Moving Objects with Multiple Transportation Modes

Der Fakultät für Mathematik und Informatik
der FernUniversität in Hagen
vorgelegte

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

von
Jianqiu Xu
Geburtsort: Nanjing, China

Hagen, 2012

Abstract

The new applications on detecting transportation modes and advanced trip recommendations make it crucial to investigate moving objects with different transportation modes. Practically, the complete trip of a person can cover several real world environments such as road network, bus network and indoor. However, in the database community, existing methods focus on a single environment and can only manage a sub trip of the continuous movement. Consequently, moving objects passing through various environments are not represented in a database system, and novel queries regarding transportation modes cannot be supported.

The aim of the thesis is to (1) define a data model that represents moving objects with different transportation modes in a database system; (2) provide efficient query processing mechanisms enabling to support novel queries; (3) evaluate the performance of such a database system called GMOD (Generic Moving Objects Database) that manages complete trips for humans. Towards this goal, at first we propose a generic method that is able to represent the location of a moving object in all environments. Since each environment consists of a set of geographic objects, a group of data types is designed to represent them. A data type for generic moving objects is defined where the location is represented by referencing to geographic objects. Afterwards, a set of operators is proposed to efficiently access the data. Definitions of both syntax and semantics are given. The proposed operators are also used to formulate queries. The data model is implemented in a database system.

In order to evaluate the performance of GMOD, a tool is developed to create real world environments based on roads and floor plans. All underlying geographic objects such as roads, buses, and buildings are managed by GMOD. Within each environment, a graph as well as the algorithm is proposed for the shortest path searching, e.g., trip planning for pedestrians, indoor navigation. A complete navigation algorithm is provided to find a route through several environments. As a second step, such a route is used to produce a moving object with different transportation modes. To simulate human behavior in a realistic scenario, some rules are defined to model movement patterns such as commuting, searching the nearest interesting place. The tool is extended to create comprehensive and scalable benchmark datasets for GMOD. A set of benchmark queries is proposed and optimization techniques are also developed. We conduct an extensive experimental study to evaluate the performance of GMOD and demonstrate the effectiveness and efficiency of proposed approaches.

Acknowledgment

First of all, I gratefully acknowledge my adviser Prof. Dr. Ralf Hartmut Güting, who provides me an opportunity to study in Germany and work with his research group. I still clearly remember the first time I met Ralf, when he warmly greeted me in his office room, offering coffee, nicely talking and introducing himself. During my ph.d study, Ralf gave invaluable academic guidance and constant support to my research, especially when I move toward the misleading way of research or make some mistakes. More broadly, he provided insightful suggestions from his vast experiences and knowledge on various aspects of doing research, e.g., how to identify a problem and define it formally. I have a great amount of benefits from this. Without his encouragement, I would not reach this point in my academic work. Ralf not only helps me on the research work, but also enriches my life in Germany, making it greatly interesting and significantly meaningful. Besides, thanks a lot for his nice family, in particular, his wife Edith Güting, who treats me kindly as one member of the family.

I am extremely thankful to all group members. In the first few days when I arrived at Germany, Anne Jahn helped me settle down in Hagen and organized many other things in daily life. Thanks to Simone Jandt, who provides the helpful module Network and reads many German letters for me. For the help on the implementation of SECONDO, I would like to thank Dr. Thomas Behr and Christian Düntgen. Dr. Thomas Behr introduced me many lower level system implementations with which I am not quite familiar and gave useful suggestions about programming in a system context. Thanks to Sara Betkas and Fabio Valdés, who help me improve the English writing, leading to a better quality of the thesis. To the other colleagues Frank Hoffmann, Mahmoud Attia, Jiamin Lu, it is a very nice experience for us to study and work together. Specially, to Frank, we stay in the same office room and have a good time together.

I express my gratitude to my friends in Hagen. Dr. Hongli Ma helps me a lot when I have problems about the language, making my stay in Germany easier. It is very good to know Dr. Tiansi Dong and Dr. Peilin Cui, who share their helpful study and living experiences in Germany. I gained a lot from them. Thanks to Prof. Dr. Zhong Li, who gave me the support in daily life. And we enjoy the time of playing soccer on the weekend.

I wish to express my deep gratitude to Chinese Scholarship Council. Without the funding provided by them, I cannot come to Germany and study for the ph.d. Thanks a lot for offering

me such an opportunity.

Finally, I would like to thank my parents Shulai Xu and Hongming Pan, for their efforts to provide me the best possible education as well as love and support through the whole ph.d study. They have done many things in the family and try to let me focus on the study without being disturbed by something else. To Xiaolei Cheng, a particularly important friend who supports me and gives me the encouragement all the time, I am really grateful to her and cherish the friendship between us. Thanks a lot to other friends in China for encouraging and supporting me to complete the ph.d study. I dedicate this thesis to all of them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

Recently, the area of moving objects with multiple transportation modes receives much attention in the literature [19, 69, 103, 81], due to novel applications related to detecting transportation modes and providing advanced trip plannings or recommendations. To fully understand the mobility of a traveler, researchers start to discover movement segments with determined motion modes from raw GPS data such as walking, driving or taking a bus, as these pieces of information denote significant characteristics of a mobile user's context.

In Microsoft's project *GeoLife* [3] researchers work on exploring and learning transportation modes of people's movement from raw GPS data to enrich the mobility with informative and context knowledge [105, 104, 103]. By mining multiple users' location histories, one can discover the top most interesting locations, classical travel sequences and travel experts in a given geo-spatial region, hence enable a generic travel recommendation.

Besides communication, mobile phones are being employed as devices to investigate the habits and situations of individuals and communities. In this context, one of the applications is to determine the transportation modes of an individual when outside. A convenient classification system is proposed in [69] with the aim of detecting transportation modes by using a mobile phone with a built-in GPS receiver and an accelerator. Besides the GPS sensor on the mobile device, the knowledge of the underlying transportation network such as bus stops and railway lines is also utilized to improve the accuracy and classification effectiveness [81].

In a transportation system, it is very meaningful to support trip planning with different kinds of motion modes (e.g., *Walk*$\rightarrow$ *Bus* $\rightarrow$ *Train*) and the constraint on a specific mode, for example, less than two bus transfers. Paper [19] presents a data model for trip planning with uncertainty in a multimodal transportation system in order to answer queries like

"*find a trip home from work, using public transportation, that minimizes the number of intermodal transfers made*",

"*find a trip home from work that arrives by 7pm with certainty* $\geq$ *0.8, and spends the least time*

possible on buses".

An interesting query called *Isochrones* is considered in [15] with the aim to find the set of all points on a road network where a specific point of interest can be reached within a given time span. Two transportation modes are taken into account, *Walk* and *Bus*. In ubiquitous and context aware computing [51, 50, 104], understanding the mobility of a client from sensor data is an important area of research. Additionally, finding the correlation between mobile users and locations will benefit from understanding the complete trajectory with precise transportation modes, e.g., recommending friends based on location history [106].

In respect of all the above work, taking into account transportation modes for moving objects becomes increasingly important. From the viewpoint of the database community, new challenges are imposed on a database system to manage trips passing through several environments (e.g., road network, indoor, bus network) and provide efficient query processing mechanisms to answer new queries. That is, the management of continuously changing locations with determined transportation modes requires dedicated support from the underlying database system. Consider the following two example movement scenarios of *Bobby*, drawn in Fig. 1.1. Only the outdoor movement is shown.

(a) Example Movement 1

(b) Example Movement 2

Figure 1.1: Trips with Multiple Transportation Modes

M_1: *starts from his house* A*, drives the car along the road to to the parking lot* B*, walks to his office building* C*, enters the building and goes to his office room.*

M_2: *walks from the house* A *to a bus stop* B*, and then takes a bus to the bus stop* C*, walks to the building* D *and goes to his office room.*

The two trips can be described by a sequence of sub movements with transportation modes:

- M_1: A $\xrightarrow{Car}$ B $\xrightarrow{Walk}$ C $\xrightarrow{Indoor}$ office room
- M_2: A $\xrightarrow{Walk}$ B $\xrightarrow{Bus}$ C $\xrightarrow{Walk}$ D $\xrightarrow{Indoor}$ office room

In fact, Google map supports a sequence of places to be visited and the user can choose private vehicles, walking or public vehicles, but different modes can not be specified for sub trips. The two trips above pass several environments and each environment owns its possible transportation modes.

To manage the location data in a database system, a generic data model is required where the model is intended to precisely represent the data in a system and has the capability of manipulating them efficiently and effectively. In spite of extensive research having been conducted on modeling moving objects, existing methods only target a single environment and can not manage the complete trip passing through several environments, e.g., M_1. According to the environment, the current state-of-the-art can be classified into three categories:

(1) *free space* [78, 96, 32, 38, 35];

(2) *road (spatial) network* [92, 40, 80, 37];

(3) *indoor* [45, 44].

Free space is an environment without movement constraint. But in practice, objects usually move on a pre-defined set of trajectories as specified by the underlying network (road, highway, etc). The first two are concerned with outdoor. In daily life, people also spend a large amount of time in indoor space such as office buildings, shopping centers, etc. According to the Nokia report [2], the percentage of time for people staying inside buildings (GPS activates only outdoor) is up to 90%. However, the models above are heterogeneous, designed independently. Different techniques are proposed for location representation and data manipulation. Each model can only manage the data in one environment without considering transportation modes. There is no interaction between different environments and one cannot know the place where people leave one environment and enter another. A complete trip with several transportation modes like M_1 (M_2) cannot be managed by previous techniques. For different applications, the movement is split into several parts each of which fits into one environment.

Besides free space, road network and indoor, there are still two environments that receive little attention: Public Transportation Network (PTN) and Region-based Outdoor (RBO). PTN seems to be similar to road network because both have pre-defined paths for moving objects. But there are still significant differences: (1) the movement in PTN depends on not only bus routes but also schedules; (2) in the road network one can move any distance along the road and change the direction at any junction if allowed, but bus travelers can only start and end their movement at bus stops, sole places for transfer. RBO represents the area for people walking outside such as pavement and footpath. The overall area in a city for outdoor walking can be considered as a relatively large polygon with many obstacles inside. The obstacles

denote places such as building blocks and junctions areas. People can move freely inside the polygon but not directly pass through obstacles.

Since real world environments have different characteristics such as constraint or free movement, 2D or 3D, movement depending on other vehicles, one needs to identify a general method of location representation for all of them. This imposes new challenges regarding (1) the representation and management of mobile data in the presence of accurate transportation modes; and (2) efficient query processing in a database system.

Due to the limitation of previous data models, interesting queries regarding transportation modes and environments cannot be answered. Consider the following examples:

(i) "*where is Bobby at 8am, e.g., at home, in the street or in the office room?*";

(ii) "*how long does Bobby walk during his trip?*"

The two queries cannot be supported in one system if the complete trip is not managed. To answer (i), one first has to identify the environment that *Bobby* belongs to at the given time instant. For (ii), the walk movement might have several parts located in different places. One first has to find all walking parts in the whole trip. Most existing methods [78, 96, 32, 38] use a pair (x, y) to identify a location; obviously, (x, y) cannot apply for indoor as it is a 3D environment. Of course, it is possible to extend the pair to a triple (x, y, z) to represent the location. But the method only focuses on geometric properties (e.g., coordinates). The raw location data (x, y, z) means nothing else than three real numbers so that the walking part cannot be recognized. Using the speed value might help determine a sub trip with a certain mode, but the speed suffers from the traffic condition, leading to imprecise data.

When designing a new DBMS technology, evaluating its effectiveness and testing the system behavior are crucial. The overall procedure often involves selecting a set of test databases, generating representative query workloads, and executing these workloads on the databases to evaluate the system performance. Such a systematic evaluation helps identify design problems, validate hypothesis, and evaluate the robustness and quality of the proposed approach. In particular, it allows one to examine a system's capability and helps determine its strengths or potential bottlenecks. A benchmark [24, 82, 29, 68, 73], 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 system. The well-known benchmark TPC is domain-specific and is increasingly irrelevant to the multitude of new data-centric applications, so that one should develop tools for application-specific benchmarking [85].

In the literature, there are various benchmarks designed for different applications. TEXTURE [31] is a benchmark to measure the relative strengths and weaknesses of combining text processing with a relational workload in an RDBMS. Due to the widespread adoption of the Resource Description Framework (RDF) for the representation of both open web and enterprise data, RDF data management systems proliferate and benchmarks [73, 29] are designed to test the scalability and performance of these systems. Jackpine [68] is a benchmark to evaluate spatial database performance including micro benchmarks and macro workload scenarios.

For the novel cloud services, some initial ideas are presented for a new benchmark that fits to the characteristics of cloud computing such as scalability, fault-tolerance [17, 16].

In the field of moving objects databases, simulation is widely accepted to provide synthetic data for designing and testing novel data types and access methods due to the difficulty of getting a large amount of real data. Besides, real data might not be comprehensive enough to thoroughly evaluate the system in consideration. Therefore, researchers develop tools to create synthetic data in a realistic scenario. GSTD [89] defines a set of parameters to control the generated trajectories. A network-based moving objects generator is proposed in [21, 22] for the traffic application. Indoor moving objects [45, 102] are created using floor plans by some pre-defined movement rules. [87] proposed 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. BerlinMOD [30] is a benchmark that uses the SECONDO DBMS [36] for generating moving-object data. Benchmarking moving objects indices is studied in [61, 46, 25], focusing on location update, current and near future positions.

However, existing data generators and benchmarks only deal with the movement in one environment (e.g., road network) and do not process transportation modes. Consequently, these data cannot be used for representing complete trips each of which can cover distinct environments (outdoor and indoor). Getting a large amount of real data with precise transportation modes is also difficult. The thesis deploys a method to create the benchmark datasets including moving objects and the environment data such as pavements, bus routes and buildings. Since the dataset for moving objects is to simulate the trips of humans, the method defined to create the data should follow movement patterns in practice. Creating moving objects in a pure random fashion makes no sense as usually there is an evident motivation to start a trip, accomplishing some tasks or performing an activity. This motivates the need to (1) develop a data generator capable of creating realistic datasets that can simulate human movement in the real world (2) effectively and efficiently manage the data in a database system.

1.2 Research Problems and Methods

The thesis focuses on three topics: (1) data model, (2) data generator and (3) benchmark. The data model is to define a framework and represent the data in a database system. Since it is not easy to get a large amount of real data, a tool is developed to create the data. Additionally, we introduce how these data are managed in the system. To evaluate the system performance, a benchmark is proposed. We elaborate the content of each topic in the following subsections.

1.2.1 Data Model

Usually, a person's trip contains several transportation modes, e.g., *Bus*, *Walk*. The trip should be represented in such a way that (1) a complete movement is managed; (2) the precise lo-

cation in each environment as well as the transportation mode is defined; and (3) one can efficiently retrieve a sub trip according to the transportation mode and distinguish sub trips with different modes.

Basically, there are two contrasting approaches to modeling geometric location data [77]: *field-based* model and *object-based* model. The first one considers the world as a continuous surface (layer) over which features (e.g., elevation) vary [27]. Some operations can be defined to manipulate different layers. The second treats the world as a surface littered with recognizable and geographic objects (e.g., cities, rivers). This method is widely used for modeling the location of moving objects [78, 96, 38, 72, 93, 20, 59, 37]. The dissertation follows the method of the *object-based* approach but models all available environments instead of one. We let the geographic space be covered by a set of so-called infrastructures, denoting real world environments. Five infrastructures are considered in total: (1) Public Transportation Network, (2) Region-based Outdoor, (3) Free Space, (4) Road Network, and (5) Indoor. Each infrastructure consists of a set of components called infrastructure objects (IFOBs) that represent available places for moving objects. For example, in road network, IFOBs are roads and streets. Pavements and footpaths represented by polygons constitute RBO. The location of moving objects is represented by referencing to these IFOBs.

The continuously changing location data is abstractly described by a function from time to a location in geographical space. In free space, a pair (x, y) [78, 96, 32, 38] is used to identify a location. A road network position is denoted by (rid, pos) [37] where rid ($\in D_{\underline{int}}$) records a road identifier and pos ($\in D_{\underline{real}}$) records the relative position on the road. Compared with outdoor, an indoor model should consider both horizontal and vertical positions to uniquely identify a location.

The goal of the data model is to have a representation method that is general and consistent in all cases rather than limited to one. At the same time, the unique feature of each individual situation should not be lost. There are two challenges: (1) IFOBs (e.g., roads, polygons) have diverse characteristics and are represented by different data types in the database system; (2) the location is related to these heterogeneous IFOBs. Additionally, although there is no IFOB in free space, the location model must maintain the feature for free space in order to seamlessly integrate such an environment, as the aim is to represent moving objects in all cases. We also call moving objects with different transportation modes generic moving objects.

The new data model [101] is designed in such a way that on one hand a generic method is proposed to represent the precise location in all infrastructures. On the other hand, the location representation of previous work in free space and road network [1] can be integrated so that former techniques can be supported without too much effort. In the sequel, the generic location representation regresses to the specific representation in free space and road network, and previous models are integrated as two instances into the new model. One does not have

[1] indoor location is not precisely represented by existing models

to design new data types and operators for environments that have already been well established. The new model is not simply a wrapper because (i) PTN and RBO are not treated by previous techniques but modeled by our method; (ii) indoor location is precisely represented.

The data model defines (1) a robust method that can represent the continuously changing location in all available environments; (2) a set of data types representing IFOBs and moving objects; (3) a group of operators for efficiently accessing and manipulating the data. Since environments have different characteristics, IFOBs in each case are represented by different data types in the database system. It is straightforward that a line represents a road and a polygon denotes the area for a pavement. But for PTN and Indoor, new data types are needed as objects have special features that cannot be represented by existing data types. To efficiently support query processing, operators have to be defined in the system for users to access the data and formulate queries.

1.2.2 Data Generator

The database system managing generic moving objects is named GMOD (Generic Moving Objects Database). The data managed by GMOD include two parts: infrastructure data and moving objects. We represent the location of a moving object by referencing to the underlying infrastructure, more specifically, mapping to IFOBs. There is a large amount of such objects, roads, pavements, buses, rooms, etc. These data are heterogeneous and it is not easy to get all of them in a real and consistent environment. Consequently, a tool called MWGen (Mini World Generator) is developed to create all real world environments.

The tool produces all outdoor data from a road dataset which is easy to attain (many public resources) and indoor data from public floor plans (e.g., [11, 8]). MWGen takes the roads as input, defines some parameters and creates the infrastructures: Region-based Outdoor (pavement areas) and Public Transportation Network. The second part includes Bus Network and Metro Network, each of which contains routes, stops and a set of trips following certain schedules. A set of buildings is generated by MWGen with floor plans as input. Since the input data (roads and floor plans) have a lot of public resources, the proposed method is available for other researchers to create their own data.

As a second step, moving objects are created based on the route planning where the start and end locations can be in any environment. If they belong to the same environment, then the navigation algorithm within that environment is called. For each infrastructure, a graph as well as the algorithm is designed for the shortest path searching, e.g., trip planning for pedestrians, indoor navigation, bus routing. If the start and end locations belong to different environments, the complete navigation algorithm through all available environments is needed. We develop such an algorithm to create a route like *Indoor*$\rightarrow$*Walk* $\rightarrow$ *Bus* $\rightarrow$*Walk*. The resulting path and a set of defined speed values (e.g., car speed, bus speed, walking speed) are used to create a generic moving object.

Besides the main purpose of creating benchmark data for evaluating GMOD, MWGen can be utilized for some other applications:

(1) **Route Planning and Travel Recommendation**. MWGen can offer a trip passing through different environments, while most existing works [54, 75, 43] are limited to one environment. Consider such a query, "*tell me how to reach my apartment from the office with the minimum traveling time*". The environments include Indoor, Road network, Region-based Outdoor, possibly also Bus (Metro) Network depending on whether the traveler would like to travel by car or public transportation system. The traveling time between by car or bus and by metro might have a large deviation if there is a heavy traffic jam. The system can recommend the transportation modes for a traveler as long as the traffic condition is up-to-date. Additionally, ride sharing (using public vehicles) might be suggested for relieving traffic congestion. Trip planning in a single environment is also available by using MWGen. For example, the shortest path for pedestrians can be provided, and one can get a precise indoor shortest path between two locations inside a building.

(2) **City and Traffic Simulation.** The created data include all transportation environments of a city as well as buildings. This could simulate a metropolitan environment where urban planners can perform some investigation (e.g., buildings distribution, optimizing bus routes), and adjust time-dependent traffic regulations for intelligent transportation services and policies. As moving objects contain multiple transportation modes, one can analyze the traffic state by populating a large amount of mobile data and test whether the traffic jam can be relieved by improving and adjusting the public transportation system.

1.2.3 Benchmark

A benchmark called GMOBench [99] is proposed to evaluate the performance of GMOD by a broad range of novel queries on moving objects regarding transportation modes and the underlying environments. We process the complete past movement (also called trajectory) and generate scalable and comprehensive datasets to simulate a variety of real life scenarios. For the evaluation to be meaningful, input data sets must be carefully chosen to exhibit a wide range of patterns and characteristics in a realistic way. To achieve such a goal, a set of movement rules is defined to create trips with some features to model human movement in practice. People's trajectories exhibit *regular patterns* [34] most of the time, e.g., traveling between home and work. On the weekend, based on the habits and preferences they may have some trips to interesting places such as home of friends, shopping malls, etc. To perform an activity, people usually prefer nearby to distant places (nearest neighbor queries). We define a set of parameters with the aim of letting users create the desired data by configuring different settings.

Two groups of queries are proposed to test a variety of operator constellations and data access methods where one copes with the underlying environments and the other deals with

moving objects. Most of the queries are not supported by existing methods for the reason that previous work is limited to one environment. Optimization strategies are developed to improve the query efficiency. The transportation modes of a moving object are represented and stored in a compact way. Indices are built on moving objects to accelerate the procedure of accessing data. We carry out a thorough experimental study for performance evaluation under comprehensive datasets and a wide range of queries to validate the effectiveness and efficiency of the proposed techniques.

1.3 Summary of Contributions

The contributions of the thesis include three parts.

A Generic Data Model

The dissertation formulates the concept of space and infrastructure and defines their components. A general definition for heterogeneous IFOBs in all environments is given. We propose a method to represent the location of moving objects in different environments and design a framework for moving objects representation. Both outdoor and indoor movements are uniformly treated. We model available places for each infrastructure and propose the data type representing its components, i.e., IFOBs. Applying the proposed framework, we show how the location is represented in each infrastructure. For the indoor environment, we define a graph model for indoor navigation that supports optimal route searching with respect to different costs (e.g., distance, time).

A type system is defined for the generic model and a relational interface is provided to exchange information. A set of operators is designed on the proposed data types where both syntax and semantics are defined. A group of example queries on all available environments and various transportation modes is formulated, demonstrating the expressiveness of the querying framework. The data model is implemented in an extensible database system SECONDO, where the development includes (1) data types representing all IFOBs and moving objects; (2) infrastructures; (3) a group of operators to access the data such as **trajectory** and **passes**; (4) the relational interface.

MWGen: A Mini World Generator

We develop a tool named MWGen to create a set of real world infrastructures (both outdoor and indoor). The outdoor part contains RBO, Bus Network and Metro Network. The indoor infrastructure consists of a set of buildings. A global *space* which is built on top of all infrastructures is constructed to manage them. We introduce how all infrastructure data and the space are stored and managed in GMOD. A 3D viewer is developed in the system to display indoor objects such as rooms and paths. The animation of indoor moving objects is also available.

Within each infrastructure, a graph as well as the algorithm is proposed for trip plannings

so that one can search an optimal route. The following navigation algorithms are developed: shortest path for pedestrians, bus routing, metro routing, and indoor navigation. Optimization techniques are also proposed for each algorithm to reduce the time cost. We analyze the time complexity of each algorithm. In particular, an efficient algorithm of trip planning for pedestrians is proposed where a sub routine for searching visible points achieves a significant performance improvement. Both theoretical analysis and experimental results confirm the advantage of the technique.

A complete navigation algorithm is developed to provide a route passing through multiple environments. The resulting path is used to create a generic moving object. A detailed experimental evaluation on MWGen is reported, including the statistics of created databases, efficiency of trip planning algorithms, and the scalability of creating large amounts of generic moving objects.

GenMOBench: A Benchmark for Generic Moving Objects

A benchmark is proposed to evaluate the performance of GMOD where a set of movement rules is designed to guide the generation of benchmark data in order to have a realistic scenario. For example, we consider the *regular* movement pattern that is between home and work, and simulate the real case by taking into account the motivation of starting a trip, e.g., nearest neighbor searching, interesting places visiting. The data can be generated by configuring different parameters to produce certain distributions according to application requirements. The data generation algorithm is presented by using an example movement.

We propose two groups of queries for the benchmark workload with one on infrastructure data and the other on generic moving objects. Optimization techniques are developed to reduce the cost of query processing. Transportation modes of a moving object are represented in a compact way and indices are built on moving object units to accelerate data accessing. Besides the two techniques available to almost all queries, we develop a method for a specific query. As a result, the time is greatly reduced. An extensive experimental study is conducted on comprehensive datasets including infrastructure data of two cities and different sets of moving objects. The experimental results confirm the effectiveness and efficiency of the solutions.

1.4 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, the related work is reviewed. The data model is introduced in Chapter 3 and the data generator is presented in Chapter 4. We show the benchmark and evaluate the performance of GMOD in Chapter 5. Finally, the conclusions of this study are made in Chapter 6 where we summarize the contributions and discuss future directions.

Chapter 2

Literature Review

2.1 Data Models

In the database literature, although there has been a large body of recent works [96, 38, 84, 95, 40, 80, 20, 28, 59, 37, 52, 66, 67, 47, 44] on modeling moving objects, all of them deal with the movement in one environment and do not investigate transportation modes.

The closest to our work are [20, 59, 37] where the models consider the underlying environment for location representation. In [20], a semantic model is proposed for representing trajectories based on background geographic information. An algorithm is developed to map the positions of vehicles into a road network. Thus, the spatial aspect of trajectories is modeled in terms of a network, which consists of edges and nodes. However, the method is restricted to a specific application domain. To answer mobility pattern queries [59], a model is designed that relies on a discrete view of the underlying space for moving objects. The authors partition the space into a set of zones, each of which is uniquely identified by a label. Afterwards, the location is represented by mapping into zones, and a trajectory is defined as a sequence of labels. A so-called *route-oriented* model is presented in [37] for moving objects in networks. The method represents a road network by routes and junctions, and trajectories are integrated with the road network. A line is used to describe the geometrical property of a road. Then, a network location is represented by a road id and the position on the road.

Nevertheless, the aforementioned models have the following drawbacks: First, the methods can only represent the location in one environment, making the model not general. Only a sub trip can be defined and positions of a complete trip passing several environments cannot be managed in a database system. Consequently, queries are limited to one environment. Second, the location in [59] is not precisely represented, only identified by a symbol pointing to a zone. The model cannot answer a precise location query. Furthermore, moving objects are represented in a discrete way (a sequence of timestamps) instead of continuous. As a result, the method is limited to one application. Third, transportation modes are not handled by these techniques.

GTFS (General Transit Feed Specification) [1] defines a common format for public transportation schedules and associated geographic information so that people can use the GTFS specification to provide schedules and geographic information to Google Maps. The specification contains some files (required and optional), and each file with a defined format records a list of items each of which stores a field name and the value. For example, a stop file includes the following required field names: *stop_id*, *stop_name*, *stop_lat*, *stop_lon*. Locations of a bus traveler are a set of items recording bus stops as well as arrival and departure time at each stop. But the method does not specify how to determine the positions between two *adjacent* stops, and the representation for moving buses is not included. In fact, the locations of a bus traveler depend on the bus and do not have to be additionally represented.

GPS is the dominant positioning technology for outdoor settings. For the indoor environment, new techniques are required. A graph model is proposed in [44] for indoor tracking moving objects using the technology RFID. The method assumes that RFID readers are embedded in the indoor space in some known positions. This is based on the condition that the indoor space is partitioned into a set of cells corresponding to vertices in the graph. An edge in the graph indicates the movement between cells and such a movement is detected by RFID readers. A raw trajectory is in essence a sequence of RFID tags. A method is developed to construct and refine the trajectory with the goal of improving the indoor tracking accuracy. *Jensen et al.* [45] present an index structure for moving objects in a symbolic indoor space. The trajectory model is composed of records in the format $(oid, symbolicID, t)$ where oid is the moving object identifier, $symbolicID$ is the identifier for a specific indoor space region and t indicates the time. The method defines the location in an imprecise way and does not give the data type representing a space region. The room where the object is located can be known, but the precise location inside is not determined. In some cases, an indoor space may occupy a large area such as a hallway or an amphitheater, leading to the approximate location representation being not feasible for the application.

Recently, semantic trajectory [13, 79, 18, 23] receives much attention in the literature, complementing the recording of moving positions. The aim is to enrich trajectories with semantic annotations and allow users to attach semantic data to some specific parts of the trajectory. A trajectory is defined as two parts called *stop* and *move*, where a *stop* is defined as a specified spatial location according to the application and a *move* is the part of a trajectory delimited by two consecutive stops. However, those works do not propose a method for modeling moving objects but involve a data pre-processing model to add semantic information to moving objects trajectory before query processing. That is, semantic data is not represented by the data model. Besides, the meaning of semantic data specifies an interesting place, e.g., a hotel or a touristic place. This is different from the intention of the present paper, dealing with moving objects with multiple transportation modes.

2.2 Transportation Modes

2.2.1 Multimodal Transportation System

A data model presented in [19] gives the framework of an advanced transportation system. The model is able to provide a trip consisting of several transportation modes, e.g., *Walk* → *Bus* → *Walk* → *Train*. Moving objects databases and graph-based databases are integrated to facilitate trip planning in urban transportation networks. The authors deal with returning the shortest path with multiple transportation modes to connect the origin and the destination. Besides returning a path with the minimum time or distance, the result can have more constraints and choices, e.g., different motion modes, the number of transfers. A graph model is defined where each vertex corresponds to a place in a transportation network. The place has a name and a geometric representation, e.g., a point or a region. Each edge is associated with a particular transportation mode. Edges with different modes can be incident on the same vertex indicating that a transfer between different modes can happen. A trip is defined as a sequence of *legs* each of which represents a path with the transportation mode.

However, the model aims to provide trip plannings with various constraints rather than model moving objects in all transportation environments. The authors assume that moving objects with different transportation modes are already obtained and managed by the database system. The proposed *leg* and multimodal trip are not represented by defining data types in a database system, but described conceptually and abstractly. Besides, the indoor environment is not considered. The dissertation focuses on representing moving objects in all real world environments and managing multimodal trips in a database system so that users can issue queries in a system context. In the general model, transportation modes are seamlessly integrated into moving objects.

An interesting query is proposed in [15] that computes isochrones in multimodal and schedule-based transport networks. The goal is to find a set of points on a road network, and from them a specific point of interest can be reached within a given time span. But the method only considers two transportation modes: *Walk* and *Bus*.

2.2.2 Detect Modes from GPS data

GPS data-based activity recognition has received considerable attention during the past years due to the rapid development of mobile devices. Besides communication, mobile phones are being employed as instruments for introspection into habits and situations of individuals and communities, especially detecting transportation modes. Existing work on detecting the transportation mode of a mobile client can be distinguished by raw input data: (1) only use GPS data [105, 103, 69]; (2) use GPS track and some other information [64, 55, 81, 97].

In Microsoft's project GeoLife, the works [104, 105, 103] aim to discover and infer trans-

portation modes from raw GPS trajectory data. By mining multiple users' location histories, one can discover the most interesting locations, classical travel sequences and travel experts in a given geo-spatial region. The difficulty is how to partition a user's trajectory into several segments and identify the transportation mode for each piece. This is because the user's trajectory can contain multiple kinds of modes and the velocity of a mode suffers from the variable traffic condition. The procedure comprises three phases: (1) a GPS trajectory is decomposed into several segments of different transportation modes, while maintaining a segment of one mode as long as possible. A set of features being independent of the velocity is identified. (2) the determined features are fed into a classification model to output the probability of each segment with different transportation modes. (3) a graph-based postprocessing algorithm is developed to further improve the inference performance. The identified features include basic features such as velocity, acceleration and advanced features consisting of heading change rate, stop rate and velocity change rate. Based on the proposed segmentation method and Decision Tree-based inference model, an inference accuracy greater than 71% is achieved.

