Geoinformatica (2015) 19:227–276
DOI 10.1007/s10707-014-0211-9

GMOBench: Benchmarking generic moving objects

Jianqiu Xu · Ralf Hartmut Güting · Xiaolin Qin

Received: 24 July 2013 / Revised: 16 February 2014 /
Accepted: 15 May 2014 / Published online: 26 June 2014

Abstract In real world scenarios, people's movement include several environments rather than one, for example, road network, pavement areas and indoor. This imposes a new challenge for moving objects database that the complete trip needs to be managed by a database system. In the meantime, novel queries regarding different transportation modes should also be supported. Since existing methods are limited to trips in a single environment and do not support queries on moving objects with different transportation modes, new technologies are essentially needed in a database system. In this paper, we introduce a benchmark called GMOBench that aims to evaluate the performance of a database system managing moving objects in different environments. GMOBench is settled in a realistic scenario and is comprised of three components: (1) a data generator with the capability of creating a scalable set of trips representing the complete movement of humans (both indoor and outdoor); (2) a set of carefully designed and benchmark queries; (3) Mode-RTree, an index structure for managing generic moving objects. The generator defines some parameters so that users can control the characteristics of results. We create the benchmark data in such a way that the dataset can mirror important characteristics and real world distributions of human mobility. Efficient access methods and optimization techniques are developed for query processing. In particular, we propose an index structure called Mode-RTree to manage moving objects in different environments. By employing the proposed index, the cost of benchmark queries

A short version appears in *ACM SIGSPATIAL*, Nov.6-9, Redondo Beach, California, 2012

Part of the work is done when the author is a Ph.D student in FernUniversität in Hagen, Germany

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

X. Qin
e-mail: qinxcs@nuaa.edu.cn

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

is greatly reduced. GMOBench is implemented in a real database system to have a practical result. We perform an extensive experimental study on comprehensive datasets to evaluate the performance. The results show that by using the Mode-RTree we achieve significant performance improvement over the baseline method, demonstrating the effectiveness and efficiency of our approaches.

Keywords Moving objects · Transportation modes · Benchmark · Query processing · Index

1 Introduction

Recently, the area of moving objects with different transportation modes becomes an interesting topic due to novel applications on detecting outdoor transportation modes and advanced trip plannings. Based on raw GPS data, motion modes such as walking, driving and cycling can be inferred in order to have more contextual information of mobile users [38, 42]. In Microsoft's project GeoLife, researchers aim to discover transportation modes from GPS data by identifying a set of sophisticated features such as velocity, acceleration and heading changing rate [57, 58]. In an advanced transportation system [1, 5], a realistic traveling plan includes a trip with different choices of modes and constraints with modes, e.g., less than two bus transfers, the walking distance is smaller than 300 meters.

As a kind of human behavior, transportation modes are closely related to the movement. Recognizing such pieces of information can enrich the knowledge of a user's mobility. From the viewpoint of the database community, the management of continuously changing location data and transportation modes (environments) requires dedicated support from the underlying database system. However, existing techniques are only able to manage the data in one environment such as free space or road network, and cannot answer new queries on moving objects traveling through several environments. The first problem needs to be solved is to have a consistent and efficient location representation in all available environments such as road network, bus network and indoor. Based on a data model proposed in [55], we develop new techniques in a database system to support efficient query processing for moving objects in different environments, for example, *Indoor* → *Walk* → *Bus* → *Walk*. To evaluate the performance of such a database system, a benchmark is needed to test the system under a wide range of queries.

A benchmark [4, 9, 13, 37, 41, 43, 48], consisting of a set of comprehensive and scalable datasets and a group of well defined queries, plays a crucial role in evaluating the functionality and performance of a database both for application users and developers. Additionally, a benchmark allows one to test a system's capabilities and helps determine its strengths or potential bottlenecks. In the field of moving objects databases, simulation is widely accepted [11, 14] to provide synthetic data for designing and testing new data types and access methods due to the difficulty of getting a large amount of real data. In particular, real data might not be comprehensive enough to thoroughly evaluate the system in consideration. Consequently, researchers develop tools to create synthetic data in a realistic scenario. Nevertheless, existing data generators [6, 40, 47] and benchmarks [11, 14, 46] only deal with the movement in one environment without considering transportation modes. The data do not represent the complete trips of humans. It is also not easy to get the real data of moving objects with precise transportation modes including both outdoor and indoor. To solve the problem, new methods are needed to generate moving objects with multiple transportation modes in a realistic way.

A large amount of indices have been proposed to efficiently support query processing on trajectories in the literature [8, 10, 12, 24, 34]. However, none of them consider the important property of moving objects: transportation modes. They only tackle the problem in a single environment without managing the complete trajectory, i.e., a portion of the movement is indexed. Additionally, in the current state-of-the-art indices work by grouping object extents on spatial and temporal without maintaining other useful information such as the transportation mode and geographic objects. For some environments such as road network and indoor, the location of a moving object is closely related to the underlying environment. This calls for an index structure that is equipped with new properties to speed up query processing.

Previous works [53–55] are fundamental steps to the present benchmark where a data model is designed to represent generic moving objects by referencing to the underlying environments, and a data generator is developed to create all real world environments: road network, bus network, metro network, pavement areas and indoor. After that, a meaningful analysis and evaluation of such a system necessitates a comprehensive benchmark.

In this article, we propose a benchmark called GMOBench to evaluate the database performance by a broad range of novel queries on moving objects considering transportation modes and different environments. We process the complete past movement (also called trajectory) and generate scalable and comprehensive datasets to simulate a variety of real life scenarios. It makes no sense to create human movement in a pure random fashion as usually there is an evident motivation to start a trip, accomplishing some tasks or performing an activity.

To achieve the goal of realistic simulation, we define a set of rules to create trips with various properties to model human movement behavior in practice. For example, people's trajectories exhibit *regular patterns* [20] most of the time, e.g., commuting. On the weekend, based on the habits and preferences they may have some trips to interesting places such as home of friends, shopping malls, and cinemas. To perform an activity, people usually prefer nearby to distant places, e.g., look for the nearest hotel. Since moving objects are generated based on real life behavior, our benchmark datasets can be used for some other applications. For example, one can monitor the traffic condition (e.g., rush hour) by creating trips between home and work places during a certain time period. As these moving objects contain multiple transportation modes, one can analyze the data and test whether the traffic jam can be relieved by improving and adjusting the public transportation system. One can also investigate people's movement inside buildings by generating indoor moving objects.

Complementary to the benchmark data generator, we design a set of queries to test a variety of operator constellations and data access methods. Two groups of queries are proposed where one deals with the underlying environments and the other considers moving objects. Most of the queries are not supported by existing methods for the reason that previous work is limited to one environment. In the baseline method, optimization strategies are developed to improve the query efficiency.

Another important aspect in the benchmark is that we develop an index structure named Mode-RTree to manage generic moving objects. We intend to develop general access methods that can minimize the cost of overall benchmark queries instead of a particular query. Such an index supports moving objects with different transportation modes. By thoroughly analyzing benchmark queries, we find a key step in the query processing. That is, the relationship between moving objects and the underlying geographic objects such as roads, buses and buildings has to be well maintained. To solve the problem, we propose a mapping method and integrate an integer in the tree node to build the relationship. During the query processing, we can prune objects that do not fulfill the query condition. As a result, the benchmark performance is further improved by employing the index.

With the aim of producing a practical and systematic result, the data generator, the proposed index and related query algorithms are all implemented in a real database system SECONDO [22]. We present a thorough experimental study for performance evaluation under comprehensive datasets. The results demonstrate the effectiveness and efficiency of the proposed method. Among all queries, the improvement is between 2 times and orders of magnitude in terms of CPU and I/O accesses using the large dataset. The evaluation is performed in SECONDO, as to our knowledge there is not yet any other available DBMS that can represent generic moving objects and execute the given set of queries. The contribution of the paper is summarized as follows:

- We develop a data generator to produce moving objects in different environments. Some rules are defined in order to create the data in a realistic way. One can generate a trip containing only one mode (e.g., an indoor trip) or several transportation modes. We present the algorithm of creating a trip passing through several environments with precise locations and transportation modes.
- A set of benchmark queries is carefully defined including 4 queries on geographic information and 17 queries on generic moving objects.
- In the baseline method for answering queries, some optimal techniques are developed to improve the query efficiency.
- We propose an index structure called Mode-RTree to efficiently manage moving objects in different environments and maintain the relationship between moving objects and the underlying geographic objects. By employing the Mode-RTree, the performance of benchmark queries is significantly improved.
- Extensive experiments are conducted to test the benchmark performance. The results confirm the efficiency and effectiveness of the proposed technique, and demonstrate that using the Mode-RTree substantially outperforms the baseline method, achieving orders-of-magnitude performance improvement.

The rest of the paper is organized as follows: A concise overview of related work is presented in Section 2. In Section 3, we elaborate the configuration of benchmark data and the generating algorithm. Benchmark queries are defined in Section 4. We introduce the baseline method in Section 5 and propose the index in Section 6. We perform the experimental evaluation in Section 7, discuss the advantage of GMOBench in Section 8 and conclude the paper in Section 9.

2 Related work

2.1 Moving objects with transportation modes

In the research literature, there is some work related to transportation modes on moving objects, which can be classified into two categories: (1) trip plannings considering transportation modes; (2) discover transportation modes from GPS data.

The paper [5] presents a data model to support querying a trip consisting of several transportation modes, e.g., *Bus*, *Walk*, *Train*. The authors deal with returning a shortest path (SP) with multiple transportation modes to connect the origin and the destination, where SP can have more constraints and choices, e.g., different motion modes, the number of transfers. An interesting query is proposed in [1] that computes isochrones in multi-model and schedule-based transport networks. The goal is to find the set of points on a road network, from which a specific point of interest can be reached within a given time span.

To provide more contextual information and enrich a user's mobility with informative knowledge, Zheng et al. [57, 58] develop a method based on supervised learning to automatically infer users' transportation modes. [38] creates a classification system that uses a mobile phone with a built-in GPS receiver and an accelerometer to determine the transportation mode of an individual when outside. Stenneth et al. [42] propose an approach to inferring a user's mode based on the GPS sensor on the mobile device as well as the knowledge of the underlying transportation network, e.g., bus stop locations, railway lines.

The above work is different from ours. The data model [5] does not represent the precise location and transportation modes for moving objects, but describes them conceptually and abstractly. They assume the data including trajectories and modes are already known in the database. Besides, indoor environment is not considered. In [1], they address the issue of query processing in a transportation system without involving moving objects, and only two transportation modes are supported: *Walk* and *Bus*. Transportation modes discovered from GPS data are only for outdoor movements because a GPS receiver will lose signal indoors. We do not focus on inferring motion modes. Based on a data model for generic moving objects [55], we evaluate the performance of a database system that supports a group of queries on moving objects in different environments.

2.2 Benchmarking

A benchmark proposed in [50] deals with 3-dimensional spatio-temporal data that require significant temporal processing and storage capabilities, and has provisions for evaluating the ability of a spatio-temporal database to handle 3-dimensional data. The work expands on the Sequoia 2000 and Paradise benchmarks, and is oriented towards general operating system and database system performance comparison. In the context of moving objects databases, [46] proposes a benchmark that includes a database description and a group of representative SQL-based queries. Ten benchmark queries plus two operations for loading and updating data are proposed. The authors give an ER diagram of a database for location-based services where the entities include humans, buildings and roads. Humans visit buildings (shops) for their interests and requests on products. Each road stores two kinds of data: (1) a polyline; (2) the time when people pass the road. But the paper does not present a method to create the benchmark database and there is no performance evaluation.

BerlinMOD [14] is a benchmark that uses the SECONDO DBMS [22] for generating moving object data. A scenario is simulated where a number of cars move within the road network of Berlin and sampled positions from such movements are used as the data. The method models a person's trips to and from work during the daytime on workdays as well as some additional trips in the evening and on the weekend. Long-term observations of moving objects are available, e.g., a month. A set of carefully selected SQL-based queries constitute the workload. However, these benchmarks only process moving objects and queries in one environment and do not consider transportation modes. Compared with the aforementioned work, our benchmark is general in the context of moving objects where the system manages trips passing through different environments and supports new queries on these data.

Benchmarking moving objects indices is studied in [11, 25, 30], focusing on location update, current and near future positions. [30] presents a benchmark termed DynaMark for dynamic spatial indexing, that is towards location-based services. Three types of queries are defined that form the basis of location-based service applications: proximity queries, *k*NN queries and sorted distance queries. A benchmark called COST [25] is concerned with the indexing of current and near-future moving objects positions and aims to evaluate the index ability to accommodate uncertain object positions. Three types of queries are investigated, timeslice query, window query and moving window query. In [11], three types of datasets are generated: (1) uniform distribution; (2) Gaussian distribution and (3) road-network-based datasets. The goal is to measure the overall index efficiency and to simulate certain real-world scenarios. The query workloads consist of the range query and the *k*NN query. A set of aspects is proposed for the performance evaluation such as data size, update frequency and buffer size.

2.3 Spatio-temporal data generators

The so far developed tools have been using random functions and road networks to model different physical aspects of moving objects. A network-based moving objects generator is proposed in [6, 7] for the traffic application. Objects are created in a random way and appear and disappear when their destinations are reached. Important concepts of the generators are the maximum speed, the maximum edge capacity, etc. Two kinds of methods are used for setting positions and velocities for moving objects generation [40], uniform distribution and skewed distribution.

GSTD [47], a widely used spatio-temporal generator, defines a set of parameters to control the generated trajectories: (1) the duration of an object instance; (2) the shift of objects and (3) the resizing of objects. Initialized by a certain distribution of points or rectangle objects, GSTD computes at every time step the next position and the shape of objects based on parametrized random functions. Later, the generator is extended to produce more realistic moving behavior such as group movement and obstructed movement, by introducing the notion of clustered movement and a new parameter [35]. G-TERD [49] is a generator for time evolving regional data in an unconstrained space and Oporto [39] is a realistic scenario generator for moving objects motivated by a specific application, fishing. A set of rules is defined to create indoor moving objects in [56], like an object in a room can move to the hallway or move inside the room. In summary, these generators only consider a single environment and cannot generate moving objects passing through different environments.

There are also various spatio-temporal simulators for different applications. ST-ACTS [19] is a simulator that uses geo-statistical data sources and intuitive principles to model *social* and *geo-demographic* aspects of human mobility. The model is based on commercial source data describing some statistics of Denmark's population. Some principles are defined to govern the social aspects of mobility, e.g., home-work and home-school. STEPS [31] is a parametric mobility model for human mobility, which makes abstraction of spatio-temporal preferences in human mobility by using a power law to rule the movement. The work focuses on human geographic mobility and defines the human mobility as a finite state. The mobility is modeled by a discrete-time Markov chain in which the transition probability distribution expresses a movement pattern. GAMMA [23] is a framework in which trajectory generation is treated as an optimization problem and solved by a genetic algorithm. Two examples are given to show how to configure the framework for different simulation objectives, in a cellular space and a real-life symbolic moving behavior. However, the aforementioned methods do not consider the detailed routing between locations, that is, how people move from one place to another. As a result, transportation modes and movement environments are not addressed.

The goal in [28] is to provide a friend-finder service. The considered places are not general, including only home and entertainment places. Also the observed time periods are limited to Friday and Saturday nights. The method sets some parameters for the simulation, like source, destination, and starting time. In order to have a realistic model of distributions, a survey is prepared to collect the data of real users based on interviews of more than 300 people. In the simulation, home places are distributed almost uniformly on the map, with a minor concentration on central zones of the city.

2.4 Indices on trajectories

In the last decade, a substantial amount of index structures have been proposed to efficiently access trajectories. A good survey on trajectory indexing and retrieving is given in [26, 59]. 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 [34], TB-Tree and STR-Tree, assuming that the motion is piecewise linear, where TB-tree is to bundle segments from the same trajectory into leaf nodes. MV3R [44] 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 the spatial and temporal dimensions. The Multiple TSB-tree is proposed in [60] to support the historical and spatial range close-pair queries for moving objects. The paper [32] 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 [12, 15, 33] where the first level is for roads and the second is for trajectories. The paper [36] proposed an index called T-PARINET, which is a structure 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 [24, 27] are also developed to support range and nearest neighbor queries over indoor objects. Xie et al. [52] propose a composite index scheme that integrates indoor geometries, indoor topologies and indoor uncertain objects to support indoor distance-aware queries efficiently.

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

3 Benchmark data

3.1 Data model

We let the *space* for generic moving objects be covered by a set of infrastructures (environments), each of which corresponds to an environment and contains its possible transportation modes. A notation is defined for each infrastructure, listed in Table 1. Transportation modes are summarized in Definition 3.1.

Table 1 Components for space

Space	I_{rn}: Road network	*Car, Taxi, Bike*
	I_{rbo}: Region-based Outdoor	*Walk*
	I_{bn}: Bus Network	*Bus*
	I_{mn}: Metro Network	*Metro*
	I_{indoor}: Indoor	*Indoor*

Definition 3.1 Transportation Mode
TM = {*Car*, *Walk*, *Indoor*, *Metro*, *Taxi*, *Bike*, *Bus*}

Each infrastructure consists of a set of infrastructure objects (IFOBs) representing available places for moving objects. For example, streets and roads constitute I_{rn} and polygons representing pavement areas compose I_{rbo}. The bus network I_{bn} comprises bus routes, bus stops and moving buses. The location of a moving object is represented by referencing to the underlying IFOBs. We give the definition below.

