MWGen: A Mini World Generator

Jianqiu Xu, Ralf Hartmut Güting

Database Systems for New Applications, Mathematics and Computer Science
FernUniversität Hagen, Germany
{jianqiu.xu,rhg}@fernuni-hagen.de

February 20, 2012

Abstract

GMOD (Generic Moving Objects Database) is a database system that manages moving objects traveling through different environments and with multiple transportation modes, like $Walk \rightarrow Car \rightarrow Indoor$, as humans' movement can cover several different environments (e.g., road network, indoor) instead of a single environment. To evaluate the performance of GMOD, a comprehensive and scalable dataset consisting of all available environments (e.g., roads, bus network, buildings) and moving objects with multiple modes is essentially needed, where the location of a moving object is represented by *referencing* to the underlying environment. Due to the difficulty of gaining real datasets, in this paper we present a tool that creates the overall space, which is composed of the following environments: road network, bus network, metro network, pavement areas and indoor. Each environment is also called an infrastructure. All outdoor infrastructures are produced from a real road dataset and the indoor environment consisting of a set of buildings is generated from public floor plans. Within each infrastructure, we design a graph as well as the algorithm for trip plannings, like indoor navigation, routing in bus network. The time complexity of the algorithm is also analyzed. A complete navigation system through all environments is developed, which is used to guide data generation for moving objects covering all available environments. The generated data, including all infrastructures and moving objects, is managed by GMOD. We report the experimental results of the data generator by conducting experiments on two real road datasets and a set of public floor plans.

1 Introduction

Recently, researchers start to explore the area of moving objects with different transportation modes [39, 12, 38, 27] as humans' movement can cover several different environments, basically *outdoor* and *indoor*. The outdoor movement could be further classified into different categories according to the transportation mode, like walking in the pavement, driving along the road, or taking a bus. At the same time, people also spend a large amount of time inside buildings. This leads to taking into account transportation modes becoming increasingly essential for moving objects. Although moving objects databases have been extensively studied in the last decade, the existing work is only limited within a single environment (e.g., free space [16], road network [18] and indoor [21, 20]). GMOD (Generic Moving Objects Database) is a database system that manages the complete movement passing through different environments, for example, walk from home to the parking lot, drive the car along the road, and then walk to the office room. Such a system needs to be adequately tested before deployment and the testing needs to use a scalable and well-defined dataset, real or synthetic. The well known benchmark TPC is domain-specific, which is increasingly irrelevant to the multitude of new data-centric applications, so that one should develop tools for application-specific benchmarking [33]. In this paper, we develop a generator, called MWGen (Mini World Generator), to create synthetic databases for generic moving objects where the result will be used for a GMOD benchmark.

In benchmarking and DBMS testing, it is common that comprehensive real data is often hard to come by, which leads researchers to develop their own data generators for specific tasks [14, 11, 10]. Specially, in the filed

of moving objects databases there are few published real large datasets as most people do not want to publish their own movement. Also, real data might not be comprehensive enough to thoroughly evaluate the system in consideration. Consequently, researchers design data generators [36, 28, 13, 26] to produce varied in size and representative datasets in a realistic simulated scenario. [15] provides a benchmark for moving objects in free space. Indoor moving objects [21, 37] are created using floor plans by some predefined movement rules. However, the datasets produced from existing generators only record the movement in a single environment and without transportation modes. They cannot be used for representing the complete movement for humans, which can cover distinct environments (outdoor and indoor), like $Indoor \rightarrow Bus \rightarrow Walk$. It is also difficult to get the real data of moving objects with different transportation modes. Meanwhile, the datasets include not only generic moving objects, but also the underlying referenced infrastructure objects. There is a large amount of such objects, roads, pavements, buses, rooms, etc. These data are heterogeneous and not easy to get all of them in real and a consistent environment. This motivates the need to (1) develop a data generator capable of generating realistic datasets including infrastructure data and moving objects (2) effectively and efficiently manage them in a GMOD.

Our data generator is based on a generic data model for moving objects, proposed in [22], where the *reference* method is used to represent the locations of moving objects in all available environments. The idea of reference method is to let the *space* where moving objects are located consist of a set of so-called *infrastructures* and then the location of a moving object is represented by referencing to the infrastructure. We partition the whole space into five infrastructures: *Road Network, Region-based Outdoor, Bus Network, Metro Network and Indoor*. Based on a real road dataset, we propose a method to create all the other outdoor infrastructures as there are many public resources which can be utilized. The indoor environment (a set of buildings) is produced from public floor plans (e.g., [8, 5]). [23] demonstrates the result of generated infrastructures. Within each infrastructure, a graph is designed for trip plannings, like finding an optimal route for pedestrians and indoor navigation. The global space, managed by GMOD, is built on all infrastructures and handles the interaction places between different environments where transportation modes can change (for example, bus stops are located in the pavement areas), yielding the system be capable of providing an optimal route passing through distinct places, e.g., buildings, pavements, roads. The result of route planning, a path through different environments, is used to create generic moving objects fusion of transportation modes. Note that moving objects in a single environment can also be created by the generator, e.g., indoor moving objects, moving objects in networks. Some applications of the generator are listed as below.

Benchmarking and DBMS testing. One goal of this paper is towards a benchmark for GMOD. To evaluate the performance of DBMSs, besides a comprehensive dataset a wide range of meaningful queries is also required. [22] formulates a group of queries on infrastructures and generic moving objects where three examples are given below:

(1) *Did anyone who was at the University on floor H-2 between 4:30pm and 5:00pm take a bus to the main (train) station?*

(2) *Who arrived by taxi at the university in December?*

(3) *At 8:00am on Monday, who sits in the same bus as Bobby.*

Our generator only needs roads and floor plans as input to create all the other infrastructures where both of them have a lot of public resources, enabling other researchers to create their own world by MWGen.

Route Planning and Travel Recommendation. We can offer a trip passing through several environments, while most existing work [25, 29, 19] are limited to one environment. Consider such a query, "*tell me how to reach my apartment from my office with minimum traveling time*". The environments include indoor, road network, pavement areas, 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. Also, 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, we can offer the shortest path for pedestrians, and inside a building one can get a precise indoor shortest path between two locations.

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 public transportation system.

The contribution of the paper is summarized as follows. We develop a generator MWGen to create a set of real world infrastructures (both outdoor and indoor), which are components for the global *space*. In detail, outdoor infrastructures are constructed from real roads and indoor environment is built on floor plans. Afterwards, a global *space* is constructed managing all infrastructures and interactions between them. The space and all infrastructure data are managed by GMOD. Within each infrastructure, a graph model is proposed for trip plannings so that one can search the optimal route. We analyze the time complexity of each algorithm. A complete navigation algorithm through all available environments is also developed to generate moving objects with multiple transportation modes. A detailed experimental evaluation is reported including the statistics of created databases, efficiency of trip planning algorithms and the scalability of creating large amounts of moving objects.