The work in [69] is to determine the transportation modes of an individual when outside where five transportation modes are considered: stationary, walking, running, biking and in motorized transport (car, bus, train). A convenient classification system is proposed that used a mobile phone with a built-in GPS receiver and an accelerator where an accuracy level of 93.6% is achieved. The classifier system is able to run on a commodity mobile device, and the overall classification system consists of a decision tree followed by a first-order discrete Hidden Markov Model.

Only using GPS information might reduce detection accuracy, thus the underlying transportation networks are utilized to improve the result. An approach to inferring a user's mode is proposed in [81] where the method is based on the GPS sensor from the mobile device as well as the knowledge of the underlying transportation network, e.g., bus stop locations. With these pieces of information, different motorized transportation modes can be distinguished such as car and bus. Furthermore, if the transportation mode *Bus* is determined, the algorithm can additionally provide the information about which particular bus the traveler is on. Such an improvement is not achieved by the other method. New features are identified to detect transportation modes besides traditional features such as speed and acceleration. The accuracy over 93.5% is obtained for inferring various transportation modes.

Three transportation modes are learned in [97]: driving, biking and walking. The method relies on user's historical data and the geographical information. A Hidden Markov Model is established with the hidden states being transportation modes, whose switch and maintenance follow the Markov property. In [64, 55], an unsupervised learning technique is proposed to use GPS tracks as well as GIS data to classify a user's transportation mode as either *Bus*, *Walk* or *Car*, and predict the most likely routes. In addition, historical information including past trips, parking places is also used to predict the traveler's destination and the trip purpose. The system proposed in [55, 56] first detects a set of significant places for the user, and then

recognizes activities like shopping and dinning that can take place at those places.

Compared with the aforementioned work, this thesis has a different focus. The thesis does not discover and predict transportation modes but concentrate on modeling moving objects with various and precise transportation modes. We model both *outdoor* and *indoor* movement while the existing work only take into consideration the *outdoor* movement because they infer based on GPS data where a GPS receiver will lose signal indoors. When people move inside a building or travel by underground trains (subways), the mode can not be detected.

2.3 Benchmark

2.3.1 Data Generators

In the literature, there are a lot of generators for moving objects data [89, 88, 70, 22, 65, 30]. These tools use random functions and road networks to model different physical aspects of moving objects. However, the so far developed tools only consider the movement in one environment including (1) free space; (2) road network; and (3) indoor. Thus, they are not able to create a moving object passing through several environments. The created data by existing generators do not include transportation modes and cannot represent generic moving objects. We review the generators for each environment as follows:

free space: GSTD [89], a widely used spatio-temporal generator, defines a set of parameters to control the produced trajectories: (1) the duration of an object instance; (2) the shift of objects; (3) the resizing of objects. Initialized by a certain distribution of points or rectangles, 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 behaviors such as group movement and obstructed movement [65], by introducing the notion of clustered movement and a new parameter. [88] develops a method to create a set of moving points or rectangles in accordance with user requirements, thus providing a tool that can simulate a variety of possible and practical scenarios. A tool is proposed in [30] to generate moving objects based on trip plannings, e.g., from home regions to work regions. The authors create cars moving in Berlin streets and record complete histories of movements. Two kinds of methods are used for setting positions and velocities for moving objects generation in [71], uniform distribution and skewed distribution.

road network: A network-based moving objects generator is proposed in [21, 22] for the traffic application. Objects are created in such a way that each appears from the start position and disappears when the destination is reached. Important concepts of the generator are maximum speed, maximum edge capacity, start and end nodes, etc. The speed and the route of a moving object depend on the load of network edges, i.e., the number of cars on the edge.

indoor: [45, 102] define a set of rules to generate indoor movement based on floor plans, e.g., an object in a room can move to the hallway or move in the staircase, or an object in a

staircase can move to the hallway. For each rule, an object randomly chooses a room as the destination. When the object enters the destination room, it will move inside the room for a random time duration and then start the next movement.

In addition to the above data generators, there are also various spatio-temporal simulators for different applications, but all of them do not consider the detailed routing between locations in different environments and how people move from one place to another. Thus, transportation modes and distinct environments are not addressed. A microscopic traffic simulator called SUMO [4] performs collision free vehicle movement with such features like multi-lane streets, different right-of-way rules. G-TERD [91] is a generator for time evolving regional data in an unconstrained space and Oporto [70] is a realistic scenario generator for moving objects motivated by a specific application, fishing. The simulator ST-ACTS [33] uses various 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 [63] is a parametric mobility model for human mobility with the aim of making the abstraction of spatio-temporal preferences in human mobility. The method is based on 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 [42] is a proposed framework in which the 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 the cellular space and the real life symbolic moving behavior.

The goal in [58] is to provide a friend-finder service. The considered places are not general, including only home and entertainment places. In addition, 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 distribution, 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 almost uniformly distributed on the map, with a minor concentration on central zones of the city.

2.3.2 Moving Objects Benchmark

A benchmark is proposed in [94] for 3-dimensional spatio-temporal data which require significant temporal processing and storage capability, leading to provisions for evaluating the ability of a spatio-temporal database to handle three dimensional data. The work expands on Sequoia 2000 and Paradise benchmarks, and is oriented towards general operating system and database system performance comparison. In the context of moving-object databases, [87]

proposes a benchmark that includes a database description and SQL-based queries. A set of ten representative benchmark queries are proposed plus two operations for loading and updating data. 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 [30] is a benchmark that uses the SECONDO DBMS [36] for generating moving objects. A realistic scenario is simulated where a number of cars move within the road network of Berlin and sampled positions from such movements are recorded and used as benchmark data. The method models a person's trips to and from work on workdays as well as some additional trips in the evening and on the weekend. Long-term observations of moving objects are also captured, e.g., one month movement. A set of carefully selected SQL-based queries constitute the workload.

Benchmarking moving objects indices is studied in [61, 46, 25], focusing on location update, current and near future positions. The paper [61] proposes a benchmark termed DynaMark for dynamic spatial indexing, the purpose of which is towards location-based services. Three types of queries are defined that form the basis of location-based service applications: proximity queries, kNN queries and sorted distance queries.

COST [46], a benchmark concerned with the indexing of current and near-future moving objects positions, 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. The concepts of data and query enlargement are introduced for addressing inaccuracy. Different indices such as TPR-tree and TPR*-tree are studied for the performance evaluation. In [25], the following three types of moving objects datasets are generated to simulate certain real-world scenarios and to measure the overall index efficiency: (1) uniform distribution; (2) Gaussian distribution and (3) road-network-based datasets. The query workloads consist of the range query and the kNN query. A set of aspects is proposed for the performance evaluation such as data size, update frequency and buffer size. The aim is to establish a comprehensive benchmark that offers an important insight into the behavior of different index techniques and provides the guidance on index selection and improvement.

However, these benchmarks only process moving objects and queries in one environment and do not consider transportation modes. Compared with the aforementioned work, the benchmark in the dissertation is general in the context of moving objects where the system manages trips passing through different environments and supports new queries on these data.

2.3.3 Route Planning

Trip plannings [54, 86, 75, 43, 48] have been investigated in one surrounding recently, e.g., free space or road network. The purpose is to provide an optimal and precise route according to some user defined constraints. However, these works cannot answer a query like "*find the shortest path with the minimum time from my home to the office room*", where multiple environments and transportation modes are involved.

In [54], the user specifies a set of interesting points each of which belongs to a specific category, and asks for the best route from the start location to the destination. The returned path should pass through at least one point from each category. Several approximation methods are proposed to provide near-optimal answers with approximation ratios that depend on either the number of categories or the maximum number of points per category. k-stops shortest path problem [86] seeks for the optimal path between two locations passing through exactly k intermediate points in Euclidean space. The two aforementioned works can be solved by OSR (optimal sequenced route) algorithms [75] where the authors define a sequence of locations to be passed by the user and compute the exact optimal route both in vector and metric spaces. The R-tree index is used to improve the procedure of examining threshold values, which are employed to prune locations that cannot belong to the final optimal route. The resulting path is built in reverse sequence, i.e., from the ending point to starting. For applications in a road network that cannot tolerate the Euclidean distance, a method based on progressively finding the nearest neighbors is proposed. A theoretical analysis of time and space cost is also provided.

[48] formulates a network mobility model that offers a concise representation of mobility statistics extracted from massive collections of trajectories. The paper aims to determine road segments along which an object will travel beyond the next junction. The method captures the turning patterns at junctions and the travel speeds on road segments at the level of individual objects. Taking traffic condition into account, [43] studied answering path queries in road networks with uncertainty. Three novel types of probabilistic path queries are proposed, e.g., "*what are the paths from my hotel to the airport whose travel time is at most 30 minutes with a probability of at least 90%*".

Multi-Geography Route Planning (MGRP) is investigated in [14] where the goal is to determine the least cost weighted paths from sources to destinations, both of which may reside in different geographies. That is, geographies are described in multiple representation paradigms, e.g., raster versus vector, different coordinate representations. The primary motivation of MGRP is to study the real time and safely routing through unfamiliar spaces where an emergency event occurs resulting in evacuation. The place considered is usually in a small area, e.g., a campus.

Chapter 3

A Data Model for Generic Moving Objects

3.1 Generic Data Model

In this section, we formulate the concept of space and infrastructure, and introduce their components. The representation for locations and moving objects is defined. We also present the method of representing the location in an approximate way.

3.1.1 Preliminaries

We give the carrier set of basic types that we use for definitions in the following sections [1].

Definition 3.1.1 *Base Types*

$D_{\underline{int}} = \mathbb{Z} \cup \{\perp\}$

$D_{\underline{real}} = \mathbb{R} \cup \{\perp\}$

$D_{\underline{bool}}$ = {*FALSE, TRUE*} $\cup \{\perp\}$

Each domain is based on the usual interpretation with an extension by the undefined value, denoted by $\perp$. Three data types are given for representing time: *instant*, *interval* and *range*, defined as follows:

Definition 3.1.2 *Time Types*

instant: $D_{\underline{instant}} = \mathbb{R} \cup \{\perp\}$

interval: $D_{\underline{interval}}$ = $\{(s, e, lc, rc) | s, e \in D_{\underline{instant}}, lc, rc \in D_{\underline{bool}}, s \leq e,$
$(s = e) \Rightarrow (lc = rc = true)\}$

range: $D_{\underline{periods}} = \{V \subseteq D_{\underline{interval}} | \, (u, v \in D_{\underline{interval}} \wedge u \neq v)$
$\Rightarrow$ *disjoint*$(u, v) \wedge \neg$ *adjacent*$(u, v)\}$

[1] Using the algebraic terminology that for a data type α, its domain or carrier set is denoted as D_α.

We use *instant* to denote the data type for time instant, which is based on the *real* type. The value of a time interval defines a set of time instants. The type *periods* represents a set of disjoint and non-connected time intervals. Additionally, we also give the *intime* type constructor which yields types whose values consist of a time instant and a value. Let α denote an abstract type (excluding time type), e.g., *int*, *real*.

Definition 3.1.3 $D_{\underline{intime}} = D_{\underline{instant}} \times D_{\alpha}$

Besides basic and time types, spatial and temporal data types are also employed from [32, 38]: *point*, *points*, *line* and *region*, *mbool* (moving bool) and *mpoint* (moving point). The definitions of *mbool* and *mpoint* are given in Appendix C.

3.1.2 Location Framework

3.1.2.1 Space and Infrastructures

The space for moving objects is covered by the following infrastructures: Free Space, Road Network, Public Transportation Network, Region-based Outdoor and Indoor. Each infrastructure consists of a set of IFOBs and defines available places for moving objects. For example, roads and streets constitute Road Network, and a set of polygons specifies the walking area for Region-based Outdoor. For Free Space, the object set is empty. A notation is defined for each infrastructure. Table 3.1 lists the components for space as well as possible transportation modes in each component. We summarize all transportation modes in Def. 3.1.4.

$Space$	I_{fs}	*Free*
	I_{rn}	*Car, Bike, Taxi*
	I_{ptn}	*Bus, Train, Metro*
	I_{rbo}	*Walk*
	I_{indoor}	*Indoor*

Table 3.1: Space Components

Definition 3.1.4 *Transportation Mode*
$D_{\underline{tm}}$ = {*Car, Bus, Train, Walk, Indoor, Metro, Taxi, Bike, Free*}

Let $I = \{I_{fs}, I_{rn}, I_{ptn}, I_{rbo}, I_{indoor}\}$ be the set of infrastructures and $Dom(I_i)(I_i \in I)$ be the values (IFOBs) for I_i. Then, the *space* is defined as follows:

Definition 3.1.5 $Space = \bigcup_{I_i \in I} Dom(I_i)$

Let *space* be the data type representing a space with domain: $D_{\underline{space}} = Space$.

Since people also walk in indoor space, in the following the mode *Walk* means outdoor environment by default. This is to distinguish between the modes *Walk* and *Indoor*. The dissertation only deals with transportation modes for objects moving on the ground. Of course, there are still two modes: *Ship* and *Airplane*. Normally people do not frequently change to these two cases in daily life so that they are not considered in this paper. The two modes can be easily integrated because both of them can be considered as modes in a public transportation system.

3.1.2.2 Infrastructure Components

The components for an infrastructure are objects representing background geographic information. Infrastructures have their own characteristics, resulting in geographic objects with different representations. To manage all these objects in a database system, some data types are needed. For instance, in I_{rn} a line is used to describe the geometrical property of a road, and in I_{rbo} a polygon or a region (in the following, we use terms polygon and region interchangeably) is used to identify the pavement area. The overall IFOBs include not only spatial objects but also spatial-temporal objects such as buses and trains. The data type of an IFOB depends on the infrastructure feature. To have a general description, we give the definition of an IFOB below.

Definition 3.1.6 *Infrastructure Object (IFOB)*

An IFOB is defined as $w(oid, s, \beta, name)$ *where* $oid(\in D_{\underline{int}})$ *is a unique identifier,* s *is a symbol for a data type,* β *is a value of the type, and a string describes the name.*

The meaning of s is clear for I_{rn}, I_{rbo} and I_{fs} where $s \in$ IOSymbol = {LINE, REGION, FREESPACE}. The data types $\underline{line}$ and $\underline{region}$ are already defined in spatial databases. For free space without IFOBs, we let the symbol be FREESPACE. Regarding I_{ptn}, the components are routes and moving vehicles that have special features. Taking the bus network as an example, a bus route cannot be simply represented by a line because the information of bus stops is missing. When modeling bus trips, both the route and the schedule should be considered. In I_{indoor}, a building consists of a set of rooms. Intuitively, the object representing a room should contain such two pieces of information (1) 2D area and (2) height above the ground level, enabling to uniquely identify a location inside a room. New data types are designed for I_{ptn} and I_{indoor}, defined in Section 3.2.

3.1.2.3 Location Representation

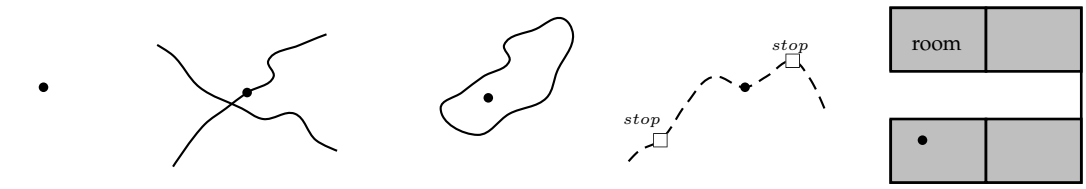
To achieve the goal of a general and precise location representation in all environments, in the proposed model the location of a moving object is described by two parts, defined as follows:

Definition 3.1.7 *Generic Location*

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

The first part is an identifier corresponding to an IFOB and the second stands for the relative position according to that object, described by (loc_1, loc_2). Each IFOB has its own geographical information such as a line for a road, a polygon for a pavement. By accessing the referenced object, the range of the location is determined. With the second value, a precise location can be obtained. Using this method, the location in all environments listed in Table 3.1 can be represented. In the following, we explain the semantic meaning of generic location for each infrastructure and give examples in Fig. 3.1.

- $(\perp, (loc_1, loc_2))$ maps to a location in free space. Since I_{fs} does not contain any IFOB, we set oid by $\perp$ and represent the position by recording coordinates (loc_1, loc_2).
- A road network location is represented by $(oid, (loc_1, \perp)$ where oid records a road id and loc_1 describes the location on the road. The value of loc_1 is between zero and the length of the road. loc_2 is not required and set as undefined.
- Regarding Region-based Outdoor, a location $(oid, (loc_1, loc_2))$ is denoted by (1) a polygon id and (2) the relative location inside the polygon, with the left lower point of the polygon bounding box being the origin point.
- In a public transportation system, a location $(oid, (loc_1, loc_2))$ is specified by a route id, a stop number loc_1 and the distance loc_2 from the stop on the route.
- For an indoor location, we let oid map to a room and (loc_1, loc_2) record the location inside the room. Since I_{indoor} is a 3D environment, the height above the ground level of a location needs to be determined. This value is recorded by an IFOB representing a room. All locations inside one room have the same height as the room, so the value does not have to be explicitly recorded. The concept of room is general, e.g., an office room, a corridor.



(a) I_{fs} (b) I_{rn} (c) I_{rbo} (d) I_{ptn} (e) I_{indoor}

Figure 3.1: Location in Each Infrastructure

To sum up, the proposed method is able to define the location for all environments. The representation first maps to an IFOB and second identifies the relative position according to that object, resulting in a precise location in space. The underlying geographical data such as the line or polygon is recorded by the referenced object and obtained by accessing the context,

leading to the capability of supporting both global and local coordinates. In addition, topological relations like "*contains*" and "*inside*" between the location and IFOBs can be directly derived by investigating the object identifier, as opposed to involving costly geometric computation.

Next, a data type called *genrange* is defined to represent sets of locations. Such a type is used on the one hand to represent the trajectory of a generic moving object, i.e., its projection into generic space. On the other hand, the type can represent any kind of curve or region in the generic space, for example, a road in the road network, a park in the region based outdoor space, a collection of rooms in a building, or an arbitrary region in free space.

Definition 3.1.8 *Generic Range*

Subrange = $\{(oid, l, m) | oid \in D_{\underline{int}}, l \in D_{\underline{points}} \cup D_{\underline{line}} \cup D_{\underline{region}} \cup \{\perp\}, m \in D_{\underline{tm}} \cup \{\perp\}\}$

$D_{\underline{genrange}} = 2^{Subrange}$

Each *genrange* value is a set of elements each of which is denoted by sub_traj_i ($\in Subrange$). sub_traj_i is composed of three attributes: oid denotes an IFOB id, l stores a set of locations and m describes a transportation mode.

When a *genrange* value is used to represent the trajectory of a moving object, then l contains the locations passed by the object (usually as a *line* value with coordinates relative to the given IFOB). Correspondingly, the transportation mode is defined. For example, the trajectory of a car moving on the road.

When the value is used to represent an arbitrary region in the generic space, l may be a *points* or *region* value or undefined (an undefined value means that all locations in the IFOB are valid). In this case the transportation mode may be undefined.

3.1.3 Moving Objects Representation

The thesis focuses on the complete past movement of moving objects instead of current and future positions. First, the abstract representation is given.

Definition 3.1.9 *Let f be the location function projecting from time to location.*

$$f : D_{\underline{instant}} \rightarrow D_{\underline{genloc}}$$

To represent the continuously changing location, a standard way to model the data is by the regression method that first segments the data into pieces or regular intervals within which it exhibits a well-defined trend, and then chooses the basis and mathematical functions most appropriate to fit the data in each piece [32, 35, 53, 90]. Using the method of *sliced representation*, a moving object is represented as a set of so-called *temporal units (slices)*. Each unit defines a time interval as well as the movement during the interval.

Definition 3.1.10 *Generic Temporal Units*

Loc = $\{(loc_1, loc_2) | loc_1, loc_2 \in D_{\underline{real}}\}$

Gentu = $\{(i, oid, i_{loc_1}, i_{loc_2}, m) | i \in D_{\underline{interval}}, oid \in D_{\underline{int}}, i_{loc_1}, i_{loc_2} \in Loc, m \in D_{\underline{tm}}\}$

Each element in Gentu contains five components: i defines a time interval, oid maps to an IFOB, i_{loc_1}, i_{loc_2} identify two positions according to the IFOB and m describes the transportation mode. A unit defines that (1) locations during i reference to an IFOB identified by oid; (2) i_{loc_1} (i_{loc_2}) precisely records the start (end) location at $i.s$ ($i.e$) according to the IFOB; (3) the transportation mode is m. Positions during $[i.s, i.e]$ are achieved by linear interpolation.

Given two units $u_1, u_2 \in$ Gentu, some relationships can be explored between them:

1. $u_1.m \neq u_2.m$

   This denotes two units with different transportation modes. For example, $u_1.m$ = *Car* and $u_2.m$ = *Walk*. Queries on transportation modes extract data from this attribute.

2. $u_1.m = u_2.m \wedge u_1.oid \neq u_2.oid$

   Transportation modes are the same, but the referenced IFOBs are different. Suppose that $u_1.m$ = $u_2.m$ = *Bus*, then $u_1.oid$ and $u_2.oid$ denote different buses.

3. $u_1.m = u_2.m \wedge u_1.oid = u_2.oid$

   In this case, both transportation modes and IFOBs are the same, but the precise locations in space can be different, distinguished by i_{loc_1} and i_{loc_2}. For example, two pedestrians walk on the same pavement or two clerks walk inside the same office room, resulting in different trajectories.

To make a compact representation for moving objects, the *mergeable* condition is defined for two units u_1 and u_2. Let $p_1(\in D_{\underline{point}})$ be the start location of u_1. In fact, p_1 is taken from the projection of u_1 in space. Again, we can have p_2 for u_2. Since a unit represents a linear movement, a function is employed for the evaluation at time t. We define T_1 (T_2) to be the relative time interval of u_1 (u_2), i.e., [0, $u_1.i.e$ - $u_1.i.s$], and give the evaluation functions.

$f_{u_1}(t) = \{(x, y) | x = p_1.x + a_1 \cdot t, y = p_1.y + b_1 \cdot t, t \in T_1, a_1, b_1 \in \mathbb{R}\}$

$f_{u_2}(t) = \{(x, y) | x = p_2.x + a_2 \cdot t, y = p_2.y + b_2 \cdot t, t \in T_2, a_2, b_2 \in \mathbb{R}\}$

Then we define u_1 and u_2 to be *mergeable* if and only if the following three conditions hold:

(i) $u_1.oid$ = $u_2.oid \wedge u_1.m$ = $u_2.m$;

(ii) $u_1.i$ is *adjacent* to $u_2.i$;

(iii) $f_{u_1}(T_1.e)$ = $f_{u_2}(T_2.s) \wedge a_1 = a_2 \wedge b_1 = b_2$.

A set of sub movements can be merged and compressed into one unit if they fulfill the *mergeable* condition. Suppose that a car is moving on the highway with constant speed for half an hour, a GPS device may record the position every five seconds. Instead of maintaining a large number of GPS tracks one unit can suffice.

A generic moving object is defined as a sequence of generic temporal units. The formal definition is given below.

Definition 3.1.11 *Generic Moving Objects*
$D_{\underline{genmo}}$ = $\{< u_1, u_2,...,u_n > | n \geq 0, n \in D_{\underline{int}}$, *and*
(i) $\forall i \in [1, n], u_i \in$ *Gentu*
(ii) $\forall i, j \in [1, n], i \neq j \Rightarrow u_i.i \cap u_j.i = \oslash \wedge u_i, u_j$ *are not mergeable* $\}$

3.1.4 Approximate Location

In some cases, a rough location can simplify the representation and at the same time still be available for the application (e.g., [59, 60]). For example, "*find all travelers passing areas A and B during their trips*". The data should be managed in such a way that one can determine whether the trajectory of a traveler intersects the area, while the precise movement inside can be ignored. There is a trade-off between precise data and performance since accurate data need much storage space and processing time.

The model in this paper supports both precise and imprecise representation so that the system can tune the level of scale to the appropriate value, making therefore the system much flexible. We define the location in an approximate way by $(oid, (\perp, \perp))$. As each IFOB covers some places, the range for the location can be known by accessing the object. Considering *Bobby's* movement M_1, one can roughly describe such a trip by recording a sequence of IFOB ids indicating: (1) roads; (2) polygons; (3) rooms.

The place covered by an IFOB may occupy a large area, e.g., the length of a highway can be hundreds of kilometers. Then, the rough representation with only an identifier may not provide enough location information for a user. To solve this problem, we can do the partition on IFOBs. The place of an IFOB is partitioned into a set of pieces and $i_{loc_1}.\delta_1$ ($i_{loc_2}.\delta_1$) stores the partition identifier. The methods are introduced in the following.

- I_{ptn}

  We believe it is enough to know which bus or train the traveler takes, while the relative location inside is ignored. Thus, no partition is needed.

- I_{indoor}

  The type $\underline{region}$ is used to represent the 2D area of a room. We partition the region into a set of cells each of which is denoted by a rectangle. The location is described by a room identifier and a cell id.

- I_{rbo}

  We employ the same method as I_{indoor} that a region is partitioned into a set of disjoint cells. As a result, the location is denoted by an IFOB id and a cell id.

- I_{rn}

A line consists of a sequence of segments and each partition corresponds to a segment. The location is identified by a line id and a segment id.

- I_{fs} No partition

3.2 Representation for Infrastructures

3.2.1 Public Transportation Network

Public transportation vehicles include buses, trains, and underground trains. Due to similar characteristics, without loss of generality we take the bus network as an example to present the infrastructure representation. A bus network contains static and dynamic components, where the first one includes bus stops and bus routes and the second one includes a set of bus trips.

3.2.1.1 Static

A bus stop identifies a location where a bus arrives, lets passengers get on and off, and then departs (if it is not the last stop). Each stop belongs to a route and several stops from different routes may map to the same location where a transfer can occur.

Bus Stop: Intuitively, a bus stop corresponds to a point in space, but it also has some other information, e.g., a route id. Instead of simply using a point, we define a data type named $\underline{busstop}$ with the carrier set:

Definition 3.2.1 *Bus Stop*

$D_{\underline{busstop}} = \{(rid, pos) | rid, pos \in D_{\underline{int}}, rid \geq 0 \wedge pos \geq 0\}$

To identify a bus stop, we use rid to denote the route id and pos to show the order on the route. Given bs_1, $bs_2 \in D_{\underline{busstop}}$, we define

bs_1 is *adjacent* to $bs_2 \Leftrightarrow bs_1.rid = bs_2.rid \wedge bs_1.pos + 1 = bs_2.pos$.

Bus Route: A bus route consists of a sequence of sub lines (partitioned by bus stops) each of which is called a *segment*. A segment represents the connection between two *adjacent* bus stops.

Definition 3.2.2 *Bus Segment*

Busseg = $\{(bs_1, bs_2, geo) | bs_1, bs_2 \in D_{\underline{busstop}}, bs_1, bs_2$ *are adjacent*, $geo \in D_{\underline{line}}\}$

Given two segments seg_1, $seg_2 \in$ Busseg, they are called *linkable* if and only if $seg_1.bs_2$ and $seg_2.bs_1$ map to the same point. Let **getbr_id**(seg) return the route id of a segment and $\underline{busroute}$ be the data type for bus routes with the domain:

Definition 3.2.3 *Bus Route*

$D_{\underline{busroute}}$ = {< seg_1, seg_2, ..., seg_n > | $n \geq 1$, $n \in D_{\underline{int}}$, *and*
(1) $\forall i \in [1, n]$, $seg_i \in$ *Busseg;*
(2) $\forall i \in [1, n$ - 1], ***getbr_id***(seg_i) = ***getbr_id***$(seg_{i+1}) \wedge seg_i$ *is linkable to* seg_{i+1}.}

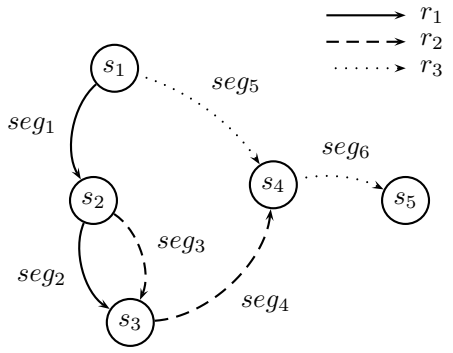


Figure 3.2: A Simple Bus Network

To demonstrate how the above data types work together, Figure 3.2 gives an example with three bus routes $\{r_1, r_2, r_3\}$ and five locations $\{s_1, s_2, s_3, s_4, s_5\}$. The location set is to identify the positions for bus stops. Figure 3.3 lists the representation of bus stops and bus routes. For brevity, the detailed geometry description is omitted. Each location from {s_1, s_2, s_3, s_4} shows an intersection between two bus routes, implying that a bus change can happen here.

Loc	Bus Stops
s_1	$bs_1 =< 1, 1 >$, $bs_7 =< 3, 1 >$
s_2	$bs_2 =< 1, 2 >$, $bs_4 =< 2, 1 >$
s_3	$bs_3 =< 1, 3 >$, $bs_5 =< 2, 2 >$
s_4	$bs_6 =< 2, 3 >$, $bs_8 =< 3, 2 >$
s_5	$bs_9 =< 3, 3 >$

(a)

Id	Bus Routes
1	$br_1 = \{seg_1(bs_1, bs_2), seg_2(bs_2, bs_3)\}$
2	$br_2 = \{seg_3(bs_4, bs_5), seg_4(bs_5, bs_6)\}$
3	$br_3 = \{seg_5(bs_7, bs_8), seg_6(bs_8, bs_9)\}$

(b)

Figure 3.3: Static Component

One issue that needs to be discussed is that in the real world a bus route has two directions (*Up* and *Down*) and for each direction there is a sequence of bus stops. *Up* and *Down* routes are normally located on two different lanes and the geometry description for them should be different. To be consistent with the reality, we define a bus route with *Up* and *Down* as two different routes, and each of them is uniquely identified and represented. In the model, this can be done by setting different *rids* to denote *Up* and *Down* routes.

3.2.1.2 Dynamic

A bus trip is determined by (1) a bus route and (2) the schedule, where the former defines the path and the latter specifies the time period of such a moving object. Each bus trip is modeled

as a moving point. Let $D_{\underline{busloc}}$ be the domain of bus locations.

Definition 3.2.4 *Bus Location*

$D_{\underline{busloc}}$ = $\{(br\_id, bs\_id, pos)\ |br\_id, bs\_id \in D_{\underline{int}}, pos \in D_{\underline{real}}\}$

A bus location is represented by a route id, a stop number on the route and the relative distance from the stop. Note that the definition is consistent with Def. 3.1.7 in Section 3.1.2.3, and here it is specified as the location on a bus route. Given $bloc_1, bloc_2 \in D_{\underline{busloc}}$, they are called *successive* if and only if the following conditions hold:

(1) $bloc_1.br\_id$ = $bloc_2.br\_id$;

(2) $bloc_1.bs\_id + 1$ = $bloc_2.bs\_id$;

(3) $bloc_1.pos$ = $bloc_2.pos$ = 0.

An operator called **geodata** is proposed to return the spatial point for a bus location or a bus stop.

geodata: $\underline{busroute} \times \underline{busloc} \rightarrow \underline{point}$

$\underline{busroute} \times \underline{busstop} \rightarrow \underline{point}$

Let p_bs_i ($\in D_{\underline{point}}$) be the point that $bloc_i$ ($\in D_{\underline{busloc}}$) corresponds to. Then, the equality of two bus locations is defined as: $bloc_1 = bloc_2 \Leftrightarrow p\_bs_1 = p\_bs_2$. Let BusU be the domain for bus trip units, defined in the following.

Definition 3.2.5 *Bus Trip Units*

BusU = $\{(i, bloc_1, bloc_2)|i \in D_{\underline{interval}}, bloc_1, bloc_2 \in D_{\underline{busloc}}$, *and*

(i) $i.s = i.e \Rightarrow bloc_1 = bloc_2$;

(ii) $bloc_1, bloc_2$ *are successive or equal*$\}$

The definition applies to the framework of generic temporal units in Def. 3.1.10, Section 3.1.3. Let $v \in$ Gentu, $u \in$ BusU, and we have

(1) $v.i \rightarrow u.i$;

(2) $v.oid \rightarrow u.bloc_1.br\_id$ $(u.bloc_2.br\_id)$;

(3) $v.i_{loc_1} \rightarrow (u.bloc_1.bs\_id, u.bloc_1.pos)$;

(4) $v.i_{loc_2} \rightarrow (u.bloc_2.bs\_id, u.bloc_2.pos)$;

(5) as u denotes a specific infrastructure unit, $v.m$ (*Bus*) is omitted in u.

