ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.7, July 2009, pp.1866−1884
doi: 10.3724/SP.J.1001.2009.00571

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

移动对象数据库模型、查询语言及实时交通流分析*

丁治明+

(中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

Data Model, Query Language, and Real-Time Traffic Flow Analysis in Dynamic Transportation Network Based Moving Objects Databases

DING Zhi-Ming+

(Fundamental Software Research Center, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: zhiming@iscas.ac.cn

Ding ZM. Data model, query language, and real-time traffic flow analysis in dynamic transportation network based moving objects databases. *Journal of Software*, 2009,20(7):1866−1884. http://www.jos.org.cn/1000-9825/571.htm

Abstract: This paper proposes a moving objects database model, dynamic transportation network based moving objects database (DTNMOD). Besides, a real-time traffic flow statistical analysis method based on moving object trajectories is provided. In DTNMOD, the underlying transportation network is modeled as a dynamic graph so that the state, topology, and traffic parameters of the transportation graph at any time instant can be tracked and queried. Moving objects are modeled as moving graph points which move only within predefined transportation networks. The data model is given as a collection of data types and operations which can be plugged as attribute types into a DBMS to obtain a complete data model and query language. To evaluate the performance of the proposed model, this paper has implemented a prototype system based on PostgreSQL and conducted a series of experiments. The experimental results show that DTNMOD provides good performances in processing region and join queries.
Key words: database; spatio-temporal; moving objects; data types and operators; PostgreSQL

摘 要: 提出一种移动对象数据库模型——Dynamic Transportation Network Based Moving Objects Database(简称DTNMOD),并给出了DTNMOD中基于移动对象时空轨迹的网络实时动态交通流分析方法.在DTNMOD中,交通网络被表示成动态的时空网络,可以描述交通状态、拓扑结构以及交通参数随时间的变化过程;网络受限的移动对象则用网络移动点表示.DTNMOD 模型包含了完整的数据类型和查询操作的定义,因此可以在任何可扩充数据库(如PostgreSQL 或 SECONDO)中实现,从而得到完整的数据库模型和查询语言.为了对相关模型的性能进行比较与分析,基于 PostgreSQL 实现了一个原型系统并进行了一系列的实验.实验结果表明,DTNMOD 提供了良好的区域查询及连接查询性能.
关键词: 数据库;时间-空间;移动对象;数据类型及操作;PostgreSQL
中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60573164 (国家自然科学基金); the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry (教育部留学回国科研启动基金)

Received 2008-01-22; Accepted 2008-10-07

1 Introduction

Moving Objects Database (MOD) is the database which can track and manage the locations of moving objects such as cars, ships, flights, and pedestrians. In recent years, the management of moving objects has been intensely investigated. In Ref.[1], Sistla and Wolfson, *et al*. have proposed a Moving Objects Spatio-Temporal (MOST) model which is capable of tracking not only the current, but also the near future positions of moving objects. Su, *et al*. in Ref.[2] have presented a data model for moving objects based on linear constraint databases. In Ref.[3], Güting, *et al*. have presented a data model and data structures for moving objects based on abstract data types (ADT). Besides, Pfoser and Jensen, *et al*. in Ref.[4] have discussed the indexing problem for moving object trajectories. However, nearly none of these works have treated the interaction between moving objects and the transportation networks in any way.

More recently, increasing research interests are focused on modeling moving objects on transportation networks. Vazirgiannis, *et al*. in Ref.[5] have discussed moving objects on fixed road networks. In Ref.[6], the authors have presented a computational data model for network constrained moving objects. Besides, the index problems of network constrained moving objects have also been studied[7]. However, all these works have only considered static transportation networks, and none of them have dealt with the database modeling problem such as the definition of data types and query languages.

In Ref.[8], Güting, *et al*. have proposed a fixed-network based MOD model with a rich set of data types and operations defined. However, the model is based on static transportation networks so that topology changes, state changes, and traffic parameters can not be expressed, which can limit its usability in real-world applications. In a running MOD system, people often need to keep the trajectory data of moving objects for a long period of time, say 3 months, during which time the topology and state changes are actually inevitable. To track the trajectory of moving objects without considering the dynamic aspect of the underlying transportation network is not sufficient to express the behavior of moving objects and to support traffic-aware navigation utilities. For instance, a route where moving objects were running three months ago may no longer exist in the current network so that the network positions of the moving objects can become invalid. Besides, route sections can be blocked by temporary constructions for several hours or even for days. If the state information is not kept in the database, the navigation algorithm can direct the drivers to an unavailable part of route. Moreover, in Ref.[8], the transportation network is defined as a single data type with an interface to relations and standard data types defined. The design may not be suitable for implementation in common extensible DBMSs such as PostgreSQL, even though it can be well implemented in SECONDO[9] which provides a lot of unique utilities.

In Refs.[10,11], Ding and Güting have discussed the modeling methods for dynamic transportation networks and for network-constrained moving objects. However, the network model in Ref.[10] is edge-based, which is not suited for location update purpose because with the edge-based model, location update happens whenever the moving object passes through a junction. The network model in Ref.[11] is route-based, but the framework is a little bit too simple to express real-world traffic networks, in which traffic flows can be very complicated. For instance, in Beijing's GIS data, there exist some routes whose traffic flow directions can have different styles for different part-part of the route can be one-way while other parts allow dual traffic flow directions, which can not be expressed by the model of Ref.[11]. Besides, in Ref.[11], traffic parameters are not considered, and data types and operations for network constrained moving objects are not defined either.

Except the above limitations, another important problem with all the existing network-constrained MOD models is that they utilize a uniform network framework (either route-based or edge-based) so that they can not accommodate location update purpose and traffic-aware navigation at the same time. In modeling network-

constrained moving objects databases, the granularity in presenting the transportation network is a key problem which can not be easily decided. On the one hand, network models with bigger granularity (for instance the route-based model[7]), can reduce location update costs, but the granularity can be too rough for traffic parameter expression and for traffic aware navigation. On the other hand, network models with smaller granularity (such as the edge-based model[7]), even though they are suitable for traffic parameter expression and traffic aware navigation, can incur higher location update costs.

To solve the above problems, we propose a new moving objects database model, Dynamic Transportation Network Based Moving Objects Database (DTNMOD), in this paper. In DTNMOD, a two-layered, route-ARS based transportation network framework is utilized, and both the moving objects and the traffic network are modeled as dynamic, so that their interrelationship can be better explored.

The remaining part of this paper is organized as follows. Section 2 describes the overall system architecture. Section 3 formally defines the DTNMOD model, including data types and operations. Section 4 presents the real-time traffic flow statistical analysis mechanism in DTNMOD. Section 5 discusses implementation issues and performance evaluation results and Section 6 finally concludes the paper.

2 Overview of the System Architecture in DTNMOD

In the following discussion, we suppose that in the whole MOD system, the transportation network is expressed as a single graph which is composed of a set of routes and a set of junctions. For simplicity, we assume that the graph is geographically embedded in the $X\times Y$ plane. The methodology proposed in this paper can be easily extended to the $X\times Y\times Z$ space.

In DTNMOD, the concept of "junction" includes two kinds of traffic network components: (1) intersections which connect two or more routes; and (2) beginning/end points of routes even though they may be associated with only one route. With each junction we associate a connectivity matrix to describe the traffic rules around the junction, so that whether or not moving objects are permitted to make left, right, and U-turns can be expressed.

For each route, its geometry is described by a polyline so that it can actually assume an arbitrary shape instead of just a straight line. The polyline is considered as directed, whose direction is from the first vertex to the last, which enables us to speak of the beginning point (or 0-end) and the end point (or 1-end) of the route.

A route is composed of a set of *Directed Atomic Route Sections* (ARS). Conceptually, a directed atomic route section is a directed segment of the traffic flow within a route, which connects two junctions and does not contain any other junctions from which moving objects can exit. The direction of ARS is indicated by the order of the two junctions (from the first junction towards the second). Since ARS is the basic unit for navigation, we choose it as the basic unit for keeping traffic states and parameters. Figure 1 illustrates the relationship of routes, junctions, and atomic route sections.