Definition 3.2 Generic Location

$$D_{\underline{genloc}} = \{(oid, (loc_1, loc_2)) | oid \in D_{\underline{int}}, loc_1, loc_2 \in D_{\underline{real}}\}$$

A generic location consists of two attributes with oid being an IFOB id and (loc_1, loc_2) describing the relative position according to that object. The representation has different semantics according to the infrastructure characteristic. For example, in a road network oid is a route identifier and $(loc_1, \perp)$ records the relative location on the route. Given a location in I_{rbo}, oid maps to a polygon and (loc_1, loc_2) represents the location inside the polygon. In spite of various data types for IFOBs such as line and region, we make an abstraction for them and only keep the object identifier to let the location model be simple.

To represent moving objects with transportation modes, we denote a generic trajectory by $tr = < u_1, u_2, ..., u_n >$, that is a sequence of temporal units ordered by time where each unit defines the movement during a time interval. In detail, we have

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

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. Consider such an example movement: *Car* $\rightarrow$ *Walk* $\rightarrow$ *Indoor*. The units record ids for (1) roads; (2) pavement areas; (3) rooms. The precise locations are identified by gl_1 (gl_2).

The method represents a moving object in a compact way. Two units u_i, u_j are merged into one if they fulfill the conditions: (1) $u_i.i$ and $u_j.i$ are *adjacent*; (2) $u_i.m = u_j.m$; (3) u_i and u_j reference to the same IFOB and the linear functions are the same. Some examples are shown in the following. (1) locations for a car (taxi or bicycle) moving on a road segment with a constant speed are compressed into one unit denoting (i) the road id and (ii) the positions of endpoints for such a segment; (2) locations for a bus (metro) traveler are not explicitly recorded but reference to the bus (metro) and this can avoid the redundant data for people who take the same bus. Consequently, the storage size for moving objects is greatly reduced.

3.2 Movement principles

In order to create realistic benchmark data, we define a set of movement rules that can reflect the characteristics and distributions of human behavior in practice. Human movement has a certain distribution in terms of time and location, and in most cases people move from one place to another with the objective of performing a certain activity at the target place. We define in total four movement rules that consider both (1) *physical* and (2) *social* aspects of mobility. The former considers a geographical impact, e.g., location, distance, while the latter regards human community behavior, preferences or habits. We believe that both *physical* and *social* factors are essential for creating the benchmark data and are able to mirror key features of human movement. The four rules are defined as follows.

- **MR1**. This rule is to represent *regular movement* [17, 20], which usually has a periodic pattern. A large majority of people go to work in the morning and come back in the evening. There are additional trips between work places during the office time, as people may travel to another place for business or conference meeting. Evidently, these regular trips occur on weekdays in most cases.
- **MR2**. Movement in a *single* environment. For example, people walk around in the city center (pedestrian areas) for shopping on the weekend, and a clerk moves from his office room to the conference room. This rule is used to generate a short trip limited to one environment.
- **MR3**. Motivated by the famous *nearest neighbor query* [16, 29, 45], we create a trip from the query location to the closest qualified location. For instance, a person wants to find the nearest hospital. If the target place is a short distance away from the query location, the traveler can go by walking. Otherwise, he may wish to travel by the public transportation system. In this case, the closest bus stop is found and then he travels by bus. Another example could be a car searching the nearest gas station. Trips generated by this method are based on the physical aspect, i.e., distance.
- **MR4**. A trip is triggered by the purpose of visiting a point of interest. The concept of an interesting location is general, a personal apartment, a sightseeing place, a restaurant, etc. On the weekend, people may visit friends, go shopping or meet in a park.

Compared with **MR1**, trips defined by **MR2**, **MR3** and **MR4** can be considered as irregular movement, but they occur frequently in daily life. There is a large amount of such trips, especially for **MR3**. An extensive study having been done on *NN* queries in moving object databases, confirms that this query is fairly common and widely used in daily life, motivating us to define a rule for the movement performing such an activity. Trips generated by **MR1**, **MR3** and **MR4** may pass through different environments, leading to multiple transportation modes.

The trips created by above rules reflect the most common movement of humans, leading to the generated data in a realistic scenario. The purpose of defining precise rules is to generate the data in a well defined method and a clear motivation. The method is flexible and one can add more rules to generate desired and interesting trips, e.g., a traveler passes a sequence of places at the minimum cost.

3.3 Parameters

To create a generic moving object, three parameters have to be configured: (1) **Location**; (2) **Time**; (3) **Transportation Modes**.

Location plays a key role for creating trips for the reason that it specifies where the trip starts and ends. No restrictions are made for the start and end locations of a traveler, i.e., they can be in any environment defined in Table 1. In particular, the location is related to an IFOB, e.g., a road or a bus. If the start and end locations are in distinct environments, a trip involving different transportation modes is created. In the following, we show how the location parameter is specified to create desired trips. Stated in Section 3.1, the whole space is partitioned into a set of infrastructures each of which consists of a set of IFOBs. A unique number is assigned to each IFOB, and each infrastructure has a range of integers denoting its element ids.

Definition 3.3 Integer Sets for IFOB ids

Let ID(I_i) denote the set of IFOB ids for one environment, $I_i \in \{I_{rn}, I_{rbo}, I_{bn}, I_{mn}, I_{indoor}\}$. Then, the overall integer set for IFOB ids is represented by IO_ID = $\bigcup$ ID(I_i).

We perform the union on IFOB ids for all environments. There is no overlapping between integer sets from different infrastructures. Since the location representation (Definition 3.2 in Section 3.1) records an IFOB id, one can set an integer range denoting certain IFOBs. The range determines the environment for the start and end locations. The parameter is defined below.

Definition 3.4 Start and End Locations

A pair $l(l_s, l_e)$ denotes the start and end locations of a trip fulfilling the condition:
(i) $l_s = (gl_s, c_s), l_e = (gl_e, c_e)$ $(gl_s, gl_e \in D_{\underline{genloc}}, c_s, c_e \subseteq$ IO_ID);
(ii) $gl_s.oid \in c_s \wedge gl_e.oid \in c_e$.

The location parameter is represented by a pair of objects defining the start and end locations as well as location domains, c_s and c_e. They are two sets each of which designates an integer range for the location. That is, the environment for gl_s (gl_e) is determined by c_s (c_e). For example, if $c_s = ID(I_{rn})$, the start location is on a road. If $c_s = ID(I_{rbo})$, gl_s is located in the pavement area. For the case $c_s = c_e = ID(I_{indoor})$, there are two possibilities: (1) gl_s and gl_e are in the same building; (2) gl_s and gl_e are in different buildings. Case (1) is simple. Case (2) implies a trip from one building to another, e.g., from home to office.

By defining c_s and c_e, one can set up the location in a flexible way. $c_s(c_e)$ can be either (i) a range indicating a set of ids or (ii) a set with a single value denoting a specific object id. For case (i), a concrete value belonging to the given set needs to be specified in order to let the trip start from a precise IFOB. We let such a value be randomly chosen from the set. For example, if $c_s = ID(I_{rn})$, a stochastic road is selected. For case (ii), the method is able to create a trip starting from (ending at) a specific IFOB, e.g., a road or a building. In the above two cases, when a concrete IFOB is determined, we randomly choose a location belonging to the IFOB and let it be the accurate start (end) location. For instance, if a road is chosen, we let a stochastic position on the road be the start (end) location of a trip. If a building is selected, a random location inside a room is chosen.

Time In principle, a trip can start at any time instant. But human movement has a certain time distribution that most trips occur in the range [6:00, 22:00] [18].

Definition 3.5 Start Time of a Trip

Let Hour = {0,1,...,23} and Min={0, 1,...,59} be two sets of integers.

The start time of a trip is defined as a four-tuple t(h,min, hc, minc) where

(i) h ∈ hc ⊆ Hour;
(ii) min ∈ minc ⊆ Min.

The two attributes h and min define the start time of a trip and each value is set according to its corresponding domain. We have *hc* for *h* and *minc* for *min*. One can generate the desired trip on time distribution by configuring *hc* and *minc*. For example, to create a trip from home to work place in the morning, the time parameter can be set by *hc* = {6, 7, 8} and *minc* = *Min*. We let the concrete value for h and min be uniformly distributed in the defined sets *hc* and *minc*. This method is general and effective, and is able to satisfy different requirements as one can shrink or enlarge the ranges for *hc* and *minc* to get certain distribution. For example, we can set *hc* = {8} and *minc* = {0, 1, ..., 30} to create a trip whose start time is between 8am and 8:30am.

Modes People have different choices on vehicles for their traveling, by car or using public transportation system. If the traveling distance is short, the mode *Walk* or *Bike* can also suffice. A trip can contain a single mode or multiple modes. The value depends on the start and end locations to some extent. Regarding the locations, we make the following assumptions for the transportation modes involved by a trip.

- Assumption 1. If l_s and l_e are located in the same building, we define the movement to be inside such a building. That is, the mode is only *Indoor*.
- Assumption 2. If l_s and l_e belong to the same outdoor environment such as I_{rn} or I_{rbo}, then the mode is single and determined by the environment, e.g., *Car* in I_{rn}.

Based on the two assumptions, the case that a trip contains different transportation modes occurs when (1) l_s and l_e are located in different buildings; or (2) l_s and l_e belong to different infrastructures. Otherwise, the modes are determined.

We investigate the components of transportation modes in a trip and find out the following behavior. First, *Walk* is the only mode that connects to another one, e.g., *Walk* → *Car*, *Indoor* → *Walk* → *Bus*. We define that the connection between two modes such as *Car* → *Indoor* and *Indoor* → *Bus* is not allowed. Usually, people do not directly change from *Bus* to *Car*, or from *Indoor* to *Taxi*. A short distance of walking is required. This is consistent with the result from [57] in which the authors infer transportation modes from raw GPS data and find that the walk segment is up to 99% as the transition between different transportation modes. Second, among outdoor transportation modes, *Car* and *Bike* are private vehicles, while *Bus*, *Metro*, and *Taxi* are public. In most cases, people travel by either private vehicles or public transportation system. It's rare that both vehicles are involved in one trip. Consequently, we distinguish between the two cases when produce trips. To sum up, we do the following partition on transportation modes based on Definition 3.1.

Definition 3.6 The Partition of Transportation Modes
Let A = {Walk} and B1={Indoor}. Public and private vehicles are defined to be B2={Bus, Metro, Taxi}, B3={Car, Bike}. Thereby, TM = A ∪ B1 ∪ B2 ∪ B3.

Note that two modes can only be connected via *Walk* and there is no mode switch between elements from $\bigcup_{i=1}^{3} B_i$. The purpose of performing the partition is to make a clear recognition on transportation modes in a trajectory. One can figure out the implicit relationship between modes and define possible modes for a trip. This will benefit the procedure

Table 2 Transportation mode transition

	A	B1	B2	B3
A	1	1	1	1
B1	1	1	0	0
B2	1	0	1	0
B3	1	0	0	1

of producing a trip containing reasonable transportation modes, making the data practical. Table 2 gives the transition matrix for TM. A is transitive to all the others, while B1, B2, B3 are only transitive to themselves and A.

Definition 3.7 Transportation Modes in a Trip
The mode is a set $mc \subset$ TM to specify possible values included by a trip.

See some instances: (1) mc = B1, movement inside a building; (2) mc = B2 ∪ A, e.g., take a bus and then a short distance walking; (3) mc = A ∪ B1 ∪ B3. It is worth noting that if the mode is already determined by l_s and l_e (the two assumptions above), mc is decided and does not have to be specified.

3.4 Configure parameters

In this part, we show how to set the three parameters to create trips simulating human movement. At first, we divide the integer set $ID(I_{indoor})$ representing buildings into three subgroups {H, W, SE}, where H refers to the set for personal houses, W denotes the ids for work buildings such as office buildings, universities, and SE (Shopping and Entertainment) is for buildings like shopping malls, cinemas. If a trip starts from or ends in the indoor environment (a building), we use the subgroup to determine the type of the building. H, W and SE are also used to create certain trips according to the rule. For example, we can set the location domain by H ∪ W for **MR1** to represent regular movement between home and work. The trip representing friends visiting on the weekend (**MR4**) can set H for both start and end locations.

A trip is an optimal route with respect to the minimum cost (e.g., distance, time) from one location to another and the result is used to create a generic moving object. The start and end locations can be in one environment or in different environments, where the latter involves multiple transportation modes. For each rule, a set of movement instances is specified to create concrete trips.

Table 3 lists some instances for each rule as well as the parameter settings. For simplicity, we only show the domain for each parameter. In some cases, transportation modes are in fact chosen by the location (see **MR2**). Otherwise, mc is variable and can be specified according to the requirement. For example, people can drive by car or use the public transportation system to travel between home and work. The values in the table are possible settings.

3.5 Trip generation algorithm

We outline the algorithm that generates a trip with different transportation modes. First, a set of graphs is introduced. A trip is created based on the shortest path (SP) where the start and end locations can be located in different environments, resulting in a set of sub trips

Table 3 Example movements and parameter settings

Rule	Movement instance	Start and end locations (c_s, c_e)	Start time (hc)	Modes (mc)
MR1	1) travel from home to work	H, W	[6, 9]	A ∪ B1 ∪ B2
	2) business travel between working places	W, W	[9, 16]	A ∪ B1 ∪ B3
MR2	1) people walk around in the city center	$ID(I_{rbo})$, $ID(I_{rbo})$	[10, 18]	A
	2) a clerk walks from one office to another	W,W	[9, 16]	B1
MR3	1) a car searches the nearest restaurant	$ID(I_{rn})$, $ID(I_{indoor})$	[10, 20]	A ∪ B3
	2) a traveler looks for the nearest bus stop	$ID(I_{rbo})$, $ID(I_{bn})$	[9, 20]	A
MR4	1) go to a shopping mall from home	H, SE	[10, 20]	A ∪ B1 ∪ B3
	2) visit friends	H, H	[10, 20]	A ∪ B2

in several infrastructures. In the sequel, different trip planning algorithms are needed, e.g., indoor navigation, trip planning for pedestrians, routing in bus network. Figure 1a lists all infrastructure graphs.

Second, to create a trip passing through a set of environments, location mapping technique is required to convert positions from one system to another. Consider the case that a pedestrian searches the nearest bus stop. To answer such a query, at first the location of the qualified bus stop is mapped to the pavement. This is because the representation of a bus stop is not simply a point but contains some other information such as the bus route id, the relative order on the route. Then, the shortest path from the query location to the bus stop is returned where the path is located in the walking area. Since a walking segment is the part connecting movements in different environments, the location mapping is actually between I_{rbo} and the other infrastructures. We denote the mapping by $M(I_{rbo}, I')$ ($I' \in \{I_{rn}, I_{bn}, I_{mn}, I_{indoor}\}$) and explain the procedure as follows.

Several data types are defined to represent IFOBs. To be more specific, the location of a bus or metro stop is defined by a point, roads are defined by spatial lines, pavements are defined by polygons, and indoor objects such as rooms and staircases are represented by proposed data types (see [55]). The mapping between each I' and I_{rbo} is actually to create a connection between two spatial objects:

1. I_{rbo} and I_{rn}: Given a location on a road, we take this point and find the closest point to it from the pavement. That is, we calculate the closest distance between a point and

Notation	Meaning
G_{rn}	Road Graph
G_{rbo}	Pavement Graph
G_{bn}	Bus Network Graph
G_{mn}	Metro Network Graph
G_{indoor}	Indoor Graph

(a)

Name	Meaning
v_r	car speed
v_m	metro speed
v_w	walking speed

(b)

Fig. 1 Infrastructure graphs and defined speed

a region. Such a pair of points is used to build the connection and denote two locations in their corresponding environment. Whenever the movement changes from I_{rn} to I_{rbo}, the point inside a region is chosen as the start location of the next trip. Conversely, we receive a location on the pavement and find the closest point to it from the road, i.e., calculate the closest distance between a point and a line.

2. I_{rbo} and I_{bn}: Since lines representing bus routes are from the road network, the mapping procedure is the same as above. A set of bus stops is identified and their mapped locations in I_{rbo} are stored in the database. Later, query processing can make use of the result if two environments are to be connected, e.g., a pedestrian searches the nearest bus stop.
3. I_{rbo} and I_{mn}: The location of a metro stop is represented by a point. We find the closest point from the pavement to the metro stop and choose it as the result. This is performed by computing the distance between a point and a region. We put a metro stop with its mapped point in I_{rbo} into a tuple and store all metro stops in a relation in the database.
4. I_{rbo} and I_{indoor}: The 2D area of a building in space is represented by a polygon. The location of the entrance/exit of a building is defined to a point inside a polygon. To perform the mapping, we find the closest pair of points between two regions, where one is for the building and the other is for the pavement. If a traveler leaves the building, the point inside the building polygon is the end of the indoor trip and its mapped location inside the pavement is the start location of walking. In the reverse way, when a traveler visits a building, we get the building entrance and choose the mapped location in I_{rbo} as the end location of walking, and then the traveler changes to *Indoor*.

