Modeling Temporally Variable Transportation Networks*

Zhiming Ding and Ralf Hartmut Güting

Praktische Informatik IV
Fernuniversität Hagen, D-58084 Hagen, Germany
{zhiming.ding, rhg}@fernuni-hagen.de

Abstract. In this paper, a State-Based Dynamic Transportation Network (SBDTN) model is presented, which can be used to describe the spatio-temporal aspect of temporally variable transportation networks. The basic idea of this model is to associate a temporal attribute to every node or edge of the graph system so that state changes (such as traffic jams and blockages caused by temporary constructions) and topology changes (such as insertion and deletion of nodes or edges) can be expressed. Since the changes of the graph system are discrete, the temporal attribute can be expressed as a series of temporal units and each temporal unit describes one single state of the node or edge during a certain period of time. 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.

Keywords. Spatio-temporal, Database, Moving Object, Algebra.

1 Introduction

The management of moving objects has been intensely investigated in recent years. However, the interaction between moving objects and the underlying transportation networks has been largely ignored. To explore this relationship by involving transportation networks into the modeling of moving objects is one of the main aims of the research project "databases for moving objects", which we participate in. Obviously, the first step along the research line is to model transportation networks themselves.

The work described in this paper arose from the observation that in many moving objects database (MOD) applications, not only the moving objects are "dynamic", but the underlying transportation networks are "dynamic" as well - new routes can be added into the network and existing routes can be blocked or become obsolete. Therefore, we need a mechanism to model the "dynamic" aspect of the transportation networks. For simplicity, we will call temporally variable transportation networks "dynamic transportation networks", or simply "dynamic graphs" throughout this paper.

* This research was supported by the Deutsche Forschungsgemeinschaft (DFG) research project "Databases for Moving Objects".

In the literature, a lot of strategies have been proposed to model the spatio-temporal aspect of geographical data. Rasinmäki in [9] has proposed a valid-period based method in which every object is associated with a pair of time stamps, one for the time of creation and one for cessation. Hamre in [6] has presented a snapshot-based model, in which the state of the world is given at regular or irregular intervals as different snapshots. Besides, Peuquet *et al*. in [8] have proposed an Event-oriented Spatio-Temporal Data Model (ESTDM). However, most of these methodologies are focused on general graph problems without considering the next step modeling of moving objects. As a result, they can not be used directly for the modeling of dynamic transportation networks which has some unique requirements.

In modeling dynamic transportation networks, one of the most important requirements is that every node or edge of the graph system should have a unique identifier associated so that the locations of moving objects can be presented by referring to these identifiers. For each node or edge, its identifier should keep constant during its whole life time, no matter whether it is opened, closed, or blocked (see Section 2). This is important because otherwise, moving objects on the same physical edge can have different edge identifiers associated at different time instants so that logically related information can be scattered among several different places in the database, making identifier-based queries hard to be processed.

Another requirement is that a mechanism should be provided to deal with blockages. In a transportation system, blockages can happen quite frequently due to car accidents, temporary constructions, and even heavy traffic jams. In the MOD system, blockages can greatly change the behavior and decision of moving objects so that they should be tracked and taken into account in evaluating query results.

In order to meet these requirements, we propose a State-Based Dynamic Transportation Network (SBDTN) model in this paper. The basic idea is to associate a temporal attribute to every node or edge of the graph system so that its state at any time instant can be retrieved. Since the changes of the graph system are discrete, we can use a series of temporal units to represent a temporal attribute with each temporal unit describing one single state during a certain period of time. In this way, the whole state and topology history of the graph system can be presented and queried.

The remaining part of this paper is organized as follows. Section 2 formally defines the data types of the SBDTN model, Section 3 defines the signatures of the corresponding operations, Section 4 provides a group of query examples, and Section 5 finally concludes the paper.

