 meters in maximum. An interesting query is considered in [5] that computes *isochrones* in multimodal and schedule-based transport networks. The goal is to find a set of points on a road network, and from them a specific point of interest can be reached within a given time span. Two transportation modes are supported: *Walk* and *Bus*.

Combining both outdoor and indoor movements, a data model is proposed in [43] to model moving objects with different transportation modes. Accurate locations in multiple environments are represented, including free space, road network, pavement areas, public transportation system, and indoor. A general location representation is designed and several data types are defined. A group of queries is formulated by an SQL-like language using proposed operators in a relational interface. However, that work primarily focused on data modeling, while range queries considering three factors (spatial, temporal and transportation modes) are not processed and precisely defined. Additionally, there is no index structure for generic moving objects.

2.2 Range queries and moving objects indices

Range queries [17, 24, 37, 39], representing the foundation queries in moving objects databases, have been extensively investigated. Given a query window consisting of a spatial region $Q.s$ and a time interval $Q.t$, the result is a set of moving objects that are located in $Q.s$ during $Q.t$. In general, $Q.t$ can also be a time instant. According to environments, the definition of $Q.s$ is different. In free space, a 2D region or a rectangle is used. In a road network, $Q.s$ is a set of road sections, each of which represents a set of disjoint and ordered intervals on a road [13, 32]. Besides range monitoring in outdoor space, techniques for indoor space are also proposed due to different distance functions and positioning technologies [27, 44]. $Q.s$ in indoor space refers to the activation range of a device such as RFID or Bluetooth, which detects and reports the observed objects at a high sampling rate. The paper [40] deals with spatio-temporal range queries over trajectory data, but does not consider transportation modes.

In the last decade, a substantial amount of index structures have been proposed to efficiently access trajectories. A good survey on trajectory indexing and retrieval is given in [25, 49]. Depending on the environment, indices can be classified into three categories: (1)

free space; (2) road network; and (3) indoor. In free space, two variations of the R-Tree for polyline indexing are suggested in [31], TB-Tree and STR-Tree, assuming that the motion is piecewise linear, where the TB-tree is to bundle segments from the same trajectory into leaf nodes. The MV3R [36] is a structure combining a standard R-Tree and a variant of the partially persistent R-Tree, supporting both time-stamp and time-interval queries. A two-level indexing structure called SETI is proposed in [10], where the structure decouples the indexing of spatial and temporal dimensions. The paper [29] offers an indexing technique capable of accurately capturing the past, present, and (near) future positions of moving objects. Practically, objects usually move on a pre-defined set of paths as specified by the underlying network, thus a couple of indices for road network are proposed. Since the movement is constrained by the underlying road network, the combination of two-level R-Trees is employed by [13, 16, 30] where the first level is for roads and the second is for trajectories. The paper [32] proposed an index called T-PARINET, combining graph partitioning and a set of composite B^+-tree local indices for trajectory data flows. Recently, some index structures for moving objects in a symbolic indoor space [23, 27] are also developed to support range and nearest neighbor queries over indoor objects.

However, previous queries do not consider the condition on transportation modes and those techniques deal with moving objects in one environment, i.e., a portion of the complete movement is managed. The proposed indices only have the capability of grouping objects on temporal and spatial dimensions, but do not manage transportation modes. As a result, existing indices can not prune objects that do not fulfill the condition on transportation modes, degrading the query performance for generic moving objects.

2.3 Spatial keywords search

Given a dataset O, each object $o \in O$ is represented by a pair (*loc*, *text*) where *o.loc* is the spatial location and *o.text* is the textual description showing the property of such a location, e.g., restaurant, bank. A spatial keywords query $q = $ (*loc*, *text*), consisting of a query area and a set of keywords, returns a list of objects in O such that their distances to *q.loc* are in ascending order and the descriptions contain the query keywords *q.text*. To efficiently answer the query, several indices [11, 14, 41, 45] are proposed to manage both spatial and textual data in an integrated way. The basic idea is to perform the query on a 2D R-Tree extended with inverted files. In the IR-Tree [11], each leaf node contains a pointer to an inverted file with text descriptions of the objects stored in the node. Each non-leaf node has an identifier for a pseudo text description that is the union of all text descriptions in the entries of the child nodes. Similarly, the entry of a non-leaf or leaf node in the IR^2-Tree [14] is extended to have an item showing the signature of the subtree or an object. The signature is to denote the textual content of all spatial objects in the subtree rooted at the node. The *bR**-*tree* [45] summarizes the keywords and stores the keyword *MBR* in the node to set up more powerful pruning rules. Spatial keyword searching in trajectory databases is studied in [12] in which trajectories contain text descriptions. The user gives a query location and a set of keywords, and the result is a set of trajectories that cover the keywords and have the shortest match distance.

We compare spatial keywords searching with our problem and point out the differences. First, spatial keywords queries deal with spatial data and additional text information, i.e., the two kinds of data are represented separately. In contrast, we seamlessly integrate transportation modes into moving objects and have a compact data representation. That is, the data type for generic moving objects includes the transportation mode information. As a

Table 1 Notations for environments

I_{rn}	Road Network
I_{rbo}	Region-based Outdoor
I_{bn}	Bus Network
I_{mn}	Metro Network
I_{indoor}	Indoor

result, all data are managed by the index without additional structures while spatial keywords searching needs an inverted file with text descriptions. Second, our queries include several cases on transportation modes. The user can request the query on a certain mode or multiple modes. Additionally, a particular order can be imposed on a set of modes. In spatial keywords queries, there is no particular order on keywords. Third, Our index has the capability to group objects on transportation modes and manage a pair of modes for two continuous pieces of movements (keeping the time order). This property is not maintained by the proposed indices for spatial keywords search.

3 Problem definition

3.1 Data representation

The problem setting aims to concisely capture the context of the data and queries. The core content is the representation of moving objects with multiple transportation modes. First, established environments are listed in Table 1 and we define a notation for each of them.[1]

Each environment consists of a set of geographic objects (GEOBs) representing available places for moving objects. For example, streets and roads are for I_{rn}, and pavement areas represented by polygons are for I_{rbo}. In the public transportation system, I_{bn} comprises bus routes, bus stops and moving buses. Several data types are defined to represent GEOBs, e.g., line and region. In addition, each environment is related to certain transportation modes, e.g., *Car* for I_{rn}. We summarize all possible values in Def. 3.1.

Definition 3.1 Transportation Modes
   $TM = \{Car, Bus, Walk, Indoor, Metro, Taxi, Bike\}$

Due to different environment characteristics such as outdoor and indoor, network-constrained and obstructed space, a general approach to representing the precise location in all cases is proposed. The capability is achieved by means of referencing representation and the method is as follows.

Definition 3.2 Generic Location
   $D_{\underline{gloc}} = \{(oid, (x, y)) | oid \in D_{\underline{int}}, x, y \in D_{\underline{real}}\}$

---

[1]we model the overall pedestrian area in a city as a large polygon with obstacles inside, denoting areas covered by buildings and roads for vehicles.

The location is represented by two parts: (1) an identifier referencing to a GEOB; and (2) the relative location according to that object. This representation has different semantics according to the environment feature. For example, in a road network, *oid* is a road identifier and $(x, \perp)$ records the relative location on the road. Given a location in I_{rbo}, we let *oid* map to a polygon and (x, y) represent the local position inside the polygon. For the location of a bus/metro passenger, one can simply use *oid* to reference the bus/metro that the traveler takes.

Then, the location of a moving object is represented by a function f_{gloc}: $D_{instant} \rightarrow D_{gloc}$. In this paper, we focus on the complete historical movement. Using the method of *sliced representation* [15], a generic moving object is represented by a sequence of temporal units ordered by time, that is $mo =< u_1, u_2, ..., u_n >$. Each unit defines the movement during a time interval as well as the transportation mode. In detail, we have

$$u_i = (i, gl_1, gl_2, m)(gl_1, gl_2 \in D_{gloc} \wedge gl_1.oid = gl_2.oid, m \in \text{TM}),$$

where i denotes the time interval, gl_1, gl_2 are the start and end locations, respectively, and m is the transportation mode. We assume that the object moves linearly during i so that the positions between gl_1 and gl_2 are calculated by a linear function. Using the example trip $mo_1 =< Indoor, Walk, Car >$ in the introduction, the units record ids for (1) rooms; (2) pavement areas; (3) roads. The actual locations are identified by gl_1 and gl_2.

3.2 RQTM

To compare locations of different environments in a consistent system, we define an operator **freespace**: $D_{gloc} \rightarrow D_{point}$ that maps a general location to free space. The meaning of such an operator depends on the environment, explained in the following. Given a location $(oid, (x, \perp))$ in I_{rn}, we map such a location to free space by a road line with the identifier *oid* and the relative position x on that road. Similarly, the position on a bus route is also mapped to free space. In I_{rbo}, we transform the local position (x, y) in the referenced polygon (with the *oid*) to the global space. For an indoor location, we omit the height value above the ground floor and project the location into 2D free space.

Let $M(mo, T) =< [m_1, m_2, ..., m_k] > (m_i \in \text{TM}, i \in [1, k])$ return a sequence of transportation modes for *mo* during a time interval T. The result can be one mode or several modes. In the context of generic moving objects, the definition of range queries is given in the following.

Definition 3.3 The query is described by $Q(t, s, m)$ where t defines a time interval, s denotes a 2D spatial bounding box, and m represents a set of transportation modes. The result is a set of trajectories $\{mo_1, mo_2, ..., mo_n\}$ where for each $mo_i (i \in [1, n])$:

(i) $|Q.m| = 1$, then $\exists T \subseteq Q.t$ such that $M(mo_i, T) = Q.m$ and $\forall t' \in T$: $\mathbf{freespace}(f_{gloc}(t')) \in Q.s$;

(ii) $|Q.m| \geq 2$, then $\forall m_i \in Q.m, \exists T \subseteq Q.t$ such that $M(mo_i, T) = m_i$ and $\forall t' \in T$: $\mathbf{freespace}(f_{gloc}(t')) \in Q.s$;

(iii) $Q.m =< [m_1, m_2, ..., m_l] >$, then exists $T_1 < T_2 < ... < T_l$ such that $\forall i \in [1, l]$: $T_i \subseteq Q.t \wedge M(mo_i, T_1) = m_1, M(mo_i, T_2) = m_2, ..., M(mo_i, T_l) = m_l \wedge T_i, T_{i+1}$ are continuous, and $\forall t' \in T_i : \mathbf{freespace}(f_{gloc}[(t')) \in Q.s$.