Third, a set of speed values are defined. We summarize them in Fig. 1b. Each road is assigned a value as the maximum speed allowed for cars. Such a value is also used for the bus moving on the road. v_m is the speed for metros. The walking speed for both indoor and outdoor is specified by v_w. Without loss of generality, we use the first movement instance of **MR1** in Table 3 to describe the procedure of the algorithm.

Step 1: Assuming that the traveler uses the bus network, the corresponding graphs $\{G_{rbo}, G_{bn}, G_{indoor}\}$ are selected according to transportation mode parameters.

Step 2: Let s be the home location of the traveler and the nearest bus stop to s is found. $M(I_{rbo}, I_{bn})$ returns the pavement location for the bus stop, denoted by bs. By running the SP algorithm on G_{rbo}, a walking path l_1 from s to bs is obtained. We create an object $< l_1$, *Walk*$>$ and put it into the path set P.

Step 3: We execute the SP algorithm on G_{bn} to find a bus connection to the destination stop. Let l_2 be the bus path and the pair $< l_2$, *Bus* $>$ is inserted into P. If a short walking is included for changing the bus, such a path is also put into P with the mode *Walk*.

Step 4: $M(I_{rbo}, I_{bn})$ maps the destination stop to the pavement location be. Again, we use G_{rbo} to find the SP from be to e, the building entrance (maps to I_{rbo}). l_3 denotes such a walking path and we insert $< l_3$, *Walk*$>$ into P.

Step 5: We run the SP algorithm on G_{indoor} to find an indoor shortest path l_4 from e to the final destination, e.g., an office room. The last sub path as well as the transportation mode is collected and we put $< l_4$, *Indoor*$>$ into P.

Step 6: For each $p_i \in P$, we take the corresponding speed value from Fig. 1b and create the sub movement.

In the end, we perform the union on all sub trips to get the complete trip, the definition of which is given in Section 3.1.

4 Benchmark queries

4.1 Query list

We present a set of interesting queries to form the benchmark workload, categorized into two groups. One requests data from infrastructures and the other deals with generic moving objects. We provide the formulation for all queries by using an SQL-like language in the Appendix A.

Infrastructure Queries.

- **Q1.** Find out all metros passing through the city center.
- **Q2.** Given a building, find all bus stops that are within a radius of 300 meters.
- **Q3.** Which streets does Bus No. 12 pass by?
- **Q4.** Where can I change between bus route No. 16 and No. 38? Where can I change between metro route No. 2 and No. 8?

The first three are concerned with interactions between different infrastructures as users may compare IFOBs from different environments. In a public transportation system, travelers need to know the place where they can switch between different routes.
Queries on Generic Moving Objects.
We consider the following aspects: (1) referenced IFOBs by moving objects; (2) transportation modes; (3) time intervals; (4) spatial data and locations.

- **Q5.** At 8am on Monday, who sits in the bus No. 32? At 8am on Monday, who sits in the metro No. 2?

This query deals with travelers who take the bus (metro) from a specific route at the given time and covers three aspects above: (1), (2) and (3). The query result is not the case for a particular bus or metro, but all available buses (metros). At the query time, there may be several buses (metros) moving on the route.

- **Q6.** What is the percentage of people traveling by public transportation vehicles?

For this query, we define the the modes {*Bus*, *Metro*, *Taxi*} to be the case of traveling by public transportation vehicles.

- **Q7.** Where does *Bobby* walk during his trip? And how long does he walk?
- **Q8.** Find out all people passing room 312 at the office building between 9am and 11am on Monday.

To answer **Q7, Q8**, we need to get a sub trip according to the transportation mode.

- **Q9.** Who arrived by taxi at the university on Friday?
- **Q10.** Who entered bus No.3 at bus stop "University" on Tuesday afternoon?

The above two queries deal with interactions between different environments. To answer these queries, one should find the place where people change transportation modes.

- **Q11.** Find out all people walking through zone *A* and zone *B* on Saturday between 10am and 3pm.

- **Q12.** Did anyone who was on floor H-5 of the office building between 2pm and 5pm take a bus to the stop "train station" on Friday?

- **Q13.** Did bus No. 35 pass by any bicycle traveler on Monday?

Q13 considers the distance between two moving objects with different transportation modes. It is interesting to discover such a relationship as the two objects move in different environments. The bus parameter means a group of buses which all belong to the route No. 35 but with different departure time, instead of a particular bus trip.

- **Q14.** Did someone spend more than 15 minutes on waiting for the bus at the bus stop Cinema on Saturday?
- **Q15.** How many people visit the cinema on Saturday?
- **Q16.** Find out all people staying at room 154 in the university for more than one hour on Thursday.

Besides the outdoor trips, indoor moving objects can also be traced.

- **Q17.** Did someone spend more than one hour traveling by bus (metro)?
- **Q18.** Find out all people changing from bus No. A to bus No. B at stop X. Find out all people changing from metro No. A to metro No. B at stop Y.

In a public transportation system, knowing the place where people do the transfer is meaningful to analyze and investigate the schedule and route distribution.

- **Q19.** Find out all trips starting from zone A and ending in zone B by public transportation vehicles with the length of the walking path being less than 300 meters.

We search trips by considering the start and end locations as well as transportation modes. A and B are defined to be a set of locations, represented by regions. They can be areas with high road density, implying a large number of residences. The query is helpful for travel recommendation. Meaningful knowledge can be discovered from past movements.

- **Q20.** Find the top *k* bus (metro) routes with high passenger flow for all workdays.
- **Q21.** Find the top *k* road segments with high traffic during the rush hour for all workdays.

By analyzing the histories of movements, routes with high passenger flow can be discovered and the schedule in a public transportation system can be adjusted to improve the traffic. For **Q21**, the traffic value of a road segment is set as the number of moving objects passing by during the rush hour ([7:00, 9:00] ∪ [16:00, 18:00]), where the following transportation modes are considered: {*Car*, *Taxi*, *Bicycle*, *Bus*}.

4.2 Discussion

One motivation of setting the above queries in the benchmark is from the application. As a kind of human behavior, transportation modes could help understand individuals' daily life in a deep way and this involves an interesting range of moving objects applications. For some recommendation services, it would be better for the system to provide trip plannings with different modes as users may have their own interest. This brings the task to capture trajectories with contextual data in the database, and the system should be capable of answering new queries to provide advanced services. Transportation modes are closely related to environment, leading to a group of queries asking for the relationship between trajectories and IFOBs. Knowing exact transportation modes would enrich mobility with informative knowledge and establish a deep understanding of movement. The proposed queries have application examples in real world, making the benchmark *understandable*.

GMOBench is a domain-specific benchmark, settled in the scenario of moving objects with multiple transportation modes. To our knowledge, majority of queries in the benchmark are not supported by existing methods or other systems. Therefore, efficiently answering those queries is essential for databases, in particular, moving objects databases. Part of queries in the benchmark is from [55] in which we propose a group of queries to test the expressiveness of data model in terms of data types and operators. New queries complement the benchmark to test typical operations within the problem domain. This is called *relevant*, which is one key requirement for domain specific benchmarks [21]. To demonstrate this, we provide the formulation of benchmark queries in the appendix. Additionally, most queries are designed in such a way that they are amenable to optimization strategies.

The detailed knowledge of data characteristics plays a crucial role in query design. The transportation mode of a moving object changes over time and such temporal data is represented by moving int. In this setting, one can ask queries such as what is the integer during a time interval, at which time the integer changes and list all integers in life time. Similar queries are defined in the benchmark that find trajectories whether a particular mode exists in the time interval, when the mode changes and whether certain modes are involved. Solely investigating transportation modes is not sufficient (too simple) and hence the location is requested in the meantime. For example, we are interested in knowing when a traveler switches from *Walk* to *Bus*, and at which bus stop.

Ultimately, we target measuring the cost of common operator constellations and access patterns as testing the implementation in a system needs a broad class of representative queries. Our queries vary in general characteristics such as *selectivity*, *join*, *aggregation* and *extraction* (retrieve a subtrip with a certain mode). Typical relationships like *pass*, *intersect*, and *distance* as well as new defined operations (e.g., *at*) are also investigated. The benchmark characterizes complex queries that a sequence of modes as well as defined locations are issued, for example, **Q10** and **Q12**. However, a thorough theoretical analysis on the completeness of benchmark queries is not available. There are some popular queries in moving objects databases, but not considered in this paper. For example, nearest neighbor query is not included because we do not identify an application in the context of transportation modes.

5 Baseline method

In this section, we introduce some optimal techniques based on performing a linear search on generic moving objects, i.e., without using a Mode-RTree.

5.1 Transportation mode access

By observing and analyzing involved operators for queries, there is a frequently called procedure in the benchmark workload. That is checking whether a transportation mode is included by a trajectory. Since a large part of queries specifies a transportation mode (e.g., **Q7**, **Q9**, **Q16**), one needs to find all qualified moving objects. To get the result, one option is to sequentially scan all units, comparing the mode attribute to see if the value is identical to the argument. However, this yields a linear searching for each trajectory. The ability to determine the existence of a mode in a fast way can reduce the overall running time.

We optimize the procedure as follows. An integer IM (32 bits) is assigned to each trajectory to denote the involved transportation modes where a bit indicates whether a mode exists or not. Each transportation mode is assigned a value as an index to locate the corresponding bit in IM. We mark the bit *true* if the mode exists and *false* if it does not exist. Suppose that the least significant (right-most) seven bits are used for the modes in Definition 3.1. We assign 0 for *Car* as the bit index, 1 for *Bus*, and so on. A moving object with the following sequence of modes *Indoor* → *Walk* → *Bus* → *Walk* defines the value 14 (binary 0001110). We calculate IM for each trajectory in advance and store it as an attribute along with the trajectory in a tuple. Later, if a query needs to determine the existence of a mode, both the query mode and the integer are taken as input. The index for the bit denoting the mode is calculated, and the stored integer is examined.

Assume that there are N trips stored in the database and one trip contains M units on average. The original method needs NM times of unit access to find all qualified trips. With the optimization technique, we can achieve the result by N integer comparisons.

5.2 Index on units

Recall that a moving object is represented by a set of units arranged in a linear order on time (Section 3.1). To retrieve a sub trip with a certain mode, at present we need to linearly traverse all units to check the value and return the qualified data. For instance, to answer **Q5** the movement with the mode *Bus* is returned, and in **Q7** walking trips are specified. For travelers who take public vehicles, one might get the sub trip on a particular bus (metro), seeing **Q10** and **Q18**. Obviously, the sequential scan yields poor performance for large data. This motivates us to build an index in order to access units according to transportation modes in a fast way.

Let $I = \{(m, l, h) | m \in \text{TM}, l, h \in D_{\underline{int}}\}$ be the index built on the units from each tr. Each element in I consists of three attributes where m records the transportation mode and l, h are entries pointing to the start and end locations of a sequence of units with the mode m. That is, each element maps to a sub movement with a single mode. For example, we have such an example movement

$$\begin{aligned} tr = \; & < (\mathit{Indoor})_1, (\mathit{Indoor})_2, ..., (\mathit{Indoor})_{10}, \\ & (\mathit{Walk})_{11}, (\mathit{Walk})_{12}, ..., (\mathit{Walk})_{15}, \\ & (\mathit{Bus})_{16}, (\mathit{Walk})_{17}, ..., (\mathit{Walk})_{20} > . \end{aligned}$$

The index built on tr is $I = \{(\mathit{Indoor}, 1, 10), (\mathit{Walk}, 11, 15), (\mathit{Bus}, 16, 16), (\mathit{Walk}, 17, 20)\}$. Although I is still a list structure, the quantity of elements depends on the number of modes, usually quite few. Each element locates the range of units with a certain mode. Consequently, given a mode m we first scan the index to determine the positions of qualified units and then access them to get the concrete data. The advantage is a large number of units that do not fulfill the condition can be skipped by visiting I.

5.3 Project the movement

The above two methods apply to almost all queries on moving objects, whereas we also develop a technique to improve the efficiency of a specific query (**Q13**) that originally needs a long processing time. To answer such a query, the procedure maps the following moving objects into free space: (1) moving objects with the mode *Bike* and (2) a set of buses belonging to one route, with the aim of computing the distance in the same environment. The distance value is represented by a moving real which consists of a sequence of units. Each unit describes the distance function in a time interval. We briefly introduce the procedure of calculating such a value.

Let mo_B and mo_b denote a *Bike* traveler and a bus, respectively. Each object is represented by a moving point that contains a set of pieces, each describing a linear movement from one location to another during a time interval. Since the object moves in free space, the location is identified by a point. The algorithm first refines the units from both mo_B and mo_b to get two subsets that have overlapping time intervals, and then sequentially scans the subsets to calculate the value between each pair of qualified pieces (time intervals have intersections). This method deteriorates over long time and suffers from poor performance if the refined subset contains a large number of elements. For example, if mo_B and mo_b have almost the same period (perhaps a long time interval), then the procedure accesses nearly all units of mo_B and mo_b to compute the distance.

Let us consider the places where mo_B and mo_b can be located. All roads are available for mo_B, but mo_b only moves along the pre-defined bus route. Evidently, the case that mo_b passes mo_B can only occur when mo_B is moving on the road segment that the bus route maps to. In other cases, it is not necessary to calculate the distance even when the time intervals of mo_B and mo_b are overlapping. For example, mo_b can not pass mo_B if they are far away from each other.

Consequently, before running the costly algorithm of calculating the distance between two moving points, we first project the *Bike* movement to road segments that the bus route maps to. This leads to a small number of units for a moving point as unqualified movement pieces are pruned. Usually, the road segments on which a bike traveler moves and a bus route do not have too many intersections. Given a bus route, we map it to a set of road segments each of which is denoted by (rid, l_s, l_e), where rid ($\in D_{\underline{int}}$) indicates the road id and l_s, l_e ($\in D_{\underline{real}}$) refer to the start and end locations of such a road segment, respectively. Recalling the location representation in Definition 3.2 of Section 3.1, the value oid corresponds to a road if the moving object is a bike traveler, resulting in fast access to the road segment.

6 Mode-RTree

In this section, we present the Mode-RTree that is for indexing generic moving objects. First, the input data for the index is introduced in Section 6.1. Then, we discuss the index architecture in Section 6.2. The index feature in detail is presented in Section 6.3. We introduce how to build the Mode-RTree in Section 6.4. Finally, the query algorithm is described in Section 6.6.

6.1 Input data

Suppose we have two trips:

- *Bobby drives the car to the parking place, then walks to the office building, and enters his office room.*
- *Bobby walks from home to a bus stop, travels by bus to the stop near to the office building, then walks to the building and finally enters the office room.*

The above two trajectories can be described by:

(1) tr_1: <*Car*, *Walk*, *Indoor*>;
(2) tr_2: <*Walk*, *Bus*, *Walk*, *Indoor*>.

Recall Section 3.1: the location of a moving object is represented by referencing to the underlying IFOBs, each of which has an unique id. Let $SubTrip(tr, m)(m \in TM)$ return a subtrip of tr with the mode m. Then, $SubTrip(tr_1, Car)$ has the value like

$$< ((i_1, (3, (0.0, \perp)), (3, (203.0, \perp)), Car), ...$$
$$(i_n, (16, (124.0, \perp)), (16, (78.0, \perp)), Car)) >,$$

in which each unit represents the movement on a road segment during a time interval. The unit records the start and end locations on that segment by storing the road id (e.g., 3, 16) and the relative position on the road. Similarly, $SubTrip(tr_2, Bus)$ has the value

$$< ((i_1, (150290, (\perp, \perp)), (150290, (\perp, \perp)), Bus),$$
$$(i_2, (151230, (\perp\perp)), (151230, (\perp, \perp)), Bus)) > .$$

That means, the person first travels by bus (id = 150290) and then transfer to another one (id = 151230). We also have $SubTrip(tr_1, Walk)$ that receives a sequence of walking units, omitted here.

As benchmark queries request data from various environments, the index should be capable of distinguishing pieces of movements with different transportation modes. A complete trajectory consists of a set of subtrips, each of which corresponds to the movement in one environment. This motivates us to decompose a generic moving object into a set of pieces. Each piece represents a subtrip and fulfills the condition: (i) only contains one transportation mode; and (ii) references to one IFOB. That is, we aim to obtain a set of pieces, each of which represents the movement related to a particular IFOB. For example, the *Car* movement of *Bobby* on a specific road fulfills the condition. However, if the road name(id) of *Bobby* moves along changes, another subtrip is produced.