2 Data Types of the SBDTN Model

In this and the next section, we formally define the SBDTN model. Obviously, to make this model readable and clean, it is crucial to have a formal specification framework which allows us to describe widely varying data models and query languages. Such a specification framework, called second-order-signature, was proposed in [3]. The basic idea is to use a system of two coupled signatures where the first signature describes a type system and the second one an algebra over the types of the first signa-

ture. In the following discussion, we will define our model with the second-order-signature, and especially, we will focus on the discrete model so that the data types and operations defined in this paper can be implemented directly in an extensible database system such as Secondo [1] to obtain a complete data model and query language. The notation of the definitions will follow those described in [4].

This section will be focused on the type system of the SBDTN model. We suppose that in the whole database system, several logically independent graphs may coexist. This assumption is necessary because in many real-life applications, moving objects can traverse several transportation networks during one single journey so that multiple graphs can be involved.

2.1 Overview of the Type System

Table 1 presents the type system of the SBDTN model. Type constructors listed in Group 1 have been defined and implemented in our earlier work [4, 2, 7] and we will use them directly without redefinition in this paper. In the following discussion, we will focus on the type constructors listed in Group 2.

Table 1. Signature describing the type system of SBDTN

Group	Type constructor	Signature	
1	*int, real, string, bool*		→ BASE
	point, points, line, region		→ SPATIAL
	instant		→ TIME
	range	BASE ∪ TIME	→ RANGE
2	*blockage, blockreason, blockpos*		→ GBLOCK
	statedetail, state		→ GSTATE
	temporalunit,temporal, intimestate		→ GTEMPORAL
	dynnode, dynedge, dyngraph		→ DGRAPH
	gpoint, gpoints, gedgesect, gline, gregion		→ GSPATIAL

Among the data types listed in Group 2, graph blockage (GBLOCK) data types and graph state (GSTATE) data types are used to describe the state of nodes or edges of the graph system. Graph temporal (GTEMPORAL) data types are used to track the state history and also the life span of a node or an edge. Dynamic graph (DGRAPH) data types enable us to define nodes, edges, and graphs. Graph spatial (GSPATIAL) data types can be used to describe static spatial objects residing on the graph system, which form a basis for our next step modeling and querying of moving objects.

2.2 Graph State Data Types and Graph Blockage Data Types

Graph state data types and graph blockage data types are used to describe the state of a node or an edge. In dynamic transportation networks, a node can have two states, opened and closed, while an edge can have three states, opened, closed, and blocked. If a node or an edge is opened, then it is entirely available to moving objects. If a node

or an edge is closed, then it is entirely unavailable to moving objects, which means that no moving objects are allowed to stay or move in any part of it. A closed node or edge is not deleted from the system. Instead, it is only temporarily unavailable to moving objects and can be reopened afterwards.

The blocked state is used to describe a special kind of state of an edge, which means "partially available" to moving objects. That is, the unblocked part of the edge is still available to moving objects, but no moving objects can move through the blocked part. Figure 1 gives an example of blocked edge.

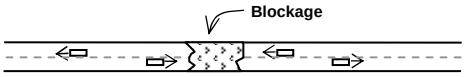


Fig. 1. A blocked edge with moving objects

Definition 1 (state) The carrier set of the *state* data type is defined as follows:

D_{state} = {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. We suppose that the total length of the road section is 1, and then every location in the road section can be represented by a real number $p \in [0, 1]$. 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 2 (blockage reason) The data type *blockreason* describes the reason of a blockage, and its carrier set is defined as follows:

$D_{blockreason}$ = {temporal-construction, traffic-jam, car-accident, others}

Definition 3 (interval) Let $(S, <)$ be a set with a total order. Intervals and closed intervals over S can be defined as follows:

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

$cinterval(S) = \{(s, e, lc, rc) \mid s, e \in S, s \leq e, lc = rc = true\}$

where *lc* and *rc* are two flags indicating "left-closed" and "right-closed" respectively. The semantics of the interval definitions can be found in [2].