Let $\underline{mpptn}$ be the data type representing *bus trips*. Based on Def. 3.2.5, we have:

Definition 3.2.6 *Bus Trips*

$D_{\underline{mpptn}} = \{< tub_1, tub_2, ..., tub_n > |n \geq 1, n \in int$, *and*

(i) $\forall i \in [1, n], tub_i \in$ *BusU;*

(ii) $\forall i, j \in [1, n], i \neq j \Rightarrow tub_i.i \cap tub_j.i = \emptyset$;

(iii) $\forall i, j \in [1, n], tub_i.bloc_1.br\_id = tub_j.bloc_1.br\_id.\}$

Condition (iii) ensures that each bus trip only belongs to one route. A bus moves along the route represented by a curve in space, and positions between two *successive* stops are determined by linear interpolation. This method yields a compact representation. Compared with the times of frequently updating raw location data (e.g., coordinates), the number of bus stops is very small. As a result, both update and storage costs are considerably reduced.

3.2.1.3 IFOBs in Bus Network

With the above data types, we specify the IFOB representation in I_{ptn}. First, IOSymbol is extended to include values for data types in I_{ptn}, IOSymbol = IOSymbol $\cup$ {BUSSTOP, BUSROUTE, MPPTN [2]}. Applying Def. 3.1.6 in Section 3.1.2.2, the result is:

- static: IFOB(oid, *BUSSTOP*, β, $name$)($\beta \in D_{\underline{busstop}}$)

  IFOB(oid, *BUSROUTE*, β, $name$)($\beta \in D_{\underline{busroute}}$)

- dynamic: IFOB(oid, *MPPTN*, β, $name$)($\beta \in D_{\underline{mpptn}}$)

Static objects are referenced by dynamic objects, while the movement of humans is represented by referencing to dynamic IFOBs. Here one has to be a bit careful. Although one may be tempted to assume that specifying the time interval and a reference to the bus are sufficient to define the movement of a bus traveler, a bus trip describes the *scheduled bus movement* from which the real bus movement may deviate (e.g., due to delays). For example, one would like to determine in a query at which bus stop a passenger enters the bus. Deriving this from the scheduled location of the bus at the time the passenger enters may lead to errors and inconsistencies with the remaining trip of the passenger. We therefore include in the unit describing a passenger's trip also the start and end locations on the respective bus route.

Let $u_{ptn}(i, oid, i_{loc_1}, i_{loc_2}, m)$ represent the movement of a bus traveler where:
(1) $oid \rightarrow$ IFOB (oid, *MPPTN*, β, $name$)($\beta \in D_{\underline{mpptn}}$);
(2) $i_{loc_1} \rightarrow (bs\_id, pos)$;
(3) $i_{loc_2} \rightarrow (bs\_id, pos)$;
(4) m = *Bus*.

The unit illustrates that during i the traveler goes by a bus identified by oid from i_{loc_1} to i_{loc_2}, where i_{loc_1} (i_{loc_2}) records the start (end) location, i.e., the bus stop. In detail, bs_id records the stop number and pos denotes the distance from the stop. This representation can significantly reduce the storage size for moving objects. Instead of recording the locations at all places where the bus speed or direction changes, one unit can suffice. If several passengers take the same bus, their locations all map to the same IFOB. Travelers may get on and off the same bus at different stops, which are distinguished by i_{loc_1} and i_{loc_2}. In addition, one does not have to update the data until the travelers get off the bus or switch to another one.

[2] MPPTN stands for moving point for public transportation network.

A bus trajectory is a set of elements sub_traj_i ($\in$ Subrange) with the attribute values:
(1) $sub\_traj_i.oid \rightarrow$ IFOB(oid, *BUSROUTE*, β, $name$)($\beta \in D_{\underline{busroute}}$);
(2) $sub\_traj_i.l$ stores the movement on the route;
(3) $sub\_traj_i.m$ = *Bus*.

3.2.2 Indoor

3.2.2.1 Modeling Indoor Space

In this environment, IFOBs represent objects such as rooms, chambers, corridors, etc. We use the term $groom$ (general room) for all of them. Each groom covers a 2D area and is located at some distance above the ground level.

Definition 3.2.7 *Region3d*
Region3d =$\{(poly, h)|poly \in D_{\underline{region}}, h \in D_{\underline{real}}\}$

The attribute $poly$ describes the 2D area and h denotes the room height above the ground level. Given $r_1, r_2 \in$ Region3d, we define $r_1 = r_2 \Leftrightarrow r_1.poly = r_2.poly \wedge r_1.h = r_2.h$. In a building, usually an office room or a corridor has only one flat surface. Nevertheless, there are also amphitheaters and chambers that have several surfaces with different altitudes. To be general, a groom is modeled as a set of objects. See below.

Definition 3.2.8 *General Room*
$D_{\underline{groom}}$ = {*GR* $\subseteq$ *Region3d* $|(gr_1, gr_2 \in GR \wedge gr_1 \neq gr_2) \Rightarrow$ *disjoint*$(gr_1.poly, gr_2.poly)\}$

Considering the staircase between two floors, such an IFOB is modeled as a set of Region3d objects each of which represents one footstep of the staircase. An elevator is represented by several grooms and each object has only one element recording the 2D area on one floor (a rectangle) accompanied with a height value. To clarify terms, when we speak of groom, it is a short description for general room, while $\underline{groom}$ denotes the data type representing a groom. The symbol set IOSymbol is extended to include the value for $\underline{groom}$, IOSymbol = IOSymbol $\cup$ {GROOM}.

Let $u_{indoor}(i, oid, i_{loc_1}, i_{loc_2}, m)$ be a temporal unit for indoor moving objects. Applying Def. 3.1.10 in Section 3.1.3, the attributes map to:
(1) $oid \rightarrow$ IFOB(oid, *GROOM*, β, $name$)($\beta \in D_{\underline{groom}}$);
(2) $i_{loc_1} \rightarrow (x, y)$;
(3) $i_{loc_2} \rightarrow (x, y)$;
(4) m = *Indoor*.

The coordinates (x, y) denote the relative position in the groom where the left lower point of the bounding box on the 2D area is set as the origin point. An indoor trajectory is specified as:

(1) $sub\_traj_i.oid \rightarrow$ IFOB(oid, *GROOM*, β, $name$)($\beta \in D_{\underline{groom}}$);
(2) $sub\_traj_i.l$ records the movement inside the IFOB;
(3) $sub\_traj_i.m$ = *Indoor*.

3.2.2.2 Indoor Navigation

In this part, we present an indoor graph for the precise shortest path searching inside a building, containing a set of grooms. Doors are used to build the connection between two grooms, and therefore we first give the representation for doors.

Definition 3.2.9 *Data Type for Doors*
Doorpos = {(*gr_id*, *pos*)| *gr_id* $\in D_{\underline{int}}$, *pos* $\in D_{\underline{line}}$}
$D_{\underline{door}}$ = {(dp_1, dp_2, $genus$, tp)| dp_1, dp_2 $\in$ *Doorpos*, $genus \in$ {*non-lift*, *lift*}, $tp \in D_{\underline{mbool}}$}

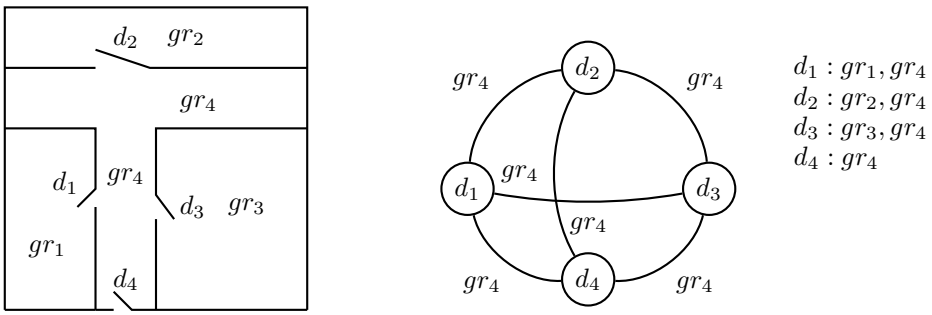
A door is shared by two grooms. We let dp_1 and dp_2 represent the door position in each groom, described by a groom id gr_id and the position pos inside. Besides the location data, a door has an attribute $genus$ describing the type for the purpose of distinguishing between non-lift doors (nld) and lift doors (ld). To model the time-dependent state of a door, i.e., open or closed, we define a moving bool tp. The value is a sequence of units where each unit has a time interval and a bool. The time-dependent state for nld can be known, while the state for ld is unknown. Usually, from 8am to 6pm an office room door is open, otherwise it is closed. But for an elevator, the floor it currently stays on is unknown. Hence the state for ld is uncertain and tp is set as undefined. We model the entrance/exit of an elevator on each floor as a door which builds the connection between different levels.

Definition 3.2.10 *An indoor graph is defined as* G_{indoor} (N, E, W, $\sum_{groom}$, $\sum_{door}$, l_v, l_e,) *where*
(1) N *is a set of nodes;*
(2) $E \subseteq V \times V$ *is a set of edges and each edge is associated with a weight value from W;*
(3) $l_v : N \rightarrow \sum_{door}$ *is a function assigning labels to nodes;*
(4) $l_e : E \rightarrow \sum_{groom}$ *is a function assigning labels to edges.*

In G_{indoor}, we let a node denote a door. An edge corresponds to a groom and builds the connection between two doors without passing through a third door. For example, an elevator is represented by several grooms each of which defines the place for the elevator on one floor. The elevator entrance/exit on each floor is modeled as a node and the connection between two *adjacent* floors shows an edge. One groom may have more than one door and each pair of doors indicates a connection, resulting in several edges in the graph. A shortest path between two doors is computed in the obstructed space for the reason that obstacles may exist inside a room. We store the path on the edge and set the weight as the path length. To create an indoor graph, all paths between doors inside one groom have to be pre-computed. Since a

room usually does not have too many doors, the cost is not high. And the computation is only done once. We let *indoorgraph* be the data type for an indoor graph.

Fig. 3.4(a) depicts an example with four rooms $\{gr_1, gr_2, gr_3, gr_4\}$ and four doors $\{d_1, d_2, d_3, d_4\}$. gr_1, gr_2, gr_3 are office rooms and gr_4 is a hallway. Each office room has a door, d_1 in gr_1, d_2 in gr_2 and d_3 in gr_3. d_4 is the entrance/exit for the hallway. G_{indoor} is shown in Fig. 3.4(b). For simplicity, we omit the shortest path on each edge.



(a) (b)

Figure 3.4: Floor Plan and Indoor Graph

The designed indoor graph supports searching the shortest path with different costs: (1) distance; (2) number of rooms; (3) time.

shortest distance: We apply Dijkstra's algorithm on G_{indoor} to find a route with the minimum distance. Let i_{loc} ($\in D_{genloc}$) be an indoor location which is represented by a groom id and a position inside the groom. A preprocessing step is required to find paths from i_{loc} to all doors in the groom where i_{loc} is located, resulting in the connection to G_{indoor}.

smallest number of rooms: This case is simple as the total cost is achieved by aggregating the number of edges in the path.

minimum traveling time: Let v_w be a person's average speed when walking inside a building. Since each edge stores the shortest path between two doors inside a groom, the time for passing an edge can be calculated by v_w and the path length. This method applies to all grooms except elevators. The time cost of moving by an elevator contains two parts: (1) waiting; (2) elevator moving. The cost of the first part is uncertain because on which floor the elevator is located cannot be determined at a query time instant. The second part can be easily calculated if the height between two floors and the elevator speed are given.

To solve the problem, we process as follows. Assume that the elevator speed is a constant value. Let d_i^{lift} and d_j^{lift} denote the lift door on the ith and the jth floor, respectively. The overall time to move from d_i^{lift} to d_j^{lift} depends on (i) the arrival time at d_i^{lift} and (ii) the path along which the lift moves to reach d_j^{lift}. Without loss of generality, we assume $0 \leq i < j$. In the best case, when a person arrives at d_i^{lift}, the lift happens to stay at floor i and directly moves up to floor j. On the contrary, the lift might just leave from floor i and moves up so that the

person has to wait until the lift goes down to the bottom floor and goes up again. Suppose that the distance between two floors is the same for the whole building, denoted by h, and there are n floors in total. Then, the length of the shortest and longest path between floor i and j is $(j-i)*h$ and $(j-i)*h+2*n*h$, respectively. The two values are the lower and upper bounds, and the set of all possible values is {$(j-i)*h$, $(j-i)*h+h$, ... ,$(j-i)*h+2*n*h$}. Each value can be assigned a membership probability depending on the elevator schedule. A simple solution is to impose the uniform distribution of the probability for each path length, while more realistic modeling should take into account the building structure, history data analysis, optimal elevator schedule, and so on. Consequently, the total time spent on the elevator can be computed.

We design an indoor graph to support an optimal route searching with respect to different costs all of which are meaningful in practice. In the following, we compare the indoor model with some others [57, 44, 102] in the literature. The precise indoor location and accurate shortest path are represented by our method. Existing techniques only locate the object inside a room, while the exact position inside is not handled. Previous indoor graphs define a room as a node, resulting in an approximate description for the indoor shortest path, i.e., a sequence of rooms. In some cases, this might not provide enough information for a traveler because some buildings may have large rooms with complex structures and some obstacles, e.g., the hall in a hotel or an airport.

We model the precise area of each room and define the paths to establish the connections between doors. This leads to the exact location of an indoor object and show a well-defined route for people to follow. Both spatial and temporal attributes for doors are modeled to have a practical and robust representation, as opposed to only illustrating the connection between rooms. As a result, the concept of doors in our model is general, not only for office rooms and corridors but also for staircases and elevators, leading to the capability of computing a complete indoor shortest path. This is not covered by previous work. Besides, existing techniques do not model the time cost of an indoor trip, and the time-dependent state of a door is not concerned.

3.2.3 Region-based Outdoor

This environment is for people walking outdoor, including places such as pavements, zebra crossings, etc. The overall area is represented by a relatively large polygon P with many holes inside. Holes denote obstacles such as building blocks and junction areas which people cannot directly pass through. To efficiently manage the data, P is decomposed into a set of polygons (e.g., triangles) to be stored in the database. Then, a location is represented by (1) a polygon id; (2) the relative position inside the polygon where the origin point is the left lower point of the polygon bounding box.

Let $u_{rbo}(i, oid, i_{loc_1}, i_{loc_2}, m)$ denote a unit for moving objects by walking and the value of

each attribute is specified as:

(1) $oid \rightarrow$ IFOB(oid, *REGION*, β, $name$)($\beta \in D_{\underline{region}}$);

(2) $i_{loc_1} \rightarrow (x, y)$;

(3) $i_{loc_2} \rightarrow (x, y)$;

(4) m = *Walk*.

Positions between i_{loc_1} and i_{loc_2} during i are obtained by linear interpolation. Fig. 3.5 shows an example in which the areas drawn by crosshatching represent the places that people cannot pass through. {pl_1,...,pl_7} denote the subareas from decomposing P. An example movement M_3 is depicted, passing areas {pl_1, pl_5, pl_7}. Locations of such a moving object can be represented in an approximate way (Section 3.1.4) where only the referenced object id is recorded, $M_3 = < (1), (5), (7) >$ (for simplicity, other values of a unit are ignored).

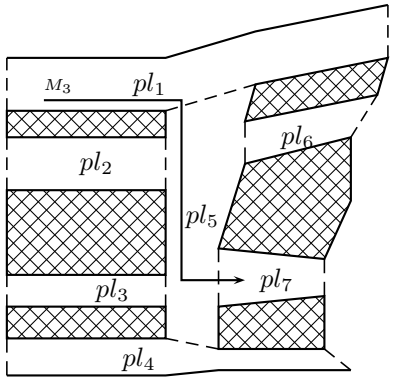


Figure 3.5: Outdoor Space Partition

3.2.4 Free Space and Road Network

Location representation in previous models [32, 38, 37] is seamlessly integrated into the proposed framework. Let $u_{fs}(i, oid, i_{loc_1}, i_{loc_2}, m)$ be a unit for moving objects in free space. Specifically,

(1) $oid \rightarrow \perp$;

(2) $i_{loc_1} \rightarrow (loc_1, loc_2)$;

(3) $i_{loc_2} \rightarrow (loc_1, loc_2)$;

(4) m = *Free*.

As there is no IFOB in I_{fs}, oid is undefined. (loc_1, loc_2) denotes the coordinates in space.

Let $u_{rn}(i, oid, i_{loc_1}, i_{loc_2}, m)$ be a unit for moving objects in a road network where

(1) $oid \rightarrow$ IFOB (oid, *LINE*, β, $name$)($\beta \in D_{\underline{line}}$);

(2) $i_{loc_1} \rightarrow (loc_1, \perp)$;

(3) $i_{loc_2} \rightarrow (loc_1, \perp)$;

(4) $m \in$ {*Car*, *Taxi*, *Bicycle*}.

A network position is represented by a road id and the position on the road. The second attribute in i_{loc_1} (i_{loc_2}) is set as undefined. In both environments, positions between i_{loc_1} and i_{loc_2} are computed by linear interpolation.

3.3 The Type System and An Interface

3.3.1 Data Types

The proposed data types, summarized in Table 3.2, include generic types for all environments and those arising from some specific infrastructures developed in the previous sections. We give the type system in Table 3.3 [3] where SPATIAL and GRAPH [38, 37] are still used in one infrastructure. Generic data types *genloc*, *genrange* and *genmo* are available in all cases. For example, *genloc* represents the location in any infrastructure and is specified as *point* in free space and *gpoint* in road network. The type *mpptn* that we define for moving buses is an instance of *genmo* in I_{ptn}. Similarly, moving objects representation in I_{fs} and I_{rn} is also an instance of *genmo*. To help clarify the difference between the new model and previous models, we give the type system of free space and road network in Appendix C.

	Name	**Meaning**
generic data types	*tm*	transportation modes
	genloc	generic locations
	genrange	generic sets of locations
	genmo	generic moving objects
PTN	*busstop*	bus stops
	busroute	bus routes
	busloc	locations on bus routes
	mpptn	moving buses
Indoor	*groom*	general rooms
	door	doors

Table 3.2: A Summary of Proposed Data Types

3.3.2 Space: A Relational View

We provide an interface to exchange information between values of proposed data types and a relational environment. First, all IFOBs are managed in the database system as they are referenced by moving objects. Table 3.4 shows the defined infrastructures each of which may

[3] basic types such as *int*, *bool* are omitted

	→ SPATIAL	*point*, *points*, *line*, *region*
	→ GRAPH	*gpoint*, *gline*
	→ PTN	*busstop*, *busroute*, *busloc*
	→ INDOOR	*groom*, *door*
	→ GENLOC	*genloc*
	→ SPACE	*genrange*, *space*
	→ TM	*tm*
GENLOC	→ TEMPORAL	*moving*, *intime*

Table 3.3: The Type System in General Model

have several components. For each component, we assign a symbol and create a relation where each tuple corresponds to an IFOB. All relation schemas are shown in Table 3.5.

Infrastructure	Infrastructure Component
I_{ptn}	BUSSTOP BUSROUTE BUS
I_{indoor}	ROOM DOOR ROOMPATH
I_{rbo}	OUTDOOR
I_{rn}	ROAD
I_{fs}	

Table 3.4: Infrastructures and Their Components

The data type to define an IFOB is embedded as an attribute in each relation. To have a unique identifier for each IFOB, we define a range of *int* values to denote IFOB ids for each infrastructure and different infrastructures are assigned disjoint values. Based on all infrastructure relations, the space is constructed in two steps:

1. create a space initialized by one relation:

   createspace: *rel* →*space*

2. add more infrastructures to the space:

   put_infra: *space* ×*rel* →*space*

Rel_Busstop	`(BusStopId: int, Stop: busstop, Name: string)`
Rel_Busroute	`(BusRouteId: int, Route: busroute, Name: string, Up: bool)`
Rel_Bus	`(BusId: int, BusTrip: mpptn, Name: string)`
Rel_Room	`(RoomId: int, Room: groom, Name: string)`
Rel_Door	`(DoorId: int, Door: door)`
Rel_Roompath	`(RoomPathId: int, Door1: int, Door2: int, Weight: real, Room: groom, Name: string, Path: line)`
Rel_Rbo	`(RegId: int, Reg: region, Name: string)`
Rel_Road	`(RoadId: int, Road: line, Name: string)`

Table 3.5: Infrastructure Relations

In the first step, the input relation can be empty, then we have the free space. If the relation stores roads, we have the space with road network. In the second step, more infrastructures can be added. One can create a full or non-full space depending on the application. For example, a non-full space might be road network plus bus network. Afterwards, the infrastructure data can be accessed by:

get_infra: $\underline{space} \times \underline{int} \rightarrow \underline{rel}$

The value of the second argument is from the set {BUSSTOP, BUSROUTE, BUS, ROOM, DOOR, ROOMPATH, OUTDOOR, ROAD}, i.e., the names of infrastructure components from Table 3.4, whose elements are assumed to be available as integer constants. Suppose that we have infrastructure relations for a city called Gendon. Applying the two steps (**createspace** and **put_infra**) a full space is created, denoted by SpaceGendon. The following examples illustrate queries on infrastructure data.

- **Q1**. "Show me the information of *Alexander* street."

```
SELECT *
FROM get_infra(SpaceGendon, ROAD) AS r
WHERE r.Name = "Alexander"
```

- **Q2**. "Where can I switch between bus No. 12 and No. 37?"

```
SELECT bs1, bs2
FROM get_infra(SpaceGendon, BUSSTOP) AS bs1,
     get_infra(SpaceGendon, BUSSTOP) AS bs2,
     get_infra(SpaceGendon, BUSROUTE) AS br1,
     get_infra(SpaceGendon, BUSROUTE) AS br2
```

```
WHERE br1.BusRouteId = 12 AND br2.BusRouteId = 37 AND
      bs1.Stop.rid = 12 AND bs2.Stop.rid = 37 AND
      geodata(br1,bs1) = geodata(br2,bs2)
```

To answer **Q2**, one needs to perform a join on two relations: bus routes and bus stops. **geodata** is defined in Section 3.2.1.2, returning the location of a bus stop.

Assume that we also have some trajectory data of citizens living and working in Gendon. A trip is represented by a tuple recording the trip id, the trajectory and a name. The relation is named MOGendon and has the schema:

```
MOGendon(Mo_id: int, Traj: genmo, Name: string)
```

3.3.3 Create A Graph for Indoor Navigation

Recall that in Section 3.2.2.2 we define nodes by doors and edges by paths between doors inside one groom for an indoor graph. Two relations are used to store graph nodes and edges, respectively. The relation schemas are included in Table 3.5 in which Rel_Door is for doors and Rel_Roompath is for paths. An indoor graph is created by the following operator.

createindoorgraph: $\underline{rel} \times \underline{rel} \rightarrow \underline{indoorgraph}$

Then, we can run shortest path queries using

indoornavigation: $\underline{genloc} \times \underline{genloc} \times \underline{instant} \times \underline{int} \times \underline{indoorgraph} \rightarrow \underline{genrange}$

where the first two arguments specify the start and end locations, the third indicates a query time and the fourth argument of type $\underline{int}$ denotes the cost type of such a shortest path (e.g., distance, time).

3.4 Operations

Before proposing operators and query examples, we first introduce some notations employed from [38, 37] to achieve a smooth integration with abstract data types and their operations. The command `LET <name> = <query>` creates a new database object `name` whose value is given by the query expression. A query can thus be formulated in several steps (separated by `;`), defining intermediate results by `LET`. The last expression determines the result of the query. We define expressions for a time instant and an interval, e.g., **instant**(2010, 12, 5, 8) (8am on Dec. 5, 2010), **interval**((2010, 12, 5, 8), (2010, 12, 5, 9)) (between 8am and 9am on Dec.5, 2010). To define operator semantics, some notations are needed. Considering Def. 3.1.6, a space can be denoted by

$Space$ = $\{(oid\ ,\ s,\ \beta,\ name) | oid \in D_{\underline{int}},\ s \in$ IOSymbol, $name \in string\}$.

By default, $Space$ is available for all operators, i.e., not needed as an explicit argument, and one element of $Space$ is denoted by w. We let u, v be single values of a data type and correspondingly U, V be generic sets of values of a type. u (U) refers to the first argument and v (V) refers to the second argument in an operator. mo denotes a moving object and u_i be one unit of mo. Semantics of some operators is given in this section while that of the others is put into Appendix A due to complex definitions.

3.4.1 Extended Operators

Starting from scratch here would not make sense. The design considers aspects such as systematic construction of the type system, definition of generic operations ranging over large collections of data types, and consistency between non-temporal and temporal (i.e. time dependent) operations. Operators in this part have the same semantics in previous work [38, 37] for free space and road network, listed in Table 3.6. The syntax is extended in order to support generic data types. See **Q3**.

Name	Signature
$=, \neq$	$\underline{tm} \times \underline{tm} \rightarrow \underline{bool}$
deftime	$\underline{genmo} \rightarrow \underline{periods}$
duration	$\underline{periods} \rightarrow \underline{real}$
present	$\underline{genmo} \times \underline{intime} \rightarrow \underline{bool}$
	$\underline{genmo} \times \underline{periods} \rightarrow \underline{bool}$
initial, final	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$
atinstant	$\underline{genmo} \times \underline{instant} \rightarrow \underline{intime}(\underline{genloc})$
atperiods	$\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$
val	$\underline{intime}(\underline{genloc}) \rightarrow \underline{genloc}$
inst	$\underline{intime}(\underline{genloc}) \rightarrow \underline{instant}$

Table 3.6: Operators by Extending the Syntax

- **Q3**. Where is *Bobby* at 8:00 am?

```
LET qt = instant(2010, 12, 5, 8);

SELECT val(mo.Traj atinstant qt)
FROM MOGendon AS mo
WHERE mo.Name = "Bobby"
```

The result is expressed by a value with $\underline{genloc}$, denoted by gl. One can investigate $gl.oid$ and $Space$ to know the infrastructure where gl is located. Assuming the infrastructure is Road Network, the referenced road can be retrieved.

```
SELECT *
FROM get_infra(Space, ROAD) AS r
WHERE r.RoadId = gl.oid
```

3.4.2 Spatial and Spatial-Temporal

The spatial operators are collected in Table 3.7. The meaning of **distance** should be clear if the two arguments belong to the same infrastructure, e.g., Euclidean distance in I_{fs}, network distance in I_{rn}. If the two input locations belong to different infrastructures, the value is defined to be the minimum length of a trip connecting them.

Table 3.8 lists operators on spatial-temporal data types. **Trajectory** projects a moving object into the space. Given a location, **at** restricts the trip to the specified place. One can also restrict the movement to a set of places by giving a *genrange* argument. A formal definition of the semantics in this case is a bit lengthy and omitted.

The operator **trip** takes two locations and a query instant as input. The locations are general (i.e., can be situated in any infrastructure), and the result is described in the form of *genmo*. Consider the query "*find a trip from my office room to my home with minimum traveling time*". The resulting trip described by a sequence of transportation modes might be *Indoor* $\rightarrow$ *Car* $\rightarrow$ *Walk* or *Indoor* $\rightarrow$ *Walk* $\rightarrow$ *Bus* $\rightarrow$ *Walk*.

Name	Signature	Semantics
$=, \neq$	*genloc* $\times$ *genloc* $\rightarrow$ *bool*	Def. 1 in Appendix A
inside	*genloc* $\times$ *genrange* $\rightarrow$ *bool*	Def. 2 in Appendix A
intersects	*genloc* $\times$ *genrange* $\rightarrow$ *bool*	the same as Def. 2 in Appendix A
	genrange $\times$ *genrange* $\rightarrow$ *bool*	Def. 3 in Appendix A
distance	*genloc* $\times$ *genloc* $\rightarrow$ *real*	Def. 4 in Appendix A

Table 3.7: Spatial Operators

Name	Signature	Semantics
trajectory	*genmo* $\rightarrow$ *genrange*	Def. 5 in Appendix A
at	*genmo* $\times$ *genloc* $\rightarrow$ *genmo*	Def. 6 in Appendix A
	genmo $\times$ *genrange* $\rightarrow$ *genmo*	omitted
passes	*genmo* $\times$ *genloc* $\rightarrow$ *bool*	**at**$(mo, v) \neq \emptyset$
	genmo $\times$ *genrange* $\rightarrow$ *bool*	**at**$(mo, v) \neq \emptyset$
trip	*genloc* $\times$ *genloc* $\times$ *instant* $\rightarrow$ *genmo*	omitted

Table 3.8: Spatial-Temporal Operators

We give query examples for these operators below.

- **Q4**. Between 8am and 9am, who sits in the same bus as *Bobby*?

```
LET qt = interval((2010, 12, 5, 8), (2010, 12, 5, 9));

SELECT mo1.Name
FROM MOGendon AS mo1,
     MOGendon AS mo2
WHERE mo2.Name = "Bobby" AND
      val(mo1.Traj atperiods qt) = val(mo2.Traj atperiods qt)
```

- **Q5**. Did *Bobby* visit the same place during such two periods: [8am, 10am] and [3pm, 5pm]?

```
LET qt1 = interval((2010, 12, 5, 8), (2010, 12, 5, 10));
LET qt2 = interval((2010, 12, 5, 15), (2010, 12, 5, 17));

SELECT intersects(trajectory(mo.Traj atperiods qt1),
                  trajectory(mo.Traj atperiods qt2))
FROM MOGendon AS mo
WHERE  mo.Name = "Bobby"
```

- **Q6**. Find all people passing zone A and zone B as well as a location p in space. (An interesting query could be: find all people passing the Christmas Market and the city center as well as the cinema.)

  Assume the names for the zones are “Zone-A" and “Zone-B". Let $p = (\perp, (x, y))(\in D_{\underline{genloc}})$ denote the location.

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, OUTDOOR) AS R1,
     get_infra(SpaceGendon, OUTDOOR) AS R2
WHERE R1.Name = "Zone-A" AND R2.Name = "Zone-B" AND
      mo.Traj passes R1.Reg AND
      mo.Traj passes R2.Reg AND
      mo.Traj passes p
```

3.4.3 Sets and Decomposition

The design of types and operations in the abstract model for moving objects [38] emphasized compatibility with a relational model, and therefore proposed only “atomic” data types, i.e.,

types suitable as attribute types in a relation. Presumably for this reason, there is no $\underline{set}$ type constructor applicable to atomic types so that for example, a type $\underline{set}(\underline{periods})$ would be available.

However, it is recognized that a tool for decomposing values of a data type into components is needed. For example, for a $\underline{region}$ value consisting of several disjoint components ("faces") it should be possible to obtain each face as an independent $\underline{region}$ value; similarly for a $\underline{moving}(\underline{point})$ one would like to have a decomposition into continuous pieces, each as a separate $\underline{mpoint}$ value. It is obvious that a natural operation to perform such decompositions would have a signature

components: $\alpha \rightarrow \underline{set}(\alpha)$

for any atomic type α consisting of several components. Therefore, we leave this restriction behind and do introduce a $\underline{set}$ constructor. In the implementation, it has turned out that one does not even need an explicit data structure to represent such sets, but can handle sets of values as streams of values at the executable level of the system. Of course, it is also possible to introduce an explicit data structure for sets. Together with the set constructor we define a few generic operations:

contains: $\underline{set}(\alpha) \times \alpha \rightarrow \underline{bool}$
card: $\underline{set}(\alpha) \rightarrow \underline{int}$

Furthermore, the operation **components** is offered for decomposition:

For $\alpha \in \{\underline{range}(\beta), \underline{points}, \underline{line}, \underline{region}, \underline{moving}(\gamma)\}$:
components: $\alpha \rightarrow \underline{set}(\alpha)$

Note that type $\underline{periods}$ is just an abbreviation for $\underline{range}(\underline{instant})$, hence the signature

components: $\underline{periods} \rightarrow \underline{set}(\underline{periods})$

is also available.

3.4.4 Transportation Modes and IFOBs

Table 3.9 lists the proposed operators. Given a moving object, we can get its transportation modes. Using *Bobby*'s trajectory M_1 as an example, **get_mode** returns {*Car*, *Walk*, *Indoor*}. With **get_mode** and **contains** (Section 3.4.3), one can examine whether a moving object includes a specific transportation mode. See **Q7**.

- **Q7**. Find all people using public transportation vehicles.

Name	Signature	Semantics
get_mode	$\underline{genmo} \rightarrow \underline{set(tm)}$	$\{u_i.m \mid u_i \in mo\}$
at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	Def. 3.4.2
ref_id	$\underline{ioref} \rightarrow \underline{int}$	$u.oid$
	$\underline{busroute} \rightarrow \underline{int}$	$u.bs_1.rid$, $u \in U$
	$\underline{mpptn} \rightarrow \underline{int}$	$u_i.bloc_1.br\_id$, $u_i \in mo$
get_ref	$\underline{genloc} \rightarrow \underline{ioref}$	Def. 3.4.3
	$\underline{genrange} \rightarrow \underline{set(ioref)}$	Def. 3.4.4
	$\underline{genmo} \rightarrow \underline{set(ioref)}$	Def. 3.4.5

Table 3.9: Operators on Transportation Modes and Infrastructure Objects

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE get_mode(mo.Traj) contains Bus OR
      get_mode(mo.Traj) contains Train OR
      get_mode(mo.Traj) contains Metro
```

Given a transportation mode, a sub trip can be extracted with respect to the mode, done by **at** (in Section 3.4.2, **at** restricts a trip to a given space and now the operator is extended).