As shown in Fig.1, route 1 has two traffic flow directions, "+" and "–" ("+" traffic flow is from 0-end to 1-end, and "–" is from 1-end to 0-end). Therefore, the atomic route sections are also in two directions. Some ARSs are symmetrical (for instance ars1 and ars10), but essentially, ARSs of the two route sides can be asymmetrical (for instance ars2 and ars9, ars3 and ars8).

In DTNMOD, every route, junction, or ARS has an identifier associated so that the system can differentiate them even though their geometry could be the same. For instance, two parallel routes, one upper and the other lower, can have identical geometries, but they have different identifiers so that the system can deal with them correctly.

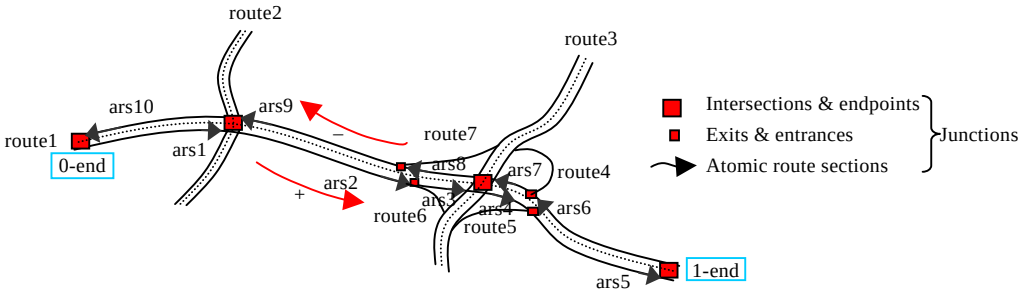
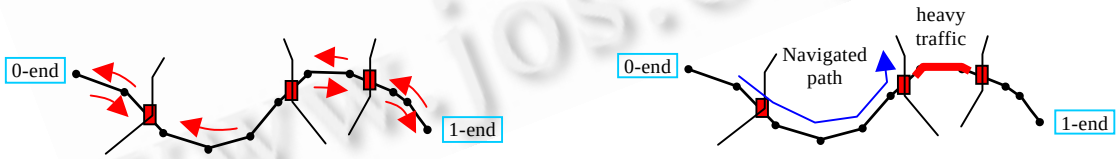


Fig.1 Two-Layered, route-ARS based presentation of transportation networks

Through the above two-layered (route-ARS) presentation for traffic networks, the system can describe complicated traffic network and the traffic flows in it. The benefit of this network framework is two-fold. First, since routes are explicitly presented, the MOD system can take routes as the basic unit for network position representation and for location updates. Therefore, if a moving object moves in a certain route with roughly steady speed, no location updates will be triggered even though it may pass through several junctions along the route. As a result, the location update cost can be reduced. In this aspect, it is superior to the edge-based network model (see Ref.[7]). Second, through atomic route sections, we can describe traffic parameters and state information in more detail so that important utilities such as traffic-aware navigation can be better supported, and in this aspect, it is superior to the route-based network model (also see Ref.[7]).

To better understand the route-ARS based traffic network framework, we give some examples. In the GIS data set of Beijing's traffic network, there exist a lot of routes whose traffic flows can have complicated patterns. Some parts of the route are one-way, while others are bidirectional, as shown in Fig.2(a). If we use the network model described in Refs.[8,11], this kind of situations can not be expressed in a natural way. Certainly we can reorganize the routes according to the traffic flows. However, the inner expression can be different from the actual route naming system which can cause confusion in real-world applications. With the two-layered network framework, the above problem can be easily solved.

In real-world applications, we can often find that the traffic conditions of a certain route are different from part to part. One part of the route can have heavy traffic so that the average speed is quite slow even though other parts are quit smooth, as shown in Fig.2(b). If we don't have ARS, we will not be able to express the complicated traffic conditions, as a result, the traffic-aware navigation cannot be well supported. On the other hand, with the route-ARS based network framework, we can express the detailed traffic conditions so that the smooth parts can still be adopted in traffic-aware navigation.



(a) A route with complicated traffic flows (b) A route with complicated traffic conditions

Fig.2 Two route examples with complicated traffic information

Moving objects can move inside the graph and transfer from one route to another via junctions. In the system, both moving objects and the underlying transportation network are dynamic-moving objects change their locations continuously, while transportation networks change their states and topologies discretely. The historical as well as

the current information of both the moving objects and the underlying transportation network is kept in the database.

In order to envisage the above ideas, we give an example. This example shows how a modern logistic system works. We suppose that in the system, transportation vehicles are uniquely identified and each of them is equipped with a portable computing platform and some integrated location tracking equipments (such as GPS (global positioning system) and wireless interface) so that its location at any time instant can be retrieved.

We assume that such a logistic system, which is responsible for cargo delivery services, exists in Beijing. The whole Beijing transportation network is expressed as a graph in the database, and the vehicles can move inside the transportation network during their journey, as shown in Fig.3.

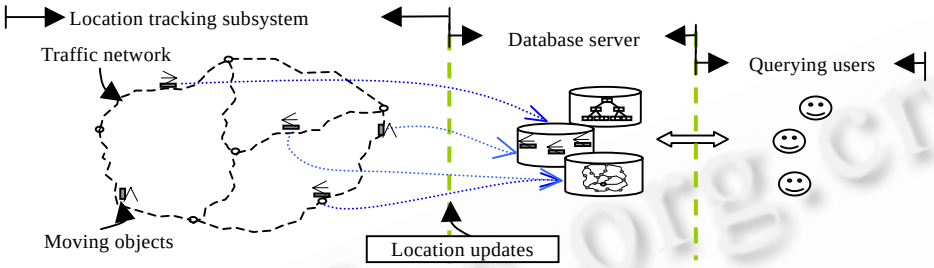


Fig.3 Architecture of the DTNMOD system

Since both historical and current location information is kept in the database, the system can support the following queries: “tell me the location of vehicle x310 at 2:00 PM of last Friday” and “find all vehicles that are currently in the Zhong-Guan-Cun street”.

Besides, since the temporal status of the transportation network (such as blockages, insertion and deletion of routes or junctions) is also tracked and stored in the dynamic graph, the following queries: “tell me the topology of the Beijing network at time *t*” and “find the current traffic jams in the Zhong-Guan-Cun street and the moving objects affected by them” can also be handled.

To speed up query processing, both moving objects and dynamic graphs should be indexed. The database records and the index structures contain location information covering a time period from the past until the near future. Therefore, when location updates occur, both database records and the index need to be modified. The location update operations will trigger the online graph refreshing procedure so that the blockage and traffic parameters of the transportation network can be recomputed and tracked in the database. Location update strategies, indexing structures, and parameter refreshment mechanisms for network constrained moving objects databases are out of the scope of this paper and will be discussed in other papers.

3 Model of Moving Objects on Dynamic Transportation Networks

In DTNMOD, the transportation network is modeled as “dynamic” graph which allow us to express traffic parameters, state changes (such as traffic jams and blockages caused by temporary constructions), and topology changes (such as insertion and deletion of junctions or routes). For simplicity, “dynamic transportation network” and “dynamic graph” will be used interchangeably throughout this paper.

In modeling the dynamic graph, we utilize a “*base-state plus event*” method. The basic idea is to associate a temporal attribute to every junction or route of the graph so that the state of the junction or route at any time instant can be retrieved. Each temporal attribute includes a base state and a series of events which describe the changes of the state or the traffic parameters so that the whole temporal history of the junction or route can be represented. This

is very useful in moving objects databases since a lot of queries (such as traffic-aware shortest path queries, which take traffic conditions into consideration) can only be processed efficiently by accessing the states of the transportation network. Besides, through the temporal attribute, we can also know the life span of any junction or route of the transportation graph so that the topology changes of the transportation networks can also be queried.

In DTNMOD, moving objects are modeled as moving graph points. A moving graph point is a function from time to graph point, which can be represented as a group of motion vectors in the discrete model. The methodology proposed in this paper can be extended to deal with more complicated situations where moving objects need to be modeled as moving graph lines or moving graph regions.