Since the meanings of spatial and temporal parameters are straightforward, we give more description about $Q.m$ as several cases are involved. This parameter can be either one mode

Table 2 Summary of notations

Notation	Description
$Q.m$	query mode
$Q.t$	query time
$Q.s$	query spatial box
mo, mo_i	a moving object
m	a transportation mode

(case (i)) or several modes. If $Q.m$ represents several modes, they can be specified by a certain order (case (iii)) or without any order (case (ii)). We show examples for each case as below.[2]

- Who rode a bicycle through the central park on Sunday?

    - $Q.t$: Sunday
    - $Q.s$: park area
    - $Q.m$: *Bike*

- Find all people passing through the city center by bus and walking on Saturday.

    - $Q.t$: Saturday
    - $Q.s$: city center
    - $Q.m$: $\{Bus, Walk\}$

- Who arrived by taxi at the university on Friday?

    - $Q.t$: Friday
    - $Q.s$: university area
    - $Q.m$: *<Taxi, Walk, Indoor>*

Compared with traditional range queries, RQTM contains one more parameter on transportation modes, the information of which is included by the data type for moving objects. Therefore, the query performance can be improved if the index can manage the data. Based on this motivation, we present the TM-RTree in the next section. For clarity, we summarize main notations used in the paper in Table 2.

4 The TM-RTree

4.1 Input data

To efficiently answer RQTM, we develop an index based on a 3D R-tree [38]. The key idea of R-tree is to group nearby objects and represent them with their minimum bounding boxes in the next higher level of the tree. We intend to group pieces of movements with the locality on three factors: temporal, spatial and mode. The proposed structure TM-RTree is built on pieces of movements, each of which represents a subtrip of a generic moving object and only includes one transportation mode. To achieve the goal, we create a set of

---

[2]more queries see Appendix A

so-called *movement tuples* for each generic moving object and repeat the same procedure for all moving objects. Afterwards, the index is built on these movement tuples.

Definition 4.1 Movement Tuples

A movement tuple mt is derived from a subtrip of mo, represented by a three-tuple $(traj\_id, box, m)$ where $traj\_id$ ($\in D_{int}$) records the trajectory identifier, box stores a 3D box built on temporal and spatial data, and m ($\in TM$) refers to the transportation mode. The subtrip includes only one mode.

We present how to retrieve movement tuples in the following. Recalling the representation of mo, a unit u_i ($\in mo$) records a subtrip where the location is represented by referencing to a GEOB. One can simply build a movement tuple for each u_i. However, this method results in the spatial locality of tree nodes being not well maintained as the bounding box sizes for different movement tuples have a large deviation. The spatial range for each u_i depends on the referenced GEOB. For example, u_i can be a bus trip, a car movement or an indoor trip. Some are in a small spatial range such as indoor trips or a short distance walking, but others are in a large scale such as traveling on a long road or a bus trip.

To solve the problem, we develop a method to (i) minimize the deviation of spatial range (good locality in TM-RTree nodes) and (ii) reduce the quantity of produced movement tuples. Basically, the procedure is to access each unit in mo, collect one or several units as one subtrip and build a movement tuple on the subtrip. Depending on transportation modes (environments), the methods are different. We use $U(mo, m)$ to denote a set of continuous units from mo with the mode m ($\in TM$) and $Len(U(mo, m))$ to show the path length of such a subtrip, respectively.

- I_{rn}: In this environment, we collect a set of units as one group $U(mo, m)(m \in \{Car, Taxi, Bike\})$ fulfilling the condition that $Len(U(mo, m))$ does not exceed a threshold l, e.g., 1000m. Then, a movement tuple is created on $U(mo, m)$.
- I_{rbo}: Using the same threshold l as I_{rn}, we directly build a movement tuple on $U(mo, Walk)$ if $Len(U(mo, Walk)) < l$. Otherwise, a new movement tuple will be created.
- I_{bn} (I_{mn}): Using the bus network, $U(mo, Bus)$ references to a unit recording the bus id as well as the start and end bus stops of such a trip. We split the movement into several pieces, each of which represents the trip between two *adjacent* stops and is used to create a movement tuple.
- I_{indoor}: Since the spatial range of a building is in a small scale, we create a movement tuple on $U(mo, Indoor)$, which represents a complete indoor trip such as first from the office room to the corridor and then to the staircase. We perform the union on those pieces and get the movement tuple.

A movement tuple manages temporal and spatial data as well as the transportation mode. The value denotes either a piece of the complete trip or the union of several pieces . We apply the procedure for all generic moving objects and get a set of movement tuples, denoted by MT, to be the input for a TM-RTree. Using the example trips mo_1 and mo_2, Fig. 2 shows the resulting movement tuples. To be concise, we only show the mode in each mt and omit the other information. For mo_1, we get seven movement tuples: $\{mt_1(Indoor), mt_2(Walk), ..., mt_7(Car)\}$. For mo_2, we have $\{mt_1'(Indoor), mt_2'(Walk), mt_3'(Indoor)\}$.

In order to have a good locality for TM-RTree nodes, before building the index we process MT as follows. First, the time range for all movement tuples is divided into a set of intervals and each mt is set an id for its time interval. For example, if all moving objects
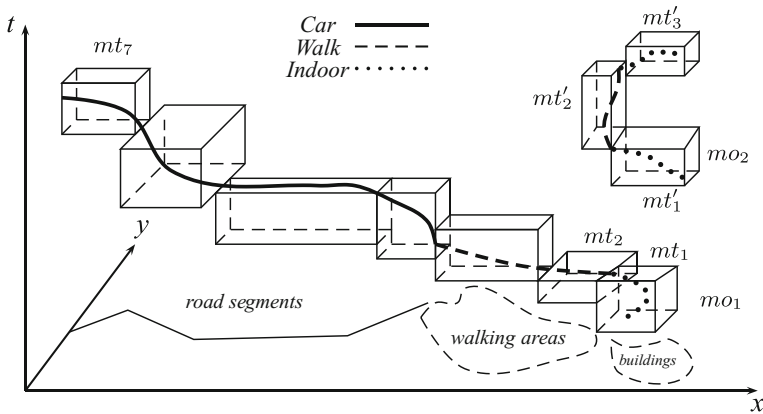
Fig. 2 *MT* on mo_1 and mo_2

represent one week movement, one can set 12 (or 24) hours as one interval. Second, the space is partitioned into a set of equal size cells represented by rectangles, and each *mt* is assigned two values *cellx* and *celly* to show *x* and *y* positions of the cell where *mt* is located. Third, we sort movement tuples in the following order: (1) time interval id; (2) *cellx*, *celly*; (3) *mt.m*, and these sorted movement tuples are the input for a TM-RTree.[3]

4.2 TM-RTree nodes

In addition to spatial and temporal boxes, each node in a TM-RTree maintains the information of transportation modes. Thus, a node has the form:

1. *MBR*: the bounding box that contains temporal and spatial data in the subtree below.
2. *ObjPtr*: an array of entries pointing to child nodes or objects.
3. *M*: an integer indicating transportation modes represented by the subtree.

In a TM-RTree, entries in non-leaf nodes are pointers to subtrees. In a leaf node, an entry references to a movement tuple. To preserve the proximity of transportation modes, we only store movement tuples with the same mode in a leaf node. Correspondingly, *M* indicates one transportation mode in such a node. In a non-leaf node, the value of *M* is the union of modes contained by its sub-trees, denoting either a single mode (all son nodes have the same mode) or a set of modes. The method of keeping modes in nodes benefits the query processing because nodes that do not contain qualified transportation modes can be pruned when traversing the index. In consequence, the following lemma is used.

Lemma 1 *Given a node N in a TM-RTree, let **tm**(N.M) ($\subseteq$ TM) return the set of modes. Then, **tm**(N.M) $\cap Q.m = \varnothing \Rightarrow N$ can be pruned.*

---

[3]In the implementation, a movement tuple is developed to be a relational tuple containing three attributes (*traj_id*, *box*, *m*). In order to efficiently access the data in the future, we combine each movement tuple with its corresponding subtrip in one relational tuple. The sub trip is represented by a moving object.

Table 3 Bit values for modes

Car: 0	Bus: 1	Walk:2	Indoor:3	Metro:4	Taxi:5	Bike: 6

Next, we introduce in detail how transportation modes are represented by M. A bit in M is set for one mode, which is assigned a value as an index to locate the bit. We mark the bit *true* indicating the existence of such a mode and *false* for nonexistence. Suppose that the least significant (right-most) seven bits are used for the modes in Def. 3.1. We assign 0 for *Car* as the bit index, 1 for *Bus*, 2 for *Walk* and so on. Table 3 shows the bit value for each mode. For example, mo_1 containing modes $\{Indoor, Walk, Car\}$ defines the value 11 (binary 0001101). This representation incurs little storage overhead and results in a fast method of checking transportation modes due to the relatively high speed of bitmap operations. Figure 3 shows an example of TM-RTree nodes using movement tuples in Fig. 2.

4.3 Insertion algorithm

We build the TM-RTree by a bulkload method [6, 7], which works as follows. The index retrieves a set of sorted movement tuples and puts them into a leaf node until one of the following things occurs:

- the leaf node is full;
- the coming movement tuple contains a new transportation mode.

If the case happens, the current leaf node is inserted into a node at the higher level and a new leaf node will be created. We do not consider the distance deviation between the new box and the current bounding box of the leaf node when splitting a leaf node. The reason is, when creating movement tuples, we already process the spatial and time ranges for size controlling by defining the time interval for sorting and the threshold for splitting the movement, see Section 4.1.

Following this procedure, the index is built in a bottm-up way. The method of inserting movement tuples into the TM-RTree is shown in Algorithm 1. An important aspect is that
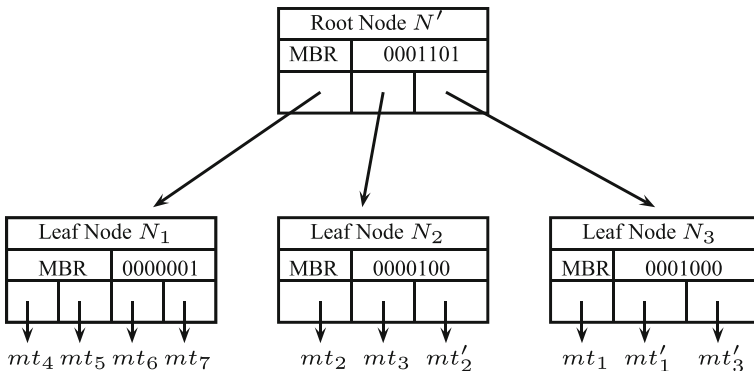

Fig. 3 Example of TM-RTree nodes

only movement tuples with the same transportation mode are maintained in a leaf node, keeping the proximity of objects.