- **Q8**. How long does *Bobby* walk during his trip?

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

We define a data type named $\underline{ioref}$ to have a light representation of referenced IFOBs, whose value may need a large storage space, e.g., a region.

Definition 3.4.1 *Reference Data Type*
$D_{\underline{ioref}} = \{(oid, ref) \mid oid \in D_{\underline{int}}, ref \in$ *IOSymbol*[4] $\}$

Operator **ref_id** returns the referenced object id and **get_ref** gets the IFOB in a light representation (See **Q9**). When the underlying data is needed, one can get the full representation by accessing the $Space$.

- **Q9**. Find all people taking Bus No. 527.

[4]IOSymbol={LINE, REGION, FREESPACE, BUSSTOP, BUSROUTE, GROOM}

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUS) AS bus
WHERE ref_id(bus.BusTrip) = 527 AND
      get_ref(mo.Traj at Bus) contains  bus.BusId
```

Definition 3.4.2 $\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$ ***at***
The result is $< u_1, u_2,...,u_n >$ *where* $u_i.m = v$.

Definition 3.4.3 $\underline{genloc} \rightarrow \underline{ioref}$ ***get_ref***
The result is $(u.oid, w.s)$ *where* $\exists\, w \in Space$*:* $u.oid = w.oid$.

Definition 3.4.4 $\underline{genrange} \rightarrow \underline{set}(\underline{ioref})$ ***get_ref***
The result is $\{(u.oid, w.s) \mid u \in U \wedge (\exists\, w \in Space : u.oid = w.oid)\}$.

Definition 3.4.5 $\underline{genmo} \rightarrow \underline{set}(\underline{ioref})$ ***get_ref***
The result is $\{(u_i.oid, w.s) \mid u_i \in mo \wedge (\exists\, w \in Space : u_i.oid = w.oid)\}$.

3.4.5 Subtype Relationships and Conversions

Infrastructure	Generic Types		
	genloc	*genrange*	*genmo*
I_{ptn}	*busstop*, *busloc*	*busroute*	*mpptn*
I_{indoor}		*groom*, *door*	
I_{rbo}			
I_{rn}	*gpoint*	*gline*	*mgpoint*
I_{fs}	*point*	*points*, *line*, *region*	*mpoint*

Table 3.10: Subtype Relationships

Sometimes one needs to apply generic operations (defined for *genloc*, *genrange* or *genmo*) to objects of specific types. This is possible through subtype relationships. In Section 3.2, data types for different infrastructures have been shown to fit into the generic framework; hence arguments of the specific types can be substituted in operations for the generic types. The valid subtype relationships are shown in Table 3.10. We give an example **Q10** that is between *genloc* and *busstop*.

- **Q10**. How long does *Bobby* wait at the bus stop “Uni"?

```
SELECT duration(deftime((mo.Traj at Walk) at bs.Stop))
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUSSTOP) AS bs
WHERE mo.Name = "Bobby" AND bs.Name = "Uni"
```

We give some comments for the query. The movement is first restricted to the mode *Walk*, and then limited to a bus stop. We suppose that people walk to a bus stop instead of by car or some other modes. Here, a generic location is specified as the bus stop location. **deftime** gets the time period at the place and **duration** returns the time span.

Furthermore, sometimes it is necessary to compare locations or moving objects that belong to different infrastructures. To be able to evaluate such relationships, we might try to provide mappings between all pairs of infrastructures. However, it is simple to introduce a generic mapping that converts an object from any infrastructure into the free space counterpart. We introduce such an operation called **freespace**. Essentially it maps values of any of the types in Table 3.10 to the corresponding type in free space, hence offers the signatures shown in Table 3.11. For the mapping of indoor locations into free space we ignore the height value associated with a $\underline{groom}$ and project into the (x, y) plane (or the ground level of the building). This maps rooms from different floors into the same locations. Nevertheless, the mapping is still able to relate locations between different infrastructures. To help understand the operators, see the examples below.

Operator	Signature
freespace	$\underline{genloc} \rightarrow \underline{point}$
	$\underline{busstop} \rightarrow \underline{point}$
	$\underline{busloc} \rightarrow \underline{point}$
	$\underline{gpoint} \rightarrow \underline{point}$
	$\underline{busroute} \rightarrow \underline{line}$
	$\underline{groom} \rightarrow \underline{region}$
	$\underline{door} \rightarrow \underline{line}$
	$\underline{gline} \rightarrow \underline{line}$
	$\underline{genmo} \rightarrow \underline{mpoint}$
	$\underline{mpptn} \rightarrow \underline{mpoint}$
	$\underline{mgpoint} \rightarrow \underline{mpoint}$
freespace_p	$\underline{genrange} \rightarrow \underline{points}$
freespace_l	$\underline{genrange} \rightarrow \underline{line}$
freespace_r	$\underline{genrange} \rightarrow \underline{region}$
genloc	$\underline{int} \times \underline{real} \times \underline{real} \rightarrow \underline{genloc}$

Table 3.11: Conversion Operators

- **Q11**. Is the university under a thunderstorm area?

  Let R_storm ($\in D_{\underline{region}}$) be the thunderstorm area.

```
SELECT R_storm contains freespace(room.Room)
FROM get_infra(SpaceGendon, ROOM) AS room
WHERE room.Name contains "Uni"
```

We assume that the name of all university rooms has ”Uni“ as the prefix and project the area of a room to the free space.

- **Q12**. Find all buses passing the city center area.

```
SELECT bus.Name
FROM get_infra(SpaceGendon, BUS) AS bus,
     get_infra(SpaceGendon, OUTDOOR) AS outdoor
WHERE outdoor.Name = "CityCenter" AND
      freespace(bus.Bus) passes outdoor.Reg
```

To compare the bus movement (of type $\underline{mpptn}$) with the city center region (which is a $\underline{region}$ value in free space), we map the bus trip into the free space using operation **freespace**, i.e., convert the movement to an $\underline{mpoint}$.

- **Q13**. Did bus No. 527 pass any traveler going by bicycle?

  We define *pass* to mean that there exists a time instant where the distance between the two moving objects is less than 3 meters.

```
SELECT mo.Name
FROM get_infra(SpaceGendon, MPPTN) AS bus,
     MOGendon AS mo
WHERE ref_id(bus.BusTrip) = 527 AND
      sometimes(distance(freespace(bus.Vehicle),
                         freespace(mo.Traj at Bicycle)) < 3.0)
```

Here we need to determine the distance between two moving objects, namely, the bus and a traveler going by bicycle. We assume the bus passes the bicycle if at some time their distance is small enough. To determine the time-dependent distance, both moving objects have to be located within the same infrastructure. Therefore, we map both of them into free space, i.e., convert to $\underline{mpoint}$. The application of the **distance** operator returns a moving real. Comparing

this with the constant 3.0 returns an *mbool* value. Finally, **sometimes** yields true if the *mbool* ever assumes the value true.[5]

For the mapping of *genrange* values into free space, we need to introduce three operators **freespace_p**, **freespace_l**, and **freespace_r** returning *points*, *line*, or *region* values, respectively. This is necessary because *genrange* is a union type. The operators return the parts of the argument that can be mapped into the respective result type. For example, one can obtain the trajectory of a trip as a *line* value.

Finally, for querying, a construction operator **genloc** for generic locations is needed to build a generic location from an IFOB identifier and two real "coordinates".

3.5 Query Examples

In this section, we perform more queries to evaluate the data model. Some notations are used. The notation `SET(<name>, <value>)` constructs a relation with a single tuple and attribute from an atomic value. For example, `SET(Name, "Bobby")` constructs the relation r containing a tuple. Seeing the operator **components** (Section 3.4.3) with the signature *periods* $\rightarrow$ *set*(*periods*), the notation `SET(Time, components(p))` will produce a relation with schema `(Time: periods)` having one tuple for each distinct time interval in the *periods* value p.

Appendix B provides useful information to check all query formulations in detail. The schemas of the involved infrastructure relations are listed and for each query an index shows where the used operations have been defined.

- **Q14**. "Which streets does bus No. 12 pass by."

```
SELECT r
FROM get_infra(SpaceGendon, ROAD) AS r,
     get_infra(SpaceGendon, BUSROUTE) AS br
WHERE br.BusRouteId = 12 AND intersects(br.Route, r.Road)
```

- **Q15**. Find out who passed the room No. 312 in the university between 8am and 9am.

```
LET qt = interval((2010, 12, 5, 8), (2010, 12, 5, 9));
```

Assume the name of the room is "Uni-312".

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, ROOM) AS room
```

[5] **sometimes** is a derived operation, **sometimes**(mb) = **not**(**isempty**(**deftime**(mb **at** true))). See [39], Exercise 4.5.

```
WHERE room.Name = "Uni-312" AND
      get_ref((mo.Traj atperiods qt) at Indoor)
      contains room.room_id
```

To define this query, several operators are needed. First, we restrict the moving object to the given period by **atperiods** and to indoor movement by **at**. Second, we get the referenced IFOBs by **get_ref**. The result is represented by the reference type, i.e., as $\underline{set}(\underline{ioref})$. Third, we check whether the room id is included by **contains**.

- **Q16**. Where does *Bobby* walk during his trip?

```
SELECT trajectory(mo.Traj at Walk)
FROM MOGendon AS mo
WHERE mo.Name = "Bobby"
```

- **Q17**. Find all people staying at office room No. 312 at the university for more than 2 hours.

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, ROOM) AS room
WHERE room.Name = "Uni-312" AND EXISTS
  SELECT *
  FROM SET(Piece,
    components(deftime(mo.Traj at room.Room)))
  WHERE duration(Piece) > 120
```

The trajectory of mo is restricted to the time period when he/she was at room No. 312 which due to the subtype relationship can be used as a $\underline{genrange}$ value. Obviously, the query refers to a single stay at this office rather than to the aggregated time over many visits. Hence the definition time interval is decomposed into disjoint intervals using **components**. This is transformed into a relation from which we select tuples for which the duration of the stay is more than two hours. If the relation is not empty for a given mo, then this mo qualifies for the result.

- **Q18**. Who entered bus No. 527 at bus stop "University"?

```
SELECT mo.Name
FROM MOGendon AS mo,
     get_infra(SpaceGendon, BUS) AS bus,
     get_infra(SpaceGendon, BUSSTOP) AS busstop
```

```
WHERE ref_id(bus.BusTrip) = 527 AND
      busstop.Name = "University" AND
      bus.Stop = val(initial(mo.Traj at
                             genloc(bus.BusId, undef, undef)))
```

- **Q19**. Did anyone who was at the University on floor H-2 between 4:30pm and 5pm take a bus to the main (train) station?

  To be on floor H-2 means to be in any of the rooms of floor H-2. We assume a table is available associating rooms with the floors they belong to:

  ```
  Uni_Rel(Floor: string, RoomName: string)
  ```

  Then the query can be formulated as follows:

  ```
  LET qt1 = interval((2010, 12, 5, 16, 30), (2010, 12, 5, 17));
  LET qt2 = interval((2010, 12, 5, 16, 30), (2010, 12, 5, 19));

  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.RoomName = r.Name AND
        (mo.Traj atperiods qt1) passes r.Room AND
        bs1.Name = "University" AND bs2.Name = "Main station" AND
        val(initial((mo.Traj atperiods qt2) at Bus))
        = bs1.Stop AND
        val(final((mo.Traj atperiods qt2) at Bus))
        = bs2.Stop
  ```

The query implicitly requires that the person takes the bus to the main station after he was on floor H-2. Hence we define a second time period from 4:30pm to 7pm during which the bus must be taken. We then ask for trajectories of people passing through the generic locations of rooms on floor H-2 within period qt1, as well as having a bus trip during qt2. The bus trip starts at the University bus stop and ends at the main station bus stop.

- **Q20**. Who arrived by taxi at the university in December?

  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. We assume such a part of the road network has been stored in the database as an object UniDriveway of type *gline*.

```
LET December = interval((2010, 12, 1), (2010, 12, 31));

SELECT mo.Name
FROM MOGendon AS mo
WHERE EXISTS
  SELECT *
  FROM SET(Trip,
    components((mo.Traj atperiods December) at Taxi))
  WHERE val(final(Trip)) inside UniDriveway
```

We reduce the trajectory of a person to the taxi trip in December. There may be several such trips (present in the resulting trajectory), hence we apply **components** to get the continuous pieces, i.e., the individual taxi trip. In the end, we check whether the relation containing these trips includes one with the final destination within the university driveway area.

3.6 Conclusions

In this chapter, we introduce a data model with the aim of representing and managing moving objects in a database system, where the following real world environments are considered: public transportation network, indoor, region-based outdoor, road network and free space. We let the space for moving objects be covered by a set of infrastructures each of which corresponds to an environment and is composed of a set of objects representing available places for moving objects. A generic location representation is proposed and can apply in all cases. The location of moving objects maps to these infrastructure objects. New data types are proposed to define moving objects and infrastructure objects. To efficiently access the data and formulate queries, a set of operators is defined in the system. We give the definitions for the signature and the semantics of operators. A relational interface is provided to exchange the data between values of proposed data types and a relational environment. We formulate a group of interesting queries regarding transportation modes and environments for moving objects.

Chapter 4

MWGen: A Mini World Generator

4.1 Generator Architecture

In this section, we outline the procedure of creating all infrastructures and present the data management in GMOD. The infrastructures include I_{rn}, I_{rbo}, I_{ptn} and I_{indoor}. Since I_{fs} contains no IFOBs, it does not have to be created. For I_{ptn}, we have two components: Bus Network and Metro Network. To distinguish between them, we define a notation for each component, I_{ptn} = $I_{bn} \cup I_{mn}$.

4.1.1 Data Generator Workflow

Fig. 4.1 shows an overview of MWGen, where the procedure consists of three phases: (1) infrastructures generation; (2) space construction; (3) trip planning. We elaborate each phase below.

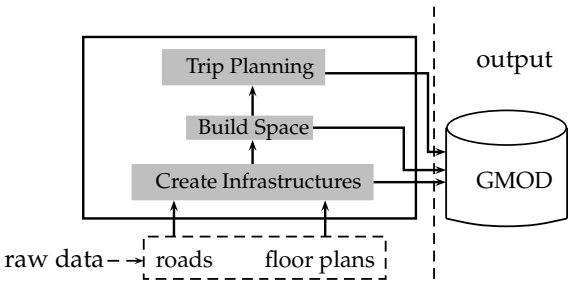


Figure 4.1: MWGen Workflow

- MWGen takes roads as input, builds I_{rn} and generates the other outdoor infrastructures in the order $I_{rbo} \rightarrow I_{bn} \rightarrow I_{mn}$. Roads are stored in a file as a relation with the schema

Road_Rel (Id: int, Type: int, Geodata: line).

Each tuple represents a road where the value *Type* indicates whether the road is a main or side street. I_{rbo} represents the walking area for people, consisting of a set of polygons. Bus routes, stops and moving buses compose the bus network. Similarly, I_{mn} comprises railways, stops and metros. The indoor environment including a set of buildings is created based on public floor plans. All created infrastructure objects as well as roads are managed by GMOD.

- Afterwards, a global space is built on top of all infrastructures. Basically, the space handles a reference representation for each infrastructure, loads infrastructure objects if they are needed for query processing and regulates the interaction places between different infrastructures.

- Trip planning is performed on the space where the start and end locations can be in any infrastructure. We let them belong to different infrastructures in order to search a path through several environments. The resulting path is used to produce a moving object with multiple transportation modes. All created moving objects are stored in GMOD.

We give more comments about the space, which is essentially important for GMOD. It has two functionalities:

- **location mapping**. Consider the two queries, "*which streets does bus route No. 12 pass by*", and "*what is the road network distance between two bus stops*". To compare IFOBs from different environments, one needs to project them into the same system. This is done by the space.

- **a bridge**. The space serves as an interface and builds the connection between the underlying IFOBs and moving objects. During the query processing, concrete data are loaded from the low level infrastructure, e.g., roads, bus routes, rooms. A reference for each infrastructure is maintained by the space with the aim of having a light representation. The location of a moving object records an identifier, while the semantic meaning of such an integer is explained by the space (e.g., a road or room id).

4.1.2 Data Management

4.1.2.1 Infrastructure Storage

We define a set of symbols to denote the type of an infrastructure.

Definition 4.1.1 *Infrastructure Symbols*
I_Symbols={ROAD, PAVEMENT, BUS, METRO, INDOOR}

In general, an infrastructure can be described by a four-tuple
$I_i(type, OS, IS, G)$ $(type \in I\_Symbols)$
where OS is a set storing all IFOBs in the environment, IS is an index set to efficiently access these objects, e.g., B-trees, and G denotes a graph for trip planning. OS is realized as a set of relations where a tuple represents an IFOB and the data type of the object is embedded as an attribute. In accordance with characteristics, a set of data types is defined to represent IFOBs from different environments. For instance, a line is used to describe the geometrical property of a road, and polygons are used to identify pavement areas for people walking. All data types are supported by GMOD. Each IFOB has a unique id and the id ranges for different infrastructures are disjoint. We explain the concrete context of OS, IS and G for each infrastructure in the following.

- I_{rn}: A collection of roads is stored in a relation (OS). The index set IS contains a B-tree with the key road id and an R-tree built on the geographical property of the road. A road graph is defined for shortest path searching. A node in G represents the endpoint of a road segment, recording a route identifier and the relative position on the route. An edge is created if two network locations (1) map to the same point in space but with different route ids (i.e., a junction) or (2) are *adjacent* on the same route.

- I_{rbo}: A set of polygons is kept in OS to cover the area for people walking. We build a B-tree with the key polygon id and an R-tree on polygon bounding boxes. The graph in I_{rbo} is used to find the shortest path in the walking area.

- I_{bn} and I_{mn}: IFOBs are routes, stops and moving vehicles, where each group is stored in a relation and indexed by a B-tree on the route id. For routes and stops, the geographical data is also indexed by an R-tree. A bus route is a line in space and a stop position is identified by a point. Both I_{bn} and I_{mn} have a graph for searching an optimal route from one stop to another with respect to the traveling time.

- I_{indoor}: OS contains a root relation $Root\_rel$ where each tuple corresponds to a building storing the 2D outdoor area, a building id, the type (e.g., office building, university) and a pointer. Two indices are created on $Root\_rel$, a B-tree on the building id and an r-tree on the bounding box of outdoor area. The pointer of each tuple in $Root\_rel$ maps to a structure that maintains (1) a *sub* relation Sub_rel storing all rooms and doors of a building; (2) a B-tree with the key room id; (3) a graph for indoor navigation. As a consequence, IS is a set of indices and G consists of a set of indoor graphs. Each type of building has its own floor plan, yielding distinct indoor structures. Several tuples in $Root\_rel$ can point to one Sub_rel if these buildings have the same floor plan, implying the identical internal structure.

Each infrastructure maintains an R-tree in order to efficiently process queries on geographical objects from different infrastructures. Consider the following examples: (1) A pedestrian

wants to find the nearest bus stop; (2) Given a bus route, one can search all roads it passes; (3) Choose a building and locate all bus stops within 300 meters.

4.1.2.2 Space Storage

The space is generated on top of all infrastructures, basically consisting of two parts (Ref, LS).

Definition 4.1.2 *Space Reference Representation*
$Ref = \{(ref, id\_l, id\_h) | \ ref \in I\_Symbols, id\_l \leq id\_h, id\_l, id\_h \in D_{\underline{int}}\}$

Ref is a list of elements, each of which has a label referencing to an infrastructure and the min, max object ids of that infrastructure. An element is inserted into Ref whenever an infrastructure is created. id_l and id_h define the object id range for an infrastructure. Thus, given a generic location , the environment that it belongs to can be known by investigating the reference id and Ref.

The system should be able to provide a trip passing through several environments. In order to achieve this, we have to take into consideration some places where people can leave one environment and enter another. LS is a location set for managing interaction places between different environments where the transportation mode can change. See below:

(1) bus (metro) stops and pavements. Each stop maps to a position on the pavement so that it is available to search the nearest stop for a pedestrian. Reversely, the traveler terminates the movement at a bus or metro stop and wants to find a path located in the pavement area to the target place (e.g., a shop).

(2) pavement locations and road network locations. Given a location in I_{rbo}, it can project to a network location, and vice versa. This is to build the bridge between I_{rn} and I_{rbo}, leading to the capability of mode switching between *Car* and *Walk*. Different from the location mapping above, there is an infinite set of locations for I_{rn} and I_{rbo}, so this conversion is done on-the-fly.

(3) building entrances and pavement locations. In order to build the connection between indoor and outdoor, we map a building entrance/exit to a pavement location.

4.1.3 Dataset Examples for Moving Objects

The dataset for moving objects is a set of trips where each passes several environments and includes multiple transportation modes. We let each trip include both outdoor and indoor movement and give examples below.

(1) *Walk* $\rightarrow$ *Car* $\rightarrow$ *Walk* $\rightarrow$ *Indoor* [1]

This might be the most common movement for people in their daily life, seeing below.

- *Bobby walks from home to the parking lot, drives the car along the road. After parking the car he walks to the office building, and finally arrives at his office room.*

[1] we use Car as the example for presentation, so modes $Taxi$ and $Bike$ are omitted

(2) *Indoor* → *Walk* → *Bus* → *Walk*
(3) *Indoor* → *Walk* → *Metro* → *Walk*

Instead of traveling by car, people may also choose the public transportation system. Suppose that the traveler takes a bus, and consider such a trip.

- *Bobby moves from his office room to the building exit, leaves the building and walks to a bus stop. Then, he takes a bus and travels by the bus to a stop close to his apartment, in the end walks from the bus stop to his home.*

The mode *Walk* is contained in each trip. According to the study in [103], walk segment is an important part which builds the connection between movement segments with different transportation modes. Usually people do not directly switch from *Car* to *Bus* or from *Indoor* to *Metro*. The transition relationship among modes is shown in Fig. 4.2.

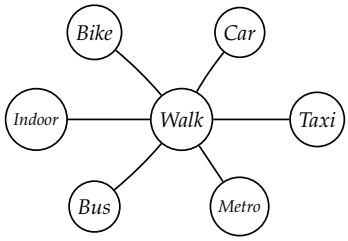


Figure 4.2: Transportation Modes Transition

4.2 Create Components for Space

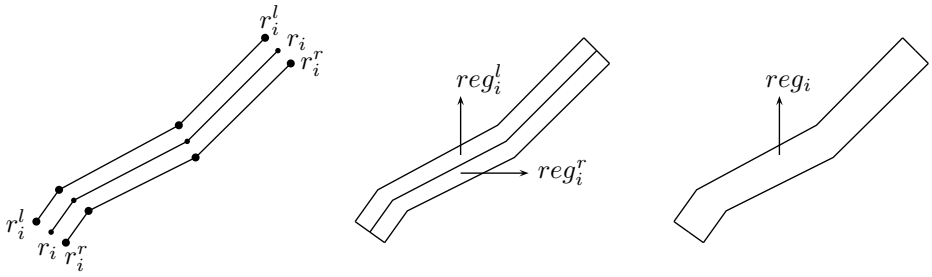
4.2.1 Region-based Outdoor

The covered places include pavements located on both or one side of roads, and zebra crossings allowing people to cross the street. Both are represented by polygons and created based on roads and a defined road width w ($\in D_{\underline{real}}$).

Pavements besides roads: We use r_i to denote a road represented by a line and extend r_i by w (e.g., five meters) to a region r whose shape is a stripe. The procedure is as follows:

1. r_i is translated to two new polylines by the distance w where one is on the left of r_i and the other is on the right of r_i, and we denote the left part by r_i^l and the right part by r_i^r.

2. We connect the two endpoints of r_i and r_i^l, and also the endpoints of r_i and r_i^r. Thus, two regions are formed, denoted by reg_i^l and reg_i^r where reg_i^l is created by r_i and r_i^l, and reg_i^r is created by r_i and r_i^r.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 4.3: Create A Region for A Road

3. A third region is produced by performing the union of reg_i^l and reg_i^r, denoted by reg_i.

Figure 4.3 depicts the three steps. Afterwards, we define another width value $w + \Delta w$ where Δw is the width of a pavement, e.g., two meters. Repeating the same three steps above, a larger region than reg_i can be created, denoted by reg_e_i. The pavements located on both sides of r_i can be obtained by reg_e_i minus reg_i. For each road, we follow the same procedure and get a set of polygons for pavements.

Zebra Crossings: These areas are used by people to cross the street and are usually located near the intersection position of two roads. So, at each junction intersected by r_i and r_j we create zebra crossings for each road. The zebra crossings are Δd (e.g., 3) meters from the junction. They cross r_i, r_j and intersect the pavements located on both sides of the road.

Finally, we perform the union on all pavements and zebra crossings to get a relatively large polygon P with many holes inside to denote the whole area for people walking. For efficient data management and query processing, P is decomposed into a set of triangles [74, 62, 41] to be stored in GMOD. The advantage is (1) the precise data of P is still maintained by these triangles no matter whether P is convex or concave, with or without holes; (2) we can build graphs on triangles to efficiently support query processing (see Section 4.3.1).

4.2.2 Bus Network

A public bus transportation system consists of two components: *static* and *dynamic*. The *static* part is comprised of bus routes and bus stops, and the *dynamic* part includes time schedules and bus trips.

4.2.2.1 Static

Bus routes. Each bus route has a start and an end location, selected as follows:

(1) The road network space is considered as a grid consisting of cells represented by rectangles. Each cell is assigned a value recording the road density, which is the number of roads intersecting the cell. Then, all cells are classified into two groups according to the road density. Cells belonging to the high density group are the areas close to the city center or with high

residential density. Cells in the low density group are the areas far away from the city center or suburbs.

(2) Two cells c_i, c_j are selected fulfilling the condition that both belong to the low density group, or one is from the high density group and the other is from the low density group. Since cells from the high density group are close to each other, the distance between c_i and c_j is too short to create a bus route. We pick up two road network points as route start and end locations where one is located inside c_i and the other is inside c_j.

(3) We compute the shortest path in the road network connecting the two locations and set the path as the bus route.

We repeat the same procedure above to create a set of bus routes. To optimize the result, we do refinement on the created bus routes for the purpose of merging two bus routes if the length of their common parts is larger than a threshold value.

Bus stops. For each route, we create a sequence of bus stops. The distance between two *adjacent* stops is selected from a pre-defined distance set. We merge several bus stops belonging to different bus routes if (i) their routes have common parts and (ii) the distance between each pair of them is smaller than a threshold value, e.g., 50m.

In the real world, one bus route has two directions (*Up* and *Down*) and for each direction there are a sequence of bus stops. To be realistic, our bus routes and bus stops are also created based on this behavior.

4.2.2.2 Dynamic

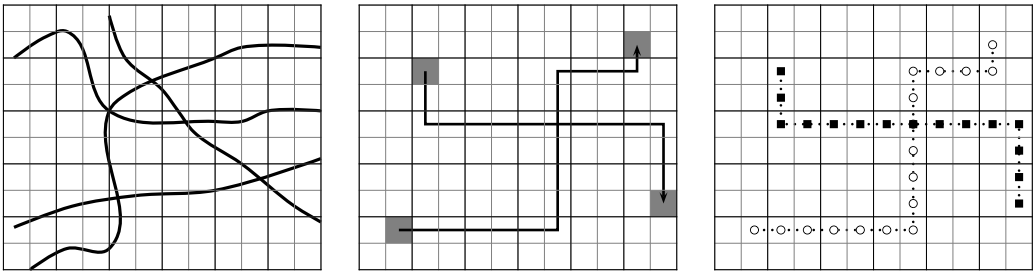
Bus trips are created based on bus routes and the speed value. A bus route defines the path on which the bus should move. The speed of a bus is determined by the road it is moving on. Recall that *Road_Rel* (Section 4.1.1) has a value for the street type, e.g., main street, side street, and this value determines the maximum speed for cars. We set the bus speed as the maximum car speed. A bus moves along the route from one stop to another, waits for passengers getting on and off, and then moves forward. Each bus route has a schedule defining the departure time for each bus trip. Different bus routes may have different schedules. We set the schedule for each route by its traffic flow, which is obtained as follows.

A set of cars moving on the roads is created. Then, for each road we record its traffic flow by aggregating the number of cars passing it during a time interval. We map a bus route to the corresponding roads and summarize the traffic flow of them to be the value for the bus route. All bus routes have daytime schedules, and the top k (e.g., 20) routes on high traffic value are also used as night bus routes. In the daytime, the schedule is set as every 15 minutes for the top k routes and every 30 minutes for the others. For each night route, we let the bus trip start every hour.

4.2.3 Metro Network

A metro network is created based on the road network and the procedure contains three steps:

- The road network space is partitioned into a set of equal size cells each of which is represented by a square. The square width is set by a value that decides the distance between two *adjacent* metro stops (e.g., 1.5km). A dual graph is built on these cells where each node corresponds to a cell and an edge is created if two cells are *adjacent*.
- A metro route has start and end locations which are center points of randomly selected cells. Note that the distance between a pair of selected cells should be larger than a threshold value (e.g., 10 km) so that the path is long enough for a metro route. A shortest path (minimum number of cells passing by) connecting the start and end locations can be found by searching the dual graph. The path is represented by a sequence of cells. We sequentially pick up the center point of each cell on the shortest path, and connect them in the same order as the cell appearing on the path to build a metro route. These points are set as the positions for metro stops. To help understand the procedure, we show an example in Fig. 4.4.
- A metro trip is created based on the pre-defined path and a given speed value (70km/h). We set the schedule for each route as every ten minutes during the time interval [6am, 10pm].



(a) Partition Road Network (b) Select Cells and Compute Paths (c) Set Routes and Stops

Figure 4.4: Create Metro Routes

4.2.4 Indoor Space

In a city, there is a large number of buildings with different types, such as personal houses, office buildings, hotels, etc. Due to the privacy problem and the difficulty of getting the data, the internal structure for personal houses is not available. Table 4.1 lists all created public buildings based on their floor plans. A floor plan is a diagram to show in scale the relationships between rooms, spaces and other physical features at one level of a structure. After finishing

one level, we translate the floor in vertical to get more floors to construct a building if the structure is the same at each level. Otherwise, levels with different structures are created separately and then built together to form a building. The university floor plan is from the author's affiliation. A 3D viewer is developed in the system to visualize the indoor data such as rooms and paths, demonstrated in Fig. 4.5.

Buildings	Resource	Buildings	Resource
officeA	[11]	hotel	[8]
officeB	[11]	hospital	[10]
shopping mall	[9]	university	
cinema	[6]	railway station	[12]

Table 4.1: Public Floor Plans

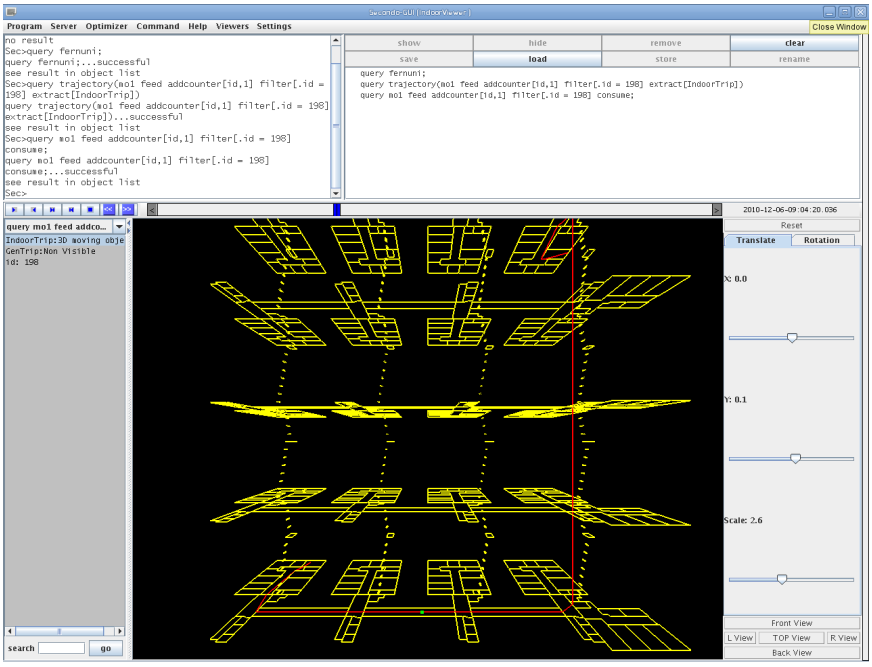


Figure 4.5: Indoor Visualization

We get the outdoor place for all buildings by taking out the area covered by roads and pavements from the overall space. A rectangle is used to represent the bounding box of the area for a building. Each rectangle is set a type value according to the building characteristic (e.g., hotel or cinema). We set the type for all rectangles as follows. Most rectangles in the city center denote office buildings, shopping malls and cinemas. Rectangles out of the city center are for personal houses, universities and surrounded by hospitals. Hotels are uniformly distributed and the railway station is located at a place with high road density.

A building is denoted by a five-tuple

$B(id, type, bbox, R, G)$ $(id, type \in D_{\underline{int}})$

where id is an identifier, $type$ describes the characteristic and $bbox$ denotes the bounding box of outdoor area. All rooms of a building are stored in a set R. An indoor graph G is also created. A building consists of a set of grooms such as office rooms, corridors, staircases, elevators, etc. A groom can have one or several doors, represented by lines. The raw data (from floor plan) for creating a building is a relation where each tuple describes the information of a groom. The relation schema is defined as follows.

RelRaw_Indoor(Oid: int, Name: string, Type: string, Room: groom, Door: line)

Each groom has an id and a name. The attribute *Type* records the type of the groom. We classify all grooms into five categories:

(1) *OR*: office rooms, chambers, conference rooms;

(2) *CO*: corridors, hallways;

(3) *BR*: bath rooms;

(4) *ST*: staircases;

(5) *EL*: elevators.

This attribute is used to select two locations for shortest path searching where the start and end locations can be in any groom, even the staircase. The type value for a groom is to guarantee that the locations are at reasonable places. In the real world, finding a path from one office room to another is common and meaningful. To arrive at the target place, people may pass through corridors, use staircases or elevators, but it seldom happens that people move from one bath room to another (of course the path can be computed).

4.2.5 Space Construction

Let $s(Ref, LS)$ denote the space to be created, initially empty. Whenever an infrastructure is generated, an element

(ref, id_l, id_h) $(ref \in I\_Symbols, id\_l, id\_h \in D_{\underline{int}})$

is inserted into $s.Ref$ where ref represents the infrastructure label and id_l (id_h) denotes the minimum (maximum) infrastructure object id. So, s is progressively filled up by infrastructures and only maintains a reference pointing to the underlying data. s also records the id range of each infrastructure for the sake of efficiently locating the environment for a moving object.

After all infrastructures are generated, one needs to handle the interaction places between different environments in order to provide a trip passing through several environments. A set of locations $s.LS$ as well as the location mapping technology is defined in the space structure. The overlapping places between the following pairs of environments are managed: (1) I_{bn} (I_{mn}) and I_{rbo}; (2) I_{rn} and I_{rbo}; (3) I_{indoor} and I_{rbo}. In fact, the considered places are between region-based outdoor and the other infrastructures. This is consistent with the data in Section 4.1.3 where the mode *Walk* is included by all moving objects.

4.3 Trip Plannings

4.3.1 Shortest Path for Pedestrians

Trip planning for pedestrians is to find the shortest path between two locations inside P, created in Section 4.2.1. P is a large polygon with many holes inside. In computational geometry, there are some main-memory algorithms on path problems in the presence of obstacles [83, 49]. Now we have to process the data in big size, as P is the whole area for outdoor walking in a city. P can contain hundreds of thousands of vertices, yielding a main-memory algorithm not practical. Besides, the technique in computational geometry has a limitation that assumes the number of holes is small. This is not the case for such an application. Each obstacle in P represents a building block or junction area, and there is a large number of such objects inside a city. Thus, an efficient and practical method is essentially needed.

We outline the solution as follows. First, two graphs DG (*Dual Graph*) and VG (*Visibility Graph*) have to be built. Instead of managing an extremely large polygon in the database, we decompose P into a set of triangles Tri, each of which is assigned a unique id. A dual graph DG is built on Tri where each node corresponds to a triangle and an edge is created if two triangles are *adjacent*. VG (*Visibility Graph*) is the most common and popular approach to computing the shortest path in an obstacle space, also adopted by our method. Each node in VG corresponds to a vertex of P where the whole set of vertices includes points from the contour and holes. If two nodes are mutually *visible* (i.e., the line connecting them does not cross any obstacles and is completely inside P), an edge is created.

A location inside P is represented by $(tri\_id, (loc_1, loc_2))$ where tri_id denotes a triangle identifier and (loc_1, loc_2) records the relative location inside the triangle by setting the left lower point of the triangle bounding box as the origin point. The procedure of searching the shortest path in P contains three steps:

(1) Given two arbitrary locations q_1 and q_2, the triangles in which q_1 and q_2 are located can be found in constant time. We let all triangles be increasingly ordered by tri_id, yielding the ability to fast access them by the id entry.

(2) All vertices of P that are *visible* to q_1 and q_2 can be found by searching DG for the purpose of building a bridge between q_1, q_2 and VG. This is because q_1, q_2 can be located anywhere in P, and if they do not belong to the vertices of P, q_1, q_2 are not represented by VG nodes.

(3) After connecting q_1, q_2 to VG, Dijkstra's algorithm can be run on VG to find the path. A^* algorithm can also be used where the heuristic value is just the Euclidean distance between a polygon vertex and the destination.

The algorithm is given in Algorithm 1. In lines 10 - 13, a simple method is to examine each vertex n_i from Q and compare n_i with the set E. The goal is to find the visible vertex to e so that e can be connected to VG. Evidently, the efficiency deteriorates over time if each vertex from Q is checked for the whole set E. To optimize the procedure, a bounding box on the point

Algorithm 1: RBOSP(s, e, VG, DG)

Input: s, e - start and end locations;
VG, DG - dual graph and visibility graph
Output: p - a shortest path from s to e