The rest of the paper is organized as follows: Related work is reviewed in Section 2. Section 3 presents the global procedure of data generation and an overview of datasets. Section 4 introduces the data generation procedure for each infrastructure. Trip planning is discussed in Section 5 and generic moving objects is generated in Section 6. The experimental result is reported in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

In the literature, there are a lot of generators for moving objects data [36, 35, 28, 13, 26, 15], but all of them consider moving objects in one environment such as free space or road network. GSTD [36], a widely used spatio-temporal generator, defines a set of parameters to control the generated trajectory: (1) the duration of an object instances; (2) the shift of objects and (3) the resizing of objects. Later, the generator is extended in [26] to produce more realistic moving behaviors such as group movement and obstructed movement, by introducing the notion of clustered movement and a new parameter. [35] generates a set of moving points or rectangles in accordance with user requirements, thus providing a tool that could simulate a variety of possible scenarios. [15] proposed a tool for generating moving objects where the datasets are generated based on trip plannings, e.g., from home regions to work regions. They create cars moving in Berlin streets and record the complete histories of movements. There is the long-term observation (a month) yielding huge histories of moving objects. In real life, objects usually move only on a pre-defined set of lines as specified by the underlying network (roads, highways, etc.). [13] developed a generator for objects moving in networks where objects are created randomly and the speed and route of a moving object depend on the load of network edges, i.e., the number of cars on it. SUMO [1] is a microscopic traffic simulator, which performs collision free vehicle movement with such features like multi-lane streets, different right-of-way rules, traffic lights. Besides outdoor movement, people also spend a lot of time inside buildings. [21, 37] define some 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 or move in the staircase. However, all these data are limited in one environment, which are not feasible for generic moving objects covering all available environments.

GPS data-based activity recognition has received considerable attention recently, especially detecting transportation modes [38, 27, 31]. The work of Microsoft *GeoLife*'s project [39, 38] focus on discovering and learning transportation modes from raw GPS data to enrich the mobility with informative and context knowledge. Mobile phones can be used to detect transportation modes when outside [27, 31]. [27] creates a classification system that uses a mobile phone with a built-in GPS receiver and an accelerometer to determine the transportation mode of an individual when outside. [31] proposed an approach to inferring a user's mode of transportation based on the GPS sensor on the mobile device as well as the knowledge of the underlying transportation network, e.g., bus stop locations, railway lines. However, their work are different from ours. We do not discover transportation modes

but create generic moving objects seamlessly integrated with transportation modes. In addition, they only take into consideration *outdoor* movement because they infer based on GPS data where a GPS receiver will lose signal indoors, while our data includes both outdoor and indoor.

Most existing work on trip plannings [25, 34, 29] are limited in one surrounding, free space or road network. There is no interaction between different environments and transportation mode is not considered. A graph model presented in [12] can provide a path connecting an origin and destination including multiple transportation modes so that the *shortest_path* has more constraints and choices, like different motion modes, number of transfers, etc. They assume the trips with transportation modes are already obtained and managed by the database system so that queries can be issued. And the movement they consider is only *outdoor*. Besides, there is a representation limitation of the graph where the vertices only represent a set of interesting places, e.g., train stations, grocery stores. This yields that query locations can only be located at these places. But a pedestrian walking along a street may issue a query like "*tell me how to get the nearest bus stop*" and his position can be anywhere inside the footpath area, but the graph does not represent all these places. Our method does not have the limitation where the location can be in any defined infrastructures.

3 Framework

In this section, we present the framework of the mini-world generator. First, we give the reference representation in Sec. 3.1. Then, we outline the data generation process in Sec. 3.2. Sec. 3.3 discusses how to physically store and manage infrastructure objects in GMOD as well as the *space*, and Sec. 3.4 summarizes the data that we create.

3.1 Reference Representation

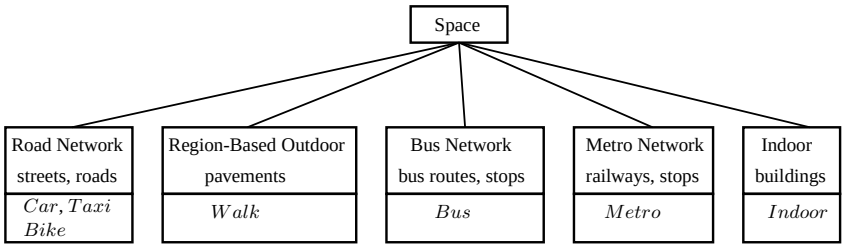

Figure 1: Sketch the Space

We consider moving objects in all available environments where each environment is called an infrastructure (in the following we use terms environment and infrastructure interchangeably). The overall space, depicted by Figure 1, consists of all available infrastructures, each of which contains a set of infrastructure objects representing available places for moving objects and is coupled with its possible transportation modes. For example, *RN* contains a set of roads and streets, and *RBO* includes a set of pavement areas. For each infrastructure, we define a notation, listed in Table 1. The transportation modes are summarized in Def. 3.1.

I_{RN}	Road Network
I_{RBO}	Region-based Outdoor
I_{BN}	Bus Network
I_{MN}	Metro Network
I_{Indoor}	Indoor

Table 1: Infrastructure Notations

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

As people also walk in the indoor space, in the following when we say mode $Walk$ it means outdoor by default, which is to distinguish the modes betweeen $Walk$ and $Indoor$. We employ the *reference* method and represent the location of a moving object via an object identifier corresponding to an *infrastructure object* as well as the relative position in that object. The location definition is given below.

Definition 3.2 *Generic Location*
$D_{\underline{genloc}} = \{(oid, i_{loc}) | oid \in D_{\underline{int}}, i_{loc} \in Loc\}$
$\overline{Loc} = \{(loc_1, loc_2) | loc_1, loc_2 \in D_{\underline{real}}\}$

Based on Def. 3.2, now we give the representation of moving objects. A generic moving object $genmo = < u_1, u_2, ..., u_n >$ is a sequence of temporal units ordered by time, each of which defines the movement during a time interval. In detail, we define $u_i = (i, oid, i_{loc_1}, i_{loc_2}, m)$ where i denotes the time interval, oid is the referenced infrastructure object identifier, i_{loc_1} and i_{loc_2} record the start and end locations according to the referenced object during i, finally comes the transportation mode m.

3.2 Data Generation Procedure

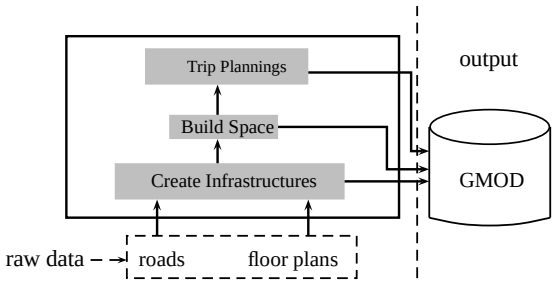

Figure 2: MWGen Workflow

Figure 2 shows the overview procedure of MWGen, which consists of three phases: (1) infrastructures generation; (2) construct the space; (3) trip plannings. We elaborate each phase below.