Given a generic moving object, the decomposing procedure is composed of two phases: First, based on modes we partition the trajectory into a sequence of subtrips, each of which contains only one transportation mode. Second, each subtrip is further split into pieces of movements and each part includes one IFOB id. Using the first trip above, we get (1) $SubTrip(tr_1, Car)$, (2) $SubTrip(tr_1, Walk)$, and (3) $SubTrip(tr_1, Indoor)$. Then, the result of splitting $SubTrip(tr_1, Car)$ is:

$$sub_1^1 = (i_1, (3, (0.0, \perp)), (3, (203.0, \perp)), Car)$$
$$sub_1^2 = (i_n, (16, (124.0, \perp)), (16, (78.0, \perp)), Car)$$

For the second trip, we first get (1) $SubTrip(tr_2, Walk)$; (2) $SubTrip(tr_2, Bus)$, (3) $SubTrip(tr_2, Walk)$, and (4) $SubTrip(tr_2, Indoor)$. Afterwards, we have the following pieces for $SubTrip(tr_2, Bus)$

$$sub_2^1 = (i_1, (150290, (\perp, \perp)), (150290, (\perp, \perp)), Bus)$$
$$sub_2^2 = (i_n, (151230, (\perp\perp)), (151230, (\perp, \perp)), Bus)$$

Based on those pieces of movements, we propose *movement units* defined in the following.

Definition 6.1 Movement Units

A movement unit is represented by $mu = (traj\_id, box, m, ref\_id)$ where *traj_id* ($\in \underline{D_{int}}$) records the trajectory identifier, *box* stores a 3D box built on spatial and temporal data, m ($\in TM$) refers to a transportation mode, and *ref_id* shows the referenced IFOB id.

In a movement unit, besides spatial, temporal and transportation mode, we record the referenced IFOB id for the purpose of building the connection between moving objects and underlying IFOBs. The value can be a road id, a bus id, or a building id. This piece of information is extremely useful for answering benchmark queries because majority of them take IFOB ids as the input parameter, e.g., **Q5**, **Q8**, **Q12**. For $\{sub_1^1, sub_1^2, sub_2^1, sub_2^2\}$, movement units are

$$\begin{aligned} mu_1 &= (1, box, Car, 3), \\ mu_2 &= (1, box, Car, 16), \\ mu_3 &= (2, box, Bus, 150290), \\ mu_3 &= (2, box, Bus, 151230). \end{aligned}$$

For simplicity, the precise data of the 3D box is omitted. We create a set of movement units for each generic moving object. The procedure is straightforward for IFOBs such as roads, buses and metros. In the following, we explain the method for other environments: I_{indoor} and I_{rbo}.

When processing indoor moving objects, the referenced id of the location is the combination of a building id and a room id (see Appendix B in detail). Given a trip inside a building, the room id keeps changing while the build id remains the same. Thus, when creating movement units we omit the room id and collect pieces of movements only based on the building id. That is, we group the movement inside a building as one movement unit instead of producing a set of units in order to have a compact representation.

Regarding walking units, the method is different from the others. We represent the overall area for walking in a city as a large polygon. To efficiently manage the large polygon, we decompose it into a set of triangles and each is assigned an unique id. The location for walking units is represented by referencing to a triangle and storing the relative location inside that triangle, with the left lower point of the bounding box being origin point. However, practically, user do not issue queries on these triangle ids which are hidden from the application level. We ignore the triangle id when processing walking units, but record the absolute location in space. A length threshold is defined to decompose the trajectory. If the length of a walking trajectory is larger than the threshold (e.g., 600m), we split the trajectory into pieces aiming to reduce the deviation of the spatial range.

In the end, we explain the difference between *movement units* and units in a moving object defined in Section 3.1. One could simply build a movement unit based on each u_i. However, this method does not have a compact representation as the trajectory referencing to an IFOB may be split into pieces. For example, several units may be needed to store in the *Car* movement on a particular road. Some effort is needed for maintaining the data and the produced result is trivial. Additionally, the quantity of produced movement units might be large. Employing the method above, the data size is reduced. Collecting movement units in all environments, we build the index structure on them.

6.2 Index structure

By conducting a thorough analysis on benchmark queries, we gain some important findings: (1) besides spatial and temporal data, transportation modes are also considered and play an important role when querying trajectories; (2) in most queries, users are interested in part of the movement, e.g., the bus trip or indoor movement, although the complete trajectory is maintained; (3) the query contains the information of a particular IFOB, which is referenced by moving objects. Through an analytical study on those behaviors, we aim to develop an index structure that can

1. manage spatial, temporal data as well as transportation modes for moving objects
2. distinguish pieces of movements with different transportation modes
3. be capable of maintaining the relationship between moving objects and referenced IFOBs

To achieve the goal, we propose an index structure called Mode-RTree to manage generic moving objects. The index is a two level structure (drawn in Fig. 2) in which the upper level contains a set of modes and records. For each transportation mode, we have a record storing the root node of a subtree. As a result, the upper level has two components:

$$(mode\_list, tree\_pointers).$$

mode_list is a list of transportation modes and *tree_pointers* is a list of records. Each transportation mode defined in Definition 3.1 has an item in *mode_list*. Correspondingly, a record for the root node of a subtree storing trajectories with that mode is put in *tree_pointers*. In Fig. 2, we show modes *Car* and *Bus* as well as the subtrees.

The lower level consists of a set of extended 3D RTrees (E3D-RTree for short). Comparing with the standard 3D R-Tree, an E3D-RTree is developed to integrate an integer per node to manage referenced ids for movement units. Thus, each node in an E3D-RTree (either a leaf or non-leaf) contains entries of the form (*MBR*, *ObjPtr*, *R*), where

- *MBR* is the bounding box for temporal and spatial data in the subtree
- *ObjPtr* is an array of pointers for child nodes or objects
- *R* is an integer representing referenced object ids in the subtree.

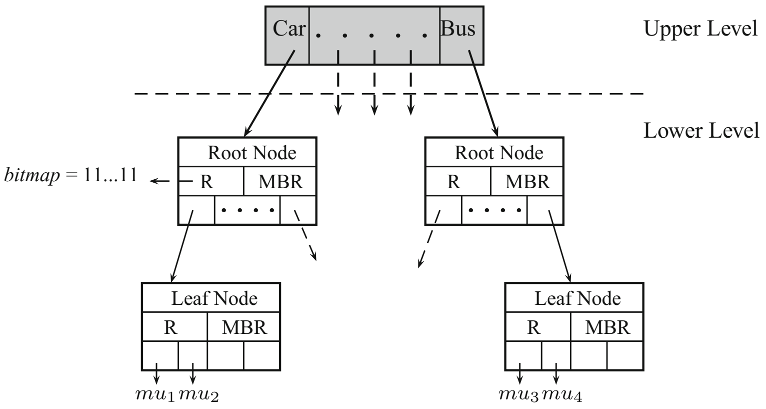


Fig. 2 The architecture of mode-RTree

Note that the semantic meaning of the integer here is different from the one in Section 5.1. Each E3D-RTree manages pieces of movement with only one mode. That is, there is one E3D-RTree for trips with *Car*, one E3D-RTree for trips with *Walk* and so on. The input data for an E3D-RTree are movement units presented in Section 6.1. The root node of an E3D-RTree is recorded at the upper level, which performs the mapping from a transportation mode to the corresponding E3D-RTree. In a leaf node of an E3D-RTree, each item points to a movement unit.

Next, we introduce the reason for managing reference IFOB ids in the index. On one hand, among benchmark queries, many of them request the data by issuing a particular IFOB, e.g., a bus/metro route (**Q5**, **Q10**, **Q13**), a building and even a room (**Q8**, **Q9**, **Q12**). If these pieces of information are maintained by the index, obviously, we can prune objects according to this condition and minimize the portion of the dataset being processed. On the other hand, a moving object contains referenced IFOB ids and we aim to index this kind of data. This motivates us to develop a data structure to support additional functionality for efficient query processing. Each node in an E3D-RTree maintains the data for a set of IFOB ids that movement units in the subtree reference to. In a leaf node, the integer marks all IFOBs that are referenced by movement units. In a non-leaf node, the value is the union of its child nodes. Using this property, we can prune the node if the integer does not contain the requested data from the query when traversing the index.

As an example to illustrate the idea, we take **Q5**.

- **Q5.** At 8am on Monday, who sits in the bus No. 32? At 8am on Monday, who sits in the metro No. 2?

The query searches people taking a particular bus No. 32 at the query time. Given a node of the E3D-RTree storing trips with bus movement, if movement units in that subtree do not reference to any bus from route 32, the required data is not contained by the integer in the node. Then, such a node can be safely pruned. We elaborate the feature of the integer in the next subsection.

6.3 The mapping

We now turn to the method of integrating the integer R in each node to represent IFOB ids. Such a value plays a pivotal role in managing the data. The idea is to maintain the relationship between moving objects and referenced IFOBs in the index. Since benchmark queries issue the data on some specific IFOBs, we can prune unqualified nodes according to the query condition. An integer is integrated in each node, and more specifically its bit value is used to manage IFOB ids. We propose a mapping method between IFOB ids and a set of bits in the integer. Given a node of an E3D-RTree, we mark the bit for a set of IFOB ids, contained by moving units. Consequently, we check the integer when access a node. If movement units managed by this node do not reference to a particular IFOB requested by the query, we prune the node. As a result, The following lemma is used.

Lemma 1 *Given a node N_i in an E3D-RTree, let bitmap($N_i.R$) return the bitmap of the integer. For a benchmark query Q, let $Q.o$ denote the requested IFOB id and bitmap($Q.o$) return the bitmap. Then, we prune N_i iff bitmap($N_i.R$) $\cap$ bitmap($Q.o$) $= \varnothing$.*

Integers are represented in a computer by a group of bits. We let $|bitmap(N_i.R)|$ (32 or 64) denote the number of bits for $N_i.R$. Each E3D-RTree manages movement units with one transportation mode, thus the integer in each node records IFOB ids in one environment. A

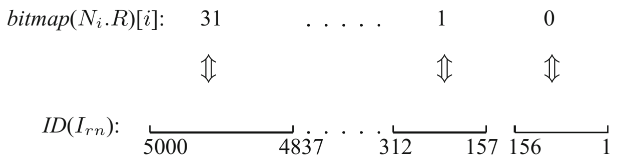


Fig. 3 The *bitmap* for roads

city may have thousands of roads and buildings, much larger than 32(64). To build the connection between IFOB ids and *bitmap*($N_i.R$), we divide IFOB ids into a set of groups, each of which contains disjoint ranges of values and maps to a bit. The number of groups depends on the quantity of bits. Suppose that $N_i.R$ has 32 bits, we need a precise specification of how 32 bits and a set of ids are related. The method is as follows.

In general, we perform the mapping between 32 bits and a set of ids, building the connection between data stored under different representations. Since the number of IFOB ids is different in each environment, several mapping methods may be used. As listed in Table 1 (see Section 3.1), there are five environments in total. We define a set of disjoint integers for reference objects ids, e.g., [1, 5000] for I_{rn}, [6000, 7000] for I_{bn}. For each environment I_i ($\in \{I_{rn}, I_{rbo}, I_{bn}, I_{mn}, I_{indoor}\}$), we let $|ID(I_i)|$ return the quantity of IFOBs. For example, $|ID(I_{rn})|$ shows the number of roads. Comparing $|bitmap(N_i.R)|$ and $|ID(I_i)|$, we have two possibilities:

- case (i): $|bitmap(N_i.R)| \geq |ID(I_i)|$
- case (ii): $|bitmap(N_i.R)| < |ID(I_i)|$

Case (i) is simple and the basic strategy is used. We let each IFOB id be one bit in $N_i.R$. For example, I_{mn} fits to this case as a city usually does not have so many metro routes. Each metro route contains a unique id, and a bit in $N_i.R$ is used for that route. We mark the value if movement units stored in the node reference to metros belonging to that route. Otherwise, the bit is *false*.

In most cases, $|bitmap(N_i.R)|$ is much smaller than $|ID(I_i)|$ (e.g., I_{rn}, I_{indoor}) and multiple IFOB ids are placed into a bit. The method is to partition $|ID(I_i)|$ into $|bitmap(N_i.R)|$ groups, each of which identified by a bit maps to a set of IFOB ids [1]. Following this procedure, we divide a set of ids into 32 groups and each IFOB id is placed into a group. The number of ids per group may not be the same, depending on whether $|ID(I_i)|$ can be evenly divided by 32. If $|ID(I_i)|$ cannot be evenly divided, we process it as follows. The number of elements per group is kept the same for groups with bits from 0 to 30, and we leave the group for bit 31 as a special case, i.e., the number of ids is not the same as the others.

Take the example I_{rn}. Assuming that there are 5000 roads in total with the id starting from 1, we have *lowerbound*($|ID(I_{rn})|/32$) = 156 elements (one can also take the *upperbound*) in groups corresponding to bits [0, 30] and 5000-31*156 = 164 elements in the group for bit 31. Let *bitmap*($N_i.R$)[i] locate the ith bit in the integer. Figure 3 depicts the result of the mapping between *ID*(I_{rn}) and *bitmap*($N_i.R$).

[1]Given a set of values, we produce a set of bin numbers and each value corresponds to a bin. This procedure is called *binning* in bitmap indexes [51].

6.4 How to build the index

In order to have a good property for the index, specifically, the locality for E3D-RTree nodes, before building the index we process movement units as follows. First, the overall time range is divided into a set of intervals and each mu is set an id for its time interval. For example, if all moving objects 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 mu is assigned two values $cellx$ and $celly$ to show x and y positions of the cell where mu is located. Third, we sort movement units in the following order: (1) $mu.m$; (2) time interval id; (3) *cellx*, *celly*, and these sorted movement units are the input data [2]

Initially, *mode_list* and *tree_pointers* are set empty. For a coming movement unit, we first check its transportation mode. If the mode does not exist in *mode_list*, a new item is created. At the same time, a new E3D-RTree is created and the root node record is put in *tree_pointers*. If the mode of a following movement unit already exists, we take the corresponding root node of an E3D-RTree and insert the movement unit into the subtree. To reduce the cost of frequently calling the root node and adjusting the structure when inserting the data, we build each E3D-RTree in a bulkload method [2, 3]. For this reason, we sort moving units by transportation modes in the first step and then build each E3D-RTree one by one.

The bulkload method works as follows. The index retrieves a set of sorted movement units and puts them into a leaf node until the node is full. 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. Following this procedure, we build the subtree for movement units with *Car*, and then the subtree for movement units with *Walk*, and so on. An important aspect is that each E3D-RTree only manages movement units with one transportation mode.

For each E3D-RTree, after inserting all movement units, we update the integer in each node to set the correct value. This is done in a bottom-up way. Given a node, we first calculate the value of its son nodes or objects, and then perform the union. Using the recursively calling method, we repeat the procedure for each node. In the end, the integer of the root node marks all referenced ids of movement units managed in this tree.

6.5 Update

In this part, we discuss how to dynamically maintain the Mode-RTree to keep track of current data. Instead of processing new movement units one by one, we insert them into the Mode-RTree by bulkload, which is the same as creating the index. Choosing this strategy is intended to avoid the high cost of frequent insert and delete operations in the index. Given a set of recently coming movement units, we sort them by (1) $mu.m$; (2) time interval id; (3) *cellx*, *celly*. Since movement units with different modes are put into their corresponding trees, each E3D-RTree is handled individually for a particular mode. For units with the same mode, we preserve the constraint that they are ordered by time, which is

[2]In the implementation, a movement unit is developed to be a relational tuple containing four attributes (*traj_id*, *box*, *m*, *ref_id*). In order to efficiently access the data in the future, we combine each movement tuple with its corresponding subtrip in one relational tuple. The subtrip is represented by a moving point. Movement units with the same $mu.m$, time interval id, *cellx* and *celly* are sorted by the moving point.

consistent with the organization of objects in the tree. Moreover, new arrival units are inherently chronologically ordered. A subtree is created for new objects and inserted into an appropriate node of the E3D-RTree. That is, a pointer to the root of the subtree is added to an entry of a node in the chosen E3D-RTree. This is done by traversing the E3D-RTree in a top-down way and stopping at the level whose height is the same as the subtree. The place to insert the subtree is recorded as well as the path from the root. Then, we update the integer as well as bounding boxes of each node in the branch.

If a new transportation mode is involved for some applications, e.g., airplane, the Mode-RTree is updated as follows. First, movement units with the new mode are produced, which is similar to the procedure of creating other units. Second, an E3D-RTree is created to store pieces of movements containing the new mode. Third, at the upper level of the Mode-RTree, the new mode as well as the root record for the new E3D-RTree are appended. Since subtrees for different modes are managed separately, the integration of a new mode does not influence subtrees for other modes.

6.6 Query algorithms

In general, the query procedure contains two phases: (i) subtree retrieval; and (ii) access the subtree to find objects that fulfill the query condition. The first phase is simple. Depending on the transportation mode, we take the E3D-RTree in the beginning. In some cases, more than one subtree are needed as the query condition requests several modes, e.g., **Q19** (*Bus, Metro, Taxi, Walk*). In the second phase, we traverse the subtree and prune some branches that do not contain qualified objects.

There are 17 queries related to moving objects in the benchmark. By employing the proposed index, we can improve the performance of 14 queries in total, excluding **Q6**, **Q7** and **Q21**. Due to the quantity of benchmark queries, we do not introduce the algorithm for each of them. We take **Q12** as an example to present the method as the query procedure is similar.