Algorithm 1: *InsertMTuple(mt)*

let N_L be the leaf node to insert mt;
if N_L *is not full and* $\boldsymbol{tm}(N_L.M) = mt.m$ **then**
$\quad$ $N_L[i] \rightarrow mt$;
$\quad$ $N_L[i\text{++}]$;
else
$\quad$ *insert N_L into its father node*;
$\quad$ let N_L' be a new leaf node;
$\quad$ $N_L'.M \leftarrow mt.m$;
$\quad$ $N_L'[0] \rightarrow mt$;

After inserting all movement tuples, we need to update the mode value, especially non-leaf nodes. The reason is some operations such as splitting and adjusting are performed on these nodes when building the TM-RTree and modes are changed after that. Algorithm 2 illustrates the pseudo code. The procedure is straightforward for a leaf node. For a non-leaf node, the value depends on its son nodes. We iteratively call the same procedure for each son node and perform the union to update *M*. The mode value is updated in a bottom-up manner, and is returned when all son nodes are processed. To update the complete tree, we call the function *UpdateMode* by taking the root node as input. Suppose that there are totally *n* nodes in a TM-RTree, the time complexity of *UpdateMode* is $O(n)$ as each node is visited once.

Algorithm 2: *UpdateMode(node_id)*

let N be the node by *node_id*;
reset($N.M$);
if N *is a leaf node* **then**
$\quad$ **return** $N.M$;
else
$\quad$ **foreach** $i \in N.entrycount$ **do**
$\quad\quad$ $m = UpdateMode(N[i].pointer)$;
$\quad\quad$ $N.M \leftarrow N.M \cup m$;
$\quad$ **return** $N.M$;

---

5 Single transportation mode

We first consider a single mode in the query condition, case (i) in Def. 3.3. Applying the TM-RTree, the query procedure follows a two-phase approach: filter and refinement. The filter step accesses the TM-RTree to find candidate objects. In the refinement step, the accurate value for each object is checked and a set of trajectory ids is returned as the result. We give the pseudo code in Algorithm 3.

Algorithm 3: *RQSin(TM-RTree, Q, MT)*

$CL = RQSin\_Filter(TM\text{-}RTree, Q)$;
let *Tr* be a binary tree with the key *traj_id*;
foreach $cl_i \in CL$ **do**
    get *mt from MT* by cl_i;
    if *SubTrip(mt) fulfills the condition on Q.t*
    and Q.s **then**
        if $mt.traj\_id$ *does not exist in Tr* **then**
            insert $mt.tr\_id$ into *Tr*;

let *RL* be the result list;
foreach $n_i \in Tr$ **do**
    $RL \leftarrow key[n_i]$;
return *RL*;

In the beginning, the filter step (Algorithm 4) is called. We create a 3D query box based on $Q.t$ and $Q.s$. Then, starting from the root node, we traverse the TM-RTree in breadth-first search to examine nodes according to the query condition.

Algorithm 4: *RQSin_Filter(TM-RTree, Q)*

$q\_box \leftarrow create3Dbox(Q.t, Q.s)$;
$CL \leftarrow \varnothing$;
$L \leftarrow TM\text{-}RTree.RootId()$;
while *L is not empty* **do**
    $N_i = GetNode(TM\text{-}RTree, L.head())$;
    if $Q.m \in \boldsymbol{tm}(N_i.M)$ **then**
        if N_i *is a leaf node* **then**
            foreach $j \in N_i.entrycount$ **do**
                if $N_i[j].box$ *intersects* q_box **then**
                    $CL \leftarrow N_i[j].pointer$;

        else
            foreach $j \in N_i.entrycount$ **do**
                if $N_i[j].box$ *intersects* q_box **then**
                    $L \leftarrow N_i[j].pointer$;

return *CL*;

A list *L* is used to store TM-RTree nodes, initialized by the root node. Given a node N_i, $\textbf{tm}(N_i.M)$ ($\subseteq TM$) returns the transportation modes that the node represents (Lemma 1 in Section 4.2). If $Q.m \in \textbf{tm}(N_i.M)$, we open such a node and proceed to check spatial and temporal conditions. For each entry in N_i, we check whether the box of the child node intersects the query box. If N_i is a leaf node (only contains one mode), we insert qualified pointers (tuple ids for movement tuples) into the candidate list *CL*. If N_i is a non-leaf node, its child nodes will be inserted into *L* for further consideration if their boxes intersect the query box. During the procedure, nodes that do not fulfill conditions on spatial, temporal and modes are pruned.

In the refinement step, we use a binary tree *Tr* with the key *traj_id* to hold the result. The notation *SubTrip(mt)* is used to denote the subtrip that a movement tuple corresponds to. For each candidate from *CL*, we get the movement tuple and check the precise value for spatial and temporal data. The transportation mode is already determined in the filter step. If *SubTrip(mt)* satisfies spatial and temporal conditions, we insert *mt.traj_id* into *Tr* if such an object does not exist in *Tr*. Otherwise, *mt* is ignored. A moving object includes a set of movement tuples and several of them can fulfill the query condition. For example, assuming $Q.m = Car$, $\{mt_4, mt_5, mt_6, mt_7\}$ from mo_1 (Fig. 2) are all located in $Q.s$ during $Q.t$. It is not necessary to access all of them if mo_1 already satisfies the query condition. Finally, we traverse *Tr* to collect the result.

6 Multiple modes

The goal is to find a list of trajectories, each of which contains all transportation modes requested by $Q.m$. We also apply the filter and refinement method, and give the algorithm in Algorithm 5. Since the filter step is similar as the procedure for answering a single mode, we omit the algorithm *RQMul_Filter(TM-RTree, Q)*. The only exception is, as $Q.m$ includes multiple values, the filter step here needs to examine the existence of each mode contained by $Q.m$. If a TM-RTree node includes one mode in $Q.m$, such a node has to be further considered.

Algorithm 5: RQMul(*TM-RTree*, *Q*, *MT*)

$CL = RQMul\_Filter(TM\text{-}RTree, Q)$;
let *Tr* be a binary tree with the node form (*traj_id*, *modebits*);
foreach $cl_i \in CL$ **do**
    get *mt* from *MT* by cl_i;
    if *SubTrip(mt) fulfills the condition on Q.t and Q.s* **then**
        if *mt.traj_id does not exist in Tr* **then**
            create a node (*mt.traj_id*, *mt.m*) and insert into *Tr*;
        else
            $n = GetNode(Tr, mt.traj\_id)$;
            if $|bitmap[n]| < |Q.m|$ **then**
                mark the bit in n by *mt.m*;

let *RL* be the result list;
foreach $n_i \in Tr$ **do**
    if $|bitmap[n_i]| = |Q.m|$ **then**
        $RL \leftarrow key[n_i]$;
return *RL*;

The refinement step is as follows. We use a binary tree *Tr* to hold candidates where the node in *Tr* has the form (*traj_id*, *modebits*). The first value is a key and the second is a *bitmap* that marks corresponding bits for transportation modes. For each candidate *mt*, the spatial and temporal conditions are first examined. Afterwards, we determine whether *mt* is

to be inserted into *Tr* or not. If the key *mt.traj_id* does not exist in *Tr*, a new node is created and inserted into *Tr*. For an existing object in *Tr*, there are two possibilities. Case (i): such a trajectory already satisfies the query condition. That is, all bits of *Q.m* are marked *true*. Then, this candidate does not have to be processed. Case (ii): if the corresponding bit for *mt.m* is *false*, we mark the value. Finally, we traverse *Tr* to produce the result. For each node n_i in *Tr*, if $|bitmap[n_i]| = |Q.m|$, such a trajectory satisfies the query condition and is returned.

Figure 4 gives an example to illustrate the refinement step. For simplicity, we focus on processing transportation modes. Assume that $Q.m = \{Walk, Indoor\}$. At the current state there are some objects in *Tr* including mo_1 (Fig. 4a), for which only the mode *Indoor* ($bitmap = 0001000$) is found from the processed movement tuples. Suppose that the candidates in *CL* are $\{mt_2(Walk), mt'_1(Indoor), mt'_2(Walk), mt'_3(Indoor)\}$, and they all satisfy spatial and temporal conditions. We explain how to process them regarding transportation modes. First, $mt_2(Walk)$ is received from the list. As mo_1 already exists in *Tr*, we check the *bitmap* and mark the *Walk* bit. mo_1 will be returned because all requested modes are found (Fig. 4b). Second, we get $mt'_1(Indoor)(\in mo_2)$, which is not in *Tr*. We insert a new node (mo_2, 0001000) into an appropriate position in the tree, see Fig. 4c. Next, another movement tuple mt'_2 from mo_2 is taken from *CL*. Correspondingly, the bit for *Walk* is marked *true* and we have (mo_2, 0001100), shown in Fig. 4d. Such a node fulfills the query condition and will be returned. Finally, we derive $mt'_3(Indoor)$ ($\in mo_2$) and ignore it because the trajectory already satisfies the condition.

7 A sequence of transportation modes

We proceed to consider the case that the user imposes a particular order on a set of modes. First, we introduce the baseline method. Then, we develop a method to let the index be capable of maintaining a pair of modes following the time order, resulting in a more efficient method to answer the query.

7.1 Baseline method

Following the filter-refinement step, we can directly employ the TM-RTree to first prune movements that do not contain transportation modes requested by *Q.m*. The filter step is the same as the procedure for multiple modes. Given a node N_i in a TM-RTree, if $N_i.M$ does not contain any mode in *Q.m*, N_i is pruned. Next, we check the precise value of spatial and temporal data for candidate objects. In comparison with the refinement step before, there are some differences here. Since *Q.m* refers to a sequence of modes, we need to maintain the time order on transportation modes. Assuming that $Q.m =< Car, Walk, Indoor >$,
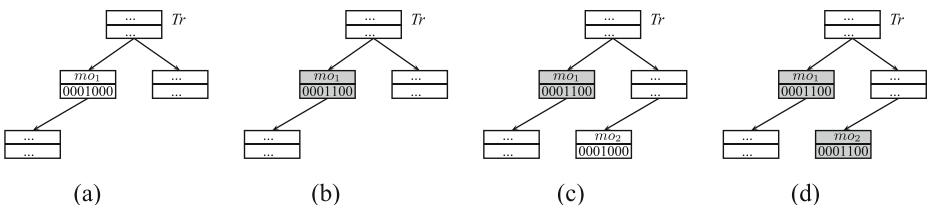

Fig. 4 Example of the refinement step

movement tuples $\{mt_1(Indoor), mt_2(Walk), mt_3(Walk), mt_4(Car)\}$ (from mo_1) are in the candidate list. Suppose that they are located in $Q.s$ during $Q.t$. Although mo_1 contains all modes requested by $Q.m$, we need to check the time order on transportation modes to see if they are equal to the query condition. In the example here, mo_1 should not be returned because from movement tuples we get the sequence $M(mo_1, Q.t) = <Indoor, Walk, Car>$, which is not equal to $Q.m$.