- The program takes reading of roads and floor plans, and generates all infrastructures. We use the real dataset for road network and create all the other outdoor environments. Roads are stored in a file with the nested list format as a relation with the schema ***Road_Rel*** $(Id : int, Type : int, Geodata : line)$. Each tuple represents a road where the value $Type$ says whether the road is a main street, a side road, or else, which will be used for determining the speed value on the street (later used to create moving objects). I_{RBO} represents the walking areas 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. Each building consists of a set of rooms and doors. All created infrastructure objects plus roads are managed by GMOD.

- Afterwards, a global space is built on top of all infrastructures. Basically, the space handles an reference representation for each infrastructure, loads infrastructure objects if they are needed for query processing and regulates the interaction or connection parts between different infrastructures. The space managed by GMOD functions as a communciation level between underlying infrastructure objects and the high level of query processing and moving objects.

- Trip plannings are executed on the space where the start and end locations can be in any created infrastructure. A trip can cover one environment or several environments, where the latter produces a moving object with multiple transportation modes. The result of this phase is a set of generic moving objects, stored in GMOD.

We give more comments about the space, which is essentially important for GMOD and in fact it builds the connection between underlying infrastructure objects and moving objects. It has two functionalities: (1) **an interface**. As stated before, moving objects are generated based on the result of trip plannings. During query processing, an infrastructure is loaded for the underlying information if the trip covers that part but is not necessarily considered if the path is not through that environment. For example, for indoor queries only I_{Indoor} has to be loaded. Each infrastructure is represented by an reference record in space where the concreate data (e.g., roads, bus routes) is stored in the low level infrastructure and only loaded if the query requests. (2) **interaction**. Elements from different infrastructures have overlappings places and one may compare or manipulate objects from different infrastructures. For example, "*which streets does bus route 512 pass by*", or "*what is the road network distance between two bus stops*". In this case, objects from different environments have to be mapped into the same system, which is done by the global space.

There is an order of creating these infrastructures as some of them have the dependence on the others. First, I_{RN} has to be created because the other outdoor infrastructures I_{RBO}, I_{BN} and I_{MN} are generated based on roads, but there is no specific order among them. Second, I_{Indoor} is generated, which has to be processed after I_{RBO} because the 2D outdoor areas for buildings are extracted from the places not covered by I_{RN} and I_{RBO}. Regarding all of them, we take the order $I_{RN} \rightarrow I_{RBO} \rightarrow I_{BN} \rightarrow I_{MN} \rightarrow I_{Indoor}$ in the paper.

3.3 Data Storage and Management

Infrastructure Storage. Each infrastructure consists of a set of infrastructure objects and different data types are used to represent these objects in accordance with the infrastructure characteristic. For instance, in *RN* a line value is used to describe the geometrical property of a road, and in *RBO* it turns out that polygons are to identify pavement areas for people walking. The overall definitions for these data types refer to [22] and all data types are supported by GMOD. A set of symbols is defined for infrastructures.

Definition 3.3 *Infrastructure Symbols*
I_Symbols={*ROAD, PAVEMENT, BUS, METRO, INDOOR*}

In general, an infrastructure in GMOD can be denoted by a four-tuple
$I_i = (type, OS, IS, G)$ $(type \in I\_Symbols)$
where OS is the object set storing all infrastructure objects in this environment, IS is an index set for efficiently accessing objects, e.g., b-trees, and G denotes the graph for trip plannings in this infrastructure. OS is a set of relations where an infrastructure object is a tuple and the data type representing the object is embedded as an attribute. Each infrastructure object has a unique object id and the id ranges for different infrastructures are disjoint. For example, [1,5000] is for roads, [5001,6500] is for bus routes and moving buses. We explain more concretely the context of OS, IS and G for each infrastructure in the following.

- I_{RN}: The route and junction relations constitute $I_{RN}.OS$, and $I_{RN}.IS$ contains two b-trees (e.g., one built on route id and the other on junction id) and an r-tree built on roads. In $I_{RN}.G$, used for shortest path searching in road network, a node corresponds to a network location denoted by $(rid, (pos, \perp))$ where rid is a route identifier and pos records the relative position along that route. An edge is created if two locations (1) have the same spatial position in space but from different routes (a junction) or (2) are *adjacent* on the same route.

- I_{RBO}: A set of polygons are stored in $I_{RBO}.OS$ to cover the area for people walking. A b-tree is built on polygon id and an r-tree is built on polygon bounding boxes. The graph in I_{RBO} is used to find a shortest path connecting two locations inside the walking area, that is trip planning for pedestrians.

- I_{BN} and I_{MN}: For I_{BN} (I_{MN}), infrastructure objects are routes, stops, moving vehicles, each of them are stored in a relation where a tuple corresponds to a route, a stop or a moving bus (metro). A btree is built on each relation where the key value is route id. For routes and stops relations, the geographical data is 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 its 2D outdoor area, building id and the type (e.g., office building, university, hotel). Each building (a tuple in $Root\_rel$) corresponds to a *sub* relation Sub_rel storing all rooms and doors in that building. Several tuples in $Root\_rel$ can correspond to one Sub_rel if the buildings they indicate have the same floor plan, implying the same internal structure but located at different outdoor places. Two indices are built on $Root\_rel$, a b-tree on building id and an r-tree on the bounding box of the building outdoor area. For each Sub_rel, a b-tree is created where the key is room id. $I_{Indoor}.G$ consists of a set of graphs, each of which corresponds to a *sub* relation for providing indoor navigation in the building. Each type of building has its own floor plan, yielding distinct indoor structures and graphs.

For each infrastructure, we have an r-tree index, which is to efficiently process interaction queries on geographical objects from different infrastructures. Consider the following examples: (1) A pedestrian located inside I_{RBO} wants to find the nearest bus stop; (2) Given a bus route, we can find which roads it passes; (3) For a building, one can find all bus stops within 300 meters.

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

Ref=$\{(ref, id\_l, id\_h) |\ ref \in I\_Symbols, id\_l \leq id\_h, id\_l, id\_h \in D_{\underline{int}}\}$

is a list of elements, each of which has a label referencing to an infrastructure and the min, max objects ids. When an infrastructure is created, an element is inserted into Ref. Ref can be fully loaded or not depending on the application requirement. For example, if only road network is considered, then $Ref = \{(Road, id\_l_r, id\_h_r)\}$. This equals to a single environment. In this paper, we let the space be full where all infrastructures are considered. id_l, id_h define the objects id range for each infrastructure. Thus, given a location $gl \in D_{\underline{genloc}}$, the environment it belonging to can be known by investigating $gl.oid$ 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 one can leave one enviroment and enter another. LS is a location set for managing interactions between different environments and the places where the transportation mode can switch. See below.

- bus (metro) stops and pavements: Each stop is mapped to a position in the pavement so that it is available to search the nearest stop for a pedestrian. And the shortest path from the query location to the stop is located in I_{RBO}. Reversely, the traveler terminates the movement by bus or metro at the stop position and walks to the target place (e.g., a shop) in the pavement.