- **Q12** Did anyone who was on floor H-5 of the office building between 2pm and 5pm take a bus to the stop "train station" on Friday?

To answer the query, besides checking the time period, we aim to find moving objects that fulfill the following condition on transportation modes:

1. pass floor H-5 of the office building
2. the bus trip starts from a bus stop nearby the office building and ends at a stop near to the train station

Since trips with different transportation modes are separately managed by the Mode-RTree, we access two subtrees for modes *Indoor* and *Bus*, respectively. For each subtree, we collect candidates that fulfill the condition. Afterwards, we perform the join on two sets of candidates and take trips that appear in both sets as the result. In the following, we introduce how to find candidates in each subtree.

Before showing the algorithm, some notations are introduced. In the example here, the query Q consists of two parts, $Q = Q_1 \cup Q_2$, where Q_1 is for *Indoor* and Q_2 is for *Bus*. Each part contains three components: (1) $Q_i.t$, a time interval; (2) $Q_i.m$, the transportation mode; (3) $Q_i.o$, IOFB ids ($i \in \{1, 2\}$).

We give the algorithm of **Q12** in Algorithm 1. The algorithm takes in three parameters: the Mode-RTree, Q and a set of movement units denoted by MU. There are two subroutines: searching *Indoor* and *Bus* trips, respectively.

Algorithm 1 *BenchQuery12*(*Mode-RTree*, Q, MU)

```
split $Q$ into $Q_1$ and $Q_2$ ;
$L_1 \leftarrow$ IndoorTrips(Mode-RTree, $Q_1$, $MU$) ;
$L_2 \leftarrow$ BusTrips(Mode-RTree, $Q_2$, $MU$) ;
return $L_1 \cap L_2$ ;
```

First, we present the procedure of searching candidates that fulfill the *Indoor* condition. Algorithm 2illustrates the pseudo code. In the beginning, we choose the subtree according to the transportation mode, the E3D-RTree for indoor trips.

Next, we initialize two variables, the query window q_box and the bitmap for query IFOB ids. q_box is a 3D box, used to filter nodes on spatial and temporal conditions. The time dimensional value is set by $Q_1.t$. We set the spatial value as follows. For indoor candidates, we want to find objects that pass floor H-5 in a particular building. The spatial value depends on the range of the building in space. Thus, we let the spatial value of q_box be the 2D bounding box of the building in outdoor space. Obviously, moving objects that do not pass such a region cannot be the result. The second variable to set is the bitmap of IFOB ids in the query. Using the mapping method in Section 6.3, we mark the corresponding bit for $Q_1.o$.

Algorithm 2 proceeds by setting the result list *RL*. A queue L is used to store E3D-RTree nodes. Starting from the root node, the algorithm traverses the E3D-RTree in a branch-and-bound manner to find candidate objects. Given a node N_i, we compare the bitmap of $N_i.R$ and $Q_1.o$ (initialized above). If the two bitmap intersect, we open such a node and proceed to check other conditions. Otherwise, we prune the node (Lemma 1). For each entry in N_i, we check whether the box of the child node intersects q_box. In this step, nodes that do not fulfill the condition on spatial and temporal are pruned. If N_i is a leaf node, we retrieve the movement unit mu_i that the entry points to. If the reference id of mu_i belongs to $Q_1.o$, this moving object is put into the candidate list *RL*. If N_i is a non-leaf node, we insert child nodes into L for further consideration. After processing all nodes in the queue L, we return the result list finally.

The procedure of searching trips for *Bus* is almost the same with the exception that q_box and *bitmap*($Q_2.o$) are set in a different way. Since we know the start and end bus stops for those trips, we get bus routes that the stops belong to. Each route has an id and the geographical information. We perform the union on geographical information of those bus routes and define the overall bounding box be the spatial box of q_box. Those bus route ids are collected and set as $Q_2.o$. A bus route includes a set of schedules, each of which is identified by a bus trip, referenced by a moving object. We propose another mapping between a bus route id and its bus trip ids in order to determine the bus route for a movement unit. The algorithm in detail is omitted as it is similar to Algorithm 2.

7 Performance evaluation

We report extensive experimental results in this section. The implementation is developed in an extensible database system Secondo [22] and programmed in C/C++ and Java. A standard PC (Intel 3.3 GHz, 4 GB memory, 500GB disk) running Suse Linux (kernel version 2.6.34) is used. We utilize the tool MWGen [54] to create all infrastructure data based on real road datasets and public floor plans (e.g., http://www.modulargenius.com/default.aspx, http://www.edenresort.com/home, http://www.greenhosp.org/floor_plans.asp).

Algorithm 2 *IndoorTrips*(*Mode-RTree*, Q_1, *MU*)

```
$E3D\text{-}RTree \leftarrow ChooseSubtree(Mode\text{-}RTree, Q_1.m)$;
initialize the query window $q\_box$;
initialize $bitmap(Q_1.o)$;
$RL \leftarrow \varnothing$ ;
$L \leftarrow E3D\text{-}RTree.RootId()$;
while $L$ is not empty do
    $N_i \leftarrow GetNode(E3D\text{-}RTree, L.head())$;
    if $bitmap(Q_1.o) \cap bitmap(N_i.R) \neq \varnothing$ then
        foreach $j \in N_i.entrycount$ do
            if $N_i[j].box$ intersects $q\_box$ then
                if $N_i$ is a leaf node then
                    get $mu_i$ from $MU$ by $N_i[j]$;
                    if $mu_i.ref\_id \in Q_1.o$ then
                        if $mu_i.traj\_id \notin RL$ then
                            $RL \leftarrow mu_i.traj\_id$;
                else
                    $L \leftarrow N_i[j].pointer$;
return $RL$;
```

Two road datasets are used, Berlin (http://www.bbbike.de/cgi-bin/bbbike.cgi) and Houston (http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html). The tool takes roads and floor plans as input and constructs the overall space for moving objects including road network, pavement areas, bus network, metro network and a set of buildings.

7.1 Experimental setup

Our benchmark generator provides a set of configuration parameters that allow a user to create the desired datasets, including (1) the total number of moving objects; (2) the number of trips according to each movement rule; (3) the distribution of transportation modes; (4) start and end locations; (5) time distribution. In the following, we present the setting in our experiment.

We simulate one week movement. All trips are created based on the rules presented in Section 3.2 and the distribution is listed in Fig. 4a. The notation *B* is for Berlin and *H* is for Houston. We add one more pattern to create moving objects visiting several places, e.g., home → shopping → restaurant → home. The time distribution of each rule is shown in Fig. 4b.

We summarize the settings of transportation modes in Fig. 4c. All trips except those from **MR2** include both indoor and walking movement. **MR2** defines the movement in one environment (Region-based Outdoor or Indoor), leading to the determined mode. For trips based on **MR3**, we let part of them travel by car and the other use public transportation vehicles. Since the trip is motivated by nearest neighbor query, we find that usually the

Name	Percentage(B)	Percentage(H)
MR1	40%	35%
MR2	10%	20%
MR3	10%	20%
MR4	30%	20%
Long Trips	10%	5%

(a) Rule Distribution

Name	Time Periods	Days
MR1	[6:30, 19:00]	Mon-Fri
MR2	[9:00, 17:00]	Mon-Sat
MR3	[9:00, 17:00]	Mon-Sat
MR4	[10:00, 21:00]	Mon-Sun
Long Trips	[6:30, 20:00]	Mon-Sat

(b) Time Distribution

Name	Berlin	Houston
MR1, **MR4**	$Indoor + Walk + \begin{cases} Bike : 5\% \\ Car : 50\% \\ Bus : 20\% \\ Metro : 20\% \\ Taxi : 5\% \end{cases}$	$Indoor + Walk + \begin{cases} Bike : 5\% \\ Car : 60\% \\ Bus : 15\% \\ Metro : 15\% \\ Taxi : 5\% \end{cases}$
MR2	*Walk*: 40%; *Indoor*: 60%	*Walk*: 40%; *Indoor*: 60%
MR3	$Indoor + Walk + \begin{cases} Car : 50\% \\ Bus : 35\% \\ Taxi : 15\% \end{cases}$	$Indoor + Walk + \begin{cases} Car : 60\% \\ Bus : 28\% \\ Taxi : 12\% \end{cases}$
Long Trips	*Indoor* + *Walk* + *Car*: 100%	*Indoor* + *Walk* + *Car*: 100%

(c) Transportation Modes Distribution

Fig. 4 Benchmark generator parameters

distance from the query location to the target place is not very far. A distance threshold is defined for the resulting path to determine whether the traveler will go on foot directly or by bus (taxi). This applies to the case that the query user is a pedestrian, not for a car. We do not include the mode *Metro* because usually such a mode is not contained in a short distance trip. We let the modes be *Indoor* + *Walk* + *Car* for a long distance journey. For some rules, we let the two cities have different distributions on transportation modes.

7.2 Datasets

Figure 5 lists all datasets that we use for the experiment. All infrastructure data are shown in Fig. 5a and the detailed information of buildings is reported in Fig.5b. We create a set of buildings of different types based on their floor plans to simulate a city environment. Besides static IFOBs, there are buses and metros represented by moving objects. In both bus and metro networks, we define a schedule for each route to create trips. For the bus schedule, there are more trips on the day ∈ {Mon, Tue, Wed, Thu, Fri, Sat} than on Sunday. For metros, schedules are the same for the whole week. We report the setting in Fig. 5c. Moving objects datasets are described in Fig. 5d, where $|U|$ denotes the quantity of total units for moving objects, $|A|$ shows the average number of units per trip, and $|MU|$ is the total number of movement units (input data for the index). The generator allows to create data sets of varying sizes, thus is a flexible tool for evaluation.

7.3 Benchmark performance

We use the CPU time and I/O accesses as performance metrics where the I/O accesses mean the number of pages that are read into the cache. In the database system, the cache size is set as 64M and one page size is 4k. The results of each query are averaged over five runs. For

	Berlin	Houston
X Range	[0, 44411]	[0, 133573]
Y Range	[0, 34781]	[0, 163280]
Roads	3,250	4,575
No. Vertices in *P*	116,516	437,279
Bus Routes	89	92
Metro Routes	10	16
Buildings	4,996	5,992
Size	2.5 G	9.7 G

(a) Infrastructure Data

Type	Berlin	Houston	Rooms NO.
house	3,713	5,000	
officeA	600	530	294
officeB	487	266	214
shopping mall	80	79	360
cinema	6	8	21
hotel	39	40	584
hospital	46	48	89
university	20	20	431
train station	1	1	56

(b) Statistics of Buildings

	Berlin	Houston
No. Bus Trips in One Day (From Mon to Sat)	5,220	5,594
No. Bus Trips On Sun	2,660	2,788
No. Metro Trips in One Day (From Mon to Sun)	1,897	3,046

(c) Mobile Infrastructure Data

Name	Trips No.	Disk Size (G)	$\|U\|$ (M)	$\|A\|$	$\|MU\|$ (M)
B10K	11,792	0.103	0.702	59.52	0.382
B50K	52,364	0.474	3.23	61.72	1.82
B200K	210,328	1.89	12.85	61.1	7.27
B500K	525,636	4.7	32.3	61.4	18.21
B1M	1,050,200	9.48	64.6	61.5	36.36
B2M	2,100,208	18.87	128.45	61.2	49.57
H1M	1,033,452	10.42	58.87	56.96	23.86

(d) Moving Objects

Fig. 5 Classification of datasets

the constant value of a query such as a particular person, a university or an office building, we manually select an arbitrary value from the possible set.

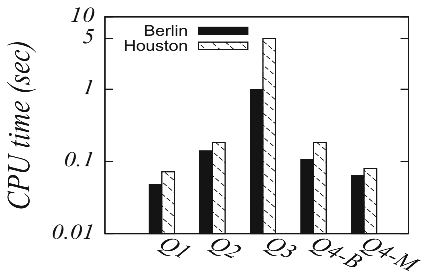
7.3.1 Queries on infrastructures

Infrastructures in both Berlin and Houston are used for evaluation, produced by the tool MWGen. In detail, IFOBs include roads represented by lines, pavements defined by a large polygon, bus (metro) routes and stops, moving buses (metros), and buildings. We use a line to represent geographical information of a route, and the location of a stop is identified by a point. Moving points are utilized for buses and metros. Given a building, we denote its 2D area in space by a polygon and use proposed data types (see [55]) for objects such as rooms, staircases and corridors. The statistics of all IFOBs are reported in Fig. 5a and b. Parameters in each query are set as follows. In **Q1**, the city center is defined to be a rectangle. The size is 4000 in length and 3000 in width for Berlin, and 9000 in length and 9000 in width for Houston. The building in **Q2** is randomly selected from the domain. The query route is also chosen in a stochastic way in both **Q3** and **Q4**.

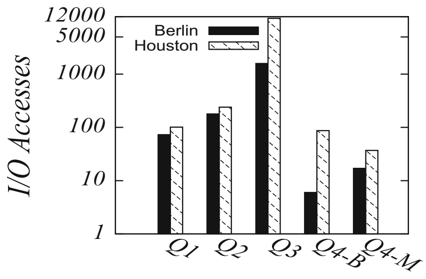
We report the cost of infrastructure queries in Fig. 6, where both time and I/O measurements are plotted in logarithmic scale. The results show that the cost of Houston is higher than that of Berlin. This is because Houston contains more infrastructure objects and occupies a larger area. **Q3** takes more time and I/O accesses than other queries as the procedure involves the costly intersection computation between roads and bus routes.

7.3.2 Scaling datasets

In this part, we report the result of evaluating the system performance between the Mode-RTree and the baseline method when the size of the dataset increases. We choose the infrastructure data of Berlin and generate different numbers of trips, from Berlin10K to Berlin2M. Figure 7 depicts the cost where the result is the sum over all queries. When



(a) CPU Time



(b) I/O Accesses

Dataset	Q1	Q2	Q3	Q4-B	Q4-M
Berlin	0.06	0.13	0.95	0.1	0.07
Houston	0.08	0.18	3.1	0.16	0.09

(c) CPU Time (sec)

Dataset	Q1	Q2	Q3	Q4-B	Q4-M
Berlin	73	180	1,542	6	17
Houston	111	236	10,386	87	36

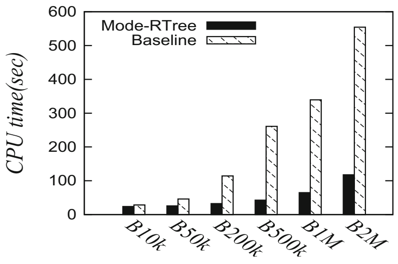
(d) I/O Accesses

Fig. 6 Query cost on infrastructures

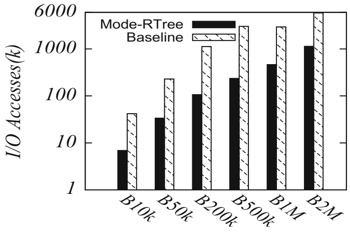
the number of moving objects increases, both CPU time and I/O accesses rise proportionally. The Mode-RTree is much faster than the baseline method. Specially, the deviation is obvious when the dataset becomes large. Note that CPU and I/O measurements are plotted in logarithmic scale. The detailed value of each query is reported in Figs. 14 and 15 in Appendix C.

7.3.3 Employ the Mode-RTree

Next, different datasets are used to evaluate the performance by employing the Mode-RTree. We use datasets B2M and H1M, and report the running cost of each query. The effect and advantage of the index is demonstrated in Figs. 8 and 9. The experimental results confirm that the Mode-RTree achieves orders-of-magnitude better performance. We put the accurate cost of each query in Appendix C from Fig. 16 to Fig. 19. **Q6**, **Q7** and **Q21** are not supported by the index and not included here.



(a) CPU Time



(b) I/O Access

Fig. 7 Scale the dataset

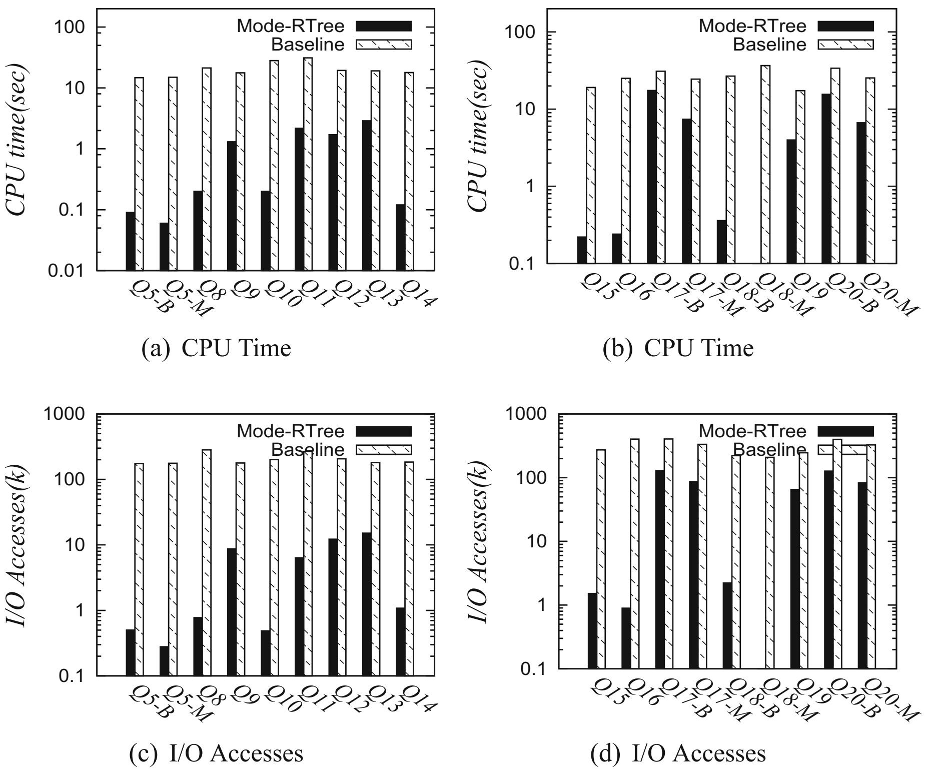


Fig. 8 B2M

7.3.4 Query parameters

By analyzing the execution procedure of benchmark queries, we find that majority of them have a common step. That is, given a time interval, a transportation mode and an IFOB id, the query gets all objects that reference to that IFOB with the mode during the time interval. For example, **Q5** returns all moving objects referencing to a particular bus at the query time. To answer **Q16**, we find objects that stay in a specific room. We extract this step and test the efficiency. Besides benchmark queries, this procedure has many applications like

- who pass Alexender street by bike on Sunday morning?
- Did some enter room 123 in the office building this morning?