To maintain the time order, we propose the concept *temporal mode*, which stores transportation modes with time intervals. The definition is given below.

Definition 7.1 Temporal Mode

$UMode = \{(i, m)|i \in D_{interval}, m \in TM\}$

$D_{tmode} = \{< u_1, u_2, ..., u_n > |n \geq 1, n \in D_{int}, and \forall i \in [1, n], u_i \in UMode\}$

The *temporal mode* consists of a sequence of units, each of which records a time interval and a transportation mode. To keep candidate objects in the refinement, a binary tree Tr is employed in which each node has the form $(traj\_id, T\_Mode, s)$ $(T\_Mode \in D_{tmode}, s \in D_{bool})$. The first is the key, the second maintains transportation modes following the time order and the third records the status of such an object. For each mt from the candidate list, after checking spatial and temporal conditions, we insert a new node into Tr if $mt.traj\_id$ does not exist. Otherwise, we update T_Mode for the current mt. For the two elements in a *temporal mode* unit, the time interval is from one dimension of $mt.box$ and the transportation mode is from $mt.m$. Afterwards, we check whether T_Mode satisfies $Q.m$. If yes, we modify the status of such a node, $s = true$. Then, this node will be returned as a result and movement tuples coming later with the same key do not have to be processed.

7.2 A pair of modes

According to the study in [46], the walk segment is an important part for moving objects with multiple transportation modes as such a segment builds the connection between movements with different motion modes. Usually people do not directly switch from *Car* to *Bus* or from *Indoor* to *Metro* because a short distance walking is needed. The property of capturing transportation modes following the time order in a TM-RTree is crucial to efficiently process a sequence of modes. Motivated by this, we extend the semantic meaning of $mt.m$ (Def. 4.1) to represent not only a single mode but also a pair of modes $< m_1, m_2 > (m_1, m_2 \in TM$ and $m_1 \neq m_2)$. Such a pair fulfills the following condition: (i) the movement with m_1 appears before m_2; and (ii) $m_1 = Walk$ or $m_2 = Walk$. Since the number of modes (Def. 3.1 in Section 3.1) is finite, we can determine the set for all possible pairs of modes. That is, we combine *Walk* with another mode and treat them as one transportation mode. As a consequence, two modes are kept in such a traj and the time order on them is also maintained. This results in a new and more efficient method to answer the query.

The method is as follows. We get a new mode set excluding *Walk*, $TM' = TM \setminus \{Walk\}$. Depending on the time order, there are two cases for a pair of modes: (i) $< Walk, m' >$ $(m' \in TM')$; and (ii) $< m', Walk > (m' \in TM')$, e.g., $< Walk, Car >$ or $< Walk, Bus >$. For each kind of combinations, an integer bit is used to represent the value. Let $I(m)(m \in TM)$ return the bit for a single mode. In case (i), we let $|TM| + I(m')(m' \in TM')$ be the bit for such a pair, and in case (ii) we let $2 \times |TM| + I(m')(m' \in TM')$ return the bit. Plus the number of bit values for a single mode, we need $2 \times (|TM|-1)+|TM| = 19$ values in total ($<Walk, Walk>$ is not required). As we use a 32 bits integer, $mt.m$ is still available after the extension. Table 4 lists the bit for each mode value. For the pair

Table 4 Bit values for modes: an extension

Car: 0	Bus: 1	Walk:2	Indoor:3	Metro:4	Taxi:5	Bike: 6
<Walk, Car>: 7	<Walk, Bus>: 8		<Walk, Indoor>: 10	<Walk, Metro>:11	<Walk, Taxi>:12	<Walk, Bike>:13
<Car, Walk>: 14	<Bus, Walk>: 15		<Indoor, Walk>: 16	<Metro, Walk>:17	<Taxi, Walk>:18	<Bike, Walk>:19

(*Walk*, *Walk*), we keep the bit to have a simple value assignment system. The definition of transportation modes is extended and the new one is given in Def. 7.2.

Definition 7.2 Extended Transportation Modes

$TM\_Ext = TM \cup \{<Walk, m'> | m' \in TM'\} \cup \{<m', Walk> | m' \in TM'\}$, where $TM' = TM \setminus \{Walk\}$

7.3 Revisit movement tuples

After the mode extension, the procedure of getting movement tuples is modified accordingly. Besides movement tuples for a single mode, a pair of modes are to be represented. As mentioned above, *Walk* is the only value connecting two different modes. Compared with the previous procedure of getting movement tuples (introduced in Section 4.1), the difference is that we merge two continuous units with different modes and represent them as one movement tuple. Given a sequence of units $< u_1, u_2, ..., u_n >$, we create a movement tuple for u_i and u_{i+1} if (1) $u_i = m'(\in TM')$ and $u_{i+1} = Walk$; or (2) $u_i = Walk$ and $u_{i+1} = m'(\in TM')$. Using mo_1 as an example, originally, the movement tuples are $\{mt_1(Indoor), mt_2(Walk), mt_3(Walk), mt_4(Car), ..., mt_7(Car)\}$. Now, the new movement tuples are $\{mt_a(< Indoor, Walk >), mt_b(< Walk, Car >), mt_c(Car), mt_d(Car), mt_e(Car)\}$. The old four movement tuples $\{mt_1, mt_2, mt_3, mt_4\}$ are merged into two new movement tuples $\{mt_a, mt_b\}$, each of which represents the movement with a pair of modes. Several movement tuples with the mode *Car* are also created for mo_1. They are produced by the same method as before and fulfill the condition on the path length.

Considering mo_2, before the extension we have $\{mt'_1(Indoor), mt'_2(Walk), mt'_3(Indoor)\}$. One can merge mt'_1 and mt'_2 as one movement tuple with the mode $<Indoor, Walk>$ and leave mt'_3 as another movement tuple. However, this causes that such a pair $<Walk, Indoor>$ (mt'_2 and mt'_3) is missing in the representation, leading to extra work in the future to find such a sequence. To overcome the shortcoming, we split the walk movement into two parts, each of which is combined with another movement. Therefore, we have $\{mt'_a(< Indoor, Walk >), mt'_b(< Walk, Indoor >)\}$ as new movement tuples. If the walk movement is a long trip and represented by several units, we get the result $\{mt'_a(< Indoor, Walk >), mt'_b(Walk), mt'_c(< Walk, Indoor >)\}$ where the first and last walk units are combined with another movement. The rest walking units are represented as one movement tuple if the path length does not exceed the threshold. Otherwise, more than one movement tuple are created.

Next, we discuss the procedure of building a TM-RTree that supports a pair of modes. In total, there are three groups of transportation modes represented by movement tuples: (1) $m \in TM$; (2) $< Walk, m' > (m' \in TM')$; and (3) $< m', Walk > (m' \in TM')$. For each possible value, a unique integer is assigned and movement tuples are sorted according to this value. We keep the property that a leaf node only contains movement tuples with the same mode. Although we extend the semantic meaning of the mode integer and modify the method of producing movement tuples, the algorithm of creating the index is the same as before. The new feature is expressed by the mode integer in each TM-RTree node.

Due to the semantic extension, the query algorithms for a single mode and multiple modes have to be modified, i.e., the procedure of transportation modes examination. In the filter step, given a leaf node N_L, if $N_L.M$ denotes a single mode ($\in TM$), we can prune the node if (i) $\mathbf{tm}(N_L.M) \neq Q.m$ for answering a single mode and (ii) $\mathbf{tm}(N_L.M) \notin Q.m$ for answering multiple modes. However, if $N_L.M$ represents a pair of modes, we can only prune N_L if $\mathbf{tm}(N_L.M) \cap Q.m = \varnothing$. In the refinement step, since the movement tuple may

correspond to a trip with two modes, one has to extract a part of the trip if only one mode is requested by $Q.m$. Spatial and temporal checking is performed on the extracted part instead of the whole trip.

7.4 The algorithm

We present the algorithm that uses the TM-RTree capable of maintaining a pair of modes. First, the filter step is introduced. Given a node N_i, we compare the modes contained by $N_i.M$ with $Q.m$ to see if N_i is to be pruned or not. Since $Q.m =< m_1, m_2, ..., m_l >$ represents a sequence of modes, instead of checking the existence of each m_i in $N_i.M$, we process $Q.m$ as follows. Each two continuous modes are combined as a pair, that is $Q.m' = \{< m_1, m_2 >, < m_2, m_3 >, ..., < m_{l-1}, m_l >\}$. Then, for each node, we check the existence of each pair in $Q.m'$. This can significantly improve the query performance. Many leaf nodes containing a single mode are pruned, as only nodes containing a pair of modes in $Q.m'$ are further considered. Some movement tuples recording a single mode may contain one mode from $Q.m$, but such a value is included by another movement tuple with a pair of modes.

For example, consider that leaf nodes $\{N_L^1(< Indoor, Walk >), N_L^2(Walk), N_L^3(Walk, Car)\}$ are to be processed. Given that $Q.m =<$ Indoor, Walk, Car $>$, we have $Q.m' = \{<$ Indoor, Walk $>, < Walk, Car >\}$, then the leaf node N_L^2 is to be pruned. Remarkably, this can prune many leaf nodes during the query processing (e.g., a node contains the mode *Car*) and improve the performance. Applying Lemma 1 in Section 4.2, a function is used to check the modes between N_i and $Q.m'$. If the result is *true*, N_i will be further considered. Otherwise, N_i is pruned. We give the function in Algorithm 6. The filter step is almost the same as Algorithm 4 except that line 6 is replaced by *CheckMode*.

Algorithm 6: *CheckMode*$(N_i.M, Q.m')$

> **foreach** $m_i \in Q.m'$ **do**
>> **if** $m_i \in tm(N_i.M)$ **then**
>>> **return** *True*;
>
> **return** *False*;

We give the algorithm in Algorithm 7 where the detailed procedure of the filter step is omitted. In the refinement step, a binary tree Tr is employed to store candidates. The field T_Mode in Tr is to maintain the time order on transportation modes. Definition 7.1 is extended to include a pair of modes, that is

Definition 7.3 $UMode = \{(i, m) | i \in D_{interval}, m \in TM\_Ext\}$

8 Performance evaluation

We report experimental results in this section. The implementation is developed in an extensible database system Secondo [18] and programmed in C/C++. A standard PC (Intel(R) Core(TM) i7-4770CPU, 3.4GHz, 8GB memory, 2TB hard disk) running Suse Linux 13.1 (32 bits, kernel version 3.11.6) is used.

8.1 Datasets

We utilize a tool called MWGen [42] to create generic moving objects. Such a tool takes in roads and some public floor plans (e.g., office building [4], hotel [3]), then creates the underlying environments including road network, pavement areas, bus network, metro network and a set of buildings. Afterwards, moving objects are generated based on the result of trip planning over different environments. In the experiment, we use road datasets of Berlin [1] and Houston [2], shown in Fig. 5a, and eight floor plans in total (listed in [42]). Each trip may contain several transportation modes and possible values are listed in Fig. 5b.