```
1  let S ← VisiblePoints(s, DG) ;
2  let E ← VisiblePoints(e, DG) ;
3  let Q be a priority queue, initially empty;
4  let T_se be the shortest path tree with the root node s;
5  foreach s_i ∈ S do
6      enqueue(Q, s_i);
7      insert s_i into T_se;
8  while head(Q) ≠ e do
9      n_i ← dequeue(Q);
10     if Distance(n_i, BBox(E)) = 0 then
11         if ∃e_i ∈ E such that e_i is equal to n_i then
12             enqueue(Q, e);
13             insert e into T_se;
14     foreach n_{i+1} ∈ Adj(n_i, VG) do
15         enqueue(Q, n_{i+1});
16         insert n_{i+1} into T_se;
17 select p from T_se;
18 return p;
```

set E is built. Before checking the existence of n_i in E, we first compute the distance between n_i and E. If such a value is not equal to zero, n_i cannot belong to E and is safely pruned. We only compare n_i and E when the distance is zero, implying that n_i might be equal to one vertex of E. In the following, we present creating the dual graph on triangles in Section 4.3.1.1 and introduce the sub procedure of searching visible points in Section 4.3.1.2.

4.3.1.1 Dual Graph on Triangles

P contains a set of contours $\{C_0, C_1, ..., C_n\}$ including the outer boundary and holes. We let C_0 be the outer contour and C_i ($i \in [1, n]$) be a inner contour (hole). Each C_i consists of a sequence of vertices which are sorted in counterclockwise order. Then, a vertex of P can be denoted by

v_i = (c_id, id, loc) (c_id, $id \in D_{\underline{int}} \wedge c\_id \geq 0 \wedge id \geq 0$, $loc \in D_{\underline{point}}$)

where c_id denotes the contour number and id identifies the vertex order in the contour. If c_id equals to zero, the vertex belongs to the outer contour. Otherwise, the vertex belongs to

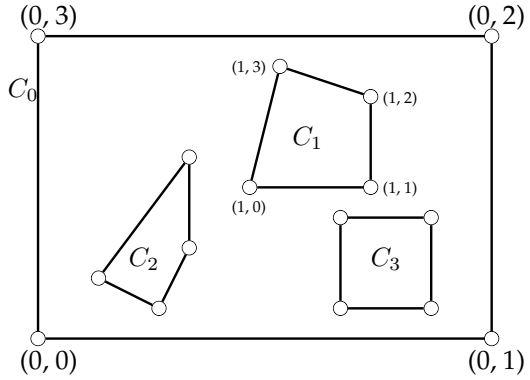


Figure 4.6: A Polygon with Holes

an inner contour. Fig. 4.6 depicts a polygon with three holes inside and the representation of its vertices.

Suppose that we get the result of performing polygon triangulation [74, 62, 41] on P, a set of triangles Tri. Each triangle is denoted by $tri(v_1, v_2, v_3) \in Tri$ (using the representation above for v_i ($i \in \{1, 2, 3\}$)). DG is built on these triangles where a node represents a triangle and an edge is created for two *adjacent* triangles. One can perform the spatial join algorithm to get the result in two steps: (1) find all pairs of triangles fulfilling the condition of bounding boxes intersecting and (2) filter the cases that (i) self intersecting (ii) without sharing any triangle edge. The proposed vertex representation can benefit to find a pair of triangles that have a common edge. The equal condition of two vertices can be examined by comparing the values of c_id and id.

4.3.1.2 Search Visible Points

Given a query location loc_q inside P, let tri_q denote the located triangle. After calculating the bounding box of tri_q, the operator

geodata: $\underline{genloc} \times \underline{region} \rightarrow \underline{point}$

returns the global point for loc_q. For simplicity, we ignore the second parameter in the following, that is **geodata**(loc_q). Let (v_i, v_j) denote an edge of tri_q. We have three relationships between loc_q and tri_q: (1) *inside*; (2) *meet*; and (3) *on_border*, defined below.

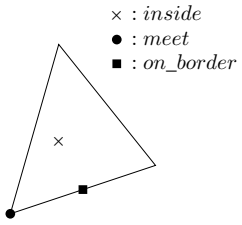


Figure 4.7: Relationships between loc_q and tri_q

Definition 4.3.1 *inside*

loc_q *is inside* $tri_q \Leftrightarrow \neg\exists\ (v_i, v_j)$: $i \neq j \wedge (v_i, v_j)$ *Contains* ***geodata***(loc_q).

Definition 4.3.2 *meet*

loc_q *meets* $tri_q \Leftrightarrow \exists v_i \in tri_q$: $v_i.loc$ = ***geodata***(loc_q).

Definition 4.3.3 *on_border*

loc_q *is on_border of* $tri_q \Leftrightarrow$
$\exists v_i, v_j$: (v_i, v_j) *Contains* ***geodata***$(loc\_q) \wedge v_i.loc \neq$ ***geodata***$(loc\_q) \wedge v_j.loc \neq$ ***geodata***(loc_q).

We give the visible point searching algorithm in Algorithm 2. First, the triangle for loc_q is found. Then, the algorithm checks the relationship between loc_q and tri_q and calls the corresponding subroutine. Note that for a query location and the located triangle, only one of the three conditions can hold at the same time.

Algorithm 2: VisiblePoints

Input: loc_q - a query point;
DG - a dual graph on Tri
Output: VP - the set of visible points to loc_q

```
1 let tri_q be the triangle where loc_q is located;
2 if loc_q is inside tri_q then
3     VP = Inside(loc_q, DG);
4 if loc_q meets tri_q then
5     VP = Meet(loc_q, DG);
6 if loc_q is on_border of tri_q then
7     VP = OnBorder(loc_q, DG);
8 return VP;
```

Before elaborating each sub algorithm, we introduce a structure called *Clamp* to be used for searching visible points. A clamp consists of three points with one being called *apex* and the other two being called *feet*, represented by $clamp(a, f_1, f_2)$ (a, f_1, $f_2 \in D_{\underline{point}}$). Two connections are defined among the three points: $\overrightarrow{af_1}$ and $\overrightarrow{af_2}$, see Fig. 4.8(a). Each connection starts from the *apex* and passes through one of the *feet*. A clamp partitions the space into two parts, $clamp.A$ and $\overline{clamp.A}$, as shown in Fig. 4.8(b). We define that $clamp.A$ does not include lines $\overrightarrow{af_1}$ and $\overrightarrow{af_2}$. One can calculate the angle of a clamp, that is $\angle f_1af_2$ or $\angle f_2af_1$. We let $clamp.\alpha$ denote the angle and define the value to be between $(0, 180)$. Given a clamp and a point q, let β_1=$\angle qaf_1$ and β_2=$\angle qaf_2$ ($\beta_1, \beta_2 \in (0, 180)$) be two angles.

Lemma 1 *clamp covers q*

A clamp covers q if and only if $clamp.\alpha = \beta_1 + \beta_2$. *This implies that the line* $\overrightarrow{aq}$ *is located inside* $clamp.A$.

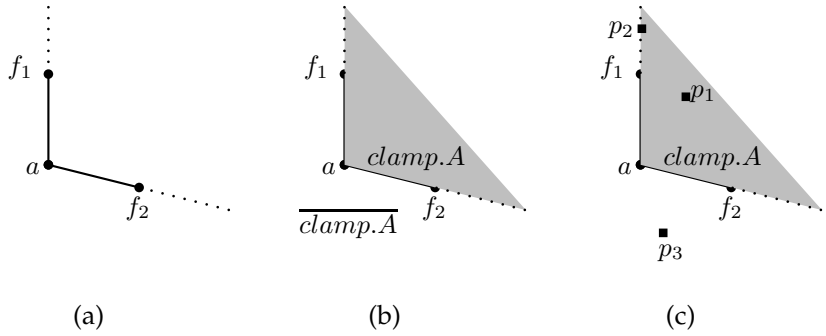


Figure 4.8: Clamp Structure

We prove that if $\overrightarrow{aq}$ inside $\overline{clamp.A}$, the *cover* condition cannot hold.

Proof *Proof by contradiction.*

If $\overrightarrow{aq}$ is inside $\overline{clamp.A}$, we have β_2 + β_1 + $clamp.\alpha$ = 360. Let β_2 = 360 - (β_1 + $clamp.\alpha$). As $clamp.\alpha$ = β_1 + β_2 (cover condition), then $clamp.\alpha$ = β_1 + 360 - (β_1 + $clamp.\alpha$) by replacing β_2. It contradicts. □

Fig. 4.8(c) depicts an example with three points $\{p_1, p_2, p_3\}$. According to the *cover* condition, the clamp covers p_1 but does not cover p_2 and p_3. The case for p_3 is straightforward. For p_2, as we define $\beta_1, \beta_2 \in (0, 180)$, the *cover* condition does not hold if the three points $\{a, f_1, p_2\}$ are co-linear.

In the following, we introduce how to use the clamp structure to find visible points for loc_q. For the sake of clear presentation, we use the notation $C_i.v_j$ to denote a vertex v_i (Section 4.3.1.1) of P where i is the value of c_id and j is the value of id. We consider the three relationships: inside, meet and on_border.

case1: loc_q is inside tri_q

If the *inside* condition holds, evidently, all vertices of tri_q are *visible* to loc_q, as shown in Fig. 4.9. We collect the three points and need to find more *visible* points inside P if exist. We create three clamps by setting loc_q as the *apex* for each case, seeing below:

$clamp_1(loc\_q, C_2.v_2, C_2.v_3)$

$clamp_2(loc\_q, C_2.v_2, C_3.v_0)$

$clamp_3(loc\_q, C_2.v_3, C_3.v_0)$.

The three clamps partition tri_q into three sub triangles. For each sub triangle, we apply the *depth-first* method to search its *adjacent* triangles on DG to find visible points to loc_q. Each sub triangle determines a searching space, and we only tackle the points that are *covered* by the corresponding clamp. Without loss of generality, we take $clamp_3$ as an example to describe the procedure. By searching DG, we find the *adjacent* triangle to $tri(loc\_q, C_2.v_3, C_3.v_0)$, that is $tri(C_1.v_0, C_2.v_3, C_3.v_0)$. Since the two triangles share two vertices, only $C_1.v_0$ needs to be checked. In the following, we use p to denote the point to be checked. The following lemma is used to determine whether p is visible to loc_q.

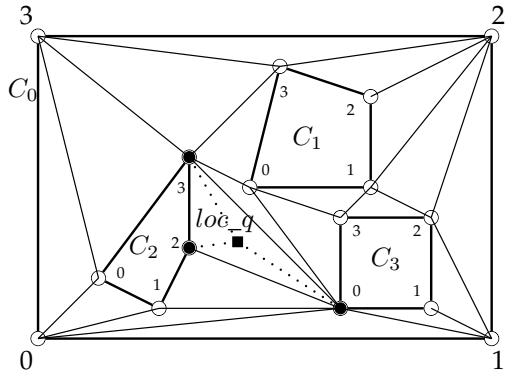


Figure 4.9: loc_q inside tri_q

Lemma 2 *p is visible to loc_q*

Let $clamp(loc\_q,\ f_1,\ f_2)$ be the current clamp structure with loc_q being the apex and $Poly_q$ be a polygon with four vertices $\{loc\_q, f_1, f_2, p\}$. We say p is visible to $loc\_q \Leftrightarrow Poly_p$ is a strictly convex polygon [2]*.*

Note that if loc_q, $f_1(f_2)$ and p are co-linear, p is not visible to loc_q as p is blocked by $f_1(f_2)$. Given a polygon with n vertices, the time complexity for checking convexity is $O(n)$. Since $Poly_p$ always has four vertices, the time complexity is $O(1)$. In Fig. 4.9, $C_1.v_0$ is visible to loc_q and is collected. When p (the checked point) is visible, we split the clamp into two subclamps. For each subclamp, the apex remains the same (loc_q), one foot is the same as before and the other foot is set as p. $clamp_3$ is split into $clamp_3^1(loc\_q,\ C_1.v_0,\ C_2.v_3)$ and $clamp_3^2(loc\_q,\ C_1.v_0,\ C_3.v_0)$, shown in Fig. 4.10(a). For each subclamp, we repeat the same procedure of visiting adjacent triangles to find visible points. Whenever a clamp is split, the angle of a subclamp becomes smaller to decrease the searching space in the sub procedure.

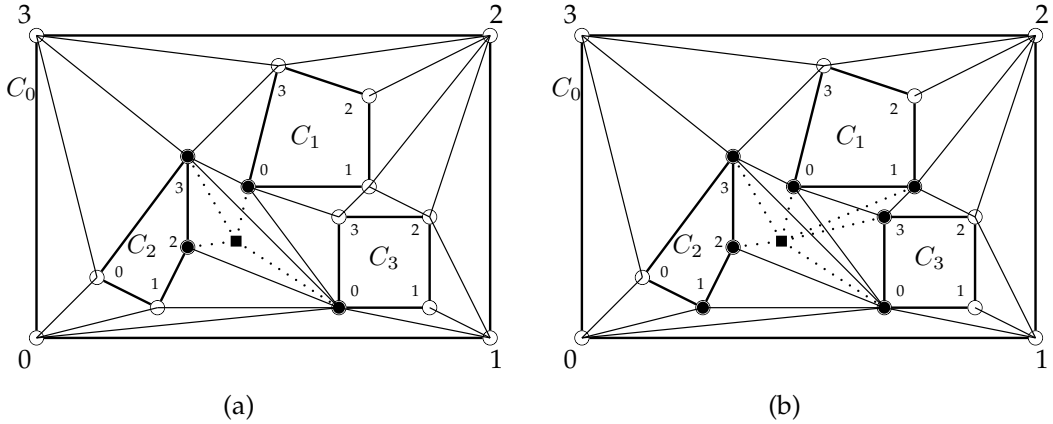


Figure 4.10: Split the *Clamp*

Now we consider the case that p is not visible. Considering $clamp_3^1(loc\_q,\ C_1.v_0,\ C_2.v_3)$, by searching DG we find the triangle $tri(C_1.v_3,\ C_2.v_3,\ C_1.v_0)$ and $C_1.v_3$ is not visible to loc_q. The

[2] every internal angle is less than 180 degrees

algorithm checks the edges (p, f_1) and (p, f_2) to determine whether the procedure of searching *adjacent* triangles should forward or stop. At current state, p (the point to be checked) is $C_1.v_3$, f_1 is $C_2.v_3$ ($C_1.v_0$) and f_2 is $C_1.v_0$ ($C_2.v_3$). The searching stops on the edge if one of the following conditions holds.

- If the edge belongs to the *outer* or *inner* contour of P, no *adjacent* triangles can be found.
- If there exists an unvisited *adjacent* triangle to $tri(p, f_1, f_2)$ with the shared edge $(p, f_{1(2)})$, the searching stops when the **internal** angle $\angle loc\_qf_{1(2)}p$ in $Poly_q$ is equal or larger than 180. The reason is if $\angle loc\_qf_{1(2)}p \geq 180$, the vertex belonging to the *adjacent* triangle cannot be *covered* by the current clamp (see the *cover* condition in Lemma 1).

Seeing Fig. 4.10(a), $C_1.v_3$ is not visible to loc_q. The searching stops at $(C_1.v_0, C_1.v_3)$ since the edge belongs to the hole C_1. The procedure also stops at the edge $(C_2.v_3, C_1.v_3)$ because $clamp_3^1(loc\_q, C_1.v_0, C_2.v_3)$ does not cover $C_0.v_3$. The vertices of $Poly_q$ are $\{loc\_q, C_2.v_3(f_1), C_1.v_0$ $(f_2), C_0.v_3\}$ where p is $C_0.v_3$. As a result, the sub procedure for $clamp_3^1(loc\_q, C_1.v_0, C_2.v_3)$ terminates at $tri(C_1.v_0, C_1.v_3, C_2.v_3)$. For the subclamp $clamp_3^2(loc\_q, C_1.v_0, C_3.v_0)$, the algorithm finds $tri(C_1.v_0, C_3.v_0, C_3.v_3)$ and checks $C_3.v_3$, which is visible. Then, $clamp_3^2(loc\_q, C_1.v_0, C_3.v_0)$ is split into two parts and the searching procedure is repeated. In the end, the searching for $clamp_3^2(loc\_q, C_1.v_0, C_3.v_0)$ returns $\{C_3.v_3, C_1.v_1\}$, seeing Fig. 4.10(b).

We summarize the result of $clamp_3^1(loc\_q, C_1.v_0, C_2.v_3)$ and $clamp_3^2(loc\_q, C_1.v_0, C_3.v_0)$, and get the visible point set $\{C_1.v_0, C_1.v_1, C_3.v_3\}$ for $clamp_3$. The procedure is the same for $clamp_1$ and $clamp_2$. Note that $C_2.v_1$ is visible to loc_q, but it is only found by $clamp_2(loc\_q, C_2.v_2, C_3.v_0)$ for the reason that $clamp_1$ and $clamp_3$ do not cover $C_2.v_1$. The algorithms are given in Algorithm 3 and Algorithm 4.

Algorithm 3: Inside

Input: loc_q - query point;
DG - dual graph on Tri.
Output: VQ - the set of *visible* points to loc_q

```
1 VQ = ∅;
2 put all vertices of tri_q into VQ;
3 let CL be the set of three clamps created by loc_q and tri_q;
4 for each clamp_i ∈ CL do
5     let e(clamp_i.f_1, clamp_i.f_2) be an edge;
6     VQ = VQ ∪ DepthTraversal(DG, clamp_i, e);
7 return VQ;
```

case2: loc_q meets tri_q

Algorithm 4: DepthTraversal

Input: DG - dual graph on Tri;
$clamp_i$ - a *clamp* structure;
e - an edge of a triangle.

Output: VQ - the set of *visible* points to loc_q covered by $clamp_i$

```
1  VQ = ∅;
2  let tri(loc_q, clamp_i.f_1, clamp_i.f_2) denote the created triangle;
3  if exists tri_i in DG that is adjacent to tri on the edge e then
4      let loc_q = clamp_i.a;
5      get p from tri_i;
6      if p is visible to loc_q then
7          VQ = VQ ∪ p;
8          split clamp_i into {clamp_i^1, clamp_i^2};
9          VQ = VQ ∪ DepthTraversal(DG, clamp_i^1, e_1(clamp_i^1.f_1, clamp_i^1.f_2));
10         VQ = VQ ∪ DepthTraversal(DG, clamp_i^2, e_2(clamp_i^2.f_1, clamp_i^2.f_2));
11     else
12         if ∠loc_q f_1 p < 180 then
13             VQ = VQ ∪ DepthTraversal(DG, clamp_i, e_1(clamp_i.f_1, p));
14         if ∠loc_q f_2 p < 180 then
15             VQ = VQ ∪ DepthTraversal(DG, clamp_i, e_2(clamp_i.f_2, p));
16 return VQ;
```

In this case, loc_q equals to the point of a triangle, i.e., a vertex of P. To determine the searching space, we process as follows. First, all triangles $\{tri_1, tri_i, ..., tri_n\} \subset Tri$ that contain loc_q are collected. This step can be optimized by building an R-tree on Tri as opposed to doing a linear searching. Then, for each tri_i we create a clamp with loc_q being the *apex* and the two triangle vertices (not equal to loc_q) being the feet. In the end, we call the function *DepthTraversal* for each created clamp. In Fig. 4.11, loc_q is located at $C_2.v_2$ and the query point is contained by $tri(C_2.v_2, C_2.v_3, C_3.v_0)$ and $tri(C_2.v_2, C_2.v_1, C_3.v_0)$. Two *clamps* are created $clamp(loc\_q, C_2.v_3, C_3.v_0)$ and $clamp(loc\_q, C_2.v_1, C_3.v_0)$. The first clamp is split into several parts during searching, demonstrated in the figure. We give the algorithm in Algorithm 5.

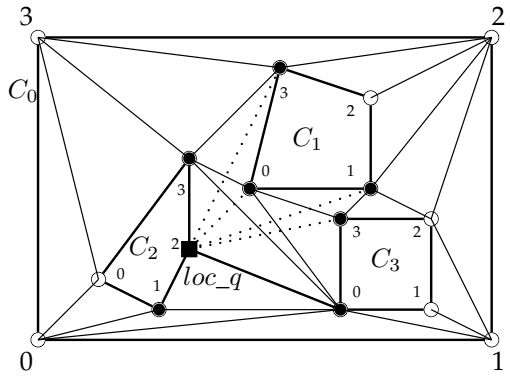


Figure 4.11: loc_q *meets* a triangle

Algorithm 5: Meet

Input: loc_q - query point;
DG - dual graph

Output: VQ - the set of all *visible* points to loc_q

```
1 VQ = ∅;
2 let S be the set of all triangles containing loc_q;
3 for each tri_i ∈ S do
4     create a clamp by tri_i and loc_q;
5     let e(clamp.f_1, clamp.f_2) be an edge;
6     VQ = VQ ∪ DepthTraversal(DG, clamp, e);
7 return VQ;
```

case3: loc_q is on_border of tri_q

Let (v_i, v_j) denote the edge that loc_q is located on. There are two cases: (1) (v_i, v_j) belongs to one contour of P; (2) (v_i, v_j) does not belong to any contour of P, demonstrated by loc_q_1 and loc_q_2 in Fig. 4.12. In the first case, two clamps have to be created to partition tri_q into two parts. We have

$clamp_1$ = (loc_q_1, $C_3.v_2$, $C_0.v_2$), and $clamp_2$ = (loc_q_1, $C_3.v_2$, $C_0.v_1$).

In the second case, the edge (v_i, v_j) is shared by two triangles both of which are considered to partition the searching space. Each triangle is split into two parts, resulting in four clamps. In this example, we have

$clamp_1$ = (loc_q_2, $C_0.v_3$, $C_2.v_0$), $clamp_2$ = (loc_q_2, $C_0.v_3$, $C_1.v_3$),

$clamp_3$ = (loc_q_2, $C_2.v_0$, $C_2.v_3$), $clamp_4$ = (loc_q_2, $C_1.v_3$, $C_2.v_3$). The algorithm is given in Algorithm 6.

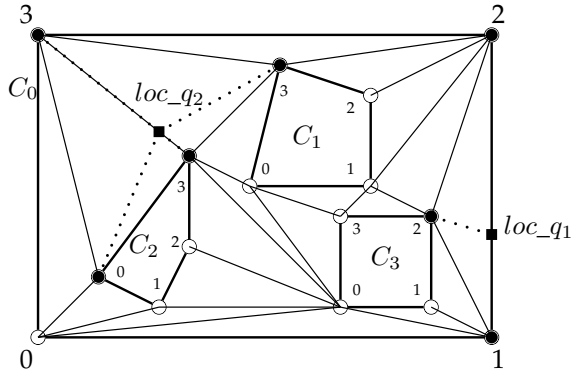


Figure 4.12: loc_q is *on_border* of a triangle

Algorithm 6: OnBorder

Input: loc_q - query point;
DG - dual graph.

Output: VQ - the set of all *visible* points to loc_q