In the following two subsections, we will formally define the DTNMOD model with the Second-Order-Signature framework, which uses a system of two coupled signatures where the first signature describes a type system and the second one describes an algebra over the types of the first signature[9]. Especially, we will focus on discrete model so that the data types and operations defined in this paper can be implemented directly in an extensible database system such as PostgreSQL and SECONDO. The notation of the definitions will follow those described in Ref.[3].

3.1 Data types

Table 1 presents the type system of the DTNMOD model. Type constructors listed in Group 1 are basic ones which have been defined and implemented in Ref.[3]. In the following, we mainly focus on the type constructors listed in Group 2. Some type constructors in Group 2, such as graph state data types, have been discussed in Ref.[10], but most of them are modified to fit into the two-layered traffic network framework and to accommodate traffic parameters.

Table 1 Signatures describing the type system of DTNMOD

Group	Type constructor	Signature	
1	*int*, *real*, *string*, *bool*		→BASE
	point, *points*, *line*, *region*		→SPATIAL
	instant		→TIME
	range	BASE∪TIME	→RANGE
	intime, *moving*	BASE∪SPATIAL	→TEMPORAL
2	*status*, *blockage*, *blockreason*, *blockpos*, *traffpara*, *traffparas*, *state*		→GSTATE
	intimeevent, *intimestate*, *g_temporal*		→GTEMPORAL
	dynroute, *dynjunct*, *ars*		→GRAPH
	gpoint, *gpoints*, *grsect*, *gline*, *gregion*		→GSPATIAL
extending	*intime*, *moving*	{*gpoint*}∪BASE∪SPATIAL	→TEMPORAL

Among the type constructors listed in Group 2, graph state (GSTATE) data types, graph temporal (GTEMPORAL) data types, and dynamic graph (GRAPH) data types are used to describe dynamic transportation networks. Graph spatial (GSPATIAL) data types describe static graph objects, which form the basis for the modeling and querying of moving objects. Temporal (TEMPORAL) data types describe moving objects.

As shown in Table 1, temporal data types are obtained by "extending" the previously defined type constructors *moving* and *intime* so that the *gpoint* data type is included as their arguments. SPATIAL data types are still reserved for these two type constructors to deal with the situations when moving objects move outside of the predefined transportation network, for instance, moving in a lake or in a big square.

3.1.1 Dynamic graph data types

In DTNMOD, the transportation network is modeled as a dynamic graph, with every junction or route associated with a *g_temporal* attribute which describes its state history.

Definition 1 (**polyline**). A polyline can be expressed by a sequence of points which correspond to the vertices of the polyline. Therefore we can define polylines as follows:

$$polyline=\{\langle p_1,p_2,\ldots,p_n\rangle | n\geq 2, \forall i\in\{1,\ldots,n\}: p_i\in D_{point}\}.$$

Definition 2 (**dynamic route**). A dynamic route can be viewed as a normal graph route with a temporal attribute associated. The carrier set of the *dynroute* data type is defined as follows:

$$D_{dynroute}=\{(rid,geo,(jid_i,pos_i)_{i=1}^{n},len,ARS,tp) | rid\in D_{int},geo\in polyline,n\geq 2,len\in D_{real},ARS\subseteq D_{ars},tp\in D_{g\_temporal}\}$$

where *rid* is the identifier of the route which is isomorphic to integer, *geo* is a polyline which describes the geographical shape of the route; (jid_i,pos_i) $(1\leq i\leq n)$ indicates the *i*th junction associated with the route, where jid_i is the identifier of the junction and $pos_i\in[0,1]$ describes its position inside the route (we suppose that the total length of the road section is 1, and then any location in the road section can be represented by a real number $p\in[0,1]$); *len* is the length of the route, *ARS* is the set of atomic route sections (see Definition 3) of the route, and $tp\in D_{g\_temporal}$ is the graph temporal attribute associated with the route (see Definition 8).

Definition 3 (**atomic route section**). The carrier set of the Atomic Route Section (ARS) data type is defined as:

$$D_{ars}=\{(aid,(jid_{from},pos_{from}),(jid_{to},pos_{to})) | aid\in D_{int},jid_{from},jid_{to}\in D_{int},pos_{from},pos_{to}\in[0,1]\}.$$

Atomic route section is considered as directed, and its direction is from the first junction towards the second. Therefore, a dual route section which allows traffic flows in 2 directions is expressed as 2 atomic route sections.

Definition 4 (**dynamic junction**). A dynamic junction can be considered as a normal junction with a temporal attribute associated. As stated earlier, the beginning/end points of routes are also considered as junctions. The carrier set of the *dynjunct* data type is defined as follows:

$$D_{dynjunct}=\{(jid,loc,((rid_i,pos_i))_{i=1}^{n},m,tp) | jid\in D_{int},loc\in D_{point},tp\in D_{g\_temporal},n\geq 1\},$$

where *jid* is the identifier of the dynamic junction, *loc* is a point value which describes the position of the junction. (rid_i,pos_i) $(1\leq i\leq n)$ in the above definition indicates the *i*th route connected by the junction, where rid_i is the identifier of the route and $pos_i\in[0,1]$ describes the position of the junction inside the route. *m* is the connectivity metrix[11], and *tp* is the graph temporal attribute associated with the junction (see Definition 8).

Definition 5 (**dynamic graph**). A dynamic graph, *G*, is composed of a set of dynamic routes and a set of dynamic junctions, which can be defined by

$$G=(R,J),$$

where $R\subseteq D_{dynroute}$ and $J\subseteq D_{dynjunct}$. In DTNMOD, each database is viewed as an instance of dynamic graph with a collection of network constrained moving objects, so that graph itself is not defined as a data type. In implementation, *R* and *J* can be implemented as relational tables.

3.1.2 Graph temporal data types

Graph temporal data types are used to track the state history and also the life span of a junction or a route.

Definition 6 (**intimestate**). The *intimestate* data type is used to describe the state (see Definition 15 below) of a junction or a route at a certain time instant. Its carrier set is defined as follows:

$$D_{intimestate}=\{(t,st) | t\in D_{instant},st\in D_{state}\}.$$

Definition 7 (**intimeevent**). The *intimeevent* data type describes the change happened to the state of a junction or a route. Its carrier set is defined as follows:

$$D_{intimeevent}=\{(t,obj,value) | t\in D_{instant},obj\in\{status,blocs,para\},value\in D_{status}\cup D_{blockages}\cup D_{traffpara}\}.$$

In the above definition, *obj* is the object of the event. The event can change the status, the blockages, or a certain traffic parameter of the junction or route. *value* specifies the new value of the event.

Definition 8 (**graph temporal**). The *g_temporal* data type is used to describe the state history of a junction or a route. Its carrier set is defined as follows:

$$D_{g\_temporal}=\{(t_{insert},t_{delete},state_{initial},(event_i)_{i=1}^{n}) | t_{insert},t_{delete}\in D_{instant},state_{initial}\in D_{intimestate},$$

$$event_i=(t_i,obj_i,value_i)\in D_{intimeevent}\ (1\le i\le n),\forall i\in\{1,\ldots,n-1\}:t_i<t_{i+1}\}.$$

In the above definition, t_{insert} and t_{delete} are two time instant values which indicate the insert time and delete time of the route or junction respectively. t_{insert} must be a defined value while t_{delete} can be either defined or undefined. If t_{delete} is "undefined" (denoted as "$\perp$"), then the corresponding junction or route is still active in the transportation network. Otherwise, t_{delete} specifies the time when it is removed from the network. In this way, the life span of each junction or route can be presented, and in this way we can decide the topology of the transportation graph at any time instant.

In DTNMOD, the value of the graph temporal attribute is maintained automatically through the real-time traffic flow statistical analysis module (see Section 4).

3.1.3 Graph state data types

Graph state data types are used to describe the state of a junction or a route. By "state" we mean three aspects: status, blockages, and traffic parameters. In dynamic transportation networks, a junction can have two statuses: opened and closed, while a route can have three statuses: opened, closed, and blocked, according to their availability to moving objects.

If the status is "blocked" for a certain route, then the detailed blockage information (for instance the cause and the position) will be described.