Algorithm 7: *RQSeq*(*TM-RTree*, *Q*, *MT*)

$CL = RQSeq\_Filter(TM\text{-}RTree, Q)$;
let *Tr* be a binary tree with the node form (*traj_id*, *T_Mode*, *s*);
foreach $cl_i \in CL$ **do**
    get *mt* from *MT* by cl_i;
    if *SubTrip(mt) fulfills the condition on Q.t and Q.s* **then**
        if *mt.traj_id does not exist in Tr* **then**
            create a new node based on *mt* and insert into *Tr*;
        else
            $n = GetNode(Tr, mt.traj\_id)$;
            if $state[n] = False$ **then**
                create (i, m) from *mt* and insert into the *temporal mode* of n;
                if $mode[n]$ *satisfies Q.m* **then**
                    update the state of n;

let *RL* be the result list;
foreach $n_i \in Tr$ **do**
    if $state[n_i]$ **then**
 $RL \leftarrow key[n_i]$;
return *RL*;

The whole time period for moving objects is four weeks and the statistics are reported in Fig. 5c. We give a notation for each dataset describing the road data and the number of moving objects. The notation B40K means 40k moving objects in Berlin, B200K means 200k moving objects in Berlin, and so on. The precise number of moving objects is provided in the column *Trips No.* $|MT|$ shows the number (millions) of produced movement tuples from trajectories and $|I|$ says the average number of units per trip. The size of disk storage for moving objects is also given.

8.2 Competing methods

We compare the performance between the TM-RTree and the other three indices, AD-RTree (Adaptive RTree), 3D-RTree and 4D-RTree. Correspondingly, the query algorithms employing the AD-RTree, the 3D-RTree and the 4D-RTree are also implemented in the system. The AD-RTree is based on a 3D-RTree and the structure adds a value in each node to store transportation modes contained by its sons (or objects). Such an index simply stores the data of transportation modes in a node, but does not group objects on transportation modes.

	Berlin	Houston
Roads No.	3,250	4,575
X Range	[0, 44411]	[0, 133573]
Y Range	[0, 34781]	[0, 163280]

(a) Road Data

$$Indoor + Walk + \begin{cases} Bike \\ Car \\ Bus \\ Metro \\ Taxi \end{cases}$$
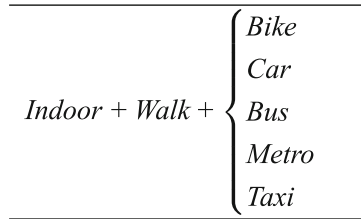
(b) Transportation Modes

Name	Trips No.	$|MT|$ (M)	$|I|$	Disk Size (G)
B40K	42,640	0.72	64	0.4
B200K	210,028	3.62	65	1.99
B400K	411,920	7.35	66	3.98
B1M	1,050,280	18.06	66	10.03
B2M	2,097,468	36.5	66	19.95
H1M	1,033,928	15.53	51	8.07

(c) Moving Objects

Fig. 5 The statistics of datasets

A leaf node does not hold the condition that only movement tuples with the same mode are contained. Instead, objects with different transportation modes are inserted, which is different from the TM-RTree. In addition, a sequence of transportation modes is not maintained by the AD-RTree.

The 3D-RTree manages spatial and temporal data without supporting transportation modes. This index does not have the capability of pruning objects that do not contain requested transportation modes, resulting in the weak pruning ability in the filter step. The refinement step checks transportation modes for each candidate. The 4D-RTree treats transportation modes as one dimension, and the mode is implemented by an integer. In order to create a 4D bounding box on movement tuples, we employ an epsilon and represent the transportation mode by an range instead of an integer. The bounding box is built on time, x, y and mode, and the 4D-RTree is created in a bulkload way.

8.3 Experimental setup

In order to have a good shape of the index, we perform some experiments to set appropriate values for the sizes of the time interval S_t and the spatial cell S_c, which are used to process movement tuples before sorting. This experiment is conducted on datasets B500K and H300K (not listed above), and several possible values are tested (e.g., $S_t = 12$ h or 24 h, $S_c = 4000.0$ or 5000.0). From the result, we find that the following settings achieve the best performance: (1) $S_t = 24$ h; and (2) $S_c = 6000.0$ for Berlin and $S_c = 20000.0$ for Houston. In addition, the threshold l used for splitting outdoor movements (see Section 4.1) is also determined by testing different values, and we get $l = 1500.0$ for Berlin and $l = 2000.0$ for the best performance. Therefore, we choose those values in the following experiments. Investigating these values in a deep (theoretical) way is out of scope of this paper.

We report the settings of query parameters in Fig. 6. The values of temporal and spatial parameters are set according to the overall range. The numbers in bold are standard settings. When we vary the value of spatial (temporal) parameter to evaluate the impact on the

Name	Values
Q_{sin}	$m \in TM$
Q_{mul}	(1) {*Indoor, Walk*};or {*Indoor, m_v*}($m_v \in TM\_v$); or {*Walk, m_v*}($m_v \in TM_v$) (2) {*Walk, Indoor, m_v*} ($m_v \in TM_v$)
Q_{seq}	(1) <*Walk, m_v* > or < *m_v, Walk*> ($m_v \in TM_v$); (2) <*Indoor, Walk, m_v* > or < *m_v, Walk, Indoor*> ($m_v \in TM_v$); (3) <*Indoor, Walk, m_v, Walk, Indoor*> ($m_v \in TM_v$)

Name	Values
Q_t	0.15%, 1.5%, **3.57%**, 7.14%, 17.85% 1h, 10h, **1d**, 2d, 5d (h = hour, d = day)
Q_s	10%, 15%, **20%**, 30%, 40%

(a) Time and Spatial

(b) Transportation Modes

Fig. 6 Query parameters settings

performance, the other parameter is set as the standard value. A notation is defined for each kind of $Q.m$, summarized in Table 5.

For the sake of easy description, we define a particular set of transportation modes.

Definition 8.1 Vehicle Transportation Modes
$TM_v = \{Bus, Metro, Taxi, Car, Bike\}$

The settings for transportation modes are shown in Fig. 6b. For Q_{sin}, a mode from *TM* is randomly selected for one query. If several modes are requested, we let $|Q.m|$ be two or three because usually a person does not have too many modes during his trip. If $|Q.m| = 2$, we let the two values be randomly chosen among *TM* excluding the case that both modes are vehicles. If $|Q.m| = 3$, we let the result be {*Indoor, Walk*} plus a vehicle. In the case Q_{seq}, we define a set of possible combinations and randomly choose one of them for each query.

For each kind of queries (Q_{sin}, Q_{mul} and Q_{seq}), we generate 100-500 cases and take the average cost as the final result. In each case, $Q.m$ is randomly chosen from its possible values. $Q.t$ and $Q.m$ are set as standard values by default, and accurate values are randomly generated within the possible set. We evaluate the performance in terms of CPU time and I/O accesses.

8.4 Transportation modes in the index

Since both the TM-RTree and the AD-RTree maintain transportation modes, we compare the average number of modes contained by a node between the two indices. Datasets B2M and H1M are used for the evaluation. Figure 7 reports the numbers at different levels. We define 0 the root level, 1 the next level and so on. There are two extreme cases: root level and leaf level. Both indices maintain the total number of modes at root level, 19 in total (according to Def. 7.2). At leaf level, the TM-RTree only contains one but the AD-RTree includes more

Table 5 Notations for queries

Notation	Description
Q_{sin}	A single mode
Q_{mul}	Multiple Modes
Q_{seq}	A sequence of modes

Level	0	1	2	3	4
TM-RTree	19	19	16	2	1
AD-RTree	19	19	18	18	10

(a) B2M

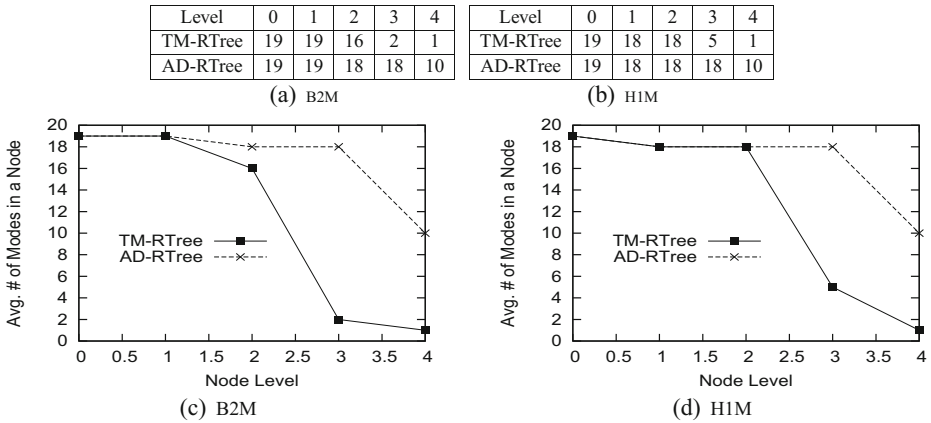| Level	0	1	2	3	4
TM-RTree	19	18	18	5	1
AD-RTree	19	18	18	18	10

(b) H1M

(c) B2M

(d) H1M

Fig. 7 Average number of modes maintained in a node

values. This factor has significant effect on the performance and we report the result in the following subsections, see Figs. 8, 11 and 12. The reason is, the less number of modes maintained in a node, the better the pruning ability of the index is. For example, root node always has to be considered as it includes all modes. By observation, the event of pruning unqualified nodes usually occurs when the traversing procedure goes into the low level as nodes at the high level have large extent in terms of spatial, temporal and transportation modes. Compared with the AD-RTree, nodes in TM-RTree keep much smaller number of modes at the low level.