```
1  VQ = ∅;
2  let (v_i, v_j) be the edge where loc_q is located;
3  if (v_i, v_j) belongs to one contour of P then
4      create clamp_1 and clamp_2 by loc_q and tri_q;
5      VQ = VQ ∪ DepthTraversal(DG, clamp_1, e_1(clamp_1.f_1, clamp_1.f_2));
6      VQ = VQ ∪ DepthTraversal(DG, clamp_2, e_2(clamp_2.f_1, clamp_2.f_2));
7  else
8      let C be the set of created clamps;
9      for each clamp_i ∈ C do
10         let e(clamp_i.f_1, clamp_i.f_2) be an edge;
11         VQ = VQ ∪ DepthTraversal(DG, clamp_i, e);
12 return VQ;
```

Time Complexity of Searching Visible Points. The algorithm *VisiblePoints* visits triangles from P to find visible vertices to loc_q. As a consequence, the complexity depends on the number of triangles of P after decomposition. Let N (≥ 3) be the total number of vertices in

P including outer and inner contours, and H be the number of holes. We use T to denote the number of triangles after polygon triangulation, calculated by a well-known formula

$$T = N + 2 * H - 2 \tag{4.1}$$

Now, we have to determine H for a polygon with N vertices. The lower and upper bounds of H can be determined:

$$H \in [0, (N-3)/3]. \tag{4.2}$$

The lower bound shows a polygon without holes. The upper bound indicates that P has three vertices for the outer contour and all the other vertices are for holes. Combine 4.1 and 4.2, we can get the size of T:

$$T \in [N-2, 5N/3-4] \tag{4.3}$$

In the worst case, the algorithm has to visit all triangles, resulting in the time complexity

$$O(5N/3-4) = O(N) \tag{4.4}$$

4.3.2 Routing in Bus Network

A bus network graph is defined for querying an optimal route with the minimum time. Before introducing the graph, we first recall the definition of a bus stop, Def. 3.2.1 in Chapter 3. A bus stop is represented by $bs_i(rid, pos)$ (rid, $pos \in D_{\underline{int}}$) where rid indicates the route *id*, pos records the stop order on the route.

By the location mapping technology in Section 4.1.2.2, a bus stop projects to a point on the pavement. Then, the walking distance between two stops can be computed applying the algorithm in Section 4.3.1, denoted by **dist**(bs_i, bs_j). We specify Δd as the length of a short distance walk, e.g., 150 meters. Two relationships are defined for bus stops: *neighbor* and *adjacent*. Given two bus stops bs_i and bs_j, we have

- bs_i is the *neighbor* of $bs_j \Leftrightarrow$ **dist**$(bs_i, bs_j) < \Delta d$
- bs_i is *adjacent* to $bs_j \Leftrightarrow bs_i.rid = bs_j.rid \wedge bs_i.pos + 1 = bs_j.pos$

Definition 4.3.4 *A bus network graph is defined as $G_{bn}(V, E, W)$ where V=$\{v_1, v_2, ..., v_n\}$ is a set of nodes representing bus stops, $E \subseteq V \times V$ are directed edges, and W is a set of weight functions returning the time cost for each edge. For each edge (v_i, v_j), an edge-delay function $w_{i,j}(t) \in W$ (t is a time variable in a time domain T) specifies how much time it takes to travel from v_i to v_j if arriving at v_i at time t.*

Let **geodata**(bs_i) return the spatial point of a bus stop. For simplicity, we ignore the parameter for a bus route. Two stops bs_i, bs_j are connected via an edge if one of the following conditions holds:

(i) **geodata**(bs_i) = **geodata**(bs_j);

(ii) bs_i is a *neighbor* of bs_j;

(iii) bs_i is *adjacent* to bs_j.

Each edge is associated with a value l recording the path from bs_i to bs_j. case (i): l is empty and $w_{i,j}(t) = 0$. case (ii): l is the shortest path located in the pavement area and $w_{i,j}(t)$ can be computed by defining a walking speed (e.g., 1m/s). case (iii): l is the bus line connecting two stops and $w_{i,j}(t)$ depends on the route schedule and arrival time at bs_i.

The query result of bus routing is a sequence of pairs (pl, m). Each pair indicates a connection between two stops where pl specifies the path (a polyline), and m shows the transportation mode for such a movement where the value is from the domain {*None*, *Walk*, *Bus*}. Each element in the domain corresponds to one of three connections. To improve query efficiency, two optimization techniques are developed.

- We run A^* algorithm on G_{bn} where the heuristic value is set as follows. Let d be the Euclidean distance between a bus stop and the destination, and $speed_b$ is the maximum speed of all buses. Then, $d/speed_b$ is computed and set as the heuristic value.

- Regarding the three connections between two nodes, let TP(v_i, v_j) return the type of an edge (v_i, v_j), where the values are from the symbol set {GEO, NEI, BUS}. Each symbol stands for a connection. GEO means v_i and v_j map to the same point, NEI says v_i, v_j are connected by a walking path and BUS is for a bus trip connecting v_i and v_j. In principle, given a node v_i, there are three edges starting from v_i, implying different values for TP(v_i, v_{i+1}). But by investigating TP(v_{i-1}, v_i), the type of (v_i, v_{i+1}) can be determined and some connections can be pruned. See below.

Lemma 3 *Access Node by Edge Type*

(1) *TP*(v_{i-1}, v_i) = *GEO* $\Rightarrow$ *TP*(v_i, v_{i+1}) = *BUS;*

(2) *TP*(v_{i-1}, v_i) = *NEI* $\Rightarrow$ *TP*(v_i, v_{i+1}) = *BUS;*

(3) *TP*(v_{i-1}, v_i) = *BUS* $\Rightarrow$ *TP*(v_i, v_{i+1}) $\in$ {*GEO, NEI, BUS*}.

For case (1), only one connection has to be considered for the reason that GEO and NEI are already processed when accessing v_{i-1}. Note that v_{i-1} and v_i are from different routes, resulting different bus trips to be processed. In case (2), bus stops with the same location as v_i are also *neighbors* of v_{i-1}, which are expanded by v_{i-1} with the connection NEI. So, the edge with type GEO can be filtered. The algorithm also does not have to process TP(v_i, v_{i+1}) = NEI. Consider the following cases. First, the total length of two walking movements by (v_{i-1}, v_i) and (v_i, v_{i+1}) could be larger than Δd, which contradicts the *neighbor* condition for two bus

stops. Second, if the length fulfills the *neighbor* condition, the union of them may be not the shortest path from v_{i-1} to v_{i+1} as the path is in obstructed space instead of Euclidean space. If the union path happens to be the shortest, then this path is already found by expanding v_{i-1} with the NEI connection. To sum up, the case TP(v_i, v_{i+1}) = NEI can be safely removed. For case (3), no edge can be pruned and all connections have to be considered.

To help understand the procedure, an example is shown in Fig. 4.13. For each location s_i, we give the bus stops at this location.

case (1): Consider the connection from <1, 3> to <2, 1> at s_4 where the two stops have the same location. Next, only the stop <2, 2> has to be accessed. Although s_1 is reachable from s_4 by walking, <3, 3> is already processed by expanding the node <1, 3>. The node <1, 4> is also visited by <1, 3>.

case (2): Suppose that the edge being processed is from <3, 3> to <1, 3>, i.e., $s_1 \rightarrow s_4$. When we expand <1, 3>, only <1, 4> is considered. <2, 1> is located at the same position as <1, 3>, but it is visited by s_1 as the NEI connection. Alg. 7 illustrates the pseudo-code of the algorithm.

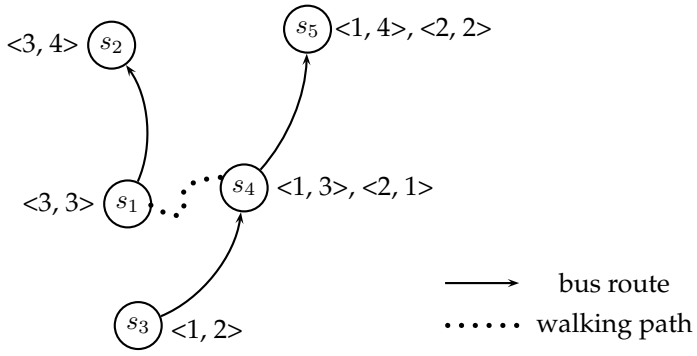


Figure 4.13: The Type of Bus Stop Connection

4.3.3 Routing in Metro Network

A graph is defined for querying the shortest path with respect to time between two stops.

Definition 4.3.5 *A metro network graph is defined as $G_{mn}(V, E, W)$ where: $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes representing metro stops; $E \subseteq V \times V$ are directed edges; W is a set of weight functions returning the time cost for each edge. For each edge (v_i, v_j), an edge-delay function $w_{i,j}(t) \in W$ (t is a time variable in a time domain T) specifies how much time it takes to travel from v_i to v_j if arriving at v_i at time t.*

A metro stop is defined by $ms_i(rid, pos)$ (rid, $pos \in D_{\underline{int}}$) where rid indicates the route *id*, pos records the stop order on the route. We define two metro stops ms_i, ms_j are *adjacent* by

ms_i is *adjacent* to $ms_j \Leftrightarrow ms_i.rid = ms_j.rid \wedge ms_i.pos + 1 = ms_j.pos$.

Let **geodata** be a function returning the spatial point of a metro stop. Two metro stops ms_i, ms_j are connected via an edge *iff* (1) **geodata**(ms_i) = **geodata**(ms_j); or (2) ms_i is *adjacent*

Algorithm 7: BNSP(bs_s, bs_e, G_{bn})

Input: bs_s, bs_e - start and end bus stops;
G_{bn} - bus graph
Output: sp - a shortest path from bs_s to bs_e by time

```
1  let Q be a priority queue, initially empty;
2  let T_se be the shortest path tree with the root node bs_s;
3  enqueue(Q, bs_s);
4  while head(Q) ≠ bs_e do
5      bs_i ← dequeue(Q);
6      foreach bs_{i+1} ∈ Adj(bs_i, G_bn) do
7          if (bs_i, bs_{i-1}),(bs_i, bs_{i+1}) fulfills one condition in Lemma 3 then
8              enqueue(Q, bs_{i+1});
9              insert bs_{i+1} into T_se ;
10 select sp from T_se;
11 return sp;
```

to ms_j. Different from G_{bn}, we do not define two metro stops are linked by a walking path because in practice usually people do not have to go to another place for a metro transfer. During the query processing, a heuristic value $d/speed_m$ is set where d is the Euclidean distance between two metro stops and $speed_m$ is the metro speed (all metros are the same). The pseudo-code of the algorithm is given in Algorithm 8.

4.3.4 Indoor Navigation

We construct an indoor graph for searching the precise shortest path inside a building. Such a graph is introduced in Section 3.2.2.2 of Chapter 3. Basically, a node represents a door and an edge denotes the shortest path between two doors inside one groom. To implement a robust indoor graph, some *virtual doors* (not existing in the real world) are created during the development. These doors aim to represent the places that connect two grooms but without explicit doors existing. For example, a staircase (modeled as a groom) builds the connection between two different floors, but maybe no such a door exists for entering the staircase.

An indoor location is represented by $(oid, (loc_1, loc_2))$ where oid is a groom identifier and (loc_1, loc_2) is the location inside the groom. The location can be anywhere inside the building where oid can map to an office room, a corridor, even a staircase. Given two locations i_s and i_e for shortest path searching, initially one needs to connect them to G_{indoor}, i.e., find all doors in the room where i_s and i_e are located, as i_s and i_e are stochastic locations (not necessarily positions for doors).

Some optimization techniques are developed to improve the query processing.

Algorithm 8: MNSP(ms_s, ms_e, G_{mn})

Input: ms_s, ms_e - start and end metro stops;
G_{mn} - metro graph

Output: sp - a shortest path from ms_s to ms_e by time

```
1  let Q be a priority queue, initially empty;
2  let T_se be the shortest path tree with the root node ms_s;
3  enqueue(Q, ms_s);
4  while head(Q) ≠ ms_e do
5      ms_i ← dequeue(Q);
6      foreach ms_{i+1} ∈ Adj(ms_i, G_mn) do
7          enqueue(Q, ms_{i+1});
8          insert ms_{i+1} into T_se;
9  select sp from T_se;
10 return sp;
```

- A^* algorithm can be applied to find a path with the minimum distance when the start and end locations are at the same level. This is done by setting the Euclidean distance as the heuristic value. We take the center point of each door (represented by a line) to compute the distance. If start and end locations are at different levels, the above heuristic value might not be efficient (still correct) as it does not involve any information about staircase and elevator which are critical for movement between different levels.

- To solve the problem above, another technique is developed for locations on different levels. The heights above the ground level of start and end locations can be retrieved by accessing their grooms. The two values define the height range for all locations on the shortest path. Then, all doors whose height values are out of the range can be pruned during searching.

- Before searching on G_{indoor}, the first step is to connect start and end locations to all doors of their grooms. This may involve much computation if the room has several doors, for example, a corridor. The reason is when a door is connected to the start location, all its adjacent doors will be processed for the shortest path computation. But some doors are not necessarily to be connected. This depends on the room rm that the door belongs to. If both the following conditions hold (1) rm has only one door and (2) rm is not the destination room, the room does not have to be visited. Consequently, we can safely prune the door of rm.

Algorithm 9: IndoorSP(i_s, i_e, G_{indoor}, $Indoor\_Rel$)

Input: i_s, i_e - start and end indoor locations;
G_{indoor} - indoor graph;
$Indoor\_Rel$ - rooms and doors of a building

Output: sp - a shortest path from i_s to i_e