For a given route, unless it is closed, it will have traffic parameters associated with each ARS to reflect its traffic status. There can be a lot of traffic parameters, for instance, average speed (including point speed and line speed), flux, vehicle density, travel time, and so forth, and we can use type IDs to identify them. As stated earlier, since atomic route sections are the basic unit for route selection in navigation, we take ARS as the basic unit for presenting traffic parameters.

Definition 9 (**status**). The carrier set of the *status* data type is defined as follows:

$$D_{status}=\{opened,closed,blocked\}.$$

In a transportation system, blockages can happen quite frequently. For instance, a road section can be blocked for hours by a car accident or by a temporary construction, or even by heavy traffic jams. Typically, the location of a blockage is static. The location of a blockage can then be expressed as a closed interval over [0,1], whose boundaries indicate the border of the blocked area.

Definition 10 (**interval**). Let $(S,<)$ be a set with a total order. Intervals and closed intervals over S are defined as:

$$interval(S)=\{(s,e,lc,rc)|s,e\in S,lc,rc\in bool,s\le e,(s=e)\Rightarrow(lc=rc=\text{true})\},$$

$$cinterval(S)=\{(s,e,lc,rc)|s,e\in S,s\le e,lc=rc=\text{true}\}.$$

where lc and rc are two flags indicating "left-closed" and "right-closed" respectively.

Definition 11 (**blockage reason**). The data type *blockreason* describes the cause of a blockage. Its carrier set is

$$D_{blockreason}=\{construction,traffic\text{-}jam,accident,\perp\},$$

where $\perp$ means "undefined" and is used for other reasons.

Definition 12 (**blockage position**). The data type *blockpos* is used to describe the position of a blockage, and its carrier set is defined as follows:

$$D_{blockpos}=\{\psi|\psi\in cinterval([0,1])\}.$$

Definition 13 (**blockage**, **blockages**). The *blockage* data type is used to describe a blockage, including its reason and its location. The *blockages* data type is used to describe multiple blockages inside one single route. Their carrier sets are defined as follows:

$$D_{blockage}=\{(br,\psi)|br\in D_{blockreason},\psi\in D_{blockpos}\},$$

$$D_{blockages}=\{B|B\subseteq D_{blockage}\}.$$

The definition of the ***blockages*** data type is based on the fact that several blockages can coexist in one route at the same time so that they should be described as a set of blockages instead of as a single blockage value.

Definition 14 (**traffic parameter**, **traffic parameters**). The ***traffpara*** data type is used to describe a single traffic parameter of an atomic route section, and the ***traffparas*** data type is used to represent a set of traffic parameters of a certain atomic route section. Their carrier sets are defined as follows:

$$D_{traffpara}=\{(typeid,ars,value)|typeid\in D_{int},ars\in D_{int},value\in D_{real}\},$$

$$D_{traffparas}=\{((typeid_i,ars_i,value_i))_{i=1}^{n}|n\geq 1,typeid\in D_{int},ars\in D_{int},value\in D_{real}\}.$$

In the above definition, *typeid* and *ars* are integers which indicate the traffic-parameter type ID and the atomic route section ID, and *value* is a real number to specify the value of the parameter. In DTNMOD, each atomic route section can have 5 parameters to describe its traffic state (see Section 4).

Definition 15 (**state**). The data type ***state*** is used to describe the detailed state (including status, blockages, and traffic parameters) of a junction or a route, and its carrier set is defined as follows:

$$D_{state}=\{(s,blocs,paras)|s\in D_{status},blocs\in D_{blockages},paras\in D_{traffparas},s\neq blocked\Leftrightarrow blocs=\varnothing\}.$$

3.1.4 Graph spatial data types

Based on the above definitions for dynamic transportation networks, we can then define some useful data types, graph point, graph points, graph route section, graph line, and graph region, which form the basis of the modeling and querying of moving objects.

Definition 16 (**graph point**, **graph points**). The ***gpoint*** data type describes a point inside the transportation graph, and the ***gpoints*** data type describes a set of graph points. Their carrier sets are defined as follows:

$$D_{gpoint}=\{jid|jid\in D_{int}\}\cup\{(rid,pos)|rid\in D_{int},pos\in[0,1]\},$$

$$D_{gpoints}=\{PS|PS\subseteq D_{gpoint}\}.$$

The position of a graph point can have two possibilities. If it is located in a junction, then it is specified by the identifier of the junction. If it is located in a route, it is specified by the (*rid*,*pos*) pair, where *rid* decides a unique route *r* of the graph and $pos\in[0,1]$ indicates a position inside *r*.

Definition 17 (**graph route section**). The ***grsect*** data type represents a section of a route, and its carrier set is

$$D_{grsect}=\{(rid,S)|rid\in D_{int},S\in cinterval([0,1])\}.$$

Definition 18 (**graph line**). A graph line is defined as a consecutive chain of route sections inside the graph. Its carrier set is defined as follows:

$$D_{gline}=\{\langle\omega_i\rangle_{i=1}^{n}|n\geq 1,\omega_i=(rid_i,S_i)\in D_{grsect},$$

where

(1) $\forall i\in\{1,n\}:S_i\in cinterval([0,1])$;

(2) $\forall i\in\{1,\ldots,n-1\}:adjacent(\omega_i,\omega_{i+1})\}$

where $adjacent(\omega_i,\omega_{i+1})$ means that ω_i, ω_{i+1} meet with their end points geographically so that all route sections of the graph line can form a chain.

Definition 19 (**graph region**). A graph region is defined as an arbitrary set of graph route sections. The carrier set of the ***gregion*** data type is defined as follows:

$$D_{gregion}=\{W|W\subseteq D_{grsect}\}.$$

3.1.5 Moving object temporal data types

In Ref.[3], Güting *et al*. have defined the ***moving*** and ***intime*** type constructors which take BASE data types and SPATIAL data types as arguments. In the following we extend these two type constructors by taking the ***gpoint*** data type also as its argument so that we can get the ***moving***(***gpoint***) and ***intime***(***gpoint***) data types.

Conceptually, a moving graph point *mgp* can be defined as a function from time to graph point:

$$mgp=f:D_{instant}\rightarrow D_{gpoint}.$$

In implementation, we should translate the above definition into a discrete representation. That is, a moving graph point is expressed as a series of motion vectors with each motion vector describing the traffic parameters (such as location, direction, speed, and so forth) of the moving object at a certain time instant.

Definition 20 (**motion vector**). The carrier set of the data type *motionvector* is defined as follows:

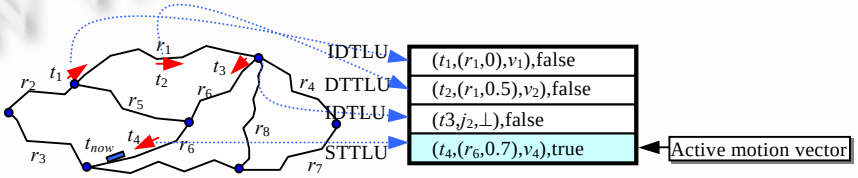
$$D_{motionvector}=\{(t,gp,\vec{v})|t\in D_{instant},gp\in D_{gpoint},\vec{v}\in D_{real},isinjunct(gp)\Rightarrow(\vec{v}=\perp)\},$$

where $\vec{v}$ is the speed measure. Its abstract value is equal to the speed of the moving object, while its sign (either positive or negative) depends on the direction of the moving object. If the moving object is moving from 0-end towards 1-end, then the sign is positive. Otherwise, if it is moving from 1-end to 0-end, the sign is negative. If *gp* is in a junction (*isinjunct*(*gp*)), then $\vec{v}$ is undefined.

Definition 21 (**moving graph point**). A moving graph point can be represented by a "motion vector plus active flag" sequence. The carrier set of the *moving*(*gpoint*) data type (or *mgpoint* for short) is defined as follows (in some other literatures, moving graph point is also called "trajectory" of moving object):

$$D_{mgpoint}=\{(mv_i,activeflag_i)_{i=1}^{n}|n\geq 1,activeflag_i\in D_{bool},mv_i=(t_i,gp_i,\vec{v}_i)\in D_{motionvector},\forall i\in\{1,\ldots,n-1\}:t_i<t_{i+1}\}.$$

For a running moving object, its location at any time instant during its life span can be computed through interpolation from its motion vectors. The motion vectors are generated by location updates (the location update mechanism for DTNMOD is discussed in Refs.[11,12]), and the last motion vector is called "active motion vector", which contains the current moving pattern of the moving object, and is the key information for computing the current or near future locations of the moving object and for triggering the next location update. We call the moving graph point with active motion vector "active" moving graph point, as shown in Fig.4.