We conduct experiments to compare the efficiency of this common step between the Mode-RTree and baseline method. The query contains three parameters: (1) the time interval is randomly generated; (2) we let the transportation mode be a value from *M*' = {*Bus*, *Metro*, *Car*, *Taxi*, *Bike*, *Indoor*}; and (3) the reference id is defined to be a random value from its possible set. For each transportation mode from *M*', we execute the query and collect the running cost. In the end, we summarize the cost of all transportation modes and let the overall value be the final result (six queries in total). Figures 10 and 11 show the result of datasets Berlin and Houston, respectively. The proposed index offers orders-of-magnitude

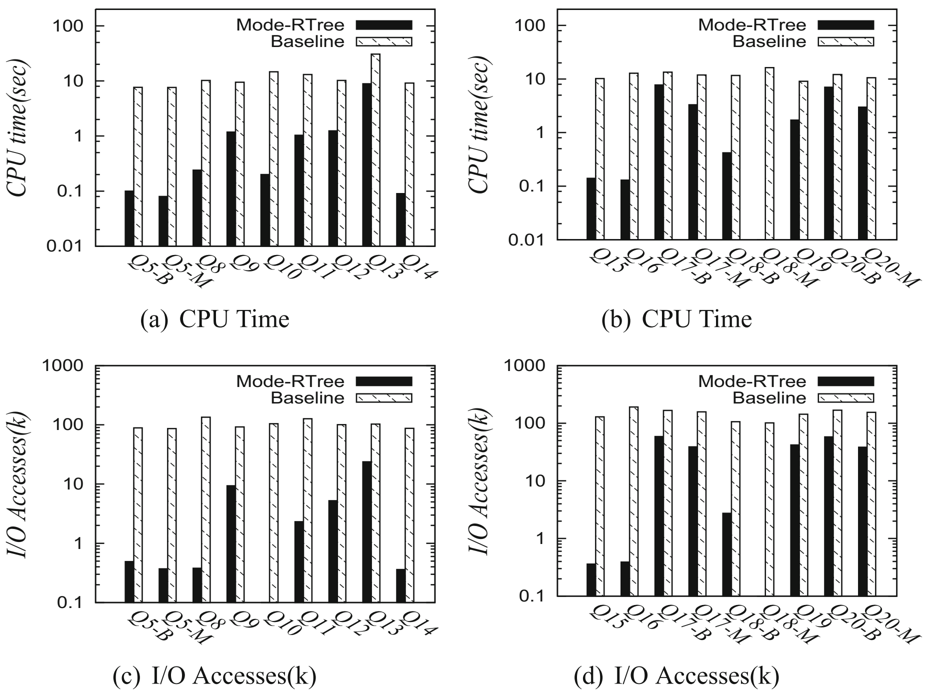


Fig. 9 H1M

performance improvement over the baseline method. The precise value of the running cost is shown Figs. 20 and 21 in Appendix C.

7.3.5 Discussion

Observe that the performance of most queries is significantly improved when the Mode-RTree is employed. The reason is, these queries request the data by considering transportation modes and IFOBs, which are efficiently managed by the index. Thereby, besides spatial

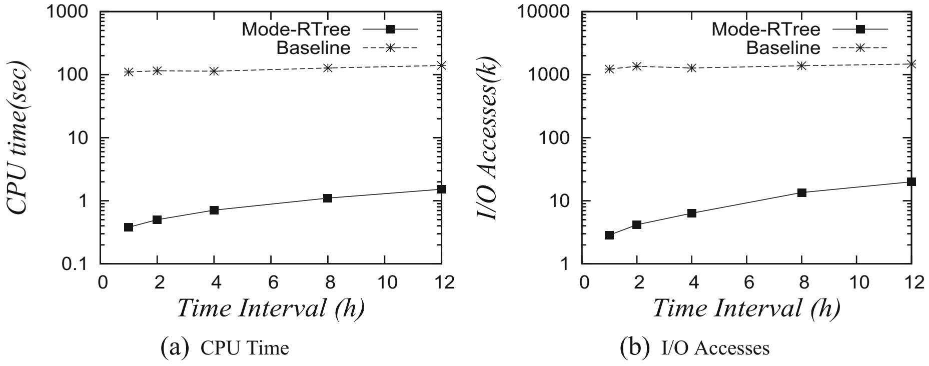


Fig. 10 Parameters: B2M

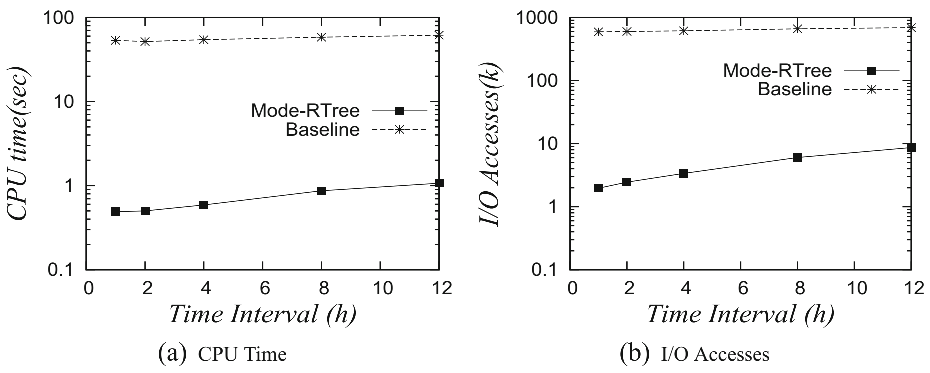


Fig. 11 Parameters: H1M

and temporal dimensions, we can prune unqualified objects on transportation modes and referenced IFOBs. Usually, one or several subtrips (e.g.,*Bus*, *Indoor*) are involved to answer the query instead of the complete trajectory. Thus, some branches in the tree can be safely cut if they do not fulfill the condition. Experimental results show that the pruning strategy substantially decreases the cost in terms of CPU time and I/O accesses. Using large datasets such as 1 or 2 million moving objects, the CPU time is less than for 3 sec in most cases.

There are three queries not supported by the Mode-RTree, **Q6**, **Q7** and **Q21**. We explain the reason as follows. To answer **Q6**, the procedure needs to collect all moving objects containing public transportation modes. If we employ the Mode-RTree, three subtrees (*Bus*, *Metro* and *Taxi*) have to be retrieved. Afterwards, we traverse each subtree to find qualified objects. Since there is no information about IFOB ids, i.e., no pruning condition, the complete tree has to be accessed. In the end, we have to group movement units on *traj_id* to remove duplicate data. There is no advantage of using the Mode-RTree. Regarding **Q7**, we need to find a particular trajectory in the database according to its id. Again, the features of the Mode-RTree is not utilized.

The last query **Q21** takes the most cost among all queries. By analyzing the procedure, we find that the majority of time is spent on mapping bus routes into road segments, up to 90%. The reason is, we need to know the roads that bus passengers travel along. This procedure purely processes the spatial data, and is not related to moving objects. As a result, the Mode-RTree can not speed up the query processing.

8 The advantage of GMOBench

Compared with other benchmarks on moving objects [11, 14, 25, 46], the superiority of GMOBench includes the following three parts: (1) data; (2) queries; (3) index.

Data We design a data generator which is able to produce a complete trip containing both indoor and outdoor movements. Several environments are defined as well as places where transportation modes change. Trips in different environments are seamlessly integrated to constitute a comprehensive movement. Besides the location data, transportation modes are precisely determined to enrich the mobility. In contrast, other works solely process moving objects in free space and ignore transportation modes. The location is accurately represented, but environment and transportation modes are missing. The objective of this work is to evaluate the performance of a system that represents moving objects in a comprehensive way. Another favourable aspect is we formulate parameters to create moving objects, which is applicable to generate more representative trips.

Queries Due to the representation issue, queries in existing benchmarks do not support different environments and transportation modes. We target the problem in this benchmark. In particular, there are queries referring to at which place the moving object switches from one environment to another (e.g., *Bus* → *Walk*, *Walk* → *Indoor*) and how human movement occurs by public vehicles. The indoor trip is also well investigated as we not only find out the located indoor space such as rooms and staircases, but also determine the precise location. To our knowledge, this is the first benchmark for moving objects with multiple transportation modes. Since the precise location is represented by our method, some well known queries are supported, e.g., range queries, nearest neighbors. Those works have been extensively studied in previously proposed benchmarks, and thereby not fully included in this work.

Index The proposed index Mode-RTree has the capability of managing trajectories in different environments. Although moving objects show various features depending intrinsically on the environment, they are unified into one framework in this task. Such a property is not covered by current indices for moving objects. Furthermore, the reference relationship between IFOBs and moving objects is maintained by the index, which supplies the power for accelerating query processing. This is motivated by answering a number of queries that require the database to find out the correlation between moving objects and particular IFOBs. The problem is not extensively studied in the current state-of-the-art. Moving points in BerlinMOD [14] are not index-based. Papers [11, 25] perform the evaluation of moving objects indices that target current and near-future positions in free space. However, those indices only support one environment and can not answer queries referring to transportation modes as well as the request of receiving trajectories according to IFOBs. The present benchmark solves the issue.

9 Conclusions

We propose a benchmark GMOBench that evaluates the performance of a database system managing moving objects traveling through different environments. A data generator is developed to create benchmark data in a realistic scenario and reflect the distribution and characteristics of human movement in practice. Some parameters are defined to produce the data in a flexible way. A group of queries on infrastructure data and moving objects is set as the benchmark workload. Based on the baseline method, several optimal techniques are proposed. More importantly, we design an index structure to manage moving objects with multiple transportation modes. By employing the index, the performance of majority of benchmark queries is significantly improved over the baseline method. The efficiency and effectiveness of proposed techniques are studied through comprehensive experiments, and detailed experimental results are reported.

In this benchmark, we implicitly assume that moving objects follow the shortest path between two locations. This might not always be true in practice, especially for buses and pedestrians. In addition, there are some irregular trips for humans, which cannot be formalized by some rules. At this moment, we do not consider defining a complete set of rules to simulate all cases, but common trips in daily life. This is in accord with assumptions made by most existing data generators. Thus, it is of interest to take into account trips with some special behavior in the future. Regarding benchmark queries, a popular query called nearest neighbor is not investigated. The reason is, this query focuses on location but not transportation modes. We do not find an application in this context. Another interesting query is to find trajectories fulfilling the condition on transportation modes in a certain order, e.g., *Indoor* → *Walk* → *Bus*. The present benchmark establishes such a query (**Q12**) but there are more examples to study. We only deal with histories of moving objects in this work. It is also important to find an approach to supporting on-line update.

Acknowledgment This work is supported in part by NSFC under grants 61300052, the Scientific Research Foundation for the Returned Overseas Chinese Scholars (State Education Ministry), and Natural Science Foundation of Jiangsu Province of China under grants BK20130810.

Appendix A - Formulate Benchmark queries

To formulate queries, we provide a relational interface to exchange information. Several relations are provided to manage infrastructure data and moving objects, summarized in Table 4. $\underline{route}$ is a data type that we propose to represent bus (metro) routes. A bus (metro) route is represented by a sequence of segments each of which defines the locations of start and end bus stops as well as the connection described by a line. For the indoor environment, we design a data type called $\underline{groom}$ used in Rel_Room to represent all rooms of a building, e.g., office rooms, corridors, staircases. Since in some cases one room can have several floors such as an amphitheater and a chamber, a $\underline{groom}$ object consists of a set of elements each of which defines the 2D area of a floor and the height above the ground level.

To access an infrastructure, we assign a symbol to each relation (summarized in Fig.12a) and obtain the data from the operator **get_infra** described in Fig. 12b. Figure 13 lists the operators that are used to access the data and formulate the queries. To represent the trajectories of moving objects (i.e., the projection into space), we propose a data type $\underline{genrange}$ which defines a set of elements. Each element records the path according to a referenced object as well as the transportation mode.

We use qt to denote the query time parameter. For the queries (**Q4**, **Q5**, **Q17**, **Q18**, **Q20**) on both bus and metro, it is sufficient to show the query expression for a bus network as the procedure is similar for metros.

Table 4 Data relation schemas

Road network	Rel_Road	(R_id:int, GeoData: line, Name: string)
Region-based outdoor	Rel_Rbo	(Reg_id:int, Reg: region, Name: string)
Bus network	Rel_BStop	(Br_id:int, Stop_id: int, GeoData: point, Name: string)
	Rel_BRoute	(Br_id:int, GeoData: route, Name: string)
	Rel_Bus	(Bus_id:int, Br_id:int, Traj: genmo, Name: string)
Metro network	Rel_MStop	(Mr_id:int, Stop_id: int, GeoData: point, Name: string)
	Rel_MRoute	(Mr_id:int, GeoData: route, Name: string)
	Rel_Metro	(Metro_id:int, Mr_id:int, Traj: genmo, Name: string)
Indoor	Rel_Building	(B_id: int, GeoData: region, Name: string)
	Rel_Room	(B_id: int, Room_id: int, GeoData: groom, Name: string)
Generic moving objects	MOGendon	(Mo_id:int, Traj: genmo, Name:string)

ROAD	RBO	BUSSTOP	BUSROUTE	BUS
METROSTOP	METROROUTE	METRO	BUILDING	ROOM

(a) Infrastructure Symbols

Name	Signature
get_infra	$\underline{space} \times \underline{int} \rightarrow$ rel

(b) Access Relations

Fig. 12 Access infrastructures

- **Q1**. Find out all metros passing through the city center.
  Let Reg_C be a region denoting the city center area.

```
SELECT mr.Mr_id
FROM get_infra(SpaceGendon, METROROUTE) as mr
WHERE mr.GeoData intersects Reg_C
```

- **Q2**. Given a building named by X, find all bus stops within 300 meters.

```
SELECT bs FROM get_infra(SpaceGendon, BUILDING) as
b,
               get_infra(SpaceGendon, BUSSTOP) as bs
WHERE b.Name = X and distance(b.GeoData,
bs.GeoData) < 300
```

- **Q3**. Which streets does bus No. 12 pass by?

```
SELECT r.Name
FROM get_infra(SpaceGendon, BUSROUTE) as br
     get_infra(SpaceGendon, ROAD) as r
WHERE br.Br_id = 12 and br.GeoData intersects
r.GeoData
```

Name	Signature
intersects	$\underline{route} \times \underline{region} \rightarrow \underline{bool}$ $\underline{route} \times \underline{line} \rightarrow \underline{bool}$ $\underline{periods} \times \underline{periods} \rightarrow \underline{bool}$
inside	$\underline{genloc} \times \underline{line} \rightarrow \underline{bool}$ $\underline{genloc} \times \underline{region} \rightarrow \underline{bool}$
distance	$\underline{region} \times \underline{point} \rightarrow \underline{real}$
theloc	$\underline{int} \times \underline{real} \times \underline{real} \rightarrow \underline{genloc}$
length	$\underline{genrange} \rightarrow \underline{real}$
duration	$\underline{periods} \rightarrow \underline{real}$
deftime	$\underline{genmo} \rightarrow \underline{periods}$
val **initial** **final**	$\underline{intime}(\underline{genloc}) \rightarrow \underline{genloc}$ $\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$ $\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$
ref_id	$\underline{genloc} \rightarrow \underline{int}$

(a) Basic

Name	Signature
atinstant **atperiods**	$\underline{genmo} \times \underline{instant} \rightarrow \underline{intime}(\underline{genloc})$ $\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$
contains	$\underline{genmo} \times \underline{tm} \rightarrow \underline{bool}$ $\underline{genmo} \times \underline{int} \rightarrow \underline{bool}$
at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$ $\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$ $\underline{genmo} \times \underline{point} \rightarrow \underline{genmo}$
freespace	$\underline{genmo} \rightarrow \underline{mpoint}$ $\underline{genloc} \rightarrow \underline{point}$
passes	$\underline{genmo} \times \underline{region} \rightarrow \underline{bool}$ $\underline{genmo} \times \underline{groom} \rightarrow \underline{bool}$
trajectory	$\underline{genmo} \rightarrow \underline{genrange}$

(b) Advanced

Fig. 13 Operators used in queries

- **Q4**. Where can I switch between bus route No. 16 and No. 38?

```
SELECT bs1, bs2
FROM get_infra(SpaceGendon, BUSSTOP) as bs1,
     get_infra(SpaceGendon, BUSSTOP) as bs2,
WHERE bs1.Br_id = 16 AND bs2.Br_id = 38 AND
bs1.GeoData = bs2.GeoData
```

- **Q5**. At 8am on Monday, who sits in the bus No. 32?

```
SELECT mo.Name
FROM get_infra(SpaceGendon, BUS) as bus, MOGendon as
mo
WHERE ref_id(val(mo.Traj atinstant qt)) = bus.Bus_id
and bus.Br_id = 32
```

- **Q6**. What is the percentage of people traveling by public transportation vehicles?

```
Let no_mo = SELECT COUNT(*) FROM MOGendon

Let no_submo = SELECT COUNT(*)
                 FROM MOGendon as mo
                 WHERE mo.Traj contains BUS or
                       mo.Traj contains METRO or
                       mo.Traj contains TAXI

SELECT DIV(no_submo, no_mo)
```

- **Q7**. Where and how long does *Bobby* walk during his trip?

```
SELECT mo.Traj at Walk
FROM MOGendon as mo
WHERE mo.Name = "Bobby"

SELECT duration(deftime(mo.Traj at Walk))
FROM MOGendon as mo
WHERE mo.Name = "Bobby"
```

- **Q8**. Find out all people passing room 312 at the office building between 9:00am and 11:00am on Monday.

```
SELECT mo.Name
FROM MOGendon as mo,
     get_infra(SpaceGendon, BUILDING) as b,
     get_infra(SpaceGendon, ROOM) as rm
WHERE b.Name = "Office-X" and b.B_id = rm.B_id and
      and rm.Room_id = 312 and
      ((mo.Traj at Indoor) atperiods qt) contains
rm.Room_id
```

- **Q9**. Who arrived by taxi at the university on Friday?
  To arrive by taxi at the university means that the final location of the passenger within the taxi belongs to some driveway area close to the university. Such a part of

the road network is also managed in the database as an object UniDriveway of type line.