- pavement locations and road network locations: Given a location in I_{RBO}, it can be mapped to a network location, and vice verse. This is to build the connection between I_{RN} and I_{RBO}. Consider such a movement: *Bobby drives the car, parks the car on one side of the road, and then walks to a shop*. We need to be able to know the place where the mode switches between *Car* and *Walk*. Different from the mapping between bus (metro) stops and pavements, there is an infinite set of locations for I_{RN} and I_{RBO}, so this conversion is done on-the-fly.

- entrances of buildings and pavement locations: This is to build the connection between indoor and outdoor movement. All entrances/exits of buildings are mapped to a set of locations inside pavement areas, thus from the entrance one can search its nearest bus or metro stop. Similarly, from the bus or metro stop one can find the path to a building. Also from the building entrance, one may walk to the place where the car

is parked. Here, there are two mappings: (1) indoor and outdoor; (2) pavement locations and road network locations.

Moving Objects Storage. Sec. 3.1 gives the representation of generic moving objects, which is implemented by a data type called *genmo*. For a set of moving objects, a relation interface is provided where the type *genmo* is embedded as an attribute and a tuple corresponds to a moving object, thus leveraging the full power of relational operators, e.g., select.

3.4 An Overview of Dataset

The whole dataset consists of two parts: infrastructures and moving objects. The infrastructures include the following five parts.

- I_{RN}: roads, streets;
- I_{RBO}: footpaths, zebra crossings;
- I_{BN}: bus routes, bus stops and buses;
- I_{MN}: metro routes, metro stops and metros;
- I_{Indoor}: universities, hotels, hospitals, etc., and personal houses.

For personal buildings, only 2D outdoor areas are managed due to lack of information about the internal structure and the privacy problem. Based on the created full space, several types of generic moving objects are created. We focus on moving objects in multiple environments instead of a single environment. In fact, the movement with a single mode is covered by generic moving objects. Each type of movement includes the mode *Walk*, as according to the study in [38], the walk segment is an important part which builds the connection between segments with different transportation modes. Usually people do not directly switch from car to bus or metro. The following are the types of generic moving objects.

- *Car + Walk* [1];

  This is to simulate the case that people walk from a building to the parking lots, drive the car to another place and then walk to the destination. Here is an example movement.

  - *Bobby walks from home to the parking lots, drives the car along the road to the parking place, and then walks from the parking place to his office building.*

- *Bus + Walk*; *Metro + Walk*

  Instead of traveling by car, people may also choose public transportation system. Assuming the traveler takes the bus, consider such a movement.

  - *Bobby walks from home to a bus stop, takes the bus (without or with transfers) to a bus stop close to his office building, and then walks from the bus stop to the building.*

- *Indoor + Walk*;

  Indoor + Car + Walk;

  Indoor + Bus + Walk; *Indoor + Metro + Walk*

  These movements are to combine outdoor and indoor movement as people usually start and end their trips inside buildings. We give examples for each of them.

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

– *Bobby moves from his office room to the exit of the building, walks along the pavement to McDonald's or a bread shop to buy some food for lunch, and then comes back.*
– *Bobby moves from his office room to the exit of the building, walks to the parking lot and drives the car home.*
– *Bobby moves from his office room to the exit of the building, walks to a bus stop and takes the bus to a stop close to his apartment, finally walks from the bus stop to his home.*

4 Infrastructure and Space Generation

4.1 Pavement Areas

This infrastructure is for people walking where the places include pavement areas located on both or one side of roads, and zebra crossings allowing people to cross the street. We create both areas based on roads and a defined road width w (e.g., five meters). The procedure is as follows. Each road is extended by w from a polyline to a region r, more precisely the shape of r is a stripe. We let r be the area for cars moving instead of a line. Then, we enlarge the width by $\triangle w$ (e.g., two meters) to create a larger region lr including the areas for both cars and pavements. Afterwards, the pavement can be achieved by lr minus r. Zebra crossings, usually located around the junctions by two intersecting roads, are generated at each junction by creating some rectangles. Finally, we perform the union on all created pavements and zebro crossings to get a relatively large polygon P with many holes inside, denoting the whole area for people walking. For efficient query processing, P is decomposed into a set of triangles. And these triangles are stored in the database.

4.2 Bus Network

The public bus transportation system consists of bus routes, bus stops and moving buses. The whole bus routes can be considered as a subset of roads. A bus route is created by (1) picking up road network points as route start and end locations; (2) computing the shortest path between them and setting it as the result. For each bus route, a sequence of bus stops is generated by defining the distance between two *adjacent* stops. A time schedule is also defined for creating bus trips. Different bus routes have different time schedules depending on the traffic value. To get the traffic value of each route, we first create a set of cars moving on the streets and record the traffic flow for each road by aggregating the number of cars passing it during a time interval. Then, we map the bus route to its corresponding road and get the traffic flow of it. The routes with high traffic value are also used as night bus routes. In the real world, one bus route has two directions and for each direction there are a sequence of bus stops. To be realistic, our bus routes and stops are also created based on this behavior.

4.3 Metro Network

A metro network is created based on road network and the procedure has three steps.

- The road network space is partitioned into a set of equal size cells. Each cell is represented by a square and the square width is set by a value l, deciding the distance between two *adjacent* metro stops. 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*.
- Each 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) can be found by searching the dual graph, which connects the start and end locations and is represented by a sequence of cells. We sequentially pick up the center point of each cell from the start to the end 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.

- 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 by every ten minutes and the movement starts from 6am and ends at 10pm.

4.4 Indoor

In a city, there is a large number of buildings with different types, e.g., office buildings, shopping malls, houses, each of which occupies a certain area in the outdoor space. Taking out the areas covered by roads and pavements, we get the places for buildings where a rectangle represents the bounding box of each building. There are public buildings and personal houses. Table 2 lists all created public buildings, which try to simulate a city environment. For each rectangle, we set its building type as follows. Most rectangles in the city center denote office buildings, shopping malls, cinemas. Rectangles located out of the city center are for personal houses and universities, surrounded by hospitals.

For each public buliding, we create the internal structure based on the floor plan. 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 one level is created, we translate the level in vertical to get more floors to construct a building if the building has the same structure at each level. Otherwise, levels with different structures are created separately and then built together to form a building. The floor plan of a university is from the authors' affiliation. [23] demonstrates the building in a 3D visualization viewer. The internal structure for personal houses is not considered due to the difficulty of getting the data.

Buildings	**Resource**	**Buildings**	**Resource**
officeA	[8]	hotel	[5]
officeB	[8]	hospital	[7]
shopping mall	[6]	university	
cinema	[3]	railway station	[9]

Table 2: Public Floor Plans

We denote a building by a five-tuple $B(id, type, bbox, R, G)$ ($id, type \in D_{\underline{int}}$, $bbox \in D_{\underline{region}}$) where each building has a unique id, a value describing its characteristic (e.g., a hotel or cinema) and a bounding box for the outdoor area. All rooms of a building are stored in a set R and an indoor graph G is built for indoor navigation.