(a) Journey of a moving object (b) Corresponding moving graph point value

Fig.4 An example moving graph point value

It is possible that a moving graph point value do not contain any active motion vectors. In this case we call the moving graph point "non-active".

Definition 22. The *intime*(*gpoint*) data type is used to describe a graph position of a moving object at a certain time instant, and its carrier set is defined as follows:

$$D_{intime(gpoint)}=\{(t,gp)|t\in D_{instant},gp\in D_{gpoint}\}.$$

Example 1. Based on the data types defined above, we can then define the following database schemas for the application example described in Section 2. The whole database contains a graph instance and a set of moving objects (we extend the *Create Table* command by allowing the "*as routes* | *juncts* | *mobjs*" clause).

Create Table BJRoutes (name: *string*, droute: *dynroute*) as routes;

Create Table BJJuncts (name: *string*, djunct: *dynjunct*) as juncts;

Create Table BJMObjs (mname: *string*, mid: *int*, mgp: *mgpoint*) as mobjs;

Besides, we suppose that in the database there exist several auxiliary relational tables which allow us to translate logic junction, route, and traffic parameter names to the corresponding identifiers.

3.2 Operations

In the earlier work on moving objects databases[3], Güing, *et al*. have defined and implemented a rich set of operations on the data types listed in Class1 of Table 1. In this subsection, we will show how these predefined operations can be systematically adapted to the DTNMOD model by an "Extending" technique. The basic idea of the "Extending" technique is to add the newly introduced data types into the signatures of the previously defined operations so that the extension of these operations can be expanded to include the newly defined data types in this paper. In addition to the extended operations, we will also define a set of new operations, which are mainly focused on graph specific data types. Table 2 gives a summary of the operations in the DTNMOD model.

Table 2 Operations of the DTNMOD model

Group	Class	Operations
Non-Temporal	Predicates	isempty, =, ≠, <, ≤, >, ≥, intersects, inside, before touches, attached, overlaps, on_border, in_interior
	Set operations	intersection, union, minus, crossings, touch_points, common_border
	Aggregation	min, max, avg, avg[center], single
	Numeric	no_components, size, perimeter, size[duration], size[length], size[area]
	Distance & direction	distance, direction
	Base type specific	and, or, not
Temporal	Projection to domain/range	deftime, rangevalues, locations, trajectory, routes, traversed, inst, val
	Interaction with domain/range	atinstant, atperiods, initial, final, present, at, atmin, atmax, passes
	When	when
	Lifting	(All new operations inferred)
	Rate of change	derivative, speed, turn, velocity
Graph Specific	Transformation	graph_euc, euc_graph, getjunct, getroute, getid
	Construction	gpoint, grsect
	Data extraction	geo, g_temporal, g_atinstant, xstate, xstatus, xblocs, xtraffpara
	Truncation	g_atperiods, g_present
	Projection	g_deftime

3.2.1 Extending operations

In designing DTNMOD operations, "Extending" is an important technique which allows us to apply previously defined operations to the data types newly introduced in this paper. The general rules for extending can be summarized as follows:

(1) Every operation whose signature involves *point* is extended to include *gpoint* also;

(2) Every operation whose signature involves *line* is extended to include *grsect* and *gline* also;

(3) Some of the operations whose signature involves *region* are extended to include *gregion* also;

(4) Operations which are only suited for 1D data types (see Ref.[3]) and for some specific data types (such as *region*) are not extended.

According to the above rules, the underscored (line-underscored and dot-underscored) operations in Table 2 are extended while other operations are not affected.

First let us deal with the non-temporal operations listed in Table 2. In Ref.[3], the signatures of most non-temporal operations (see the line-underscored non-temporal operations, such as **isempty**) are defined with two data type variables π and σ, where $\pi \in \{int, bool, string, real, instant, point\}$ and $\sigma \in \{range(int), range(bool), range(string), range(real), periods, points, line, region\}$. Now we extend the domain of π and σ like this:

$$\pi \in \{int, bool, string, real, instant, point\} \cup \{gpoint, status\},$$

$$\sigma \in \{range(int), range(bool), range(string), range(real), periods, points, line, region\} \cup \{gpoints, grsect, gline, gregion\}.$$

As a result of this change, all operations whose signatures are defined with π, σ variables are extended to cover the newly introduced data types automatically. As for operations whose signatures are not defined with π, σvariables

(see the dot-underscored non-temporal operations in Table 2, such as crossings), we should supplementtheir signatures by including the newly introduced data types while keeping their original semantics. Detailed supplementing methods can be found in Ref.[8].

As for the temporal operations listed in Table 2, the extension can be made in a similar way. In Ref.[3], the signatures of most temporal operations (see the line-underscored temporal operations, such as **deftime**) are defined by two data type variables α and β, ($\alpha,\beta\in$BASE$\cup$SPATIAL). Now we extend the domain of α and β like this:

$$\alpha\in\text{BASE}\cup\text{SPATIAL}\cup\{gpoint\},$$

$$\beta\in\text{BASE}\cup\text{SPATIAL}\cup\text{GSPATIAL}.$$

By this extension, the operations whose signatures are defined with α, β are extended automatically. As for operations whose signatures are not defined with α, β variables (see the dot-underscored temporal operations in Table 2, such as locations), their signatures should be supplemented to include the newly defined data types.

Through the above extension, the previously defined operations are enabled to deal with the data types newly introduced in this paper. The above extension is also suited to the Lifted operations. On the other hand, "Lifting" should also be applied to the extended part of the signatures described above.

3.2.2 Graph specific operations

Table 3 gives the signature of the operations specific for dynamic graphs. For the sake of readability, *rid*, and *jid* are used instead of *int* in the signatures.

Table 3 Graphic specific operations

Class	Operation	Signature	
Transform	graph_euc	*gpoint*	→*point*
		gpoints	→*points*
		gline	→*line*
		gregion	→*line*
	euc_graph	*point*	→*gpoints*
		points	→*gpoints*
		region	→*gregion*
	getjunct	*jid*	→*dynjunct*
	getroute	*rid*	→*dynroute*
	getid	*dynjunct*	→*jid*
		dynroute	→*rid*
Construct	gpoint	*rid*×*real*	→*gpoint*
		rid×*point*	→*gpoint*
		jid	→*gpoint*
	grsect	*rid*×*range*(*real*)	→*grsect*
		rid×*point*×*point*	→*grsect*
Data extract	geo	*dynjunct*	→*point*
		dynroute	→*line*
	g_temporal	*dynjunct*	→*g_temporal*
		dynroute	→*g_temporal*
	g_atinstant	*g_temporal*×*instant*	→*intimestate*
	xstate	*Intimestate*	→*state*
	xstatus	*state*	→*status*
		g_temporal×*instant*	→*status*
	xblocs	*state*	→*blockages*
		g_temporal×*instant*	→*blockages*
	xtraffpara	*state*×*type*[*int*]×*ars*[*int*]	→*real*
		g_temporal×*instant*×*type*[*int*]×*ars*[*int*]	→*real*
Truncation& Projection	g_atperiods	*g_temporal*×*periods*	→*g_temporal*
	g_present	*g_temporal*×*instant*	→*bool*
	g_deftime	*g_temporal*	→*periods*

As shown in Table 3, operations in the "Transform" class can be further divided into two groups. The first group of operations, **graph_euc** and **euc_graph**, transform Euclidean information to the corresponding graphical representation and vice versa. Operations in the second group, **getjunct**, **getroute**, and **getid**, transform identifier

representation to graphical representation and vice versa.

Construction operations are used to construct *gpoint* and *grsect* values. There can be three ways to indicate a position in a dynamic route when constructing *gpoint* and *grsect* values. One way is to give a real number value $\alpha \in [0,1]$ directly and the other way is to give the Euclidean information of the position. In the second case, further computation is needed to transform the point value to the corresponding real number value. If the graph point is inside a junction, then the junction identifier is enough for constructing the graph point value.