```
1  if i_s.oid = i_e.oid then
2      let r be the 2D area for the room i_s.oid, obtained by accessing Indoor_Rel;
3      create the dual graph DG and the visibility graph VG on r;
4      return RBOSP(i_s, i_e, DG, VG);
5  let D_s be doors in the room i_s.oid by searching Indoor_Rel;
6  let D_e be doors in the room i_e.oid by searching Indoor_Rel;
7  let Q be a priority queue, initially empty;
8  let T_se be the shortest path tree with the root node i_s;
9  foreach d_i ∈ D_s do
10     enqueue(Q, d_i);
11     insert d_i into T_se;
12 while head(Q) ≠ i_e do
13     u_i ← dequeue(Q);
14     if ∃d_i ∈ D_e such that d_i = u_i then
15         enqueue(Q, i_e);
16         insert i_e into T_se;
17     foreach u_{i+1} ∈ Adj(u_i, G_indoor) do
18         enqueue(Q, u_{i+1});
19         insert u_{i+1} into T_se;
20 select sp from T_se;
21 return sp;
```

4.3.5 Time Complexity Analysis

In this part, we study the complexity of proposed algorithms for trip plannings. Since each algorithm is executed on a graph, the time complexity is $O(NlogN + E)$ [26] by implementing a min-priority queue where N is number of nodes and E is the number of edges in the graph. In the following, we show the size of N and E in each case.

VG: Let n be the number of nodes in VG, that is P (the polygon for pavements) contains n vertices in total. Before searching VG, one first has to connect the start and end locations to VG. To find visible points for a query location inside a polygon with n vertices, the time $O(n)$

is needed [3]. Consequently, the overall time is $O(n + nlogn + E)$ where E is n^2 in the worst case, implying that each pair of points is visible. Practically, for such an application where P represents the city walking area with many obstacles, it is impossible that each vertex is visible to all the others. The obstacles (holes in P) are the places where people cannot directly pass through, e.g., building blocks, junction areas. So, we denote the size of E by $O(kn)$ where k is the maximum number of visible points for a vertex in P. In the end, the time complexity is therefore $\Theta(kn)$ ($k \ll n$).

G_{bn}: Assuming n bus stops in total, we need to determine the cardinality for bus stop connections. Stated in Section 4.3.2, there are three cases to build an edge between two stops bs_i, bs_j: (1) **geodata**(bs_i) = **geodata**(bs_j); (2) bs_i is the *neighbor* of bs_j; (3) bs_i is *adjacent* to bs_j. For each bus stop, let c_1, c_2 be the maximum number of connections for (1) and (2) where $0 \leq c_1 \wedge c_1 \ll n$ and $0 \leq c_2 \wedge c_2 \ll n$. Additionally, there is one connection for *adjacent*. This yields $c_1 + c_2 + 1$ connections in maximum for a bus stop. In summary, the time cost is $\Theta((c_1 + c_2)n)$ ($c_1, c_2 \ll n$).

G_{mn}: Let n be the total number of metro stops, and we give the size for E. For each stop, there is one connection to its adjacent stop on the same route. At the same time, there can be c_3 ($0 \leq c_3 \wedge c_3 \ll n$) stops from other routes having the same spatial location where a transfer can occur. Only these two indicate possible connections between two metro stops. So, the number of edges in G_{mn} is $E = n(1 + c_3)$ and the overall time complexity is $\Theta((1 + c_3)n)$ ($c_3 \ll n$).

G_{indoor}: Given an indoor graph with n nodes, each node represents a door and edges are connections between two doors in the same groom. Let d be the maximum number of doors in one groom, then the edge number is $O(dn)$ ($d \ll n$). Now we consider the time on finding all paths from the start (end) location to doors in the room. This equals to shortest path searching in a polygon P_r where P_r represents the 2D area of a room. If P_r is a convex polygon, this results in $O(d)$. If P_r is a concave polygon with or without holes, we apply the algorithm in Section 4.3.1. Assume m to be the maximum vertex number for P_r. Building dual and visibility graphs costs $O(m^2)$. This is done on-the-fly for the reason that (1) m is usually small (2) much storage and maintenance effort are needed for persistent data. After that, searching the shortest path needs $O(k'm)$ on VG where k' is the maximum number of visible points for a polygon vertex, resulting in $O(m^2 + dk'm)$ to connect start and end locations to G_{indoor}. To sum up, the time complexity of indoor navigation is $O(m^2 + dk'm + dn)$ ($d < m$, $k' \leq m$, $m \ll n$).

We summarize the time complexity for all trip planning algorithms in Table 4.2.

4.3.6 Generic Moving Objects Generation

To create a moving object, two parameters are needed: (1) path; (2) speed. Trip planning produces the path and the located environment of the path determines the speed for mov-

[3] the classical *rotational plane sweep algorithm* [76] needs $O(nlogn)$ in an open space, but here we have a closed area P and a new optimal searching algorithm is developed, seeing Section 4.3.1.2.

Environment	Time Complexity
Region-Based Outdoor	$\Theta(kn)$ $(k \ll n)$
Bus Network	$\Theta((c_1 + c_2)n)$ $(c_1, c_2 \ll n)$
Metro Network	$\Theta((1 + c_3)n)$ $(c_3 \ll n)$
Indoor	$O(m^2 + dk'm + dn)$ $(d < m, k' \leq m, m \ll n)$

Table 4.2: Time Complexity for Routing

ing objects. In the following, we introduce the generation of moving objects with different transportation modes.

4.3.6.1 Car + Walk

The needed infrastructures are I_{rn} and I_{rbo}. We let the trip start from I_{rbo}, move a short distance on the pavement, then travel along the road and terminate in I_{rn}. The start location is a randomly selected point on the pavement, denoted by p_1. From p_1, the traveler walks some distances to another nearby pavement point p_1'. Using the location mapping technique (Section 4.1.2.2), p_1' projects to a road network position rp_1, from which the traveler moves by car to the destination. The complete path is through two environments and consists of (1) walk movement; (2) road path. Walking speed is defined as 1m/s and the speed for car is determined by the maximum speed allowed on the road. Recall that *Road_Rel* (in Section 4.1.1) has an attribute for the street type, and the maximum speed is set according to the type. For example, the main street is 50km/h and the side street is 30km/h.

4.3.6.2 Bus+Walk and Metro+Walk

Combining I_{rbo} and I_{bn}, we are able to create moving objects with modes *Bus + Walk*. The start and end locations are two stochastic pavement points, denoted by p_1 and p_2. First, the closest bus stop to p_1 is found, denoted by bs_1. Then, bs_1 maps to a position on the pavement for the purpose of finding the walking trip from p_1 to bs_1. Afterwards, the bus trip algorithm is called to find a connection from bs_1 to bs_2 with the minimum time where bs_2 is the closest bus stop to p_2. During this movement, a short distance walking may also be involved for a bus transfer. A bus speed is set as the maximum speed allowed on the road. In the end, the traveler leaves the bus network at bs_2 and moves from bs_2 to p_2 on foot. Similarly, we can also create a trip with modes *Metro + Walk*.

4.3.6.3 Indoor

An indoor moving object is created based on (1) the precise shortest path returned by indoor navigation where the start and end locations can be anywhere inside a building; (2) two de-

fined speed values: walking and elevator (if such a device is involved by the traveler).

4.3.6.4 Both Outdoor and Indoor Movement

Including all infrastructures, we are able to create the comprehensive movement for a person, e.g., starts from home, then moves by car or bus, enters a building, and finally reaches a room. Assuming that people start and end their trips in the indoor environment, start and end locations are defined in different buildings, say B_i, B_j. Due to lack of floor plans for private buildings, if B_i (B_j) corresponds to a personal house, we ignore the movement inside the building. Suppose that both B_i and B_j are public buildings. A trip starts from B_i where a random location in B_i is selected as the start position. The shortest path is found for routing to an exit of B_i. In Section 4.1.2.2, we state that the global space manages a set of places where the transportation mode can switch, including exits/entrances of buildings and pavement locations. Each building exit maps to a pavement location, leading to the connection between indoor and outdoor. Now consider the outdoor movement. If the distance between B_i and B_j is smaller than a defined threshold (e.g., 300m), the traveler moves by walk for outdoor. Otherwise, a car or the public transportation system (we assume people tend to use vehicles for a long distance trip) is used. The distance between B_i and B_j is set as the Euclidean distance between the bounding boxes of outdoor area. Finally, the movement inside B_j starts from the building entrance and terminates at a random indoor location.

4.4 MWGen Evaluation

MWGen is developed in an extensible database system SECONDO [36] and programmed in C/C++ and Java on a PC (AMD 3.0 GHz, 4 GB memory) installed Suse Linux 11.3. We carry out experiments using two real road datasets for cities Berlin and Houston and a set of public floor plans. The roads of Berlin are downloaded from BBBike [5] and Houston roads are from Census Tiger/Line® [7], where Berlin is the largest city in Germany and Houston is the fourth largest city in U.S.A. The original data format is long/lat, and we convert the data by Gauss Krueger.

4.4.1 Infrastructures Statistics

Table 5.3(a) shows the infrastructure data and the cost of overall generation in terms of time and disk space. Table 4.4 lists all outdoor graphs where G_{rn} denotes the graph for road network and G_{mn} denotes the graph for metro network. Table 4.5 summarizes the information for all buildings. No_B and No_H are used to denote the number of buildings for each type in Berlin and Houston, respectively. Given a building B_i, let $|B_i.R|$ be the cardinality of rooms.

For each public building, an indoor graph is created where the cardinality for nodes and edges is also reported.

	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
Bus Stops	2,852	5,498
Metro Routes	10	16
Buildings	4,996	5,992
Time Cost	63 min	170 min
Database Size	2.5 G	9.7 G

Table 4.3: An Overview of Infrastructure Data

Graph Type	Berlin		Houston	
	Nodes	Edges	Nodes	Edges
VG	116,516	1,409,621	437,353	4,120,987
DG	145,608	160,154	458,797	469,519
G_{rn}	10,628	30,002	10,063	23,982
G_{bn}	2,852	9,270	5,320	17,994
G_{mn}	644	2,476	1,344	4,240

Table 4.4: Statistics of Outdoor Graphs

4.4.2 Visible Points Searching

In this part, we evaluate the procedure of searching visible points and compare the performance between the proposed algorithm VisiblePoints (VP for short) and the classical rotational plane sweep (RPS) algorithm [76]. First, we introduce the procedure of RPS algorithm. Let p_s be the query point and $\overrightarrow{p_s p_h}$ be the starting horizontal sweep line ($p_h.x$ is larger than the axis value of p_s and all obstacle vertices). Initially, the algorithm sorts all obstacle vertices, each of which is denoted by p_i, in counter clockwise (or clockwise, in the following we use the clockwise order) according to p_s (i.e., rotating from $\overrightarrow{p_s p_h}$ to $\overrightarrow{p_s p_i}$). After sorting, the vertices are stored in a priority queue according to the angle value. A binary search tree T is used for *visible* checking. Each node in T stores the segment of an obstacle and the key value is set as

Type	No_B	No_H	$\|B_i.R\|$	Indoor Graph	
				Nodes No.	Edges No.
personal house	3,713	5,000			
officeA	600	530	294	777	17,260
officeB	487	266	214	537	17,108
shopping mall	80	79	360	902	21,022
cinema	6	8	21	47	130
hotel	39	40	584	1,471	87,628
hospital	46	48	89	271	4,104
university	20	20	431	1,063	7,608
train station	1	1	56	113	1,280

Table 4.5: Statistics of Buildings and Indoor Graphs

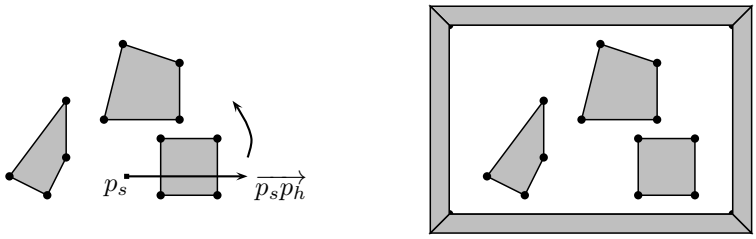
the distance between p_s and the segment. At the beginning, T stores all segments that intersect $\overrightarrow{p_sp_h}$. For each p_i popped from the queue, three operations are performed:

(1) *visible* checking;

(2) inserts segment p_ip_j into T where rotating from p_sp_i to p_sp_j is in counter clockwise order;

(3) deletes segment p_ip_k from T where rotating from p_sp_i to p_sp_k is in clockwise order.

If p_i is *visible* to p_s, this means that $\overrightarrow{p_sp_i}$ neither passes through any obstacles nor contains any obstacle vertices (excluding p_i). Let $min(T)$ be the smallest key in T and there are two cases that p_i is *visible*: (i) $dist(p_s, p_i) \leq min(T)$ and (ii) no obstacle segment in T intersects $\overrightarrow{p_sp_i}$ or the intersection point is p_s or p_i. Figure 4.14(a) shows an example.



(a) (b)

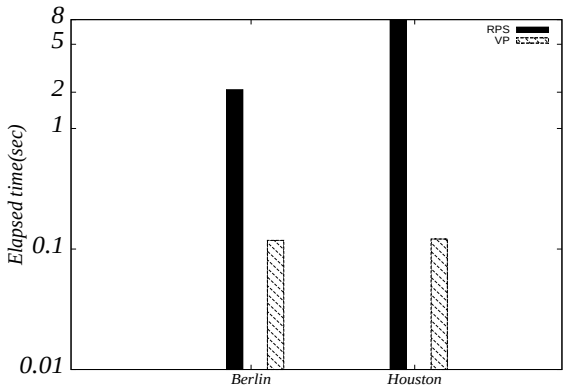
Figure 4.14: Rotational Plane Sweep

Although the original RPS method does not have the *outer* boundary covering all obstacles, the algorithm can be adapted as follows: For the points inside P, RPS can be directly applied. While for points locating on the boundary, a bounding box can be created containing the *outer* boundary as a region inside and the space between the bounding box and the *outer* boundary is treated as obstacle areas (see Figure 4.14(b)). The time complexity for RPS is $O(NlogN + N)$

for each query point, where N is the number of all obstacle vertices. $O(NlogN)$ is for sorting and $O(N)$ is for sweep processing. However, RPS algorithm is not efficient for a large polygon with many vertices, due to the sorting procedure and operations on T such as inserting and deleting.

	RPS	VP
Berlin	2.10	0.118
Houston	7.89	0.121

(a) Time (sec)



(b) Cost

Figure 4.15: Evaluation of Visible Points Searching

We compare the performance of two algorithms (1) RPS and (2) VP by using both infrastructure data of Berlin and Houston. In each city, 5,000 random pavement points are generated to be the set of query locations. We run both algorithms to find the visible vertices for each query point and measure the execution time. The final result is the average value over all runnings where the time measurements are plotted in logarithmic scale. To have a fair comparison, both algorithms have the same 5,000 query points. The complexity of RPS is $O(NlogN + N)$ for each case, while our algorithm is $O(N)$ in the worst case. The experimental results confirm the efficiency of our algorithm where VP achieves about an order of magnitude performance improvement.

4.4.3 Efficiency of Trip Plannings

We evaluate the efficiency of outdoor trip plannings and indoor navigation. The result of pedestrian routing is averaged over 5,000 pairs of random locations. For bus and metro routing, 5,000 pairs of random stops are generated and each pair is assigned a query time instant. In I_{bn}, Houston takes more time than Berlin due to its large occupied area, resulting in longer bus routes and more bus stops. The outcome is the average time over all runnings. The time cost of indoor navigation in each public building is reported in Fig. 4.16(b). In each case, we

measure the execution time and average the result over running shortest path searching on 500 pairs of random indoor locations. The experimental results demonstrate the efficiency of the algorithms.

	Berlin	Houston
I_{rbo}	0.78	2.4
I_{bn}	0.13	0.23
I_{mn}	< 0.1	< 0.1

(a) Outdoor (time in sec)

officeA	0.25	officeB	0.27
mall	0.37	cinema	0.32
hotel	1.57	hospital	0.35
university	0.123	trains station	0.1

(b) Indoor (time in sec)

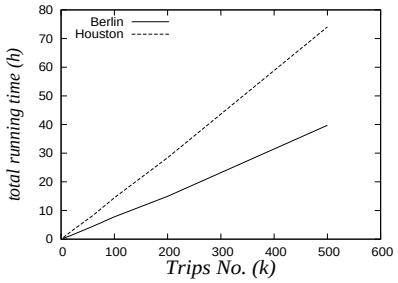
Figure 4.16: Time Cost of Trip Plannings

4.4.4 Scalability of Moving Objects Generation

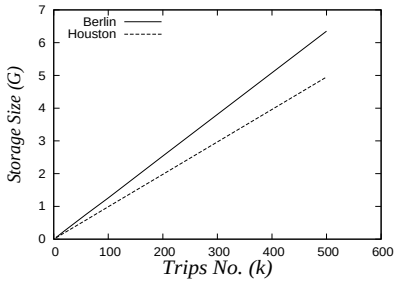
This part of experiment exhibits the scalability of generating a variable size of generic moving objects, in terms of time and storage size. For the three kinds of generic moving objects introduced in Section 4.1.3, the distribution is set by 2:1:1 assuming that the number of people traveling by car is the same as that of people traveling by public transportation system. The start and end locations of a trip are defined in the indoor environment where two buildings are selected. The start and end buildings can be public or private. The location inside a public building is randomly chosen. The distance between two buildings is defined to be larger than a value (300m) so that vehicles are involved for traveling. Figure 4.17 reports the experimental results. Overall, we see that the running time and storage size scale linearly to the number of trips.

Trip No.	Berlin		Houston	
	Time (h)	Disk Size (G)	Time (h)	Disk Size (G)
4k	0.32	0.052	0.57	0.038
8k	0.56	0.1	1.16	0.08
20k	1.39	0.257	2.85	0.198
60k	4.49	0.767	8.31	0.592
100k	7.8	1.26	14.46	0.99
200k	14.95	2.54	28.42	1.98
500k	39.75	6.35	74.06	4.95

(a)



(b) Time



(c) Disk Size

Figure 4.17: Evaluation of MWGen Scalability

4.5 Conclusions

In this chapter, we present a generator MWGen, which reads in roads and floor plans, and outputs the infrastructures and generic moving objects for GMOD. Outdoor infrastructures are created based on road networks and indoor environment is generated from floor plans. Each infrastructure has a graph for trip plannings within that environment. A new and efficient algorithm is proposed for searching visible points, the result of which benefits to return the shortest path for pedestrians. Generic moving objects are produced based on trip plannings where the movement can cover all available environments. The space manages all infrastructures and also serves as an interface between moving objects and underlying referenced objects, e.g., roads, pavements, rooms. All created data are managed by GMOD and will be used for testing GMOD performance.

Chapter 5

GMOBench: A Benchmark for Generic Moving Objects

5.1 Benchmark Data

5.1.1 Movement Rules

In order to create realistic benchmark data, we define a set of movement rules that can reflect the characteristics and distribution 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 the 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 because they can mirror key features of human movement. The four rules are defined as follows:

- **MR1**. This rule is to represent the *regular* movement, 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 a 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, 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 that has 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 the desired and interesting trips, e.g., a traveler passes a sequence of places at the minimum cost.

5.1.2 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., the location can be in any environment defined in Table 3.1 (Section 3.1.2.1). 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 3.1.2.1 of Chapter 3, 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 5.1.1 *Integer Sets for IFOB Ids*

Let IO_ID={ID_R, ID_P, ID_B, ID_M, ID_IN} be the overall integer set for all IFOB ids and the following two conditions hold:

(i) $\forall$ *U* $\in$ *IO_ID*, *U* $\subset$ $D_{\underline{int}}$;
(ii) $\forall$ *U, V* $\in$ *IO_ID*, *U* $\neq$ *V* $\Rightarrow$ *U* $\cap$ *V* = $\varnothing$.

Each element in IO_ID is a set that stores integers representing the ids of IFOBs in the corresponding infrastructure. An element is denoted by a symbol ID plus the first (two) character(s) of the environment name. For example, ID_R stores all road and street ids in I_{rn}, and ID_IN records building ids in I_{indoor}. There is no overlapping between the integer sets from different infrastructures. Since the location representation (Def. 3.1.7 in Section 3.1.2.3 of Chapter 3) records an IFOB id, one can set an integer range denoting certain IFOBs. The range can determine the environment for the start and end locations. The parameter is defined below.

Definition 5.1.2 *Location Parameter*
A pair $l(l_s, l_e)$ *denotes the start and end locations of a trip with the conditions:*
(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 the constraints, 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_R, the start location is on a road. If c_s = ID_P, gl_s is located in the pavement area. For the case c_s = c_e = ID_IN, 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_R, 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].

Definition 5.1.3 *Time Parameter*
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) $h \in hc \subseteq Hour$;
(ii) $min \in minc \subseteq Min$.

The two attributes h and min define the start time of a trip and each value is set according to its corresponding constraint. 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 the 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. Recalling Def. 3.1.4 in Section 3.1.2.1 of Chapter 3, we do the following partition on a subset of transportation modes (excluding the mode Free). Let TM denote the result, seeing below.

Definition 5.1.4 *Partition of Transportation Modes*
TM = A ∪ B where
(i) A = {Walk};
(ii) B = B1 ∪ B2 ∪ B3 where
B1 = {Indoor}, B2 = {Bus, Metro, Taxi}, B3= {Car, Bike}.

TM is first partitioned into two groups A and B where A contains only one element. The modes in B are further grouped, thus we can distinguish between (1) indoor and outdoor; (2) public transportation modes and private vehicles. We define *Walk* to be the only mode that connects to another mode, e.g., *Walk* → *Car*, *Indoor* → *Walk* → *Bus*. The connection between two modes such as *Car* → *Indoor* and *Indoor* → *Bus* is not allowed, implying there is no mode switch between elements from B. 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 in [103] where 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. Table 5.1 gives the transition matrix for TM. A is transitive to all the others, while B1, B2, B3 are only transitive to themselves and A.

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

Table 5.1: Transportation Mode Transition

Definition 5.1.5 *Transportation Modes Parameter*

The mode parameter is a set $mc \subset TM$ to specify the modes that are to be included by a trip.

See some instances: (1) mc = B1, movement inside a building; (2) mc = A $\cup$ B2, take the bus and a short distance walking; (3) mc = A $\cup$ B1 $\cup$ B3. Note 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.

5.1.3 Configuration

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_IN representing buildings in I_{indoor} 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 constraint in the location parameter by H $\cup$ W for **MR1** to represent regular movement between home and work place. The trips representing friends visiting on the weekend (**MR4**) can set H for both start and end locations.

A generic moving object is created based on an optimal route with respect to the minimum cost (e.g., distance, time) from one location to another. The start and end locations can be in the same or different environments, leading to multiple transportation modes. For each rule, a set of movement instances is specified to create concrete trips. Table 5.2 lists some instances for each rule as well as the parameter settings. For simplicity, we only show the constraint value for each parameter. Note that in some cases transportation modes are in fact determined 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.

Rule	Movement Instance	c_s, c_e	hc	mc
MR1	travel from home to work	H, W	$[6, 9]$	A ∪ B1 ∪ B2
	business travel between working places	W, W	$[9, 16]$	A ∪ B1 ∪ B3
MR2	people walk around in the city center	ID_P, ID_P	$[10, 18]$	A
	a clerk walks from one office to another	W,W	$[9, 16]$	B1
MR3	a car searches the nearest restaurant	ID_R, ID_IN	$[10, 20]$	A ∪ B3
	a traveler looks for the nearest bus stop	ID_P, ID_B	$[9, 20]$	A
MR4	go to a shopping mall from home	H, SE	$[10, 20]$	A ∪ B1 ∪ B3
	visit friends	H, H	$[10, 20]$	A ∪ B2

Table 5.2: Example Movements and Parameter Settings

5.1.4 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 in several infrastructures. In the sequel, different trip planning algorithms are needed, e.g., indoor navigation, trip planning for pedestrians, routing in bus network. Fig. 5.1(a) 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 $M(I_{rbo}, I')$ ($I' \in \{I_{rn}, I_{bn}, I_{mn}, I_{indoor}\}$).

Third, a set of speed values are defined. We summarize them in Fig. 5.1(b). 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 5.2 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 modes.

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) Infrastructure Graphs

Name	Meaning
S_c	car speed values
v_m	metro speed
v_w	walking speed

(b) Speed Values

Figure 5.1: Algorithm 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. 5.1(b) and create the sub movement. In the end, we perform the union on all sub trips to get the complete trip.

5.2 Query Workload

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. Some of the queries are taken from Section 3.4 and Section 3.5 in Chapter 3 where we present them as running examples. Some queries are proposed for the benchmark. To avoid ambiguous notations, we use the symbol **BQ** to name the following queries for the purpose of distinguishing between benchmark queries and queries in Chapter 3. We provide the formulation for new proposed queries by using an SQL-like language in the Appendix D.

Infrastructure Queries.

- **BQ1.** Find out all metros passing through the city center.
- **BQ2.** Given a building, find all bus stops that are within a radius of 300 meters.

- **BQ3.** Which streets does Bus No. 12 pass by?
- **BQ4.** 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.

- **BQ5.** 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. 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.

- **BQ6.** 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.

- **BQ7.** Where does *Bobby* walk during his trip? And how long does he walk?
- **BQ8.** Find out all people passing room 312 at the office building between 9:00am and 11:00am on Monday.

To answer **BQ7, BQ8**, we need to get a sub trip according to the transportation mode.

- **BQ9.** Who arrived by taxi at the university on Friday?
- **BQ10.** 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.

- **BQ11.** Find out all people walking through zone A and zone B on Saturday between 10am and 3pm.
- **BQ12.** Did anyone who was on floor H-5 of the office building between 2pm and 5pm take a bus to the train station on Friday?

- **BQ13.** Did bus No. 35 pass by any bicycle traveler on Monday?

BQ13 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.

- **BQ14.** Did someone spend more than 15 minutes on waiting for the bus at the bus stop Cinema on Saturday?

- **BQ15.** How many people visit the cinema on Saturday?

- **BQ16.** 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.

- **BQ17.** Did someone spend more than one hour on traveling by bus (metro)?

- **BQ18.** 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.

- **BQ19.** 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 movement.

- **BQ20.** Find the top k bus (metro) routes with high passenger flow for all workdays.

- **BQ21.** 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 **BQ21**, 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] $\cup$ [16:00, 18:00]), where the following transportation modes are considered: $\{Car, Taxi, Bicycle, Bus\}$.

5.3 Optimization

5.3.1 Transportation Modes 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 moving object mo. Since a large part of queries specify a transportation mode (e.g., **BQ7**, **BQ9**, **BQ16**), one needs to find all qualified moving objects. To get the result, one option is to sequentially scan all units of mo, comparing the mode attribute to see if the value is identical to the argument. However, this yields a linear searching for each mo. 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 mo 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 Def. 3.1.4 of Chapter 3. 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* $\rightarrow$ *Walk* $\rightarrow$ *Bus* $\rightarrow$ *Walk* defines the value 14 (binary 0001110). We calculate IM for each mo in advance and store it as an attribute along with mo 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.

5.3.2 Index on Units

Recall that a moving object is represented by a set of units arranged in a linear order on time. 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 **BQ5** the movement with the mode *Bus* is returned, and in **BQ7** walking trips are specified. For travelers who take public vehicles, one might get the sub trip on a particular bus (metro), seeing **BQ10** and **BQ18**. 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 of mo. 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 in I maps to a sub movement in mo with a single mode. For example, we have such an example movement

mo = < $(Indoor)_1$, $(Indoor)_2$, ... , $(Indoor)_{10}$,
$(Walk)_{11}$, $(Walk)_{12}$, ... , $(Walk)_{15}$,
$(Bus)_{16}$, $(Walk)_{17}$, ... , $(Walk)_{20}$>.

The index built on mo is I = {(*Indoor*, 1, 10), (*Walk*, 11, 15), (*Bus*, 16, 16), (*Walk*, 17, 20)}. Although I is still a list structure, the quantity of elements depends on the number of modes in mo, 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 in mo and then access these units 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.

5.3.3 Projecting 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 (**BQ13**) 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 nearly accesses 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 Def. 3.1.7 of Chapter 3, the value oid corresponds to a road if the moving object is a bike traveler, resulting in fast access to the road segment.

5.4 Benchmark Results

This section shows extensive experimental results of the performance evaluation. The implementation is done in an extensible database system SECONDO [36] and programmed in C/C++ and Java. A standard PC (AMD 3.0 GHz, 4 GB memory, 2TB disk) running Suse Linux (kernel version 11.3) is used. We utilize the tool [98, 100] to create all infrastructure data based on real road datasets and a set of public floor plans (e.g., [11, 8, 10]). Two road datasets are used, Berlin [5] and Houston [7]. The tool takes roads and floor plans as input and generates pavement areas, bus network, metro network and a set of buildings.

5.4.1 Experimental Setup

The benchmark generator provides a set of configuration parameters that allow a user to create the desired datasets, including (1) the total size of moving objects; (2) the number of trips according to each movement rule; (3) distribution of transportation modes; (4) start and end locations; (5) time distribution. In the following, we present the setting in the benchmark experiment. We simulate one week movement. All trips are created based on the rules presented in Section 5.1.1 and the distribution is listed in Fig. 5.2(a). We add one more pattern to create moving objects each of which visits several places, e.g., home $\rightarrow$ shopping $\rightarrow$ restaurant $\rightarrow$ home. The time distribution of each rule is shown in Fig. 5.2(b).

We summarize the distribution of transportation modes in Fig. 5.2(c). 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 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. For a long distance journey that visits more than one place, we let the modes be *Indoor* + *Walk* + *Car*. For some rules, we let the two cities have different distributions on transportation modes.

Name	Percentage
MR1	40%
MR2	10%
MR3	10%
MR4	30%
Long Trips	10%

(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* + { *Bike* : 5%, *Car* : 50%, *Bus* : 20%, *Metro* : 20%, *Taxi* : 5% }	*Indoor* + *Walk* + { *Bike* : 5%, *Car* : 60%, *Bus* : 15%, *Metro* : 15%, *Taxi* : 5% }
MR2	*Walk*: 40%; *Indoor*: 60%	*Walk*: 40%; *Indoor*: 60%
MR3	*Indoor* + *Walk* + { *Car* : 50%, *Bus* : 35%, *Taxi* : 15% }	*Indoor* + *Walk* + { *Car* : 60%, *Bus* : 28%, *Taxi* : 12% }
Long Trips	*Indoor* + *Walk* + *Car*: 100%	*Indoor* + *Walk* + *Car*: 100%

(c) Transportation Modes Distribution

Figure 5.2: Benchmark Generator Parameters

5.4.2 Benchmark Datasets

Fig. 5.3 lists all datasets that we use for the experiment. All infrastructure data are shown in Fig. 5.3(a) and the detailed information of buildings is reported in Fig. 5.3(b). We create a set of buildings of different types based on their floor plans to simulate a city environment. Besides static IFOBs such as roads and bus routes, 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 $\in$ {Mon, Tue, Wed, Thu, Fri, Sat} than on Sunday. For metros, the schedules are the same for the whole week. The detailed information is shown in Fig. 5.3(c). Moving objects datasets are described in Fig. 5.3(d). The generator allows to create data sets of varying sizes, thus is a flexible tool for evaluation.

5.4.3 Performance Evaluation

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

	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

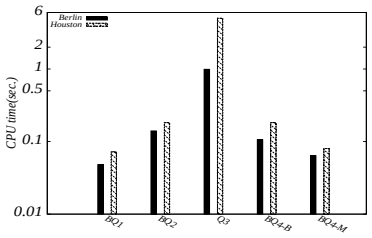
Name	Trips No.	Total Units No. (M)	Avg. Units No. Per Trip	Disk Size (G)
Berlin10k	10,068	0.585	58.1	0.086
Berlin20k	20,127	1.17	58.4	0.174
Berlin50k	50,289	2.93	58.3	0.435
Berlin100k	100,635	5.86	58.3	0.871
Berlin200k	201,209	11.75	58.4	1.75
Berlin500k	503,673	29.36	58.3	4.36
Houston500k	515,664	29.73	57.6	4.4

(d) Moving Objects

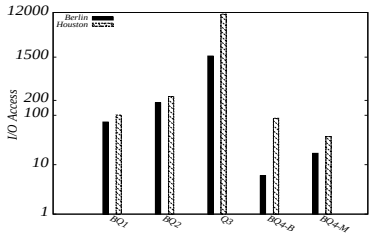
Figure 5.3: Classification of Datasets

5.4.3.1 Queries on Infrastructures

Fig. 5.4 shows the cost of infrastructure queries where both time and I/O measurements are plotted in logarithmic scale. The results show that the query cost in Houston is higher than that of Berlin. This is because Houston contains more infrastructure objects and occupies a larger area compared with Berlin. **BQ3** takes more time and I/O accesses than other queries as the procedure involves the costly intersection computation between roads and bus routes.



(a) CPU Time



(b) I/O Accesses

Dataset	BQ1	BQ2	BQ3	BQ4-B	BQ4-M
Berlin	0.048	0.14	0.99	0.106	0.064
Houston	0.072	0.182	5.02	0.182	0.08

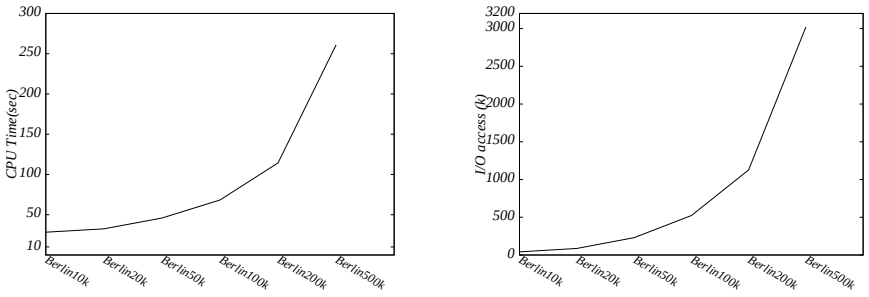
(c) CPU Time (sec)

Dataset	BQ1	BQ2	BQ3	BQ4-B	BQ4-M
Berlin	73	179	1582	6	17
Houston	101	240	11157	87	37

(d) I/O Accesses

Figure 5.4: Infrastructure Query Cost

5.4.3.2 Scaling Experiments



(a) CPU Time (b) I/O Accesses

Figure 5.5: Total Cost of Queries on Moving Objects

This part investigates the system performance when the data size increases. We choose the infrastructure data of Berlin and generate different numbers of trips, from Berlin10k to Berlin500k. Fig. 5.5 depicts the cost where the result is the sum over all queries. When the input size increases, both CPU time and I/O accesses rise proportionally. The detailed value of each query is reported in Fig. 5.6.

5.4.3.3 Optimization Techniques

We show in Fig. 5.7 the extra cost of creating the integers on transportation modes and building the indices on moving objects. The additional storage size is calculated by summarizing the value of all trips. Each trip includes two storage parts: (1) an integer built on transportation modes; (2) an index built on units. Such a formula $\sum_{i=1}^{N}(b+\sum_{j=1}^{|I_j|} 3*b)$ is used to compute the total storage cost where N refers to the number of trips in total, $|I_j|$ denotes the number of elements in the index for each trip and b shows the number of bytes for an integer. In the system, we use b = 4 bytes.

We use the symbols TO and T to denote the time cost for a query with and without optimization, respectively, and symbols AO and A refer to the number of I/O accesses for a query with and without optimization. Fig. 5.8 and Fig. 5.9 depict the query cost of Berlin and Houston, respectively. The result shows that the CPU time is reduced by optimization techniques for almost all queries, and some queries are significantly improved, e.g., **BQ6**, **BQ15**. There is one query (**BQ7**) that does not need optimization techniques so that the results are nearly the same for two cases. In a few occasions, the number of I/O accesses by AO is a little more than A (e.g., **BQ8**). But in most cases, optimization techniques still show the effect and advantage, demonstrated in both figures.

In both datasets, by optimization the CPU time is around 10 seconds for most queries. **BQ21** takes the most among all queries as a majority of time is spent on mapping bus routes into road segments, up to 80%. We put the accurate cost value for each query in Fig. 5.10 and Fig. 5.11.

Query	Berlin10k	Berlin20k	Berlin50k	Berlin100k	Berlin200k	Berlin500k
BQ5-B	0.2	0.3	0.5	0.9	1.8	4.2
BQ5-M	0.1	0.2	0.5	0.9	1.7	4.2
BQ6	0.1	0.3	0.7	1.4	2.7	7.0
BQ7	0.2	0.4	0.8	1.5	2.9	7.3
BQ8	0.2	0.3	0.7	1.3	2.5	6.6
BQ9	0.7	0.8	1.1	1.6	2.6	5.5
BQ10	0.8	1.0	1.7	3.0	5.4	12.7
BQ11	0.6	1.2	3.2	6.5	14.3	40.9
BQ12	0.7	0.9	1.3	2.0	3.4	7.5
BQ13	1.4	1.6	2.0	2.7	4.2	8.4
BQ14	0.4	0.6	1.2	2.1	4.1	10.0
BQ15	0.3	0.5	1.0	2.1	4.3	11.3
BQ16	0.2	0.4	0.8	1.5	3.0	8.0
BQ17-B	0.2	0.4	0.9	1.9	3.8	10.1
BQ17-M	0.2	0.3	0.8	1.6	3.2	8.4
BQ18-B	0.7	0.9	1.6	2.8	5.0	12.1
BQ18-M	0.7	1.2	2.4	4.4	8.7	21.3
BQ19	0.1	0.3	0.8	2.3	4.4	10.5
BQ20-B	0.4	0.7	1.4	2.6	5.0	12.7
BQ20-M	0.3	0.5	1.0	1.9	3.8	9.5
BQ21	19.7	20.0	21.5	23.4	27.8	43.0
Total	28.3	32.5	45.9	68.3	114.4	261.0

(a) CPU Time (sec)

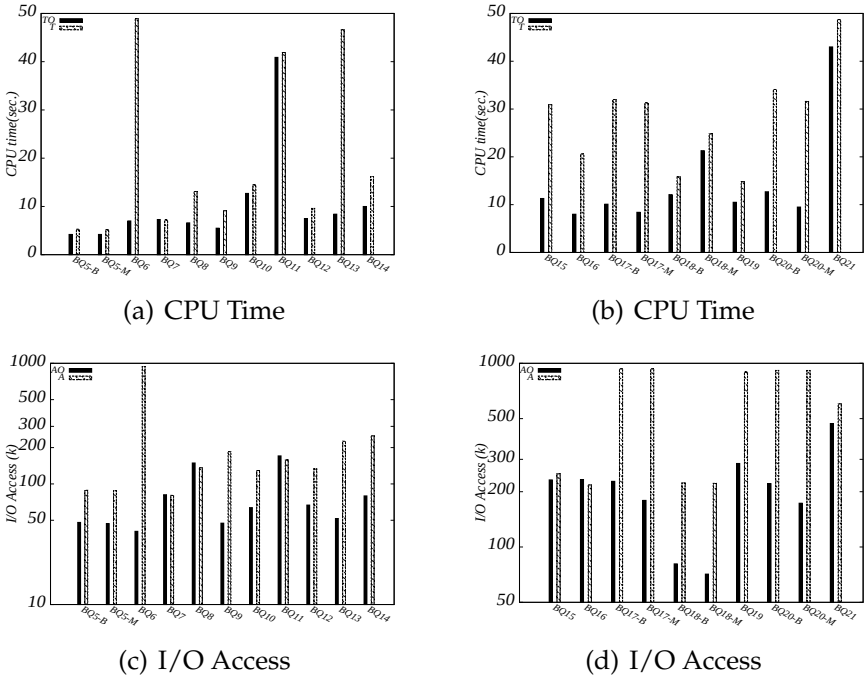
Query	Berlin10k	Berlin20k	Berlin50k	Berlin100k	Berlin200k	Berlin500k
BQ5-B	2.0	3.0	5.8	8.4	19.9	48.0
BQ5-M	0.4	0.5	0.9	1.5	7.4	46.9
BQ6	0.001	0.001	0.001	0.001	0.5	40.5
BQ7	0.03	0.03	0.3	0.3	0.08	81.4
BQ8	2.1	4.0	10.0	20.4	54.6	149.3
BQ9	1.6	1.7	2.1	8.4	19.4	47.3
BQ10	1.5	2.0	3.5	8.6	23.8	63.6
BQ11	2.6	5.3	13.3	27.9	64.0	170.8
BQ12	1.0	1.6	5.2	14.0	27.8	66.8
BQ13	1.4	1.6	2.7	6.4	18.8	51.7
BQ14	0.3	0.6	1.5	9.1	24.2	79.6
BQ15	1.5	3.2	10.1	35.4	90.4	232.0
BQ16	3.9	7.8	21.5	46.8	93.0	232.6
BQ17-B	1.8	5.3	17.7	45.3	90.7	227.8
BQ17-M	2.1	4.7	15.4	35.9	72.1	179.4
BQ18-B	1.0	1.9	5.1	16.1	32.9	81.0
BQ18-M	1.8	2.0	2.1	9.8	28.7	71.2
BQ19	4.9	9.9	24.9	51.8	109.6	285.1
BQ20-B	0.1	6.1	22.0	45.0	88.8	221.4
BQ20-M	0.03	4.4	15.2	34.8	69.7	173.2
BQ21	1.2	21.2	50.2	97.2	190.6	470.5
Total	41.8	86.8	229.1	522.5	1126.7	3020.2

(b) I/O Accesses(k)

Figure 5.6: Scaling Evaluation Results

	Additional Storage Size (M)	Creation Time (sec)
Berlin500k	93.62	86.39
Houston500k	89.38	88.28

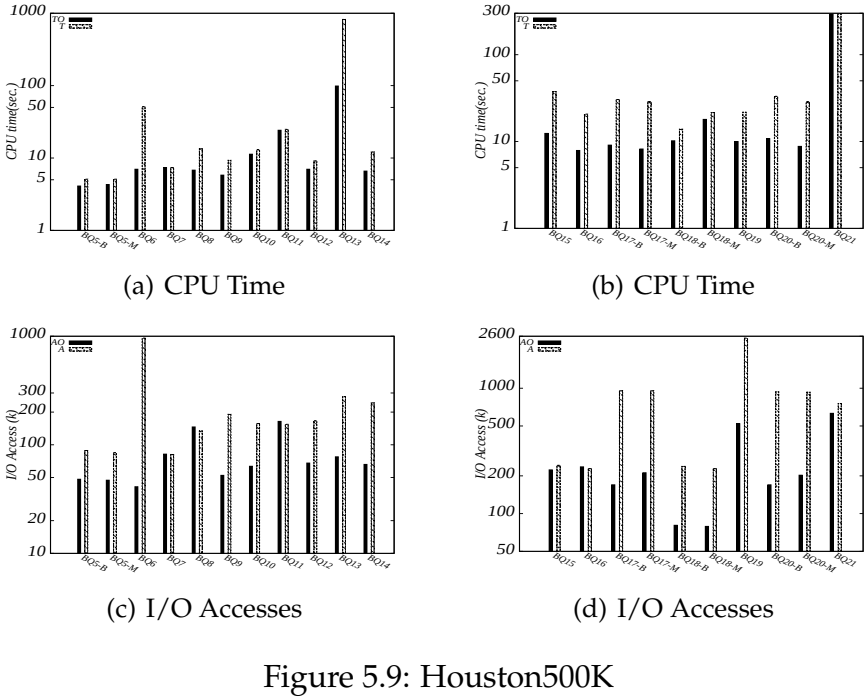
Figure 5.7: Indices Cost



(a) CPU Time (b) CPU Time

(c) I/O Access (d) I/O Access

Figure 5.8: Berlin500K



(a) CPU Time (b) CPU Time

(c) I/O Accesses (d) I/O Accesses

Figure 5.9: Houston500K

Query	TO	T
BQ5-B	4.2	5.3
BQ5-M	4.2	5.2
BQ6	7.0	48.9
BQ7	7.3	7.2
BQ8	6.6	13.1
BQ9	5.5	9.1
BQ10	12.7	14.5
BQ11	40.9	41.9
BQ12	7.5	9.6
BQ13	8.4	46.6
BQ14	10.0	16.2

(a) CPU Time (sec)

Query	TO	T
BQ15	11.3	30.9
BQ16	8.0	20.6
BQ17-B	10.1	32.0
BQ17-M	8.4	31.3
BQ18-B	12.1	15.8
BQ18-M	21.3	24.8
BQ19	10.5	14.8
BQ20-B	12.7	34.1
BQ20-M	9.5	31.6
BQ21	43.0	48.7

(b) CPU Time (sec)

Query	AO	A
BQ5-B	48.0	88.6
BQ5-M	46.9	87.8
BQ6	40.5	940.9
BQ7	81.4	80.4
BQ8	149.2	136.1
BQ9	47.3	185.2
BQ10	63.6	129.1
BQ11	170.8	157.8
BQ12	66.8	133.3
BQ13	51.7	226.1
BQ14	79.6	250.3

(c) I/O Accesses (k)

Query	AO	A
BQ15	232.0	250.2
BQ16	232.6	217.7
BQ17-B	227.8	933.0
BQ17-M	179.4	933.0
BQ18-B	81.0	223.0
BQ18-M	71.2	222.1
BQ19	285.1	897.2
BQ20-B	221.4	915.2
BQ20-M	173.2	914.3
BQ21	470.5	601.5

(d) I/O Accesses (k)

Figure 5.10: Results of Berlin500K

Query	TO	T
BQ5-B	4.1	5.1
BQ5-M	4.3	5.1
BQ6	7.0	51.2
BQ7	7.4	7.4
BQ8	6.8	13.4
BQ9	5.8	9.3
BQ10	11.3	13.0
BQ11	24.2	24.9
BQ12	7.0	9.1
BQ13	98.8	822.8
BQ14	6.6	12.1

(a) CPU Time (sec)

Query	TO	T
BQ15	12.4	37.7
BQ16	7.9	20.7
BQ17-B	9.1	30.5
BQ17-M	8.2	28.8
BQ18-B	10.2	13.9
BQ18-M	18.0	21.5
BQ19	10.0	21.8
BQ20-B	10.8	33.2
BQ20-M	8.8	28.7
BQ21	293.3	295.2

(b) CPU Time (sec)

Query	AO	A
BQ5-B	48.2	88.7
BQ5-M	47.2	84.5
BQ6	41.1	961.7
BQ7	82.1	81.5
BQ8	146.0	134.1
BQ9	52.3	190.9
BQ10	63.4	156.7
BQ11	164.4	154.5
BQ12	67.9	166.2
BQ13	77.5	277.1
BQ14	66.0	243.2

(c) I/O Accesses (k)

Query	AO	A
BQ15	223.3	243.1
BQ16	236.1	228.6
BQ17-B	169.8	953.3
BQ17-M	211.8	953.3
BQ18-B	80.7	238.2
BQ18-M	79.1	228.4
BQ19	522.2	2514.0
BQ20-B	169.8	936.6
BQ20-M	202.5	932.4
BQ21	631.3	756.6

(d) I/O Accesses (k)

Figure 5.11: Results of Houston500K

5.5 Conclusions

We propose a benchmark to evaluate the performance of a database system managing moving objects that travel through different environments. A method is developed to create the benchmark data in a realistic scenario with the aim of reflecting the distribution and characteristics of human movement in practice. A group of queries on infrastructure data and moving objects is set as the benchmark workload. Optimization techniques are proposed to reduce the query cost. The efficiency and effectiveness of the proposed techniques are studied through comprehensive experiments and the detailed experimental results are reported.

Chapter 6

Conclusions

6.1 Summary

The area of moving objects with multiple transportation modes becomes an interesting topic recently, due to its new applications such as investigating habits and situations of individuals and communities, web applications based on GPS. To fully understand the mobility, besides the location data, the underlying environments and precise transportation modes have to be recognized. These pieces of information denote key characteristics of human behavior. Since real world environments have different features such as free and constraint movement, 2D and 3D, a general location representation is designed in order to apply to all cases. In this dissertation, we investigate the following fundamental problems of managing and querying generic moving objects in a database system: (1) data model and data management; (2) data generator and query algorithms; (3) benchmark.

The thesis proposes a method that represents the location of a moving object by referencing to the underlying environments, specifically to geographic objects. The method is general and can be applied to all real world environments, including road network, region-based outdoor, bus network, metro network and indoor. Due to the unique characteristic of each environment, different data types are proposed to represent geographic objects in each case. There are not only static objects such as roads and pavements, but also dynamic objects like buses and metros. One important advantage of the referencing representation is that the storage size can be greatly reduced. A framework for moving objects is proposed for all available environments and we present how to apply the framework for each sub trip with a particular transportation mode. To efficiently support query processing, a set of operators is designed to access the data including both moving objects and geographic objects. We formally define the syntax and semantics of each operator and show running examples by formulating a group of interesting queries.

The data model is implemented in SECONDO which is an extensible database system and supports new technology to be embedded in a database system. All data types are developed,

including those for generic locations and moving objects as well as for geographic objects. An interface is provided to exchange the information between proposed data types and a relational environment. Queries can be expressed by an SQL-like language where the defined operators are registered as functions in DBMS.

A tool is developed to create all environments based on roads and floor plans, both of which have a lot of public resources, enabling the tool to be flexible. Other researchers can use the tool to create their own data. In the system, each environment is generalized to be a set object that contains all its elements and indices. All created objects are managed by the database system. Based on the created environments, we build the space on top of them with the aim of managing interactions between different environments and performing the location mapping. In essence, the space can be considered as an interface between moving objects and the low level geographic objects. In order to provide trip plannings, a graph as well as the algorithm is designed within each environment. Taking the indoor case as an example, the algorithm supports the shortest path with respect to different costs such as distance and time. A complete navigation algorithm is also developed to find a route passing through different environments. The results of trip plannings are used to produce generic moving objects.

A benchmark is proposed to evaluate the performance of the database system managing generic moving objects. To have realistic and comprehensive benchmark data, we propose some rules to generate moving objects with precise transportation modes with the goal of simulating human behavior in practice. Parameters can be configured by different settings in order to create the data according to different requirements. Two groups of queries constitute the benchmark workload, infrastructure data queries and moving objects queries. To efficiently support query processing, optimization techniques are developed to reduce the query cost in terms of the CPU time and I/O accesses. The experimental results confirm the effectiveness and efficiency of the proposed techniques.

6.2 Future Directions

One important future work is to investigate the on-line update condition as currently we deal with the past movement, assuming that there is no update for infrastructure data (e.g., construction of new roads and buildings). Since these moving objects contain transportation modes and travel through several environments, one can do data mining to find interesting movement patterns and provide advanced routing recommendations. Another application is to study the social network from the historical movement such as discovering correlations and relationships among people (e.g.,friends) because the data contain the semantic information of the mobility.

Appendix A

Operator Semantics

We extend the definition of operator **geodata** and give domains and ranges in Table A.1. **geodata** which plays a crucial rule for defining semantics, maps locations in different infrastructures into free space. The first three signatures are for PTN. Given a line and the relative position on the line, **geodata** returns a point for that position. Given a point (line) inside a region, we obtain its global position. The last one maps an indoor location to free space. This is done by ignoring the height above the ground level, i.e., projecting a room to the ground floor of a building. To have the symbol set denoting data types supported by **geodata**, we define IOSymbol' = {LINE, REGION, BUSROUTE, GROOM } ($\subset$ IOSymbol).

geodata	$\underline{busroute} \rightarrow \underline{line}$
	$\underline{busroute} \times \underline{busstop} \rightarrow \underline{point}$
	$\underline{busroute} \times \underline{busloc} \rightarrow \underline{point}$
	$\underline{line} \times \underline{real} \rightarrow \underline{point}$
	For $\alpha \in \{\underline{point}, \underline{line}\}$
	$\underline{region} \times \alpha \rightarrow \alpha$
	$\underline{groom} \times \alpha \rightarrow \alpha$

Table A.1: Extension for **geodata**

Definition 1 **=:** $\underline{genloc} \times \underline{genloc} \rightarrow \underline{bool}$

Let u, v be the two locations and the result is TRUE iff

(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.i_{loc} = v.i_{loc}$*; or*

(ii) $u.oid = v.oid \wedge u.i_{loc} = v.i_{loc}$*; or*

(iii) $u.oid = \perp \wedge (\exists\, w \in Space$*:*

$w.oid = v.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, u.i_{loc} =$ ***geodata*** $(w.\beta, v.i_{loc}))$*; or*

(iv) $v.oid = \perp \wedge (\exists w \in Space$*:*

$w.oid = u.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, v.i_{loc} =$ ***geodata*** $(w.\beta, u.i_{loc}))$

Cases (i) and (ii) are straightforward as both locations are in free space or in the same IFOB. In case (iii) and (iv), if one location is in free space and the other belongs to another infrastructure, the latter maps to free space by loading the referenced IFOB.

Definition 2 ***inside:*** $\underline{genloc} \times \underline{genrange} \rightarrow \underline{bool}$

Let u be a single location and V be a set of locations. The result is TRUE iff $\exists v \in V$ such that

(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.i_{loc} \in v.l$; *or*

(ii) $u.oid = v.oid \wedge (u.i_{loc} \in v.l \vee$
$(\exists w \in Space$: $v.oid = w.oid \wedge$
$\textbf{\textit{geodata}}(w.\beta, u.i_{loc}) \in \textbf{\textit{geodata}}(w.\beta, v.l)))$; *or*

(iii) $u.oid = \perp \wedge (\exists w \in Space$:
$v.oid = w.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, u.i_{loc} \in \textbf{\textit{geodata}}(w.\beta, v.l))$; *or*

(iv) $v.oid = \perp \wedge (\exists w \in Space$:
$u.oid = w.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, \textbf{\textit{geodata}}(w.\beta, u.i_{loc}) \in v.l)$

Case (i) shows a single location and a set of locations in free space. If u, v reference to the same object w (case (ii)), one can directly compare the location inside w or load the referenced object to transform from the relative location to the global. Similarly, if one parameter corresponds to free space and the other does not, we need to perform the location mapping by the referenced object. These are cases (iii) and (iv).

Definition 3 ***intersects:*** $\underline{genrange} \times \underline{genrange} \rightarrow \underline{bool}$

The result is TRUE iff $\exists u \in U, \exists v \in V$ such that

(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.l$ ***intersects*** $v.l$ *; or*

(ii) $u.oid = v.oid \wedge u.l$ ***intersects*** $v.l$ *; or*

(iii) $u.oid = \perp \wedge (\exists w \in Space$:
$v.oid = w.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, u.l$ ***intersects geodata*** $(w.\beta, v.l))$; *or*

(iv) $v.oid = \perp \wedge (\exists w \in Space$:
$u.oid = w.oid \wedge w.s \in$ *IOSymbol'* $\wedge\, v.l$ ***intersects geodata***$(w.\beta, u.l))$

Definition 4 ***distance:*** $\underline{genloc} \times \underline{genloc} \rightarrow \underline{real}$

$\textbf{\textit{length}}(\textbf{\textit{trajectory}}(\textbf{\textit{trip}}(u, v)))$

Definition 5 ***trajectory:*** $\underline{genmo} \rightarrow \underline{genrange}$

The result is a set of (oid, l, m) where $m = u_i.m$ and the values for oid and l are defined in the following.

(i) $u_i.oid = \perp \Rightarrow oid = \perp \wedge l = \cup f(t)(t \in u_i.i)$; *or*

(ii) $\exists w \in Space$: $u.oid = w.oid \wedge$ *(w.s = REGION* $\vee$ *w.s = GROOM), then*
$oid = u.oid$, $l = \cup f(t).i_{loc}(t \in u_i.i)$; *or*

(iii) $\exists w \in Space$: $u.oid = w.oid \wedge$ *(w.s = LINE* $\vee$ *w.s = BUSROUTE), then*
$oid = u.oid, l = \cup$ ***geodata***$(w.\beta, f(t).i_{loc})(t \in u_i.i)$; *or*

(iv) $\exists w_1, w_2 \in Space$: $u.oid = w_1.oid \wedge w_1.s$ = *MPPTN* $\wedge$ ***ref_id***(w_1) = $w_2.oid$, *then*
$oid = w_2.oid, l =$ ***trajectory(atperiods***$(w_1, u_i.i))$

The trajectory of a moving object is a set of movement projections, each of which shows the path of a unit $u_i \in mo$. The meaning for (i) and (ii) should be clear. In case (iii), the object moves along a pre-defined path, e.g., a road or a bus route. For case (iv), the location in u_i maps to a bus. Then, the trajectory is the bus movement, going to case (iii).

Definition 6 ***at:*** $\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$

Let $mo = < u_1, u_2, ..., u_n >$ *be the moving object and* v *denote the location.*

(i) $v.i_{loc} \neq \perp$, *the result is* $< u'_1, ..., u'_k >$ *where* $u'_i \in mo \wedge \forall t \in u'_i.i : f(t) = v$; *or*

(ii) $v.oid \neq \perp \wedge v.i_{loc} = \perp$, *then the result is* $< u'_1, ..., u'_k >$ *where* $u'_i \in mo \wedge$

(a) $u'_i.oid = v.oid$; *or*

(b) $u'_i.oid = \perp \wedge (\exists w \in Space$ *such that*
$w.oid = v.oid \wedge \forall t \in u'_i.i : f(t) \in w.\beta \wedge w.s \in$ *IOSymbol'*$)$

The meaning is clear if the second argument represents a precise location (case (i)). If the location only records an object id (case (ii)), then the units in the result can have the same reference id as the input, or the unit location maps to the place covered by the given IFOB.

Appendix B

Example Relations and Query Signatures

The following relations are used in the queries where each infrastructure relation is accessed by the **get_infra** operator.