A building consists of rooms, corridors, staircases, elevators, etc. To be more general, we use the term *groom* (general room) for all of them. Basically, a groom is a set of objects, each of which has a polygon denoting the 2D area and a value for the height above the ground level. A groom can have one or several doors, represented by lines. The raw data (from the floor plan) for creating a building is a relation stored in nested list format where each tuple corresponds to a groom. The relation schema is given as follows.

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

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 to guarantee that the selected start and end locations for shortest path computing are at reasonable places. An indoor location is represented by $(oid, (x, y))$ (apply Def. 3.2) where oid is a groom identifier and (x, y) shows the relative location inside the groom. In the real world, it is meaningful to find a path from one office room to another where one may pass through corridors, use staircases or elevators, but it seldomly happens that people move from one bath room to another (of course the path can be computed).

4.5 Space

Let $s(Ref, LS)$ denote a space to be created, initially empty. Whenever an infrastructure is generated, an element (ref, id_l, id_h) representing the infrastructure label and objects id range is inserted into $s.Ref$. So, s is progres-

sively filled up by distinct infrastructures and only maintains an reference pointing to the underlying environment. For efficiently locating the enviornment for a moving object, s also manages the id range for each infrastructure. After all infrastructures are generated, one needs to handle the interaction places between different enviornments in order to be capable of providing a trip passing through multiple infrastructures. A set of locations $s.LS$ as well as the location mapping technology is defined in the space structure. As a result, the interaction positions 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 switching areas between region-based outdoor and the other infrastructures, which is consistent with the data in Sec.3.4 where the mode $Walk$ is included by all moving objects.

5 Trip Plannings

5.1 Shortest Path for Pedestrians

Trip planning for pedestrians is to find a shortest path between two locations inside P, created in Sec. 4.1. In computational geometry, there are some main-memory algorithms on path problems in the presence of obstacles [32, 24]. But now we have to process a relatively big data, as P is the whole area of outdoor walking in a city with hundreds of thousands of vertices, yielding the main-memory algorithm not practical. Also, the technique in computational geometry has a limitation that assumes the number of holes is in a small number, which is not feasible for such an application. Each obstacle in P represents a building block or junction area, and there are a large number of such objects inside a city. Thus, an efficient and practical method is essentially needed.

Our solution is 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 an 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, which is 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.

The searching procedure has two steps. We describe them briefly as this paper focuses on the overall data generation instead of a specific query algorithm. (1) Using the general location representation (Def. 3.2), 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) is the relative location inside the triangle. Given two arbitrary locations $p_1(tri\_id, (loc_1, loc_2))$ and $p_2(tri\_id, (loc_1, loc_2))$, the triangles that they are located in can be found in a constant time. We let all triangles be ordered by tri_id, yielding the ability to fast access them by the id entry. (2) All *visible* points to p_1 and p_2 can be found by searching DG, which is to build a bridge between p_1, p_2 and VG. p_1, p_2 can be located anywhere in P, and if they do not belong to the vertices of P, p_1, p_2 are not represented by VG nodes. After connecting p_1, p_2 to VG, Dijkstra algorithm can be run on VG and A^* algorithm can also be used where the heuristic value is just the Euclidean distance between the vertex popped from priority queue and the destination. We give the algorithm in Alg. 1 (see Appendix).

5.2 Routing in Bus Network

A bus network graph is defined for routing where two kinds of trips are supported: (1) minimum cost time; (2) minimum number of transfers. In the presentation, we assume the trip with minimum time is considered. Before introducing the graph, we first give the definition of a bus stop. A bus stop is defined by a three-tuple $bs_i(br\_id, stop\_pos, d)$ ($br\_id \in D_{\underline{int}}$, $stop\_pos \in D_{tcreal}$, $d \in D_{\underline{bool}}$) where br_id indicates the bus route *id*, $stop\_id$ records the relative position on the route and d represents the direction (one route has two directions: *Up* and *Down*).

Let **geodata** be a function returning the spatial point of a bus stop by taking a bus stop and the route as input. For the easy of presentation, the route parameter is omitted, that is **geodata**(bs_i). Bus stops belonging to different routes can have the same spatial location, but they are treated as different bus stops. By the technology in Sec. 3.3, a bus stop can be mapped to a point in the pavement. Then, applying the algorithm of trip planning for pedestrians the walking distance between two stops can be computed, denoted by **dist**(bs_i, bs_j). We define Δd be the value denoting the length of a short walk path, e.g., 150 meters. Two relationships are defined for bus stops: *neighbor* and *adjacent*. Given two bus stops bs_i and bs_j,

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

Definition 5.1 *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) denoting the connection from v_i to 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.*

Two bus 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 the *neighbor* of bs_j;
(iii) bs_i is *adjacent* to bs_j.

The three conditions represent different connections where (i) shows a transfer without movement cost; (ii) represents a short distance walking for changing the bus stop (e.g., cross the street to take the bus in another direction); and (iii) is the movement by a bus trip. Each edge is associated with a value l recording the path from bs_i to bs_j. We explain the path and time cost in detail for each kind of connection. case (i): l is empty and $w_{i,j}(t) = 0$. case (ii): l is the shortest path in the pavement 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 time schedule of the route and arrival time at bs_i.

The query result of bus routing is a sequence of pairs (pl, m), each of which indicates the connection between two sequent stops on the path. pl is a polyline specifying the movement path, and m decides the transportation mode for such a movement where the value is from domain $\{None, Walk, Bus\}$, each of which corresponds to one connection between two stops. To improve the efficiency of query processing, two optimization techniques are developed.

- We run Dijkstra's algorithm on G_{BN} to find a shortest path. A heuristic value can be set for optimization. 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 above three connections between two nodes, let $TP(v_i, v_j)$ return the type for 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 in space, 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 the value $TP(v_{i-1}, v_i)$, some connections can be pruned so that the number of nodes to be expanded can be reduced. That is, the value $TP(v_{i-1}, v_i)$ determines $TP(v_i, v_{i+1})$. We give the result for each case.

Lemma 5.1 *Expanding Nodes 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 access v_{i-1}. In case (2), nodes with the same spatial location as v_i are pruned and no further walk movement has to be considered from v_i. We give the reason in the following. 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 program also does not have to process the case $TY(v_i, v_{i+1}) = NEI$. First, the total length of two walking movements ((v_{i-1}, v_i), (v_i, v_{i+1})) could be larger than Δd, which contradicts the defined *neighbor* condition of two bus stops. Second, if the length fulfills the 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, then all connections have to be dealed with. Alg. 2 (in Appendix) illustrates the pseudo-code of the algorithm.

5.3 Routing in Metro Network

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

Definition 5.2 *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 $ms_i(mr\_id, stop\_pos, d)$ ($mr\_id \in D_{\underline{int}}$, $stop\_pos \in D_{\underline{real}}$, $d \in D_{\underline{bool}}$) where mr_id indicates the route *id*, $stop\_pos$ records the relative location on the route and d represents the direction (*Up* or *Down*). We define two metro stops ms_i, ms_j are *adjacent* by

ms_i is *adjacent* to $ms_j \Leftrightarrow ms_i.mr\_id = ms_j.mr\_id \wedge ms_i.d = ms_j.d \wedge ms_i.stop\_id + 1 = ms_j.stop\_id$.