Operations in the "Data Extract" class are relatively simple and their only functionality is to extract information from an object. Through these operations, the detailed information contained in the transportation graph can be extracted step by step.

Conceptually, the graph temporal attribute associated with a junction or a route can be viewed as a "moving state" with its value changing only discretely. The operations in the Truncation & Projection class are designed because of this similarity. The **g_atperiods** operation generates a new graph temporal value which is corresponding to the given time periods. The operation **g_present** decides whether the graph temporal attribute is defined at a given time instant. The operation **g_deftime** return the time periods when the graph temporal value is defined.

3.2.3 Query examples

One of the main benefits of modeling moving objects on dynamic transportation networks is that the system is enabled to support logic road names or identifiers while queries based on Euclidean space can also be supported. Another advantage is that the interaction between moving objects and the underlying traffic networks, for instance, "show the current blockages in the Chang-An-Jie street and all moving objects inside the blocked area", can also be queried.

In order to envisage the above ideas, we give some query examples in this subsection, which are based on the database schema described in Example 1. Since logic road or junction names can be translated into the corresponding identifiers through the auxiliary relations (see Example 1), we suppose that the identifiers are already known and put them in the brackets behind the junction and route names.

Example 2. "Find all cargos that are currently in the Zhong-Guan-Cun street (rid)"

Select name, mid
From BJMObjs
Where **present**(**at**(mgp,**grsect**(rid,[0,1])),NOW)=TRUE;

In the above query, we use a *range*(*real*) value, [0,1], to represent a close interval over [0,1]. NOW is a real number variable whose value equals to the current time instant.

Example 3. "Find all cargos that are currently within 5 miles from my position *p*"

Select name, mid
From BJMObjs
Where **distance**(**val**(**atinst**(mgp,NOW)),*p*)≤5*MILE;

In this query, *p* is a point value and MILE is a constant real number value representing the length of one mile.

Example 4. "Find all cargo pairs which were within 500m from each other at time *t*"

Select A.mid, B.mid
From BJMObjs A, BJMObjs B
Where **distance**(**val**(**atinst**(A.mgp,*t*)),**val**(**atinst**(B.mgp,*t*)))≤500*METER;

This query shows the join of moving objects. METER is a constant real number value standing for one meter.

Example 5. "Find all current blockages in the Beijing traffic network"

Select **getid**(droute), **xblocs**(**g_temporal**(droute),NOW)

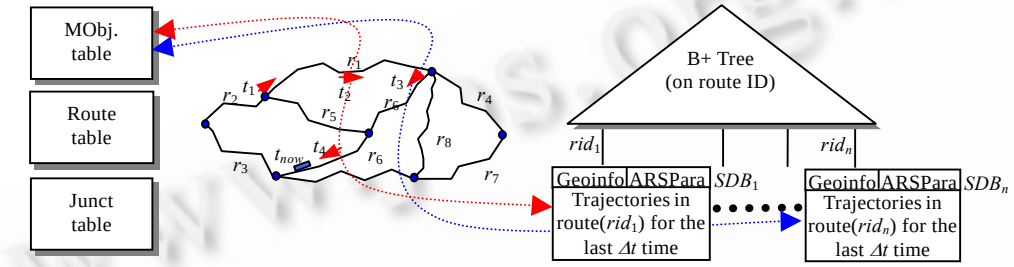
From BJRoutes

Where xstatus(**g_temporal**(droute),NOW)=blocked;

This example shows how the blockage information contained in the dynamic graph can be extracted through “Data Extract” operations.

4 Real-Time Traffic Flow Statistical Analysis Mechanism in DTNMOD

The above stated data types and operations enable us to represent traffic networks and moving objects in databases as relational tables. However, this is still not sufficient since the traffic flow analysis can not be efficiently conducted by scanning the whole relational tables once and again. To speed up the statistical analysis in DTNMOD, we maintain a statistical data structure, called Current Traffic-status Statistical Analysis Tree (CTSA-tree), to compute real-time traffic status of the transportation network, as illustrated in Fig.5. To simplify the discussion, we use “moving graph point” and “trajectory” interchangeably in this section. Besides, we suppose that the functions *route*(*rid*) and *ars*(*rid*,*aid*) return the corresponding route and ARS of the specified route and ARS identifiers respectively.



(a) Relational tables (b) Network and moving objects (c) Structure of the CTSA-tree

Fig.5 Current traffic-status statistical analysis tree in DTNMOD

As shown in Fig.5(c), the CTSA-Tree denotes the route records on the *rid* attribute into a B+ Tree structure. The leaf nodes contain records of the form (*rid*,*SDBPointer*), where *rid* is the identifier of the route, and *SDBPointer* is a pointer to the statistical data block (SDB) of the route. Each SDB takes the form (*routegeoinfo*, *ARSPara*,*datasource*), where *routegeoinfo* contains the *geo* and *len* attributes of the route (see definition 2), *ARSPara* is the set of atomic route sections of the route with each ARS associated with 5 traffic parameters (see below), and *datasource* is a set of trajectories which act as the data source for statistical computation.

Let us consider the SDB for *route*(*rid*). In the *ARSPara* field, each atomic route section *ars* has 5 basic traffic parameters associated to describe its current traffic status: (1) the number of moving objects inside *ars*, denoted as η_{mo}; (2) the average speed of moving objects inside *ars*, denoted as v_p; (3) the flux of moving objects in *ars*, denoted as θ_{mo}, where θ_{mo} describes the average number of moving objects which pass through *ars* per time unit, (4) the travel time of *ars*, denoted as T_τ; and (5) the traffic jam status β, here β can either be 1 or 0, indicating “blocked” or “unblocked” respectively. For each traffic parameter, both the last reported value $value_{last}$ (the last reported value which has been written to the relational table) and the newly computed value $value_{comp}$ (the new value which is computed when a new location update occurs) are kept.

In the *datasource* field, the latest trajectory data corresponding to route(*rid*) are kept as the statistical data source for computing the above traffic parameters. To speed up the computation, only the recent $\Delta\tau$ time period (say the last 10 minutes) trajectory data need to be stored. More precisely,

$$datasource=\bigcup_{i=1}^{n}(\psi_{t_i}(\psi_r(\xi_i,rid),[t_{now}-\Delta\tau,t_{now}]))\ ,$$

where ψ_{t_i} and ψ_r are interception operations (see definition 23) and t_{now} stands for the current time instant (we suppose that the set of all moving object trajectories is $\{\xi_1,\xi_2,\ldots,\xi_n\}$).

The new values ($value_{comp}$) of the 5 parameters are refreshed whenever a new location update related to the corresponding route occurs. When a moving object *mo* triggers a DTTLU or an STTLU in route *r*, the database server will refresh the parameters for all ARSs of *r*. When the moving object transfers from route r_s to r_e and trigger an IDTLU, the system will refresh parameters for all ARSs of both r_s and r_e. The computation methods are listed in Eqs.(1)~(5).

In the following, we describe in more detail how the parameters can be computed through the trajectory data. We first define a set of functions on moving object trajectories.

Definition 23 (**intercept operations**). Suppose that $\xi=\{\xi_1,\xi_2,\ldots,\xi_n\}$ is a set of trajectories. An intercept function ψ works as follows: (1) for each trajectory ξ_i ($1\le i\le n$), extract the *part*(s) which intersect the specified conditions; and (2) collect all the extracted parts (not null) from step (1) as the result of the function (if the extracted part for a trajectory is null, then it is discarded). We define 6 intercept operations:

$\psi_r(\xi,rid)$: for every trajectory in ξ, to extract the parts which correspond to *route*(*rid*) geographically. The result of $\psi_r(\xi,rid)$ is still a set of trajectories;

$\psi_a(\xi,rid,aid)$: for every trajectory in ξ, to extract the parts which correspond to *ars*(*rid*,*aid*) geographically. The result is a set of trajectories;

$\psi_p(\xi,npos)$: for every trajectory in ξ, to extract the parts which correspond to the network position *npos* geographically. The result is a set of motion vectors;

$\psi_{t_i}(\xi,I)$: for every trajectory in ξ, to extract the part temporally corresponding to the specified time interval $I=[t_s,t_e]$. The result is a set of trajectories;

$\psi_{tp}(\xi,t)$: for every trajectory in ξ, to extract the part temporally corresponding to the specified time instant t. The result is a set of motion vectors;

$\psi_v(\xi,v)$: for every trajectory in ξ, to extract the parts during which the speed of the moving object is lower than v. The result is a set of trajectories.