```
MOGendon(Mo_id: int, Traj: genmo, Name: string)

Rel_Busstop(BusStopId: int, Stop: busstop, Name: string)
Rel_Busroute(BusRouteId: int, Route; busroute, Name: string,
               Up: bool)
Rel_Bus(BusId: int, BusTrip: mpptn, Name: string)
Rel_Room(RoomId: int, Room: groom, Name: string)
Rel_Door(DoorId: int, Door: door)
Rel_Roompath (RoomPathId: int, Door1: int, Door2: int, Weight: real,
               Room: groom, Name: string, Path: line)
Rel_Rbo(RegId: int, Reg: region, Name: string)
Rel_Road(RoadId: int, Road: line, Name: string)
```

In the following tables we show the signatures of operations used in the queries. Operator **get_infra** is omitted as it occurs in almost every query. We define subtype to mean that the operator performs a conversion from a specific type to a generic type.

No.	Operator	Signature		Def.
Q1				
Q2	**geodata**	$\underline{busroute} \times \underline{busstop}$	$\rightarrow \underline{point}$	Section 3.2.1.2
Q3	**atinstant**	$\underline{genmo} \times \underline{instant}$	$\rightarrow \underline{intime}(\underline{genloc})$	Table 3.6
	val	$\underline{intime}(\underline{genloc})$	$\rightarrow \underline{genloc}$	Table 3.6
Q4	**atperiods**	$\underline{genmo} \times \underline{periods}$	$\rightarrow \underline{genmo}$	Table 3.6
	val	$\underline{intime}(\underline{genloc})$	$\rightarrow \underline{genloc}$	Table 3.6
	=	$\underline{genloc} \times \underline{genloc}$	$\rightarrow \underline{bool}$	Table 3.7

No.	Operator	Signature		Def.
Q5	**atperiods**	$genmo \times periods$	$\rightarrow genmo$	Table 3.6
	trajectory	$genmo$	$\rightarrow genrange$	Table 3.8
	intersects	$genrange \times genrange$	$\rightarrow bool$	Table 3.7
Q6	**passes**	$genmo \times genrange$	$\rightarrow bool$	Table 3.8
	passes	$genmo \times genloc$	$\rightarrow bool$	Table 3.8
Q7	**get_mode**	$genmo$	$\rightarrow set(tm)$	Table 3.9
	contains	$set(tm) \times tm$	$\rightarrow bool$	Section 3.4.3
Q8	**at**	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
	deftime	$genmo$	$\rightarrow periods$	Table 3.6
	duration	$periods$	$\rightarrow real$	Table 3.6
Q9	**ref_id**	$mpptn$	$\rightarrow int$	Table 3.9
	at	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
	get_ref	$genmo$	$\rightarrow set(ioref)$	Table 3.9
	contains	$set(ioref) \times int$	$\rightarrow bool$	Section 3.4.3
Q10	subtype	$busstop$	$< genloc$	Table 3.10
	at	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
	at	$genmo \times genloc$	$\rightarrow genmo$	Table 3.8
	deftime	$genmo$	$\rightarrow periods$	Table 3.6
	duration	$periods$	$\rightarrow real$	Table 3.6
Q11	**contains**	$set(region) \times region$	$\rightarrow bool$	Section 3.4.3
	freespace	$groom$	$\rightarrow region$	Table 3.11
	contains	$set(string) \times string$	$\rightarrow bool$	Section 3.4.3
Q12	**freespace**	$mpptn$	$\rightarrow mpoint$	Table 3.11
	passes	$mpoint \times region$	$\rightarrow bool$	[38]
Q13	**ref_id**	$mpptn$	$\rightarrow int$	Table 3.9
	distance	$mpoint \times mpoint$	$\rightarrow mreal$	[38]
	freespace	$mpptn$	$\rightarrow mpoint$	Table 3.11
	freespace	$genmo$	$\rightarrow mpoint$	Table 3.11
	at	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
Q14	subtype	$busroute$	$< genrange$	Table 3.10
	subtype	$line$	$< genrange$	Table 3.10
	intersects	$genrange \times genrange$	$\rightarrow bool$	Table 3.7
Q15	**atperiods**	$genmo \times periods$	$\rightarrow genmo$	Table 3.6
	at	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
	get_ref	$genmo$	$\rightarrow set(ioref)$	Table 3.9
	contains	$set(ioref) \times int$	$\rightarrow bool$	Section 3.4.3
Q16	**at**	$genmo \times tm$	$\rightarrow genmo$	Table 3.9
	trajectory	$genmo$	$\rightarrow genrange$	Table 3.8

No.	Operator	Signature		Def.
Q17	subtype	*groom*	$<$ *genrange*	Table 3.10
	at	*genmo* $\times$*genrange*	$\rightarrow$*genmo*	Table 3.8
	deftime	*genmo*	$\rightarrow$*periods*	Table 3.6
	components	*periods*	$\rightarrow$*set*(*periods*)	Section 3.4.3
	duration	*periods*	$\rightarrow$*real*	Table 3.6
Q18	**ref_id**	*mpptn*	$\rightarrow$*int*	Table 3.9
	subtype	*busstop*	$<$ *genloc*	Table 3.10
	genloc	*int* $\times$*real* $\times$*real*	$\rightarrow$*genloc*	Table 3.11
	at	*genmo* $\times$*genloc*	$\rightarrow$*genmo*	Table 3.8
	initial	*genmo*	$\rightarrow$*intime*(*genloc*)	Table 3.6
	val	*intime*(*genloc*)	$\rightarrow$*genloc*	Table 3.6
	=	*genloc* $\times$*genloc*	$\rightarrow$*bool*	Table 3.7
Q19	subtype	*groom*	$<$ *genrange*	Table 3.10
	passes	*genmo* $\times$*genrange*	$\rightarrow$*bool*	Table 3.8
	subtype	*busstop*	$<$ *genloc*	Table 3.10
	atperiods	*genmo* $\times$*periods*	$\rightarrow$*genmo*	Table 3.6
	at	*genmo* $\times$*tm*	$\rightarrow$*genmo*	Table 3.9
	initial	*genmo*	$\rightarrow$*intime*(*genloc*)	Table 3.6
	final	*genmo*	$\rightarrow$*intime*(*genloc*)	Table 3.6
	val	*intime*(*genloc*)	$\rightarrow$*genloc*	Table 3.6
	=	*genloc* $\times$*genloc*	$\rightarrow$*bool*	Table 3.7
Q20	subtype	*gline*	$<$ *genrange*	Table 3.10
	atperiods	*genmo* $\times$*periods*	$\rightarrow$*genmo*	Table 3.8
	components	*periods*	$\rightarrow$*set*(*periods*)	Section 3.4.3
	final	*genmo*	$\rightarrow$*intime*(*genloc*)	Table 3.6
	val	*intime*(*genloc*)	$\rightarrow$*genloc*	Table 3.6
	at	*genmo* $\times$*tm*	$\rightarrow$*genmo*	Table 3.9
	inside	*genloc* $\times$*genrange*	$\rightarrow$*bool*	Table 3.7

Appendix C

Type System in Previous Data Models

	→ BASE	*int*, *real*, *string*, *bool*
	→ TIME	*instant*
BASE ∪ TIME	→ RANGE	*range*
	→ SPATIAL	*point*, *points*, *line*, *region*
	→ GRAPH	*gpoint*, *gline*
SPATIAL ∪ GRAPH	→ TEMPORAL	*moving*, *intime*

Table C.1: Data Types in [32, 38, 37]

Definition 7

UBool = $\{(i, b) | i \in D_{interval}, b \in D_{bool}\}$

D_{mbool} = {$< u_1, u_2, ..., u_n > | n \geq 0, n \in D_{int}$, *and* $\forall i \in [1, n], u_i \in$ *UBool* }

Definition 8

UPoint = $\{(i, p_1, p_2) | i \in D_{interval}, p_1, p_2 \in D_{point}\}$

D_{mpoint} = {$< u_1, u_2, ..., u_n > | n \geq 0, n \in D_{int}$, *and* $\forall i \in [1, n], u_i \in$ *UPoint*}

Definition 9

NLoc = $\{(rid, pos, side) | rid \in D_{int}, pos \in D_{real}, side \in D_{bool}\}$

D_{gpoint} = $\{(n\_id, gp) | n\_id \in D_{int}, gp \in$ *NLoc* $\}$

Definition 10

NReg = $\{(rid, pos_1, pos_2) | rid \in D_{int}, pos_1, pos_2 \in D_{real}\}$

D_{gline} = $\{(n\_id, gl) | n\_id \in D_{int}, gl \in$ *NReg* $\}$

Definition 11

UGPoint = $\{(i, gp_1, gp_2) | i \in D_{interval}, gp_1, gp_2 \in$ *NLoc and*

(i) $gp_1.rid = gp_2.rid$;

(ii) $gp_1.side = gp_2.side$ $\}$

$D_{mgpoint}$ = {$< u_1, u_2, ..., u_n > | n \geq 0, n \in D_{int}$, *and* $\forall i \in [1, n], u_i \in$ *UGPoint*}

Appendix D

Formulate Benchmark Queries

To express benchmark queries, we use the same relations in Section 3.3.2 of Chapter 3 and Appendix B. One infrastructure relation is added to represent a set of buildings. In order to retrieve such a relation by the operator **get_infra**, a notation BUILDING is defined to denote the type (similar to ROAD, BUSSTOP, BUS, etc., seeing Section 3.1.2.2 of Chapter 3). For operators that are used to formulate benchmark queries, some of them are already defined in Section 3.4 of Chapter 3 and the others are listed in Table D.1.

```
Rel_Building(B_id: int, GeoData: region, Name: string)
```

Name	Signature
intersects	$\underline{line} \times \underline{line} \rightarrow \underline{bool}$ $\underline{periods} \times \underline{periods} \rightarrow \underline{bool}$
distance	$\underline{region} \times \underline{point} \rightarrow \underline{real}$
length	$\underline{genrange} \rightarrow \underline{real}$
ref_id	$\underline{busstop} \rightarrow \underline{int}$
contains	$\underline{genmo} \times \underline{tm} \rightarrow \underline{bool}$ $\underline{genmo} \times \underline{int} \rightarrow \underline{bool}$
at	$\underline{genmo} \times \underline{point} \rightarrow \underline{genmo}$

Table D.1: Benchmark Operators

We use qt to denote the query time parameter. For queries on both bus and metro, it is sufficient to show the query expression by considering the bus as the procedure is similar for metros.

- **BQ2**. 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,
     get_infra(SpaceGendon, BUSROUTE) AS br
WHERE b.Name = X and br.BusRouteId = ref_id(bs.Stop) AND
      distance(geodata(br.Route, bs.Stop), b.GeoData) < 300
```

- **BQ14**. 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.Stop)) > 15
```

- **BQ15**. 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
```

- **BQ17**. 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
```

- **BQ18**. 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  ref_id(bus1.BusTrip) = A AND ref_id(bus2.BusTrip) = B AND
       bs.Name = X AND
       val(final(mo.Traj at genloc(bus1.BusId))) =
       val(initial(mo.Traj at genloc(bus2.BusId))) AND
       val(final(mo.Traj at genloc(bus1.BusId))) = bs.Stop
```

- **BQ19**. 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, OUTDOOR) AS R1,
    get_infra(SpaceGendon, OUTDOOR) 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
```

- **BQ20.** Find the top k bus (metro) routes with high passenger flow for all workdays.

```
SELECT TOP k *
FROM
  SELECT ref_id(bus.BusTrip), 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.BusId
  GROUP BY ref_id(bus.BusTrip)
  ORDER BY NO DESC
```

- **BQ21.** 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 road id for the segment. We decompose each road 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.BusRouteId, br.Route, COUNT(*) AS NO
    FROM get_infra(SpaceGendon, BUSROUTE) AS br,
         get_infra(SpaceGendon, BUS) AS bus
    WHERE deftime(bus.Traj) intersects qt AND
```

```
          ref_id(bus.BusTrip) = br.BusRouteId
    GROUP BY br.BusRouteId
```

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 geodata(br.Route) 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
```

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 $gl \in D_{\underline{genloc}}$, 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 B_Id and R_Id can be extracted from an indoor location. Using the example before, we can get B_Id = "167654" and R_Id = "123" from $gl.oid$ = "167654123".

We provide a table in the following to show the involved operators for benchmark queries.

No.	Operator	Signature	Def.
BQ2	**ref_id**	$\underline{busstop} \rightarrow \underline{int}$	Table D.1
	distance	$\underline{region} \times \underline{point} \rightarrow \underline{real}$	Table D.1
	geodata	$\underline{busroute} \times \underline{busstop} \rightarrow \underline{point}$	Section 3.2.1.2
BQ14	**duration**	$\underline{periods} \rightarrow \underline{real}$	Table 3.6
	deftime	$\underline{genmo} \rightarrow \underline{periods}$	Table 3.6
	atperiods	$\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$	Table 3.6
	at	$\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$	Table 3.8
	subtype	$\underline{busstop} < \underline{genloc}$	Table 3.10
BQ15	**atperiods**	$\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$	Table 3.6
	contains	$\underline{genmo} \times \underline{int} \rightarrow \underline{bool}$	Table D.1
BQ17	**duration**	$\underline{periods} \rightarrow \underline{real}$	Table 3.6
	deftime	$\underline{genmo} \rightarrow \underline{periods}$	Table 3.6
	at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	Table 3.9
BQ18	**ref_id**	$\underline{mpptn} \rightarrow \underline{int}$	Table 3.9
	val	$\underline{intime}(\underline{genloc}) \rightarrow \underline{genloc}$	Table 3.6
	initial	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$	Table 3.6
	final	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$	Table 3.6
	at	$\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$	Table 3.8
	=	$\underline{genloc} \times \underline{genloc} \rightarrow \underline{bool}$	Table 3.7
BQ19	**val**	$\underline{intime}(\underline{genloc}) \rightarrow \underline{genloc}$	Table 3.6
	final	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$	Table 3.6
	initial	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$	Table 3.6
	inside	$\underline{genloc} \times \underline{genrange} \rightarrow \underline{bool}$	Table 3.7
	contains	$\underline{genmo} \times \underline{tm} \rightarrow \underline{bool}$	Table D.1
	length	$\underline{genrange} \rightarrow \underline{real}$	Table D.1
	trajectory	$\underline{genmo} \rightarrow \underline{genrange}$	Table 3.8
	at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	Table 3.9
BQ20	**ref_id**	$\underline{mpptn} \rightarrow \underline{int}$	Table 3.9
	contains	$\underline{genmo} \times \underline{tm} \rightarrow \underline{bool}$	Table D.1
	atperiods	$\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$	Table 3.6
	at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	Table 3.9
	contains	$\underline{genmo} \times \underline{int} \rightarrow \underline{bool}$	Table D.1

No.	Operator	Signature	Def.
BQ21	**deftime**	$\underline{genmo} \rightarrow \underline{periods}$	Table 3.6
	intersects	$\underline{periods} \times \underline{periods} \rightarrow \underline{bool}$	Table D.1
	ref_id	$\underline{mpptn} \rightarrow \underline{int}$	Table 3.9
	intersects	$\underline{line} \times \underline{line} \rightarrow \underline{bool}$	Table 3.9
	geodata	$\underline{busroute} \rightarrow \underline{line}$	Table A.1
	trajectory	$\underline{genmo} \rightarrow \underline{genrange}$	Table 3.8
	at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	Table 3.9
	deftime	$\underline{genmo} \rightarrow \underline{periods}$	Table 3.6
	intersects	$\underline{genrange} \times \underline{genrange} \rightarrow \underline{bool}$	Table 3.7

Bibliography

[1] http://code.google.com/transit/spec/transit_feed_specification.html (2011).

[2] http://conversations.nokia.com/2008/09/23/indoor-positioning-coming-to-life (2012).

[3] http://research.microsoft.com/en-us/projects/geolife (2011).

[4] http://sumo.sourceforge.net (2012).

[5] http://www.bbbike.de/cgi-bin/bbbike.cgi (2009).

[6] http://www.bkrfloorplans.co.uk/bkr_services/cinemas.html (2011).

[7] http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html (2011).

[8] http://www.edenresort.com/home (2011).

[9] http://www.emaarmgf.com/mallwestdelhi/floor_plan.asp (2011).

[10] http://www.greenhosp.org/floor_plans.asp (2011).

[11] http://www.modulargenius.com/default.aspx (2011).

[12] http://www.nshispeed.nl/en/stations/station-maps-floor-plan (2011).

[13] L. O. Alvares, V. Bogorny, B. Kuijpers, J. Macedo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM GIS*, page 22, 2007.

[14] V. Balasubramanian, D. V. Kalashnikov, S. Mehrotra, and N. Venkatasubramanian. Efficient and scalable multi-geography route planning. In *EDBT*, pages 394–405, 2010.

[15] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks. In *ACM GIS, Demo*, page 78, 2008.

[16] C. Bennett, R. L. Grossman, D. Locke, J. Seidman, and S. Vejcik. Malstone: towards a benchmark for analytics on large data clouds. In *KDD*, pages 145–152, 2010.

[17] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing. How is the weather tomorrow?: towards a benchmark for the cloud. In *DBTest*, 2009.

[18] V. Bogorny, B. Kuijpers, and L. Alvares. ST-DMQL: A semantic trajectory data mining query language. *International Journal of Geographical Information Science*, 23(10):1245–1276, 2009.

[19] J. Booth, P. Sistla, O. Wolfson, and I. F. Cruz. A Data Model for Trip Planning in Multimodal Transportation Systems. In *EDBT*, pages 994–1005, 2009.

[20] S. Brakatsoulas, D. Pfoser, and N. Tryfona. Modeling, Storing and Mining Moving Object Databases. In *IDEAS*, pages 68–77, 2004.

[21] T. Brinkhoff. Generating network-based moving objects. In *SSDBM*, pages 253–255, 2000.

[22] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[23] X. Cao, G. Cong, and C. S. Jensen. Mining significant semantic locations from gps data. *PVLDB*, 3(1):1009–1020, 2010.

[24] M. J. Carey, D. J. Dewitt, and J. F. Naughton. The oo7 Benchmark. In *SIGMOD*, pages 12–21, 1993.

[25] S. Chen, C. S. Jensen, and D. Lin. A benchmark for evaluating moving object indexes. *PVLDB*, 1(2):1574–1585, 2008.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[27] V. Delis, T.Hadzliacos, and N.Tryfona. An Introduction to Layer Algebra. Technical Report. CTI-94.01.01, Compurter Technology Institute, University of Patras, Greece, 1994.

[28] Z. Ding and R. H. Güting. Managing Moving Objects on Dynamic Transportation Networks. In *SSDBM*, pages 287–296, 2004.

[29] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and Oranges: a comparison of rdf benchmarks and real rdf datasets. In *SIGMOD*, pages 145–156, 2011.

[30] C. Düntgen, T. Behr, and R. H. Güting. BerlinMOD: A benchmark for moving object databases. *VLDB Journal*, 18(6):1335–1368, 2009.

[31] V. Ercegovac, D. J. DeWitt, and R. Ramakrishnan. The texture benchmark: Measuring performance of text queries on a relational dbms. In *VLDB*, pages 313–324, 2005.

[32] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *SIGMOD*, pages 319–330, 2000.

[33] G. Gidófalvi and T. B. Pedersen. ST-ACTS: a spatio-temporal activity simulator. In *GIS*, pages 155–162, 2006.

[34] M. C. González, C. A. Hidalgo R., and A. Barabási. Understanding individual human mobility patterns. *Nature*, 453:779–282, 2008.

[35] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating interpolated data is easier than you thought. In *VLDB*, pages 156–165, 2000.

[36] R. H. Güting, V. Almedia, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, and M. Spiekermann. SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching. In *ICDE, Demo Paper*, pages 1115–1116, 2005.

[37] R. H. Güting, V. T. de Almeida, and Z. M. Ding. Modeling and Querying Moving Objects in Networks. *VLDB Journal*, 15(2):165–190, 2006.

[38] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM TODS*, 25(1):1–42, 2000.

[39] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[40] C. Hage, C. S. Jensen, T. B. Pedersen, L. Speicys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. In *VLDB*, pages 1019–1030, 2003.

[41] M. Held. FIST: Fast industrial-strength triangulation of polygons. *Algorithmica*, 30(4):563–596, 2001.

[42] H. Hu and D. L. Lee. GAMMA: A framework for moving object simulation. In *SSTD*, pages 37–54, 2005.

[43] M. Hua and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[44] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, pages 122–131, 2009.

[45] C. S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pages 208–227, 2009.

[46] C. S. Jensen, D. Tiesyte, and N. Tradisauskas. The cost benchmark-comparison and evaluation of spatio-temporal indexes. In *DASFAA*, pages 125–140, 2006.

[47] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, pages 70–79, 2008.

[48] H. Jeung, M. L. Yiu, X. Zhou, and C. S. Jensen. Path prediction and predictive range querying in road network databases. *VLDB Journal*, 19(4):585–602, 2010.

[49] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. *Discrete Comput. Geom.*, 18:377–383, 1997.

[50] M. Kim and D. Kotz. Periodic properties of user mobility and access-point popularity. *Personal and Ubiquitous Computing*, 11(6):465–479, 2007.

[51] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *Ubicomp*, pages 243–260, 2006.

[52] B. Kuijpers and W. Othman. Trajectory databases: Data models, uncertainty and complete query languages. In *ICDT*, pages 224–238, 2007.

[53] J. A. Lema, L. Forlizzi, R. H. Güting, and M. Schneider. Algorithms for moving objects databases. *The Computer Journal*, 46(6):680–712, 2003.

[54] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. H. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.

[55] L. Liao, D. Fox, and H. Kautz. Learning and Inferring Transportation Routines. In *AAAI*, pages 348–353, 2004.

[56] L. Liao, D. J. Patterson, D. Fox, and H. A. Kautz. Learning and inferring transportation routines. *Artif. Intell.*, 171(5-6):311–331, 2007.

[57] B. Lorenz, H. J. Ohlbach, and E. P. Stoffel. A hybrid spatial model for representing indoor environments. In *W2GIS*, pages 102–112, 2006.

[58] S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia. On the impact of user movement simulations in the evaluation of lbs privacy- preserving techniques. In *PiLBA*, 2008.

[59] C. Mouza and P. Rigaux. Mobility patterns. *GeoInformatica*, 9(4):297–319, 2005.

[60] C. Mouza, P. Rigaux, and M. Scholl. Efficient evaluation of parameterized pattern queries. In *CIKM*, pages 728–735, 2005.

[61] J. Myllymaki and J. H. Kaufman. Dynamark: A benchmark for dynamic spatial indexing. In *Mobile Data Management*, pages 92–105, 2003.

[62] A. Narkhede and D. Manocha. *Fast polygon triangulation based on Seidel's algorithm*. Graphics Gems V, Academic Press, 1995.

[63] A. D. Nguyen, P. Sénac, V. Ramiro, and M. Diaz. STEPS - an approach for human mobility modeling. In *Networking (1)*, pages 254–265, 2011.

[64] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high-level behavior from low-level sensors. In *Ubicomp*, pages 73–89, 2003.

[65] D. Pfoser and Y. Theodoridis. Generating semantics-based trajectories of moving objects. *Computers, Environment and Urban Systems*, 27(3):243–263, 2003.

[66] R. Praing and M. Schneider. Modeling Historical and Future Movements of Spatio-Temporal Objects in Moving Objects Databases. In *CIKM*, pages 183–192, 2007.

[67] R. Praing and M. Schneider. A universal abstract model for future movements of moving objects. In *AGILE Conf.*, pages 111–120, 2007.

[68] S. Ray, B. Simion, and A. D. Brown. Jackpine: A benchmark to evaluate spatial database performance. In *ICDE*, pages 1139–1150, 2011.

[69] S. Reddy, M. Mun, J. Burke, D. Estrin, M. H. Hansen, and M. B. Srivastava. Using mobile phones to determine transportation modes. *TOSN*, 6(2), 2010.

[70] J. M. Saglio and J. Moreira. Oporto: A realistic scenario generator for moving objects. *GeoInformatica*, 5(1):71–93, 2001.

[71] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.

[72] P. Scarponcini. Generalized model for linear referencing in transportation. *GeoInformatica*, 6(1):35–55, 2002.

[73] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL Performance Benchmark. In *ICDE*, pages 222–233, 2009.

[74] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991.

[75] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *VLDB Journal*, 17(4):765–787, 2008.

[76] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.

[77] S. Shekhar, M. Coyle, B. Goyal, D. R. Liu, and S. Sarkar. Data models in geographic information systems. In *Commun ACM 40*, volume 4, pages 103–111, 1997.

[78] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *ICDE*, pages 422–432, 1997.

[79] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data Knowledge Engineering*, 65:126–146, 2008.

[80] L. Speicys, C. S. Jensen, and A. Kligys. Computational Data Modeling for Network-Constrained Moving Objects. In *ACM-GIS*, pages 118–125, 2003.

[81] L. Stenneth, O. Wolfson, P. Yu, and B. Xu. Transportation Mode Detection using Mobile Devices and GIS Information. In *ACM SIGSPATIAL*, pages 54–63, 2011.

[82] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 Benchmark. In *SIGMOD*, pages 2–11, 1993.

[83] J. A. Storer and J. H. Reif. Shortest paths in the plane with polygonal obstacles. *Journal of the ACM*, 41(5):982–1012, 1994.

[84] J. Su, H. Xu, and O. H. Ibarra. Moving objects: Logical relationships and queries. In *SSTD*, pages 3–19, 2001.

[85] Y. C. Tay. Data Generation for Application-Specific Benchmarking. *PVLDB*, 4(12):1470–1473, 2011.

[86] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis. Constrained shortest path computation. In *SSTD*, pages 181–199, 2005.

[87] Y. Theodoridis. Ten benchmark database queries for location-based services. *Comput. J.*, 46(6):713–725, 2003.

[88] Y. Theodoridis and M. A. Nascimento. Generating spatiotemporal datasets on the www. *SIGMOD Record*, 29(3):39–43, 2000.

[89] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the generation of spatiotemporal datasets. In *SSD*, pages 147–164, 1999.

[90] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, pages 791–804, 2008.

[91] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. On the generation of time-evolving regional data. *GeoInformatica*, 6(3):207–231, 2002.

[92] M. Vazirgiannis and O. Wolfson. A Spatiotemporal Model and Language for Moving Objects on Road Networks. In *SSTD*, pages 20–35, 2001.

[93] A. Voisard and B. David. A Database Perspective On Geospatial Data Modeling. *TKDE*, 14(2):226–242, 2002.

[94] P. F. Werstein. A performance benchmark for spatiotemporal databases. In *10th Annual Colloquium of the Spatial Information Research Center*, pages 365–373, 1998.

[95] O. Wolfson, S. Chamberlain, K. Kalpakis, and Y. Yesha. Modeling moving objects for location based services. In *IMWS, Invited Paper*, pages 46–58, 2001.

[96] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *SSDBM*, pages 111–122, 1998.

[97] D. Xu, G. Song, P. Gao, R. Cao, X. Nie, and K. Xie. Transportation modes identification from mobile phone data using probabilistic models. In *ADMA (2)*, pages 359–371, 2011.

[98] J. Xu and R. H. Güting. Infrastructures for Research on Multimodal Moving Objects. In *MDM, Demo Paper*, pages 329–332, 2011.

[99] J. Xu and R. H. Güting. GMOBench: A Benchmark for Generic Moving Objects. Informatik-Report 362, Fernuniversität Hagen, 2012.

[100] J. Xu and R. H. Güting. MWGen: A Mini World Generator. In *MDM*, 2012.

[101] J. Xu and R.H. Güting. A Generic Data Model for Moving Objects. *Geoinformatica*, 2012.

[102] B. Yang, H. Lu, and C. S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pages 335–346, 2010.

[103] Y. Zheng, Y. Chen, X. Xie, and W. Y. Ma. Understanding transportation mode based on GPS data for Web application. *ACM Transaction on the Web*, 4(1):1–36, 2010.

[104] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Y. Ma. Understanding mobility based on GPS data. In *UbiComp*, pages 312–321, 2008.

[105] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning Transportation Mode from Raw GPS Data for Geographic Applications on the Web. In *WWW*, pages 247–256, 2008.

[106] Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W. Y. Ma. Recommending friends and locations based on individual location history. *TWEB*, 5(1):5, 2011.

Author's Biography

Personal Information

- Name: Jianqiu Xu
- Birthday: Oct.,29, 1982
- The place of birth: Nanjing, China

Education

- 9.2001 - 6.2005: B. S., College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics.

  The title of dissertation: The Modeling Techniques based on Vega (in Chinese)

- 9.2005 - 4.2008: M. E., College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics.

  The title of dissertation: Research and Development Technologies of Moving Objects in Networks (in Chinese)

Publications

- J. Xu and R.H. Güting. A Generic Data Model for Moving Objects, Geoinformatica, 2012.
- J. Xu and R.H. Güting. Manage and Query Generic Moving Objects in SECONDO, Demo, VLDB, To Appear, 2012.
- J. Xu and R.H. Güting. MWGen: A Mini World Generator, MDM, 2012.
- J. Xu. Research on Moving Objects with Multimodal Transportation Modes, SIGMOD/-POSD IDAR, 2011.
- J. Xu and R.H. Güting. Infrastructures for Research on Multimodal Moving Objects, MDM, Demo, pages 329-332, 2011.
- R.H. Güting, T. Behr, and J. Xu. Efficient k-Nearest Neighbor Search on Moving Object Trajectories. The VLDB Journal 19:5 (2010), 687-714.

- R.H. Güting, A. Braese, T. Behr, and J. Xu. Nearest Neighbor Search on Moving Object Trajectories in Secondo. SSTD, Demo, pages 427-431, 2009.

Technical Reports

- J. Xu and R.H Güting. GMOBench: A Benchmark for Generic Moving Objects, Informatik-Report 362, FernUni in Hagen, Germany, 2012.
- J. Xu and R.H Güting. Visible Points Searching in Large Obstructed Space, Technical Report, FernUni in Hagen, Germany, 2012.