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* to ms_j. Different from G_{BN}, we do not define two metro stops are linked by a path in the pavement because in practice usually people do not have to go to another location for a metro transfer. During 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 procedure is similar as bus network, and the pseudo-code of the algorithm is given in Appendix.

5.4 Indoor Navigation

We construct an indoor graph for searching the shortest path inside a building.

Definition 5.3 *An indoor graph $G_{Indoor}(V, E)$ is defined for a building where V is a set of nodes representing all doors ; $E \subseteq V \times V$ is a set of edges, each of which denotes a shortest path connecting two doors where the doors are inside one groom.*

Each door is represented by a line and we take the middle point for path computation. The indoor path is computed in an obstructed space instead of Euclidean space as there can be obstacles (e.g., walls) inside a room. To build the graph, one needs to pre-compute the paths between each pair of doors inside one groom. As the number of doors for a groom is usually not large, the time cost of such computation is affordable. An office room may have one or two doors, while a corridor can own several doors. And this process has only to be done once. For the nodes in G_{Indoor}, besides the real doors existing in a building we also create *virtual doors* to represent the places that connect two grooms but no explicit doors exist. This is to make a robust indoor graph. For example, a staircase builds the connection between two different floors, but maybe no doors exist for entering and leaving the staircase. Note that a staircase is modeled as a groom (see Sec. 4.4). So, we create a virtual door between the staircase and the floor. The virtual door is created by the program but does not exist in the real world.

An indoor location is represented by $(oid, (loc_1, loc_2))$ where oid is a groom identifier and (loc_1, loc_2) is the relative location inside the groom. $(oid, (loc_1, loc_2))$ can be located anywhere inside the building where oid can map to an office room, a corridor, even a staircase. Two kinds of shortest paths are developed: 1) minimum length; 2) minimum traveling time. The algorithm with minimum distance is given in the appendix. For the second, we define a speed value for indoor walking to compute the time of moving along the indoor path. A building may have elevators where the time cost includes waiting and moving. We implement the elevator as a public transportation vehicle with a regular and periodic movement. Some optimization techniques are developed to improve query processing.

- Given two indoor locations i_s and i_e, the heights above the ground level can be retrieved, which are recorded by the groom they are located. If i_s and i_e are at different levels, the height of all doors on the shortest path should be between the height of the two levels. Then, all doors whose height is out of the range can be pruned during searching.
- Before searching on G_{Indoor}, one first has to connect the start and end locations to all doors of their located grooms. This may involve much computation if the room has several doors, for example, a corridor. This is because when a door is connected to the start location, all its adjacent doors will be processed for shortest path computation. But some doors are not necessarily to be connected if (1) the room via the door to enter has only one door and (2) the room is not the destination. We prune these doors during initialization.
- The Euclidean distance can be used as a heuristic value during searching, but it only benefits when two locations are at the same level. If they are at different levels, the optimal distance should be the length of such a path that is from the current location to the staircase or elevator.

5.5 Time Complexity Analysis

In this part, we study the complexity of our proposed algorithms for trip planning. As each algorithm is executed on searching on a graph, the time complexity is $O(NlogN + E)$ 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 for each graph.

- Region-based Outdoor VG: Let n be the number of vertices in P, which is also the number of nodes in VG. Before searching on VG, one first has to connect the start and end locations to VG. Searching visible points for a query location needs $O(n)$ [2]. Due to space limitation, we omit discussing in detail the procedure. 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. But for such an application where P represents the area for pedestrians with many obstacles inside, it is impossible that each vertex is visible to all the others. The obstacles (holes) in P are the places which people cannot directly pass through, e.g., building blocks, junction areas. So, we denote the size of E by $O(kn)$ $(k \ll n)$ where k is the maximum number of visible points for each vertex in P. In the end, the time complexity is $O(n + nlogn + kn)(k \ll n)$.
- Bus Graph G_{BN}: Assuming there are totally n bus stops in bus network, we need to determine the cardinality for bus stop connections. As stated in Sec.5.2, there are three cases to build an edge between two bus 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) fulfilling the condition $0 \leq c_1 \wedge c_1 \ll n$, $0 \leq c_2 \wedge c_2 \ll n$, and there is one connection to the *adjacent* stop. This yields $c_1 + c_2 + 1$ connections for a bus stop. In summary, the time cost is $O(nlogn + (c_1 + c_2 + 1)n)$ $(c_1, c_2 \ll n)$.

[2] the classical *rotational plane sweep algorithm* [30] needs $O(nlog)$ in an open space, but here we have a close area P resulting a new optimal searching algorithm.

- Metro Graph G_{MN}: Let n be the total number of metro stops, and we give the size for E. For each stop s, there is one connection from s to its adjacent one on the same route. At the same time, there can be c ($0 \leq c \wedge c \ll n$) stops from other routes having the same spatial location as s where a transfer can occur. These two are the only possibilities for a connection between two metro stops. So, the number of edges in G_{MN} is $E = n(1 + c)$ and the overall time complexity is $O(nlogn + n(1 + c))(c \ll n)$.

- Given an indoor graph with n nodes, each node represents a door and edges are connections between two doors in the same room. Let r be the maximum number of doors in one room, then the edge number is $O(rn)$. Now we consider the time on finding the path to all doors for the start and end locations inside their rooms. This equals to shortest path searching in a polgyon with holes where a polgyon represents the 2D area of a room and there can be obstacles. We apply the algorithm in Sec.5.1. Assume m be the maximum vertex number for a room polgyon. Building graphs costs $O(m^2)$ for both DG and VG, which is done on-the-fly for the reason that (1) m is usually small (2) much storage and maintenance effort are needed for each room. After that, searching a shortest path needs $\Theta(k'm)$ on VG where k' be the maximum number of visible points for a polygon vertex. Let d be the number of doors contained by the room resulting in $O(m^2 + dk'm)$ for connecting locations to G_{Indoor}. To sum up, the time complexity of indoor navigation is $\Theta(m^2 + dk'm + rn)$ $(forall\ \alpha \in \{m, d, k', r\}, \alpha \ll n)$.

We summarize the time cost of each algorithm in Table 3.

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	$\Theta(cn)$ $(c \ll n)$
Indoor	$\Theta(m^2 + dk'm + rn)$ $(forall\ \alpha \in \{m, d, k', r\}, \alpha \ll n)$

Table 3: Time Cost for Trip Planning Algorithms

6 Create Generic Moving Objects

To create a moving object, we need two parameters: (1) path recording where the object moves; (2) speed defining how the object moves. The path is from the result of trip plannings where the start and end locations are available in any infrastructure. The speed is determined by the environment.