8.5 Scaling datasets

We report the result for different queries on transportation modes in Figs. 8, 9 and 10. $Q.s$ and $Q.t$ are defined to be standard values. When the number of trips increases, the cost of all indices rises proportionally. The TM-RTree is faster than the other three in all cases. In particular, the deviation becomes large for the big dataset. Using B2M, on average TM-RTree is 5 times faster than AD-RTree and 6 times faster than 3D R-Tree. This is because when applying the AD-RTree and the 3D-RTree, the filter step does not efficiently prune
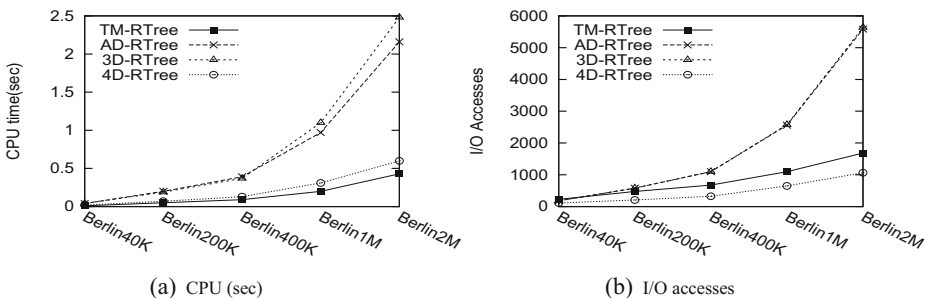

(a) CPU (sec)

(b) I/O accesses

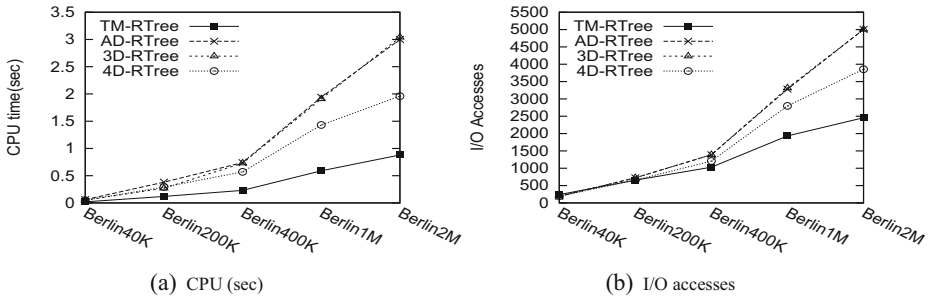**Fig. 8** $Q_s in$: a single mode - scaling datasets

Fig. 9 Q_{mul}: multiple modes - scaling datasets

unqualified nodes, incurring much more CPU and I/O cost. The 3D-RTree does not maintain transportation modes. Although the AD-RTree stores modes in a node, the index does not group objects on transportation modes. A leaf node may contain several kinds of modes, leading to the weak pruning ability and many nodes to be visited. Therefore, a large amount of movement tuples will be accessed in the refinement step, resulting in more processing time.

The time cost of the TM-RTree is better than the 4D-RTree for Q_{sin}, while the I/O cost is slightly worse. This is because the TM-RTree maintains a single mode as well as a pair of modes. When only a single mode is requested, both nodes containing a single mode and a pair of modes have to be investigated, leading to more I/O accesses. For Q_{mul} and Q_{seq}, the performance of the TM-RTree is superior to the 4D-RTree in both time and I/O cost.

We also observe that there is no obvious performance difference between the AD-RTree and the 3D-RTree (see Figs. 8, 9 and 10), demonstrating that the effect of simply storing modes in a node is not obvious for improving the query efficiency. The 4D-RTree treats transportation modes as one dimension and its performance is better than the AD-TRee and the 3D-RTree. We put the precise value of each query cost in Appendix B.

8.6 Varying spatial and temporal parameters

Using the dataset B2M, we perform the experiment on all kinds of query modes: Q_{sin}, Q_{mul} and Q_{seq}. In each case, two groups of results are reported. We evaluate the impact
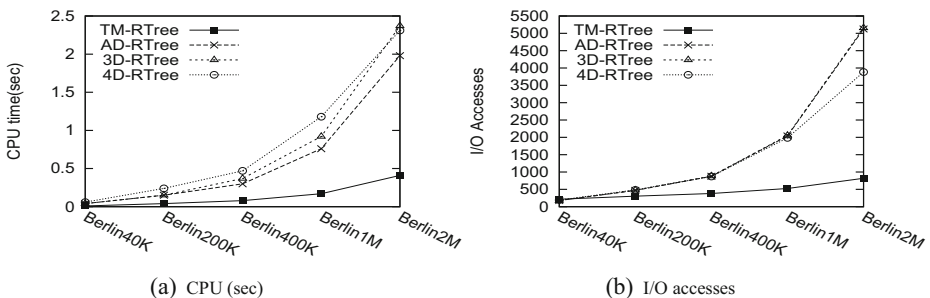

Fig. 10 Q_{seq}: a sequence of modes - scaling datasets
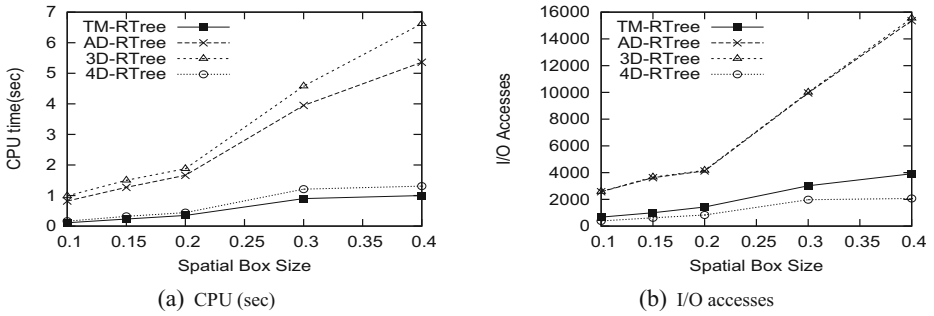
(a) CPU (sec)

(b) I/O accesses

Fig. 11 Q_{sin}: a single mode - spatial

of spatial (temporal) parameter on the performance by keeping the standard value for the other parameter. The performance of the TM-RTree is much better than the AD-RTree, the 3D-RTree and the 4D-RTree, especially when the size of the query box becomes large. On average, TM-RTree is 5 times faster than the other three indices. Similar to the result in the scaling experiment, there is a subtle difference between the performance of the TM-RTree and the 4D-RTree for Q_{sin}. Our index shows its advantages in Q_{mul} and Q_{seq}. Figures 11 and 12 report the result for Q_{sin}, Figs. 13 and 14 report the result for Q_{mul}, and Figs. 15 and 16 report the result for Q_{seq}.

8.7 The capability of TM-RTree

In the end, we compare two kinds of TM-RTrees where one only supports the movement with a single mode and the other manages both a single mode and a pair of modes. To distinguish between them, we name the former TM-RTree_sin. Two datasets are used, B2M and H1M. The evaluation is performed on different transportation mode conditions. Spatial and temporal parameters are set as standard values.

The results are different for the three cases, shown in Figs. 17 and 18. The TM-RTree_sin has better performance than the TM-RTree for Q_{sin}. Although the CPU time is almost the same, less I/O accesses are incurred for TM-RTree_sin. The reason is, after the mode extension, the total number of modes increases. The TM-RTree maintains more transportation
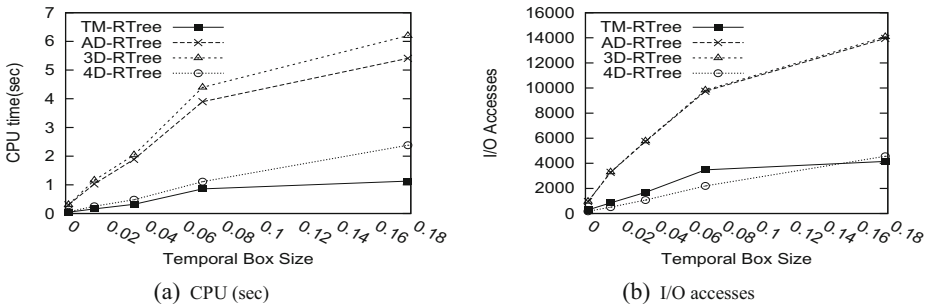

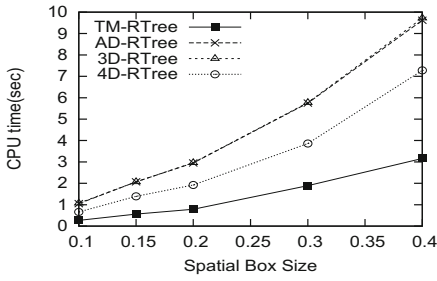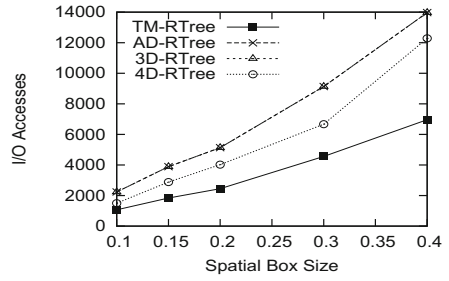
(a) CPU (sec)

(b) I/O accesses

Fig. 12 Q_{sin}: a single mode - temporal

Fig. 13 Q_{mul}: multiple modes - spatial
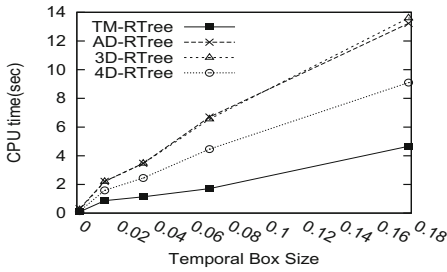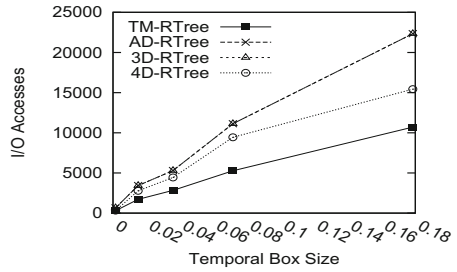
Fig. 14 Q_{mul}: multiple modes - temporal
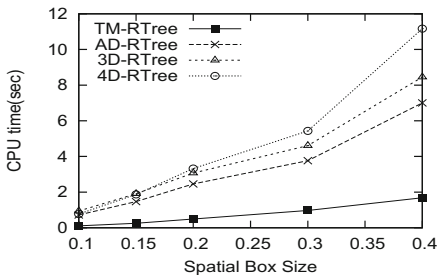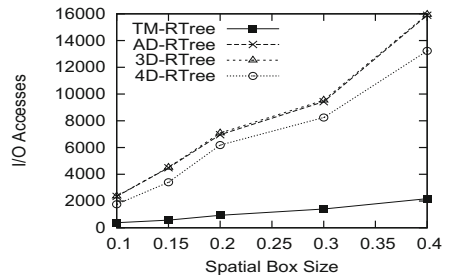
Fig. 15 Q_{seq}: a sequence of modes - spatial

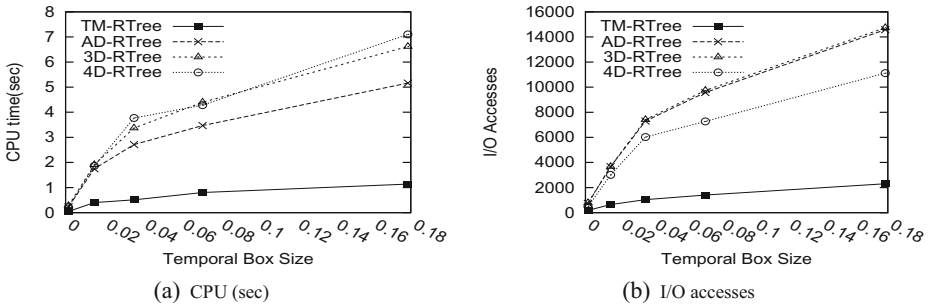(a) CPU (sec)   (b) I/O accesses

Fig. 16 Q_{seq}: a sequence of modes - temporal

modes than the TM-RTree_sin, i.e., different pairs of modes. Given a mode m ($m \neq Walk$), the procedure only has to consider leaf nodes containing m for TM-RTree_sin. However, using the TM-RTree we also have to process nodes containing the mode $< m, Walk >$ or $< Walk, m >$. In addition, there exists an extreme case for *Walk* which builds the connection between different modes, resulting in many cases to be included, e.g., <*Walk*, *Bus*> and <*Indoor*, *Walk*>. As a result, the TM-RTree incurs more nodes to be accessed.