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE val(final((mo.Traj atperiods qt) at Taxi)))
      inside UniDriveway
```

- **Q10**. Who entered bus No. 3 at bus stop University on Tuesday afternoon?

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUSSTOP) AS bs,
     get_infra(SpaceGendon, BUS) AS bus
WHERE bs.Br_id = 3 AND bs.Name = "University" AND
bus.Br_id = 3 AND
      bs.GeoData = freespace(val(initial((mo.Traj
atperiods qt) at
                                     theloc(bus.Bus_id,
undef, undef)))))
```

- **Q11**. Find out all people walking through zone *A* and zone *B* on Saturday between 10am and 3pm.

```
SELECT mo.Name
FROM MOGendon AS mo,
    get_infra(SpaceGendon, RBO) AS R1,
    get_infra(SpaceGendon, RBO) AS R2
WHERE R1.Name = "Zone-A" AND R2.Name = "Zone-B" AND
      ((mo.Traj atperiods qt) at Walk) passes
R1.Reg AND
      ((mo.Traj atperiods qt) at Walk) passes
R2.Reg
```

- **Q12**. Did anyone who was on floor H-5 of the office building between 2pm and 5pm take a bus to the stop “train station” on Friday?

  To be on floor H-5 means to be in any of the rooms of floor H-5. We create a table associating rooms with their floors:

  Uni_Rel(B_id: int, Floor: string, Room_id: int)

```
SELECT mo.Name
FROM MOGendon AS mo,
     Uni_Rel AS u,
     get_infra(SpaceGendon, ROOM) AS r,
     get_infra(SpaceGendon, BUSSTOP) AS bs1,
     get_infra(SpaceGendon, BUSSTOP) AS bs2
WHERE u.Floor = "H-2" AND u.B_id = r.B_id AND
u.Room_id = r.Room_id AND
      (mo.Traj atperiods qt) passes r.GeoData AND
      bs1.Name = "University" AND
      bs2.Name = "Main station" AND
      freespace(val(initial((mo.Traj atperiods qt)
at Bus))) =
      bs1.GeoData AND
freespace(val(final((mo.Traj atperiods qt) at
Bus))) =
bs2.GeoData
```

- **Q13**. Did bus No. 35 pass any bicycle traveler by on Monday?
  We define *pass* to mean that there exists a time instant that the distance between the two moving objects is less than 3 meters.

```
SELECT mo.Name
FROM get_infra(SpaceGendon, BUS) AS bus,
      MOGendon AS mo
WHERE bus.Br_id = 35 AND
      sometimes(distance(freespace(bus.Traj
atperiods qt),
                          freespace((mo.Traj
atperiods qt) at Bicycle)) < 3.0)
```

- **Q14**. Did someone spend more than 15 minutes on waiting for the bus at the bus stop Cinema on Saturday?

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUSSTOP) AS bs
WHERE bs.Name = "Cinema" AND
      duration(deftime((mo.Traj atperiods qt) at
bs.GeoData)) > 15
```

- **Q15**. How many people visit the cinema on Saturday?

```
SELECT COUNT(*)
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUILDING) AS b
WHERE b.Name = "Cinema" AND
      (mo.Traj atperiods qt) contains b.B_id
```

- **Q16**. Find out all people staying at room 154 in the university for more than one hour on Thursday.

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUILDING) AS b,
     get_infra(SpaceGendon, ROOM) AS r
WHERE b.Name = "University" AND b.B_id = r.B_id AND
      r.Room_id = 154 AND
      duration(deftime(mo.Traj atperiods qt) at
                theloc(r.Room_id,undef,undef))) > 60
```

- **Q17**. Did someone spend more than one hour traveling by bus?

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE duration(deftime(mo.Traj at Bus)) > 60
```

- **Q18**. Find out all people changing from bus No. A to bus No. B at stop X.

```
SELECT mo.Name
FROM MOGendon AS mo
     get_infra(SpaceGendon, BUSSTOP) as bs,
     get_infra(SpaceGendon, BUS) as bus1,
     get_infra(SpaceGendon, BUS) as bus2
WHERE bus1.Br_id = A AND bus2.Br_id = B AND
      bs.Name = X AND
      freespace(val(final(mo.Traj at
theloc(bus1.Bus_id)))) =
      freespace(val(initial(mo.Traj at
theloc(bus2.Bus_id)))) AND
      freespace(val(final(mo.Traj at
theloc(bus1.Bus_id)))) =
      bs.GeoData
```

- **Q19**. Find out all trips starting from zone A and ending in zone B by public transportation vehicles with the length of the walking path being less than 300 meters.

```
SELECT mo.Name
FROM MOGendon AS mo
     get_infra(SpaceGendon, RBO) AS R1,
     get_infra(SpaceGendon, RBO) AS R2
WHERE R1.Name = "ZoneA" AND R2.Name = "ZoneB" AND
      val(initial(mo.Traj)) inside R1.GeoData AND
      val(final(mo.Traj)) inside R2.GeoData AND
      ((mo.Traj contains Bus) OR (mo.Traj contains
Metro) OR
      (mo.Traj contains Taxi)) AND
      length(trajectory(mo.Traj at Walk)) < 300
```

- **Q20.** Find the top *k* bus (metro) routes with high passenger flow for all workdays.

```
SELECT TOP k *
FROM
   SELECT bus.Br_id, COUNT(*) AS NO
   FROM MOGendon AS mo
        get_infra(SpaceGendon, BUS) AS bus
   WHERE mo.Traj contains Bus AND
       ((mo.Traj atperiods qt) at Bus)) contains
bus.Bus_id
   GROUP BY bus.Br_id
   ORDER BY NO DESC
```

- **Q21.** Find the top *k* road segments with high traffic during the rush hour for all workdays.

  To answer such a query, we create a relation storing all road segments with the schema

  Rel_RoadSeg: (S_id: int, GeoData: line, R_id: int)

  where S_id is the unique segment id, GeoData stores the geometrical property and R_id indicates the id for the road that the segment belongs to. For each road, we decompose it into a set of pieces at the junction positions and each piece is stored as a tuple in Rel_RoadSeg.

  First, we get the traffic of each road segment by aggregrating the number of buses passing by.

```
LET Rel_BRCOUNT =
    SELECT br.Br_id, br.GeoData, COUNT(*) AS NO
    FROM get_infra(SpaceGendon, BUSROUTE) as br,
         get_infra(SpaceGendon, BUS) as bus
   WHERE deftime(bus.Traj) intersects qt AND
bus.Br_id = br.Br_id
   GROUP BY br.Br_id
```

  Each tuple in Rel_BRCOUNT indicates the number of trips for each bus route.

```
LET Rel_RES1 =
    SELECT s.S_id, SUM(br.NO) AS FLOW
    FROM Rel_BRCOUNT as br,
         Rel_RoadSeg as s
    WHERE br.GeoData intersects s.GeoData
    GROUP BY s.S_id
```

  We perform the join on bus routes and road segments to get the segments that each bus route maps to. Then, we group the tuple by segment id and call the aggregation function to get the total number of buses passing a segment from different routes.

  Second, we get the traffic from moving objects that contain one of the modes {*Car*, *Taxi*, *Bike*}. At first, we collect the paths of these trips.

```
LET Rel_Traj_C =
    SELECT trajectory(mo.Traj at CAR) AS Path
    FROM MOGendon AS mo
    WHERE deftime(mo.Traj) intersects qt

LET Rel_Traj_T =
    SELECT trajectory(mo.Traj at TAXI) AS Path
    FROM MOGendon AS mo
    WHERE deftime(mo.Traj) intersects qt

LET Rel_Traj_B =
    SELECT trajectory(mo.Traj at BIKE) AS Path
    FROM MOGendon AS mo
    WHERE deftime(mo.Traj) intersects qt
```

```
LET Rel_Traj = Rel_Traj_C union Rel_Traj_T union
Rel_Traj_B
```

Then, we do the join on the paths and road segments to get the traffic value.

```
LET Rel_RES2 =
    SELECT s.S_id, COUNT(*) AS FLOW
    FROM Rel_Traj AS p,
         Rel_RoadSeg AS s
    WHERE p.Path intersects s.GeoData
    GROUP BY s.S_id
```

Third, we merge the two traffic relations Res_RES1 and Res_RES2 to get the final result.