6.1 Car + Walk

For moving objects with modes *Car + Walk*, the infrastructures needed are I_{RN} and I_{RBO}. We let the trip start and move a short distance in I_{RBO}, then travel along the road in I_{RN} and end in I_{RBO} again. The start location is a randomly selected point inside the pavement, denoted by p_1. From p_1, the trip moves a short distance to another randomly chosen point p_1' inside the pavement. The movement from p_1 to p_1' is the mode $Walk$. With the location mapping technique introduced in Sec.3.3, p_1' is mapped to a road network position rp_1. Then, from rp_1 the trip travels by car to another network position rp_2. rp_2 is mapped to a pavement location. In the end, the trip moves in I_{RBO} for some time and terminates. To get the complete path, trip planning for pedestrians is called twice for the two sub movements in the start and end, and searching shortest path in I_{RN} is also executed. The walking speed is defined as 1m/s and the speed for car is decided by the maximum speed allowed on the street. Recall that *Road_Rel* (in Sec. 3.2) has an attribute for the street type, and the maximum speed is set according to the type. For example, main street is 50km/h, side street is 30km/h.

6.2 Bus+Walk and Metro+Walk

Combine infrastructures I_{RBO} and I_{BN}, we are able to create moving objects withs modes $Bus + Walk$. The start and end locations for such a trip are two stochastic locations inside the pavement, denoted by p_1, p_2. First, the closest bus stop to p_1 is found, denoted by bs_1. Then bs_1 is mapped to a position in the pavement so that the shortest path from p_1 to bs_1 can be found applying Alg. 1. So, the trip walks from p_1 to bs_1. Afterwards, the bus trip algorithm is called to find a connection with minimum time from bs_1 to bs_2 where bs_2 is the closest bus stop to p_2. During this movement, a short distance walking may also be involved for a transfer. In the end, the traveler leaves bus network at bs_2 and moves from bs_2 to p_2 on foot.

We can also create a trip with modes $Metro + Walk$ where the start and end locations are randomly selected points located inside I_{RBO}. Then, the nearest metro stops to such two locations can be found. By searching on G_{MN}, an optimal route can be found to connect the two stops.

6.3 Indoor

An indoor moving object is created by a shortest path with minimum length connecting two indoor locations and two defined speed values: (1) walking; (2) lift. Using the indoor navigation algorithm (Alg. 4), a shortest path is returned where the start and end locations can be anywhere inside a building. The traveler may use the lift for moving, so a lift speed is also defined to create moving objects when the movement is by the lift. [23] demonstrates the animation of indoor moving objects in a 3D viewer.

6.4 Both Outdoor and Indoor

Combine the indoor and outdoor infrastructures, we are able to create a complete movement for humans, e.g., starting from home, moving outdoor (by car or bus), and then walk to the office room. For a trip including both outdoor and indoor, we let the start and end locations from two different buildings, say B_i, B_j, assuming that people start and end their trips inside buildings. But the generator also supports either start or end location is inside a building, while the other is an outdoor location. First, we consider the movement inside two buildings. Assuming the trip starts from B_i, the movement inside B_i is from a randomly selected location to the exit of B_i. The movement inside B_j is from the entrance of B_j to a random location in B_j. Now, we consider the outdoor movement. In Sec. 3.3, we state that the global space manages a set of places where the transportation mode switches, including entrances of buildings and pavement locations. A building exit/entrance is an indoor location and is mapped to a pavement location, building the connection between indoor and outdoor. We define a distance threshold value Δ_l where if the distance between B_i and B_j in space is smaller than Δ_l, the trip moves by walk for outdoor. Otherwise, the trip may use a car or public transportation system (we assume people tend to use vehicles for a long distance trip). The distance between B_j and B_j is set as the Euclidean distance between their bounding boxes. To get the outdoor movement, the procedures in above subsections are called according to the transportation modes involved.

The buildings in Table 2 are all public buildings, but there are also many personal houses and apartments where people often start and end their trips from these buildings (e.g., go to work from home and back). Due to be lack of floor plans for such buildings, the internal structures are not known. If a personal house or apartment is selected for B_i or B_j, the trip starts and ends at the exit/entrance of the building while the movement inside is ignored.

7 Evaluation

This section shows an extensive experimental results by running MWGen. Instead of implementing a stand-alone tool, the data generator is developed in an extensible database system Secondo [17] and programed by C/C++ and Java on a AMD 3.0 GHz with 4 GB of memory installing Suse Linux. We carry out experiments using two real road datasets for cities Berlin and Houston and public floor plans. The roads of Berlin download from

$BBBike$ [2] and Houston roads are from $Census\ Tiger/Line^{®}$ [4], 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 it by Gauss Krueger.

7.1 Infrastructure Data

Table 4 shows the infrastructure data that we create and the cost in terms of time and space. The whole buildings include those listed in Table 2 and personal houses. The time cost includes creating all infrastructures, global space generation, and graph construction in each infrastructure. The database includes all infrastructure data as well as the corresponding graphs for each infrastructure.

	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
Time Cost	63 min	170 min
Database Size	2.5 G	9.7 G

Table 4: An Overview of Infrastructure Data

The following table shows the data of all outdoor graphs where G_{RN} is the graph for road network. Table 6 summarizes the information for buildings, where No_B and No_H denote the number of such a type building in Berlin and Houston, respectively. Given a building B_i, let $|B_i.R|$ be the cardinality of rooms in B_i. For each public building, an indoor graph is created where the cardinality for nodes and edges is also given.

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 5: Statistics of Outdoor Graphs

7.2 Efficiency of Trip Plannings

We evaluate the efficiency of outdoor trip plannings, i.e., pedestrian routing, bus routing and metro routing. The result for pedestrian routing is averaged over 5,000 pairs of random start and end locations. For bus and metro routing, 5,000 random pairs of stops are generated and each pair is assigned a query time instant. The result is the average time over all runnings, each of which returns an optimal route with minimum time. The experimental results demonstrate the efficiency of our algorithms.

The time cost of indoor navigation is reported in Fig. 3(b) where the four most complex buildings are considered in terms of cardinality of graph nodes and edges. In each building, two random locations are generated

Type	No_B	No_H	$\lvert B_i.R \rvert$	Indoor Graph	
				Nodes No.	Edges No.
houses	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 6: Statistics of Buildings and Indoor Graphs

	RBO	BN	MN
Berlin	0.78	0.13	< 0.1
Houston	2.4	0.23	< 0.1

(a) Outdoor (sec)

Buildings	Time (sec)
officeA	0.25
shopping mall	0.37
hotel	1.57
university	0.123

(b) Indoor

Figure 3: Time Cost of Trip Plannings

to find the shortest path from one to another. We measure the execution time and average the result over running shortest path searching on 500 pairs of indoor locations. As a result, the time cost for a building is related to the complexity of its indoor graph in terms of cardinality of nodes and edges.