The performance is almost the same between the TM-RTree and the TM-RTree_sin for Q_{mul}. In Q_{seq}, as expected, the TM-RTree is significantly better than the TM-RTree_sin (see in Figs. 17 and 18). On average, the former is 6-7 times faster than the latter, and I/O accesses by the TM-RTree are about 1/3 as the TM-RTree_sin.

8.8 Discussion

We discover some behaviors based on the experimental result and give a summary. First, the performance is significantly improved when the index manages spatial, temporal and transportation modes in an integrated way. On average, the TM-RTree is 5 times faster than AD-RTree, 3D-RTree and 4D-RTree. Second, the 4D-RTree is superior to the AD-RTree and the 3D-RTree. Especially, if a single mode is requested, the performance of a
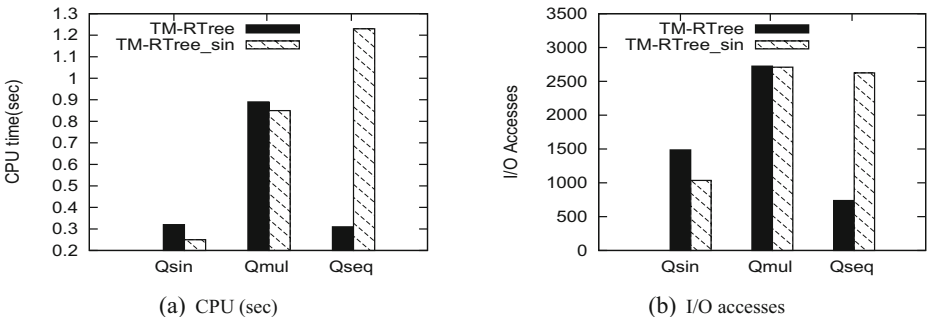

(a) CPU (sec)   (b) I/O accesses

Fig. 17 B2M: TM-RTree and TM-RTree_sin
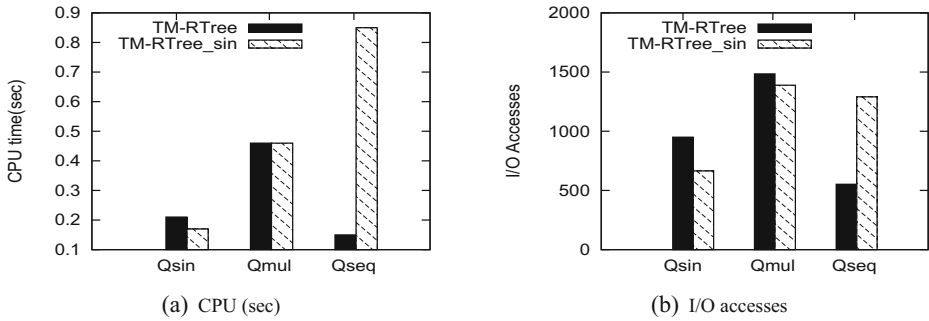
Fig. 18 H1M: TM-RTree and TM-RTree_sin

4D-RTree is even close to or slightly better than the TM-RTree. Third, in comparison with the TM-RTree_sin which only supports a single mode, the TM-RTree incurs more I/O accesses for Q_{sin}, but is much faster for Q_{seq}. The performance of the TM-RTree_sin decreases substantially for Q_{seq} as the index does not maintain the time order on transportation modes. In the end, regarding the three kinds of transportation modes, Q_{mul} needs more CPU time and I/O accesses than the other two cases. Between Q_{sin} and Q_{seq}, the performance for Q_{seq} is even better than that for Q_{sin} when employing the TM-RTree.

9 Conclusions

In this paper, we study range queries for moving objects with multiple transportation modes and consider three query cases: a single mode, multiple modes and a sequence of modes. An index structure called TM-RTree is proposed to efficiently manage moving objects with transportation modes. Such a structure manages not only spatial and temporal data but also transportation modes. The TM-RTree has the capability of maintaining the movement with a single mode as well as a pair of modes following the time order. Several algorithms are developed to efficiently answer queries. A comprehensive experimental study is reported to verify the performance of our solution in terms of efficiency and scalability.

Appendix – A Range Query Examples

- Who passed the room No. 123 in the university on Tuesday afternoon?

    - $Q.t$: Tuesday afternoon
    - $Q.s$: room No. 123 in the university
    - $Q.m$: *Indoor*

- Find all taxis passing through Alexender street on Saturday.

    - *Q.t*: Saturday
    - *Q.s*: Alexender street
    - *Q.m*: *Taxi*

- Find out all people walking through zone A and moving around in a shopping mall on Saturday between 10am and 3pm.

    - *Q.t*: [10am, 3pm] on Saturday
    - *Q.s*: a region + a building
    - *Q.m*: {*Walk*, *Indoor*}

- Find all people who pass room No. 34 at the office building between 9:00am and 12:00am on Monday, and then take a bus to the train station.

    - *Q.t*: [9am, 12am] on Monday
    - *Q.s*: a region covering the building and the station
    - *Q.m*: <*Indoor*, *Walk*, *Bus*, *Walk*, *Train*>

- Find all people who drive through the area A and then walk to the building X on Monday between [8am, 9am].

    - *Q.t*: [8am, 9am] on Monday
    - *Q.s*: a region including A and X
    - *Q.m*: <*Car*, *Walk*, *Indoor*>

Appendix– B Experimental Statistics

Tables 6, 7, 8, 9, 10, 11, and 12

Table 6 Scaling experiments: Q_{sin}

	TM-RTree	AD-RTree	3D-RTree	4D-RTree
(a) CPU Time (sec)				
B40K	0.01	0.04	0.04	0.02
B200K	0.05	0.20	0.19	0.07
B400K	0.09	0.39	0.37	0.13
B1M	0.20	0.97	1.10	0.31
B2M	0.43	2.16	2.48	0.6
(b) I/O Accesses				
B40K	228.06	180.95	181.175	108.99
B200K	475.92	578.83	581.09	208.73
B400K	677.52	1097.36	1100.82	324.75
B1M	1097.71	2561.275	2577.3	649.05
B2M	1683.62	5585.04	5636.63	1064.85

Table 7 Scaling experiments: Q_{mul}

	TM-RTree	AD-RTree	3D-RTree	4D-RTree
(a) CPU Time (sec)				
B40K	0.02	0.06	0.05	0.05
B200K	0.12	0.38	0.27	0.29
B400K	0.23	0.74	0.73	0.57
B1M	0.59	1.94	1.91	1.43
B2M	0.88	3.0	3.04	1.96
(b) I/O Accesses				
B40K	238.36	190.16	190.16	191.86
B200K	655.02	725.02	725.02	657.83
B400K	1031.43	1388.73	1388.73	1208.73
B1M	1932.99	3282.19	3314.15	2798.16
B2M	2456.38	5001.62	5001.62	3855.9

Table 8 Scaling experiments: Q_{seq}

	TM-RTree	AD-RTree	3D-RTree	4D-RTree
(a) CPU Time (sec)				
B40K	0.01	0.04	0.04	0.06
B200K	0.04	0.15	0.15	0.24
B400K	0.08	0.30	0.37	0.47
B1M	0.17	0.76	0.92	1.18
B2M	0.41	1.98	2.37	2.31
(b) I/O Accesses				
B40K	215.89	178.46	179.1	191.59
B200K	305.1	470.31	475.53	485.49
B400K	380.64	876.79	884.05	869.69
B1M	525.49	2038.58	2053.7	1983.27
B2M	819.02	5125.36	5157.24	3882.36

Table 9 Varying spatial and temporal parameters: Q_{sin}

	Q_s					Q_t				
	10%	15%	20%	30%	40%	0.15%	1.5%	3.57%	7.14%	17.85%
(a) CPU Time (sec)										
TM-RTree	0.11	0.23	0.35	0.9	1.0	0.047	0.16	0.32	0.86	1.90
AD-RTree	0.82	1.27	1.66	3.95	5.36	0.30	1.03	1.89	3.9	9.36
3D-RTree	0.99	1.50	1.88	4.58	6.62	0.31	1.16	2.04	4.4	11.22
4D-RTree	0.16	0.32	0.44	1.21	1.31	0.06	0.25	0.48	1.11	2.38
(b) I/O Accesses										
TM-RTree	674.07	1001.32	1429.26	3016.4	3912.07	287	841.83	1679.2	3478.67	7618.9
AD-RTree	2570.95	3619.0	4122.52	9958.4	15360.47	948.1	3282.4	5741.47	9740.73	25179.8
3D-RTree	2589.2	3664.34	4164.57	10011.3	15544.1	955.47	3304.2	5770.57	9826.27	25527.55
4D-RTree	384.29	624.1	831.34	1972.54	2069.67	175.63	508.57	1069.9	2195.8	4556.7

Table 10 Varying spatial and temporal parameters: Q_{mul}

	Q_s					Q_t					
	10%	15%	20%	30%	40%	0.15%	1.5%	3.57%	7.14%	17.85%
(a) CPU Time (sec)										
TM-RTree	0.27	0.56	0.79	1.89	3.16	0.09	0.87	1.14	1.71	4.66
AD-RTree	1.05	2.07	2.95	5.75	9.62	0.31	2.21	3.47	6.70	13.23
3D-RTree	1.07	2.07	2.96	5.77	9.72	0.23	2.20	3.46	6.55	13.60
4D-RTree	0.66	1.39	1.92	3.86	7.28	0.14	1.59	2.45	4.46	9.10
(b) I/O Accesses										
TM-RTree	1066.43	1839.16	2446.1	4558.45	6961.95	296.66	1722.48	2839.95	5266.5	10715.25
AD-RTree	2233.37	3888.42	5131.56	9134.55	13973.25	649.12	3437.48	5297	11125.7	22345.45
3D-RTree	2233.37	3888.42	5131.56	9134.55	13973.25	649.12	3437.48	5297	11125.7	22345.45
4D-RTree	1502.11	2880.98	4022.3	6672.7	12286.3	310.76	2778.26	4443.3	9435.8	15429.9