```
SELECT TOP k *
FROM
  SELECT r1.S_id, r1.FLOW + r2.FLOW as C
  FROM Rel_RES1 as r1,
       Rel_RES2 as r2
  WHERE r1.S_id = r2.S_id
  ORDER BY C DESC
```

Appendix B

Unique IFOB id and reference id for an indoor location

In order to have a unique id for each IFOB, we define a set of disjoint ranges each of which is used for one infrastructure, e.g., [1, 5000] for I_{rn}, [6000, 7000] for I_{bn}. Given a generic location $gl \in \underline{D_{genloc}}$, the meaning of $gl.oid$ is clear if the IFOB belongs to an outdoor infrastructure. However, for the indoor environment different buildings can have the same room id (e.g., ROOM NO 12, ROOM NO 35), only using the building id cannot uniquely identify an indoor location.

To solve the problem, $gl.oid$ is set by combining a building id and a room id for an indoor location. We introduce how to achieve the goal in the following. For the sake of clear presentation, we use two strings B_Id and R_Id to denote a building id and a room id, respectively. Suppose that gl is located in the room R_Id = "123" of a building B_Id = "167654", then $gl.oid$ is assigned by the concatenation of B_Id and R_Id, that is "167654123". Let $len(B\_Id)$ return the length of a string for B_Id. The range for all building ids is carefully chosen in order to fulfill the condition $len(B\_Id_{min}) = len(B\_Id_{max})$, i.e., the number of digits is the same for each id. Otherwise, an ambiguous case can occur.

Afterwards, $len(B\_Id)$ is a fixed value (new building construction is not considered) and we can extract B_Id and R_Id from an indoor location. Using the example before, we can get B_Id = "167654" and R_Id = "123" from $gl.oid$ = "167654123".

Appendix C - experimental statistics

Query	B10K	B50K	B200k	B500K	B1M	B2M
Q5-B	0.07	0.08	0.08	0.07	0.09	0.09
Q5-M	0.04	0.05	0.06	0.04	0.05	0.06
Q6	0.1	0.3	0.7	1.4	2.7	7.0
Q7	0.2	0.4	0.8	1.5	2.9	7.3
Q8	0.2	0.19	0.18	0.1	0.2	0.2
Q9	0.52	0.53	0.72	0.33	0.95	1.3
Q10	0.07	0.07	0.08	0.06	0.09	0.2
Q11	0.04	0.07	0.24	0.51	1.08	2.17
Q12	0.66	0.71	0.78	0.81	1.17	1.7
Q13	1.25	1.3	1.46	1.55	2.05	2.88
Q14	0.03	0.04	0.06	0.06	0.09	0.12
Q15	0.07	0.08	0.1	0.11	0.16	0.22
Q16	0.19	0.18	0.2	0.1	0.21	0.24
Q17-B	0.1	0.41	1.68	4.17	8.34	17.38
Q17-M	0.07	0.2	0.79	1.77	3.8	7.4
Q18-B	0.25	0.25	0.24	0.3	0.3	0.36
Q18-M	0.01	0.01	0.01	0.02	0.01	0.01
Q19	0.05	0.13	0.39	0.85	1.95	3.99
Q20-B	0.14	0.41	1.51	3.75	7.45	15.52
Q20-M	0.09	0.19	0.71	1.6	3.38	6.65
Q21	19.7	20.0	21.5	23.4	27.8	43.0
Total	23.85	25.6	32.29	42.5	64.77	117.79

(a) CPU time (sec)

Query	B10K	B50K	B200K	B500K	B1M	B2M
Q5-B	0.435	0.426	0.449	0.474	0.488	0.5
Q5-M	0.192	0.207	0.219	0.242	0.254	0.279
Q6	0.001	0.001	0.001	0.001	0.5	40.5
Q7	0.03	0.03	0.3	0.3	0.8	81.4
Q8	0.13	0.162	0.286	0.38	0.61	0.78
Q9	0.686	0.868	2.388	2.987	5.32	8.67
Q10	0.016	0.037	0.115	0.195	0.287	0.486
Q11	0.045	0.146	0.672	1.549	3.16	6.34
Q12	0.245	0.568	1.725	3.81	7.1	12.22
Q13	2.47	2.76	3.757	5.643	8.96	15.08
Q14	0.022	0.054	0.147	0.317	0.572	1.08
Q15	0.014	0.045	0.172	0.381	0.775	1.52
Q16	0.022	0.026	0.09	0.255	0.469	0.895
Q17-B	0.551	2.998	12.488	32.05	64.07	129.24
Q17-M	0.524	2.191	8.944	21.578	43.11	86.77
Q18-B	0.014	0.081	0.634	1.132	1.532	2.22
Q18-M	0.003	0.003	0.003	0.003	0.003	0.003
Q19	0.312	1.613	6.124	15.346	30.812	65.14
Q20-B	0.001	0.091	9.71	31.219	62.826	126.6
Q20-M	0.001	0.124	7.208	20.873	41.47	83.20
Q21	1.2	21.2	50.2	97.2	190.6	470.5
Total	6.9	33.63	105.63	235.93	463.72	1133.42

(b) I/O Accesses(k)

Fig. 14 Scaling datasets by employing the Mode-RTree

Query	B10k	B50k	B200k	B500k	B1M	B2M
Q5-B	0.2	0.5	1.8	4.2	7.63	14.76
Q5-M	0.1	0.5	1.7	4.2	7.31	14.95
Q6	0.1	0.7	2.7	7.0	10.61	20.89
Q7	0.2	0.8	2.9	7.3	14.18	27.83
Q8	0.2	0.7	2.5	6.6	10.8	21.28
Q9	0.7	1.1	2.6	5.5	9.08	17.71
Q10	0.8	1.7	5.4	12.7	14.43	28.07
Q11	0.6	3.2	14.3	40.9	15.76	31.14
Q12	0.7	1.3	3.4	7.5	10.77	19.36
Q13	1.4	2.0	4.2	8.4	10.22	19.08
Q14	0.4	1.2	4.1	10.0	9.14	17.9
Q15	0.3	1.0	4.3	11.3	9.91	19.04
Q16	0.2	0.8	3.0	8.0	12.77	25.05
Q17-B	0.2	0.9	3.8	10.1	15.74	30.98
Q17-M	0.2	0.8	3.2	8.4	12.18	24.5
Q18-B	0.7	1.6	5.0	12.1	17.92	26.78
Q18-M	0.7	2.4	8.7	21.3	24.83	36.65
Q19	0.1	0.8	4.4	10.5	49.33	17.37
Q20-B	0.4	1.4	5.0	12.7	15.5	33.91
Q20-M	0.3	1.0	3.8	9.5	12.24	25.39
Q21	19.7	21.5	27.8	43.0	49.06	81.77
Total	28.3	45.9	114.4	261.0	339.41	554.41

(a) CPU Time (sec)

Query	B10k	B50k	B200k	B500k	B1M	B2M
Q5-B	2.0	5.8	19.9	48.0	87.88	174.54
Q5-M	0.4	0.9	7.4	46.9	88.35	176.21
Q6	0.001	0.001	0.5	40.5	85.04	170.03
Q7	0.03	0.3	0.08	81.4	170.17	340.14
Q8	2.1	10.0	54.6	149.3	139.81	282.12
Q9	1.6	2.1	19.4	47.3	89.79	177.86
Q10	1.5	3.5	23.8	63.6	101.51	200.44
Q11	2.6	13.3	64.0	170.8	134.09	267.47
Q12	1.0	5.2	27.8	66.8	104.18	206.14
Q13	1.4	2.7	18.8	51.7	90.86	180.34
Q14	0.3	1.5	24.2	79.6	92.23	184.11
Q15	1.5	10.1	90.4	232.0	134.45	272.28
Q16	3.9	21.5	93.0	232.6	201.57	401.83
Q17-B	1.8	17.7	90.7	227.8	202.64	405.08
Q17-M	2.1	15.4	72.1	179.4	166.99	333.45
Q18-B	1.0	5.1	32.9	81.0	118.66	221.68
Q18-M	1.8	2.1	28.7	71.2	109.57	208.03
Q19	4.9	24.9	109.6	285.1	125.39	245.7
Q20-B	0.1	22.0	88.8	221.4	199.78	397.97
Q20-M	0.03	15.2	69.7	173.2	163.72	326.78
Q21	1.2	50.2	190.6	470.5	338.61	675.28
Total	41.8	229.1	1126.7	3020.2	2945.27	5847.48

(b) I/O Accesses(k)

Fig. 15 Scaling datasets by baseline method

Query	Baseline	Mode-RTree
Q5-B	14.76	0.09
Q5-M	14.95	0.06
Q8	21.28	0.2
Q9	17.71	1.3
Q10	28.07	0.2
Q11	31.14	2.17
Q12	19.36	1.7
Q13	19.08	2.88
Q14	17.9	0.12

(a)

Query	Baseline	Mode-RTree
Q15	19.04	0.22
Q16	25.05	0.24
Q17-B	30.98	17.38
Q17-M	24.5	7.4
Q18-B	26.78	0.36
Q18-M	36.65	0.01
Q19	17.37	3.99
Q20-B	33.91	15.52
Q20-M	25.39	6.65

(b)

Fig. 16 CPU time (sec) for B2M

Query	Baseline	Mode-RTree
Q5-B	174.54	0.5
Q5-M	176.21	0.279
Q8	282.12	0.775
Q9	177.86	8.67
Q10	200.44	0.486
Q11	267.47	6.34
Q12	206.14	12.22
Q13	180.34	15.08
Q14	184.11	1.08

(a)

Query	Baseline	Mode-RTree
Q15	272.28	1.52
Q16	401.83	0.895
Q17-B	405.08	129.244
Q17-M	333.45	86.773
Q18-B	221.68	2.22
Q18-M	208.03	0.003
Q19	245.7	65.14
Q20-B	397.97	126.60
Q20-M	326.78	83.20

(b)

Fig. 17 I/O accesses (k) for B2M

Query	Baseline	Mode-RTree
Q5-B	7.67	0.1
Q5-M	7.63	0.08
Q8	10.19	0.24
Q9	9.48	1.18
Q10	14.62	0.2
Q11	13.1	1.03
Q12	10.18	1.24
Q13	30.75	8.87
Q14	9.13	0.09

(a)

Query	Baseline	Mode-RTree
Q15	10.21	0.14
Q16	12.8	0.13
Q17-B	13.44	7.74
Q17-M	11.89	3.31
Q18-B	11.66	0.42
Q18-M	16.31	0.01
Q19	9.08	1.71
Q20-B	12.15	7.03
Q20-M	10.56	2.99

(b)

Fig. 18 CPU time (sec) for H1M

Query	Baseline	Mode-RTree
Q5-B	88.89	0.49
Q5-M	86.45	0.37
Q8	134.41	0.38
Q9	92.28	9.37
Q10	104.32	0.09
Q11	126.57	2.31
Q12	100.54	5.23
Q13	102.76	23.76
Q14	87.12	0.36

(a)

Query	Baseline	Mode-RTree
Q15	129.09	0.36
Q16	191.34	0.39
Q17-B	166.85	58.77
Q17-M	157.4	38.86
Q18-B	106.26	2.73
Q18-M	101.43	0.03
Q19	143.11	41.86
Q20-B	168.36	57.92
Q20-M	154.45	38.35

(b)

Fig. 19 I/O accesses (k) for H1M

1h	Bus	Metro	Car	Taxi	Bike	Indoor	Sum
No Index	17.98	17.12	22.48	16.33	16.23	20.21	110.35
Mode-RTree	0.09	0.07	0.05	0.03	0.04	0.1	0.38
2h							
No Index	17.73	17.55	20.87	16.77	16.79	25.59	115.3
Mode-RTree	0.15	0.1	0.06	0.06	0.04	0.09	0.5
4h							
No Index	18.91	17.65	23.31	16.26	16.12	20.96	113.21
Mode-RTree	0.23	0.18	0.1	0.05	0.05	0.1	0.71
8h							
No Index	23.15	22.05	25.97	16.49	16.37	23.09	127.12
Mode-RTree	0.38	0.36	0.14	0.05	0.07	0.1	1.1
12h							
No Index	27.27	26.69	27.69	16.78	16.6	24.28	139.31
Mode-RTree	0.54	0.52	0.2	0.08	0.07	0.11	1.52

(a) CPU (sec)

1h	Bus	Metro	Car	Taxi	Bike	Indoor	Sum
No Index	184.53	178.0	245.96	172.03	171.64	271.33	1223.49
Mode-RTree	1.06	0.54	0.7	0.056	0.034	0.465	2.86
2h							
No Index	182.63	179.05	250.35	172.48	171.89	401.83	1358.23
Mode-RTree	1.54	0.81	1.14	0.08	0.04	0.55	4.16
4h							
No Index	191.14	181.97	259.01	173.31	172.30	297.72	1275.45
Mode-RTree	2.3	1.86	1.92	0.15	0.072	0.061	6.36
8h							
No Index	205.26	192.58	284.07	175.79	173.97	352.48	1384.15
Mode-RTree	4.39	4.6	3.03	0.37	0.17	0.91	13.47
12h							
No Index	217.85	200.48	306.87	178.22	176.00	391.75	1471.17
Mode-RTree	6.97	6.62	4.45	0.61	0.32	0.97	19.94

(b) I/O Accesses(k)

Fig. 20 Query parameters: B2M

1h	Bus	Metro	Car	Taxi	Bike	Indoor	Sum
No Index	8.88	8.24	10.81	7.97	7.85	9.73	53.48
Mode-RTree	0.13	0.08	0.08	0.06	0.05	0.09	0.49
2h							
No Index	8.6	8.38	9.6	8.08	7.96	8.97	51.59
Mode-RTree	0.13	0.09	0.06	0.06	0.07	0.09	0.5
4h							
No Index	9.25	8.87	10.33	8.33	8.18	9.48	54.44
Mode-RTree	0.2	0.1	0.08	0.05	0.06	0.1	0.59
8h							
No Index	10.3	10.1	11.52	8.1	7.87	10.31	58.2
Mode-RTree	0.35	0.19	0.08	0.07	0.07	0.11	0.87
12h							
No Index	11.43	11.64	12.14	8.24	8.09	10.01	61.55
Mode-RTree	0.52	0.24	0.07	0.06	0.07	0.11	1.07

(a) CPU (sec)

1h	Bus	Metro	Car	Taxi	Bike	Indoor	Sum
No Index	95.31	90.5	109.70	87.44	87.16	120.64	590.75
Mode-RTree	0.85	0.39	0.38	0.01	0.02	0.31	1.96
2h							
No Index	96.06	90.99	112.12	87.74	87.26	125.06	599.23
Mode-RTree	1.14	0.51	0.41	0.02	0.02	0.35	2.45
4h							
No Index	97.44	91.9	116.30	88.27	87.42	133.26	614.59
Mode-RTree	1.65	0.81	0.48	0.02	0.02	0.39	3.37
8h							
No Index	102.15	95.95	128.1	89.52	88.09	157.28	661.09
Mode-RTree	2.99	1.88	0.6	0.03	0.03	0.5	6.03
12h							
No Index	105.56	98.3	136.80	90.36	88.74	170.4	690.16
Mode-RTree	4.68	2.62	0.75	0.03	0.05	0.54	8.67

(b) I/O Accesses(k)

Fig. 21 Query parameters: H1M

References

1. 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
2. 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

3. Bercken J, Seeger B, Widmayer P (1997) A generic approach to bulk loading multidimensional index structures. In: VLDB, pages 406–415
4. Bitton D, DeWitt DJ, Turbyfill C (1983) Benchmarking database systems a systematic approach. In: VLDB, pp 8–19
5. Booth J, Sistla P, Wolfson O, Cruz IF (2009) A data model for trip planning in multimodal transportation systems. In: EDBT, pp 994–1005
6. Brinkhoff T (2000) Generating network-based moving objects. In: SSDBM, pp 253–255
7. Brinkhoff T (2002) A framework for generating network-based moving objects. GeoInformatica 6(2):153–180
8. Cai Y, Ng R (2004) Indexing spatio-temporal trajectories with chebyshev polynomials. In: SIGMOD, pp 599–610
9. Carey MJ, DeWitt DJ, Naughton JF (1993) The oo7 benchmark. In: SIGMOD conference, pp 12–21
10. Chakka VP, Everspaugh A, Patel JM (2003) Indexing large trajectory data sets with seti. In: CIDR
11. Chen S, Jensen CS, Lin D (2008) A benchmark for evaluating moving object indexes. PVLDB 1(2):1574–1585
12. de Almeida VT, Güting RH (2005) Indexing the trajectories of moving objects in networks. GeoInformatica 9(1):33–60
13. Duan S, Kementsietsidis A, Srinivas K, Udrea O (2011) Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In: SIGMOD Conference, pp 145–156
14. Düntgen C, Behr T, Güting RH (2009) Berlinmod: a benchmark for moving object databases. VLDB J 18(6):1335–1368
15. Frentzos E (2003) Indexing objects moving on fixed networks. In: SSTD, pp 289–305
16. Frentzos E, Gratsias K, Pelekis N, Theodoridis Y (2007) Algorithms for nearest neighbor search on moving object trajectories. GeoInformatica 11(2):159–193
17. Giannotti F, Nanni M, Pedreschi D, Pinelli F, Renso C, Rinzivillo S, Trasarti R (2011) Unveiling the complexity of human mobility by querying and mining massive trajectory data. VLDB J 20(5):695–719
18. Giannotti F, Nanni M, Pedreschi D, Pinelli F, Renso C, Rinzivillo S, Trasarti R (2011) Unveiling the complexity of human mobility by querying and mining massive trajectory data. VLDB J Spec Issue
19. Gidófalvi G, Pedersen TB (2006) St-acts: a spatio-temporal activity simulator. In: GIS, pp 155–162
20. González MC, Hidalgo CAR, Barabási A (2008) Understanding individual human mobility patterns. Nature 453:779–282
21. Gray J (ed.) (1993) The benchmark handbook for database and transaction systems (2nd Edition). Morgan Kaufmann
22. 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
23. Hu H, Lee DL (2005) Gamma: A framework for moving object simulation. In: SSTD, pp 37–54
24. Jensen CS, Lu H, Yang B (2009) Indexing the trajectories of moving objects in symbolic indoor space. In: SSTD, pp 208–227
25. Jensen CS, Tiesyte D, Tradisauskas N (2006) The cost benchmark-comparison and evaluation of spatio-temporal indexes. In: DASFAA, pp 125–140
26. Dinh L, Aref WG, Mokbel MF (2010) Spatio-temporal access methods: Part 2 (2003–2010). IEEE Data Eng Bull 33(2):46–55
27. Lu H, Cao X, Jensen CS (2012) A foundation for efficient indoor distance-aware query processing. In: ICDE, pp 438–449
28. Mascetti S, Freni D, Bettini C, Wang XS, Jajodia S (2008) On the impact of user movement simulations in the evaluation of lbs privacy- preserving techniques. In: PiLBA
29. Mouratidis K, Hadjieleftheriou M, Papadias D (2005) Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In: SIGMOD, pp 634–645
30. Myllymaki J, Kaufman JH (2003) Dynamark: a benchmark for dynamic spatial indexing. In: Mobile data management, pp 92–105
31. Nguyen AD, Sénac P, Ramiro V, Diaz M (2011) Steps-an approach for human mobility modeling. Networking 1:254–265
32. Pelanis M, Saltenis S, Jensen CS (2006) Indexing the past, present, and anticipated future positions of moving objects. ACM TODS 31(1):255–298
33. Pfoser D, Jensen CS (2003) Indexing of network constrained moving objects. In: GIS, pp 25–32
34. Pfoser D, Jensen CS (2000) Novel approaches to the indexing of moving object trajectories. In: VLDB

35. Pfoser D, Theodoridis Y (2003) Generating semantics-based trajectories of moving objects. Comput Environ Urban Syst 27(3):243–263
36. Popa IS, Zeitouni K, Oria V, Barth D, Vial S (2011) Indexing in-network trajectory flows. VLDB J 20(5):643–669
37. Ray S, Simion B, Brown AD (2011) Jackpine: a benchmark to evaluate spatial database performance. In: ICDE, pp 1139–1150
38. Reddy S, Mun M, Burke J, Estrin D, Hansen MH, Srivastava MB (2010) Using mobile phones to determine transportation modes. TOSN 6(2)
39. Saglio JM, Moreira J (2001) Oporto: a realistic scenario generator for moving objects. GeoInformatica 5(1):71–93
40. Saltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. In: SIGMOD Conference, pp 331–342
41. Schmidt M, Hornung T, Lausen G, Pinkel C (2009) Sp2bench: a sparql performance benchmark. In: ICDE, pp 222–233
42. Stenneth L, Wolfson O, Yu P, Xu B (2011) Transportation mode detection using mobile devices and gis information. In: ACM SIGSPATIAL, pp 54–63
43. Stonebraker M, Frew J, Gardels K, Meredith J (1993) The sequoia 2000 benchmark. In: SIGMOD Conference, pp 2–11
44. Tao Y, Papadias D (2001) Mv3r-tree: a spatio-temporal access method for timestamp and interval queries. In: VLDB, pp 431–440
45. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: VLDB, pp 287–298
46. Theodoridis Y (2003) Ten benchmark database queries for location-based services. Comput J 46(6):713–725
47. Theodoridis Y, Silva JRO, Nascimento MA (1999) On the generation of spatiotemporal datasets. In: SSD, pp 147–164
48. Tözün P, Pandis I, Kaynak C, Jevdjic D, Ailamaki A (2013) From a to e: analyzing tpc's oltp benchmarks: the obsolete, the ubiquitous, the unexplored. In: EDBT, pp 17–28
49. Tzouramanis T, Vassilakopoulos M, Manolopoulos Y (2002) On the generation of time-evolving regional data. 6 3
50. Werstein PF (1998) A performance benchmark for spatiotemporal databases. In: 10th annual colloquium of the spatial information research centre, pp 365–373
51. Wu K, Shoshani A, Stockinger K (2010) Analyses of multi-level and multi-component compressed bitmap indexes. ACM Trans Database Syst 35(1)
52. Xie X, Lu H, Pedersen TB (2013) Efficient distance-aware query evaluation on indoor moving objects. In: ICDE, pp 434–445
53. Xu J, Güting RH (2011) Infrastructures for research on multimodal moving objects. In: MDM, Demo Paper, pp 329–332
54. Xu J, Güting RH (2012) MWGen: a mini world generator. In: MDM, pp 258–267
55. Xu J, Güting RH (2013) A generic data model for moving objects. GeoInformatica 17(1):125–172
56. Yang B, Lu H, Jensen CS (2010) Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In: EDBT, pp 335–346
57. Zheng Y, Chen Y, Xie X, Ma WY (2010) Understanding transportation mode based on gps data for web application. ACM Trans Web 4(1):1–36
58. Zheng Y, Liu L, Wang L, Xie X (2008) Learning transportation mode from raw gps data for geographic applications on the web. In: WWW, pp 247–256
59. Zheng Y, Zhou X (2011) Computing with spatial trajectories. Springer
60. Zhou P, Zhang D, Salzberg B, Cooperman G, Kollios G (2005) Close pair queries in moving object databases. In: GIS, pp 2–11

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.

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.

Xiaolin Qin is a professor in the department of computer science and technology in Nanjing University of Aeronautics and Astronautics. His research topic includes spatial and temporal databases, database security and sensor network. He has published more than 90 papers as well as five books. His group has built the prototype database system NHDB.