7.3 Generate Generic Moving Objects

This experiment shows the scalability of generating a variable size of generic moving objects, in terms of time and storage size. Section 3.4 presents all kinds of generic moving objects, while in the experiment we focus on the following three: (1) *Car + Walk + Indoor*; (2) *Bus + Walk + Indoor*; (3) *Metro + Walk + Indoor*. Each has both outdoor and indoor movement. In fact, the transportation modes of the others are covered by the three. We set the distribution by 2:1:1, assuming that the cardinality of people traveling by car is the same as the cardinality of people traveling by public transportation system. The start and end locations of a trip are defined in the indoor environment where the buildings can be public or private. The start and end buildings are randomly selected and the distance between them is set larger than Δ_l (introduced in Sec. 6.4) so that vehicles are involved for traveling. Table 7 reports the experimental results of generating variable number of moving objects. Overall, we see the running time and storage size scale linearly to the data size.

8 Conclusions

In this paper, we develop 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 network and the indoor environment is generated from floor plans. Each infrastructure has a graph within that environment for trip plannings. Generic moving objects are produced based on trip plannings where the movement can cover all available environments. The space manages all infrastructures and also services as an interface between moving objects and underlying referenced objects, e.g., roads, pavements, rooms. All created data are managed by GMOD, which will be used for testing GMOD performance.

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

Table 7: Generic Moving Objects

One important future work is to study the distribution of generic moving objects. Currently, the moving objects are generated in an random way. For modeling humans' movement in a practical way, the data should be generated according to the study of people daily movement, e.g., commuting between home and work, going shopping and visiting friends in the weekend. Another thing is to develop a benchmark for GMOD.

Acknowledgement

Thanks for my colleague Simone Jandt providing the program to get the traffic flow of moving cars.

References

[1] http://sumo.sourceforge.net/.

[2] http://www.bbbike.de/cgi-bin/bbbike.cgi.

[3] http://www.bkrfloorplans.co.uk/bkr_services/cinemas.html.

[4] http://www.census.gov/geo/www/tiger/tgrshp2010/tgrshp2010.html.

[5] http://www.edenresort.com/home.

[6] http://www.emaarmgf.com/mallwestdelhi/floor_plan.asp.

[7] http://www.greenhosp.org/floor_plans.asp.

[8] http://www.modulargenius.com/default.aspx.

[9] http://www.nshispeed.nl/en/stations/station-maps-floor-plan.

[10] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *SIGMOD Conference*, pages 685–696, 2011.

[11] C. Binnig, D. Kossmann, E. Lo, and M.T. Özsu. Qagen: generating query-aware test databases. In *SIGMOD*, pages 341–352, 2007.

[12] J. Booth, P. Sistla, O. Wolfson, and I.F. Cruz. A data model for trip planning in multimodal transportation systems. In *EDBT*, 2009.

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

[14] N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, pages 1097–1107, 2005.

[15] C. Düntgen, T. Behr, and R.H. Güting. Berlinmod:a benchmark for moving objects databases. *VLDB Journal*, 18(6):1335–1368, 2009.

[16] 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.

[17] 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*, 2005.

[18] 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.

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

[20] C.S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, 2009.

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

[22] J.Xu and R.H.Güting. A generic data model for moving objects. Informatik-report 360, Fernuniversität Hagen, 2011.

[23] J.Xu and R.H.Güting. Infrastructures for research on multimodal moving objects. In *MDM, Demo Paper*, 2011.

[24] 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.

[25] 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.

[26] D. Pfoser and Y. Theodoridis. Generating semantics-based trajectories of moving objects. *Computers, Environment and Urban Systems*, 27(3):243–263, 2003.

[27] 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.

[28] J.M. Saglio and J. Moreira. Oporto: A realistic scenario generator for moving objects. *GeoInformatica*, 5(1):71–93, 2001.

[29] M. Sharifzadeh, M. Kolahdouzan, and C.Shahabi. The optimal sequenced route query. *VLDB Journal*, 17(4):765–787, 2008.

[30] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.

[31] L. Stenneth, O. Wolfson, P. Yu, and B. Xu. Transportation mode detection using mobile devices and gis information. In *ACM SIGSPATIAL*, 2011.

[32] J.A. Storer and J.H. Reif. Shortest paths in the plane with polygonal obstacles. *Journal of the ACM*, 41(5):982–1012, 1994.

[33] Y.C. Tay. Data generation for application-specific benchmarking. In *VLDB, Challenges and Visions*, 2011.

[34] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis. Constrained shortest path computation. In *SSTD*, pages 181–199, 2005.

[35] Y. Theodoridis and M. A. Nascimento. Generating spatiotemporal datasets on the www. *SIGMOD Record*, 29(3):39–43, 2000.

[36] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the generation of spatiotemporal datasets. In *SSD*, pages 147–164, 1999.

[37] B. Yang, H. Lu, and C.S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, 2010.

[38] 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.

[39] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *WWW*, 2008.

Appendix

Algorithm 1: RBOShortestPath(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

let S be the set of visible points to s by searching DG;
let E be the set of visible points to e by searching DG;
let Q be a priority queue initially empty;
let T_{se} be the shortest path tree with root node s;
foreach $s_i \in S$ **do**
    enqueue(Q, s_i);
    insert s_i into T_{se};
while *head(Q)* $\neq e$ **do**
    $n_i \leftarrow$ *dequeue*(Q);
    if $\exists e_i \in E$ *such that* e_i *is equal to* n_i **then**
        enqueue(Q, e);
        insert e into T_{se};
    foreach $n_{i+1} \in$ *Adj(*n_i, VG*)* **do**
        enqueue(Q, n_{i+1});
        insert n_{i+1} into T_{se};
select p from T_{se};
return p;

Algorithm 2: BNShortestPath(bs_s, bs_e, G_{BN})

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

```
let Q be a priority queue initially empty;
let T_se be the shortest path tree with root node bs_s;
enqueue(Q, bs_s);
while head(Q) ≠ bs_e do
    bs_i ← dequeue(Q);
    foreach bs_{i+1} ∈ Adj(bs_i, G_BN) do
        if (bs_i, bs_{i-1}),(bs_i, bs_{i+1}) fulfills one condition in Lemma 5.1 then
            enqueue(Q, bs_{i+1});
            insert bs_{i+1} into T_se ;
select p from T_se;
return p;
```

Algorithm 3: MNShortestPath(ms_s, ms_e, G_{MN})

Input: ms_s, ms_e - start and end metro stops;
G_{MN} - metro graph
Output: p - a shortest path from ms_s to ms_e by time

```
let T_se be the shortest path tree with root node ms_s;
let Q be a priority queue initially empty;
enqueue(Q, ms_s);
while head(Q) ≠ ms_e do
    ms_i ← dequeue(Q);
    foreach ms_{i+1} ∈ Adj(ms_i, G_MN) do
        enqueue(Q, ms_{i+1});
        insert ms_{i+1} into T_se;
select p from T_se;
return p;
```

Algorithm 4: IndoorShortestPath(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: p - 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 by accessing Indoor_Rel;
3      create dual graph DG' and visibility graphs VG' on r;
4      return RBOShortestPath(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 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 p from T_se;
21 return p;
```