Table 11 Varying spatial and temporal parameters: Q_{seq}

	Q_s					Q_t				
	10%	15%	20%	30%	40%	0.15%	1.5%	3.57%	7.14%	17.85%
(a) CPU Time (sec)										
TM-RTree	0.11	0.25	0.49	0.97	1.68	0.05	0.4	0.51	0.8	1.14
AD-RTree	0.72	1.47	2.46	3.76	7.0	0.23	1.75	2.71	3.47	5.16
3D-RTree	0.94	1.91	3.07	4.61	8.45	0.28	1.91	3.37	4.4	6.62
4D-RTree	0.77	1.87	3.32	5.43	11.17	0.22	1.86	3.77	4.28	7.10
(b) I/O Accesses										
TM-RTree	380.65	570.3	935.8	1400.85	2172.35	185.9	647	1040.3	1404.7	2303.95
AD-RTree	2344.87	4484.82	6960.7	9431.75	15891.85	731.2	3670.45	7298.5	9572.9	14579.9
3D-RTree	2358.46	4511.2	7082.25	9536.45	15946.8	783.1	3673.3	7412.25	9737.55	14732.7
4D-RTree	1755.48	3406.8	6181.4	8240.65	13230	480.5	3006.8	6028.2	7268.85	11118.7

Table 12 Compare the performance between the TM-RTree and the TM-RTree_sin

	B2M			H1M		
	Q_{sin}	Q_{mul}	Q_{seq}	Q_{sin}	Q_{mul}	Q_{seq}
(a) CPU Time (sec)						
TM-RTree	0.32	0.89	0.31	0.21	0.46	0.15
TM-RTree_sin	0.25	0.85	1.23	0.17	0.46	0.85
(b) I/O Accesses						
TM-RTree	1485.77	2724.42	738.18	949.96	1484.77	552.34
TM-RTree_sin	1034.77	2708.93	2624.69	664.76	1389.33	1290.49

References

1. http://www.bbbike.de/cgi-bin/bbbike.cgi (2012)
2. http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html (2012)
3. http://www.edenresort.com/home (2012)
4. http://www.modulargenius.com/default.aspx (2012)
5. Bauer V, Gamper J, Loperfido R, Profanter S, Putzer S, Timko I (2008) Computing isochrones in multi-modal, schedule-based transport networks. In: ACM GIS, Demo, p 78
6. Berchtold S, Böhm C, Kriegel HP (1998) Improving the query performance of high-dimensional index structures by bulk load operations. In: EDBT, pp 216–230
7. Bercken J, Seeger B, Widmayer P (1997) A generic approach to bulk loading multidimensional index structures. In: VLDB, pp 406–415
8. Booth J, Sistla P, Wolfson O, Cruz IF (2009) A data model for trip planning in multimodal transportation systems. In: EDBT, pp 994–1005
9. Cai Y, Ng R (2004) Indexing spatio-temporal trajectories with chebyshev polynomials. In: SIGMOD, pp 599–610
10. Chakka VP, Everspaugh A, Patel JM (2003) Indexing large trajectory data sets with seti. In: CIDR
11. Cong G, Jensen CS, Wu D (2009) Efficient retrieval of the top-k most relevant spatial web objects. PVLDB 2(1):337–348
12. Cong G, Lu H, Ooi BC, Zhang D, Zhang M (2012) Efficient spatial keyword search in trajectory databases. CoRR, abs/1205.2880
13. de Almeida VT, Güting RH (2005) Indexing the trajectories of moving objects in networks. GeoInformatica 9(1):33–60
14. Felipe ID, Hristidis V, Rishe N (2008) Keyword search on spatial databases. In: ICDE, pp 656–665
15. Forlizzi L, Güting RH, Nardelli E, Schneider M (2000) A data model and data structures for moving objects databases. In: SIGMOD, pp 319–330
16. Frentzos E (2003) Indexing objects moving on fixed networks. In: SSTD, pp 289–305
17. Gedik B, Liu L (2004) Mobieyes: distributed processing of continuously moving queries on moving objects in a mobile system. In: EDBT, pp 67–87
18. Güting RH, Almedia V, Ansorge D, Behr T, Ding Z, Höse T, Hoffmann F, Spiekermann M (2005) Secondo:an extensible dbms platform for research prototyping and teaching. In: ICDE, Demo Paper, pp 1115–1116
19. Güting RH, de Almeida VT, Ding ZM (2006) Modeling and querying moving objects in networks. VLDB J 15(2):165–190
20. 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
21. Hage C, Jensen CS, Pedersen TB, Speicys L, Timko I (2003) Integrated data management for mobile services in the real world. In: VLDB, pp 1019–1030
22. Jensen CS, Lu H, Yang B (2009) Graph model based indoor tracking. In: MDM, pp 122–131
23. Jensen CS, Lu H, Yang B (2009) Indexing the trajectories of moving objects in symbolic indoor space. In: SSTD, pp 208–227

24. Kollios G, Papadopoulos D, Gunopulos D, Tsotras VJ (2005) Indexing mobile objects using dual transformations. VLDB J 14(2):238–256
25. Dinh L, Aref WG, Mokbel MF (2010) Spatio-temporal access methods: Part 2 (2003–2010). IEEE Data Eng Bull 33(2):46–55
26. Li M, Dai J, Sahu S, Naphade MR (2011) Trip analyzer through smartphone apps. In: GIS, pp 537–540
27. Lu H, Cao X, Jensen CS (2012) A foundation for efficient indoor distance-aware query processing. In: ICDE, pp 438–449
28. Mokbel MF, Xiong X, Aref WG (2004) Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD Conference, pp 623–634
29. Pelanis M, Saltenis S, Jensen CS (2006) Indexing the past, present, and anticipated future positions of moving objects. ACM TODS 31(1):255–298
30. Pfoser D, Jensen CS (2003) Indexing of network constrained moving objects. In: GIS, pp 25–32
31. Pfoser D, Jensen CS (2000) Novel approaches in query processing for moving object trajectories. In: VLDB, pp 395–406
32. Popa IS, Zeitouni K, Oria V, Barth D, Vial S (2011) Indexing in-network trajectory flows. VLDB J 20(5):643–669
33. Reddy S, Mun M, Burke J, Estrin D, Hansen MH, Srivastava MB (2010) Using mobile phones to determine transportation modes. TOSN 6(2)
34. Sistla P, Wolfson O, Chamberlain S, Dao S (1997) Modeling and querying moving objects. In: ICDE, pp. 422–432
35. Stenneth L, Wolfson O, Yu P, Xu B (2011) Transportation Mode Detection using Mobile Devices and GIS Information. In: ACM SIGSPATIAL, pp 54–63
36. Tao Y, Papadias D (2001) Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In: VLDB, pp 431–440
37. Tao Y, Papadias D, Sun J (2003) The TPR*-Tree: an optimized spatio-temporal access method for predictive queries. In: VLDB, pp 790–801
38. Theodoridis Y, Vazirgiannis M, Sellis TK (1996) Spatio-temporal indexing for large multimedia applications. In: ICMCS, pp 441–448
39. 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
40. Wang L, Zheng Y, Xie X, Ma WY (2008) A flexible spatio-temporal indexing scheme for large-scale gps track retrieval. In: MDM, pp 1–8
41. 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
42. Xu J, Güting RH (2012) MWGen: a mini world generator. In: MDM, pp 258–267
43. Xu J, Güting RH (2013) A generic data model for moving objects. GeoInformatica 17(1):125–172
44. Yang B, Lu H, Jensen CS (2009) Scalable continuous range monitoring of moving objects in symbolic indoor space. In: CIKM, pp 671–680
45. Zhang D, Chee YM, Mondal A, Tung AKH, Kitsuregawa M (2009) Keyword search in spatial databases: Towards searching by document. In: ICDE, pp 688–699
46. Zheng Y, Chen Y, Xie X, Ma WY (2010) Understanding transportation mode based on GPS data for Web application, vol 4
47. Zheng Y, Xie X, Ma WY (2010) GeoLife: A collaborative social networking service among user, location and trajectory. Invited paper. IEEE Data Eng Bull 32(2):32–40
48. Zheng Y, Zhang L, Ma Z, Xie X, Ma WY (2011) Recommending friends and locations based on individual location history. TWEB 5(1):5
49. Zheng Y, Zhou X (2011) Computing with spatial trajectories, Springer

Jianqiu Xu got his bachelor and master degree from Nanjing University of Aeronautics and Astronautics in 2005 and 2008, respectively. Then, he studied the Ph.D supervised by Prof. Dr. Ralf Hartmut Güting between 2008.9 and 2012.10 from FernUniversität in Hagen, Germany, focusing on moving objects databases and spatial databases. In 2013.1, he joined Nanjing University of Aeronautics and Astronautics in China as an assistant professor, and became an associate professor since 2013.6.

Ralf Hartmut Güting has been a full professor in Computer Science at the University of Hagen, Germany, since 1989. He received his Diploma and Dr. rer. nat. degrees from the University of Dortmund in 1980 and 1983, respectively, and became a professor at that university in 1987. From 1981 until 1984 his main research area was Computational Geometry. After a one-year stay at the IBM Almaden Research Center in 1985, extensible and spatial database systems became his major research interests; more recently, also spatio-temporal or moving objects databases. He has been an associate editor of the ACM Transactions on Database Systems and an editor of the VLDB Journal and is on the Editorial Board of GeoInformatica. He has published two German text books on data structures and algorithms and on compilers, respectively, and an English text book on moving objects databases, as well as around eighty journal and conference articles. His group has built prototypes of extensible and spatio-temporal database systems, the Gral system and the SECONDO system.

Yu Zheng is a lead researcher from Microsoft Research Asia. He is an IEEE senior member and ACM senior member. His research interests include location-based services, spatio-temporal data mining, ubiquitous computing, and mobile social applications. He has published over 50 referred papers as a leading author at high-quality international conferences and journals, such as SIGMOD, SIGKDD, AAAI, ICDE, WWW, Ubicomp, IEEE TKDE, and ACM TWEB. He has received 3 best paper awards from UIC10, ACM SIGSPA-TIAL GIS11, and ADMA'11 as well as 1 best paper nominee from Ubicomp11. He is also an editorial board of 4 international journals, such as GeoInformatica and Journal of Location Based Services. In 2008, he was recognized as the Microsoft Golden Star. He joined MSRA in July 2006 right after received his Ph.D. degree in communication & information systems from Southwest Jiaotong University.