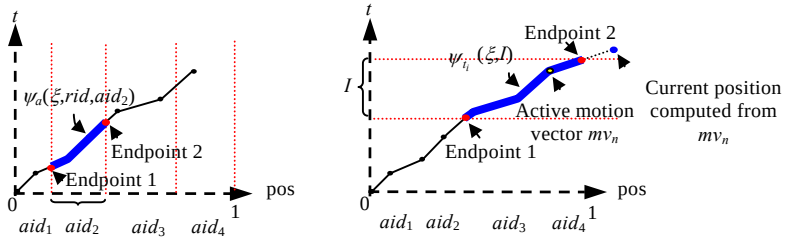
When dealing with a certain trajectory of ξ, if the intercepting point is between two consecutive motion vectors, then necessary interpolation is required to get the end points of the resulted parts (see Fig.6(a). For simplicity, in Fig.6 we only depict part of the trajectory corresponding to a certain route *route*(*rid*)). Besides, if the trajectory contains the active motion vector, then the system has to compute the current location of the moving object and append it to the trajectory first (see Fig.6(b)). After that, the computation can be conducted as for non-active trajectories.

Definition 24 (**project operations**). Suppose that ξ is a set of trajectories. Project operations compute the spatial or temporal projection of ξ. There are two project operations:

$\pi_t(\xi)$: to compute the temporal projection of ξ. The result is a set of time intervals;

$\pi_{geo}(\xi)$: to compute the spatial projection of ξ. The result is a set of network route sections.

Based on the above defined operations, in the following we describe the computation methods for the traffic parameters. Let's consider *route*(*rid*). Suppose that the statistical data source for *route*(*rid*) is $datasource=\{\xi_1,\xi_2,\ldots,\xi_n\}$, where ξ_i ($1\le i\le n$) is a moving object trajectory which describes the moving object's movement in *route*(*rid*) in the recent $\Delta\tau$ time period.



(a) Interception according to an ARS (b) Interception according to a time interval

Fig.6 Interception of moving object trajectory

(1) The number of the moving objects currently running in *ars*(*rid*,*aid*), denoted as η_{mo}.

$$\eta_{mo}=\left|\bigcup_{i=1}^{n}(\psi_{tp}(\psi_a(\xi_i,rid,aid),t_{now}))\right| \tag{1}$$

In Eq.(1), the result of $\bigcup_{i=1}^{n}(\psi_{tp}(\psi_a(\xi_i,rid,aid),t_{now}))$ is a set of motion vectors which describe the current moving states of the moving objects that are currently running in *ars*(*rid*,*aid*). Therefore, the element number of the set is η_{mo}.

(2) The average speed of the moving objects currently running in *ars*(*rid*,*aid*), denoted as v_p.

Let $\delta=\bigcup_{i=1}^{n}(\psi_{tp}(\psi_a(\xi_i,rid,aid),t_{now}))$. As explained above, δ is a set of motion vectors describing the current state of the moving objects which are currently running in *ars*(*rid*,*aid*). Suppose $\delta=\{mv_1,mv_2,\ldots,mv_m\}$, then v_p can be computed with the following formula:

$$v_p=\sum_{i=1}^{m}(|\,mv_i.\vec{v}\,|)\Big/m \tag{2}$$

(3) The travel time of *ars*(*rid*,*aid*), denoted by T_τ.

As stated earlier, the computation of travel time needs to involve historical trajectory information. Suppose that the time window size for the statistics of T_τ is Δt ($\Delta t\le\Delta\tau$), that is, the system will use trajectory information inside the time window $I=[t_{now}-\Delta t,t_{now}]$, where t_{now} stands for the current time instant.

Let $\delta=\bigcup_{i=1}^{n}(\psi_{t_i}(\psi_a(\xi_i,rid,aid),I))$, and from analysis we can see that δ is the set of trajectories which describe the movements of the moving objects which are running in *ars*(*rid*,*aid*) during the time interval *I*. Suppose that $\delta=\{tseg_1,tseg_2,\ldots,tseg_m\}$. For $tseg_i$ ($1\le i\le m$), the corresponding travel time τ_i can be computed as follows:

$$\tau_i=\frac{t_{end}(tseg_i)-t_{start}(tseg_i)}{|\,pos_{end}(tseg_i)-pos_{start}(tseg_i)\,|}\times len(ars(rid,aid)),$$

where the functions $t_{start}()$, $pos_{start}()$, $t_{end}()$, and $pos_{end}()$ return the starting time/position and end time/position of the trajectory respectively; and *len*() returns the length of the ARS.

Therefore, the average travel time of all moving objects passing through *ars*(*rid*,*aid*) in the time window *I* is:

$$T_\tau=\sum_{i=1}^{m}(\tau_i)\Big/m \tag{3}$$

(4) The flux of moving objects in *ars*(*rid*,*aid*), denoted as θ_{mo}.

Suppose that *npos*=(*rid*,*pos*) is the network position of the endpoint of *ars*(*rid*,*aid*), and the time window for statistics of θ_{mo} is $I'=[t_{now}-\Delta t',t_{now}]$ ($\Delta t'\le\Delta\tau$). Then θ_{mo} can be computed with the following formula:

$$\theta_{mo}=\left|\bigcup_{i=1}^{n}(\psi_p(\psi_{t_i}(\psi_a(\xi_i,rid,aid),I'),(rid,pos)))\right|\Big/\Delta t' \tag{4}$$

From analysis we can see that $\bigcup_{i=1}^{n}(\psi_p(\psi_{t_i}(\psi_a(\xi_i,rid,aid),I'),(rid,pos)))$ is the set of motion vectors corresponding to all moving objects passing through the endpoint of *ars*(*rid*,*aid*) during the time window I'. Therefore, the element number of the set divided by the duration is the number of moving objects passing through *ars*(*rid*,*aid*) per time unit, that is, θ_{mo}.

(5) The traffic jam state of *ars*(*rid*,*aid*), denoted as β.

Let $I''=[t_{now}-\Delta t'',t_{now}]$ ($\Delta t''\leq\Delta\tau$) be the time window for the statistics of β. We first compute the jammed area of *ars*(*rid*,*aid*) as follows:

$$\alpha_{jam}=\bigcap_{i=1}^{n}(\pi_{geo}(\psi_v(\psi_{t_i}(\psi_a(\xi_i,rid,aid),I''),v_{slow}))),$$

where v_{slow} is a predefined slow-speed threshold.

We can see that α_{jam} is a section of *ars*(*rid*,*aid*) where all moving objects moves through with speed lower than v_{slow}. Therefore, If α_{jam} is not NULL, then *ars*(*rid*,*aid*) is blocked. Otherwise, no blockage exists. That is:

$$\beta=\begin{cases}\text{true, if } \alpha_{jam}\neq\varnothing\\ \text{false, if } \alpha_{jam}=\varnothing\end{cases} \tag{5}$$

For each traffic parameter of a certain ARS, whenever the new value $value_{comp}$ is computed, the system will compare $value_{comp}$ with the last reported value $value_{last}$. If the difference between them exceeds a certain predefined threshold, a database update will be conducted to append the new value to the *g_temporal* attribute of the dynamic route. In this way, the route records in the relational table are not refreshed every time when the parameter value changes. Instead, only when the change is significant enough, the database record is refreshed so that the performance can be improved.

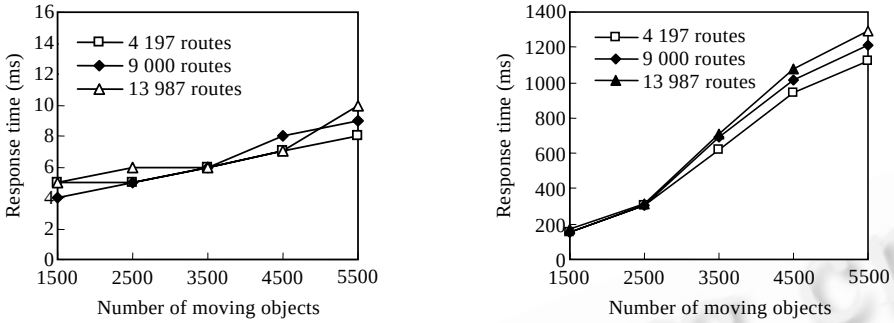
5 Implementation and Performance Evaluation

The above stated DTNMOD model has been implemented as a prototype within the PostgreSQL 8.2.3 extensible database system (with PostGIS 1.2.1 extension for spatial support), running on an IBM x205 with 17G hard disk, 256M memory, and Fedora Core 4 Linux operating system. PostgreSQL is an open source object-relational database system which supports the extension of the DBMS kernel with new data types and operators so that it is possible to accommodate complicated data types and operations defined in this paper.

The DTNMOD data types and operations have been implemented in *C* as two algebra modules, dynamic graph algebra and moving object algebra, in PostgreSQL. With these two algebras, the database system is extended to support the modeling and querying of traffic networks and moving objects. Besides, we have designed a direct trajectory appending technique to speed up the location update procedure at the server end, so that location update messages can be appended to the corresponding moving graph point values directly instead of by executing database update commands.

To evaluate the query processing performance of the DTNMOD model, we have conducted a series of experiments based on the prototype system and some additionally implemented modules. The experiments are conducted based on the real GIS data of Beijing for the traffic network. The original data set is in MIF format and we have implemented a data transformation program to get the desired data format described in this paper. In order to test the performance of DTNMOD on different map scales, we generate three data sets with 4 197, 9 000, and 13 987 routes respectively by selecting different route levels. Besides, we have implemented a network constrained moving objects generator, NetMO-Generator, which can create moving object trajectories according to predefined traffic networks.

In the experiments, we mainly focus on the query response time. We choose query examples 3 and 4 of Section 3.2.3 to test the query performance for region queries (to search moving objects inside a region) and for join queries (to search moving object pairs). In order to test the "pure" query processing performances, we do not utilize any index structures for moving objects, even though traffic networks are indexed with B+ tree (on *jid* and *rid*) and R-tree (on *geo*) respectively. The experimental results are shown in Fig.7.



(a) Average response time for region queries (b) Average response time for join queries

Fig.7 Average query response time of the DTNMOD model

From Fig.7(a) we can see that region queries can be processed efficiently in the DTNMOD model. When the number of moving objects reaches 5 500, the average response time is around 10 ms, which is quite acceptable for typical querying users. Besides, with the map scale increasing, the query response time only increases slightly. This is because the route/junction tables are indexed with B-tree on the *rid*/*jid* attributes so that the transformation from (*rid*,*pos*)/*jid*→(*x*,*y*) can be finished very quickly.

From Fig.7(b) we can see that the response time for join queries increases when the number of moving objects going up. Nevertheless, the response time is still acceptable when the number of moving objects amounts to 5 500. It is obvious that an effective index structure will dramatically speed up the query processing for join queries, and we are currently developing such an index structure, which is discussed in another paper.

6 Conclusion

One of the key research issues with moving objects databases (MOD) is the modeling of moving objects. In this paper, a new moving objects database model, Moving Objects on Dynamic Transportation Networks (DTNMOD), is proposed. In DTNMOD, transportation networks are modeled as dynamic graphs and moving objects are modeled as moving graph points. Compared with previously proposed MOD models, DTNMOD has the following features:

(1) The transportation network framework is two-layered (route-plus-ARS), which is suitable both for location updates and for traffic parameter expression and traffic-aware navigation purposes;

(2) The transportation network is modeled as dynamic, and traffic flow analysis can be conducted in real-time, so that the whole temporal history of traffic parameters, state changes, and topology changes of the networks can be tracked automatically and queried;

(3) The data types and operations are defined directly on network components so that the model is more suitable to be implemented in common extensible DBMSs such as PostgreSQL and SECONDO. Currently the DTNMOD model has been implemented in PostgreSQL and the experimental results show good performance in processing region and join queries.

Based on the DTNMOD system, we have implemented a taxi-positioning system deployed at Shenzhen InTech

Co., Ltd, and a traffic condition statistical analysis system deployed at Beijing Satcom ITS Co., Ltd, which show satisfactory usability of the proposed model.

In the future research, we will deal with the interaction between moving objects and other data types such as spatial data and location dependent data (LDD). We have implemented the LDD module in ProstgreSQL so that all the above mentioned data types can work together under the same DBMS framework. Besides, we will also deal with spatio-temporal data warehouse and OLAP, and MOD-based data mining.

Acknowledgements The author would like to thank Prof. Ralf Hartmut Güting of Fernuniversität in Hagen, Germany, and Prof. Xiaofeng Meng of Renmin University of China for their valuable discussions and advices on the first version of this paper. Also thank Dr. Man Li for her valuable work in the experiments.

References:

[1] Sistla AP, Wolfson O, Chamberlian S, Dao S. Modeling and querying moving objects. In: Gray WA, Larson PA, eds. Proc. of the 13th Int'l Conf. on Data Engineering (ICDE'97). Washington: IEEE Computer Society Press, 1997. 422−432.

[2] Su J, Xu H, Ibarra O. Moving objects: Logical relationships and queries. In: Jensen CS, Schneider M, Seeger B, Tsotras VJ, eds. Proc. of the SSTD 2001. Berlin: Springer-Verlag, 2001. 3−19.

[3] Güting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, Vazirgiannis M. A foundation for representing and querying moving objects. ACM Trans. on Database Systems, 2000,25(1):1−42.

[4] Pfoser D, Jensen CS, Theodoridis Y. Novel approaches in query processing for moving object trajectories. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, Whang KY, eds. Proc. of the 26th VLDB. San Francisco: Morgan Kaufmann Publishers, 2000. 395−406.

[5] Vazirgiannis M, Wolfson O. A spatiotemporal model and language for moving objects on road networks. In: Jensen CS, Schneider M, Seeger B, Tsotras VJ, eds. Proc. of the SSTD 2001. Berlin: Springer-Verlag, 2001. 20−35.

[6] Speicys L, Jensen CS, Kligys A. Computational data modeling for network-constrained moving objects. In: Proc. of the 11th ACM Int'l Symp. on Advances in Geographic Information Systems (GIS 2003). New York: ACM Press, 2003. 118−125.

[7] Almeida V, Güting RH. Indexing the trajectories of moving objects in networks. GeoInformatica, 2005,9(1):33−60.

[8] Güting RH, Almeida VT, Ding ZM. Modeling and querying moving objects in networks. VLDB Journal, 2006,15(2):165−190.

[9] Güting RH, Almeida V, Ansorge D, Behr T, Ding ZM, Höse T, Hoffmann F, Spiekermann M. SECONDO: An extensible DBMS platform for research prototyping and teaching. In: Proc. of the 21st Int'l Conf. on Data Engineering (ICDE 2005). Washington: IEEE Computer Society Press, 2005. 1115−1116.

[10] Ding ZM, Güting RH. Modeling temporally variable transportation networks. In: Lee YJ, Li J, Whang KY, Lee D, eds. Proc. of the 9th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2004). Berlin: Springer-Verlag, 2004. 154−168.

[11] Ding ZM, Güting RH. Managing moving objects on dynamic transportation networks. In: Proc. of the 16th Int'l Conf. on Science and Statistical Database Management (SSDBM 2004). Washington: IEEE Computer Society Press, 2004. 287−296.

[12] Ding ZM, Zhou X. Location update strategies for network-constrained moving objects. In: Haritsa JR, Ramamohanarao K, Pudi V, eds. Proc. of the 13th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2008). Berlin: Springer-Verlag, 2008. 644−652.

DING Zhi-Ming was born in 1966. He is a Ph.D. and professor at the Institute of Software, the Chinese Academy of Sciences and a CCF senior member. His current research areas are database systems, mobile computing and information retrieval.