Definition 4 (blockage position) The data type *blockpos* is used to describe the position of a blockage, and its carrier set is defined as follows:

$D_{blockpos} = \{ \Psi \mid \Psi \in cinterval([0,1])\}$

In Definition 4 we assume that a blockage can not move during its life time. In most cases this is true. However, sometimes, a blockage can also be "dynamic" if we take the blockages caused by floods or parades into consideration. In these cases, a blockage should be modeled as a moving interval over [0, 1] and can be handled by using the techniques proposed in [4], which is out of the scope of this paper.

Definition 5 (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 edge. Their carrier sets are defined as follows:

$D_{blockage} = \{(br, \Psi) \mid br \in D_{blockreason}, \Psi \in D_{blockpos}\}$

$D_{blockages} = \{B \mid B \subseteq D_{blockage}\}$

Definition 6 (state detail) The data type *statedetail* is used to describe the detailed state of a node or an edge, and its carrier set is defined as follows:

$D_{statedetail} = \{(s, B) \mid s \in D_{state}, B \in D_{blockages}, s \neq \text{blocked} \Leftrightarrow B=\varnothing\}$

Definition 6 is based on the fact that several blockages can exist in one road section at the same time so that they should be described as a set of blockages instead of a single blockage value.

2.3 Graph Temporal Data Types

Graph temporal data types are used to track the state history, and also the life span of a node or an edge. During its life time, a node or an edge can discretely assume a series of states, and each state can last for a certain period of time.

Definition 7 (temporal unit) The *temporalunit* data type describes the state of a node or an edge during a certain time period. Its carrier set is defined as follows:

$D_{temporalunit} = \{(I, sd) \mid I \in interval(D_{instant}), sd \in D_{statedetail}\}$

Definition 8 (temporal) The *temporal* data type is defined as a sequence of temporal units which describe the state history of a node or an edge. Its carrier set is defined as follows:

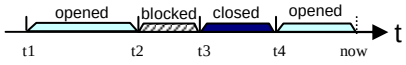
$D_{temporal} = \{<\mu_1, \ldots, \mu_n> \mid n \geq 1,\ \mu_i = (I_i, sd_i) \in D_{temporalunit}\ (1 \leq i \leq n)$, and:

(1) $\forall i, j \in \{1, \ldots n\}, i \neq j: I_i \cap I_j = \varnothing$

(2) $\forall i \in \{1, \ldots n\text{-}1\}: I_i \lhd I_{i+1}$ ($\lhd$ means "before" in time series)}

For a certain temporal unit $\mu_i = (I_i, sd_i)$ $(1 \leq i \leq n)$, I_i is composed of two time instant values $\min(I_i)$ and $\max(I_i)$, which indicate the starting point and the endpoint of I_i respectively. $\min(I_i)$ must be a defined value while $\max(I_i)$ can be either defined or undefined. If $\max(I_i)$ is an "undefined" value $\perp$, then I_i is called an open temporal unit. Otherwise, it is called a closed temporal unit. Semantically, $\perp$ means "until now". Therefore, if a node or edge is still active in the transportation network, its temporal attribute will contain exactly one open temporal unit, which forms its last temporal unit. Otherwise, if it has been deleted from the transportation network, then its temporal attribute will only contain closed temporal units.

The insertion and deletion time of a node or an edge can be decided by $\min(I_1)$ and $\max(I_n)$ respectively. In this way, we can decide the topology of the graph system at any time instant. Figure 2 illustrates an example temporal attribute value.



[t1, t2), (opened, ∅)
[t2, t3), (blocked, {(traffic-jam, [0.2,0.3])})
[t3, t4), (closed, ∅)
[t4 , ⊥), (opened, ∅)

a) state changes of an edge b) the corresponding temporal units

Fig. 2. An example temporal attribute value

Definition 9 (intimestate) The *intimestate* data type is used to describe the state of a node or an edge at a certain time instant. Its carrier set is defined as follows:

$D_{intimestate} = \{(t, sd) \mid t \in D_{instant}, sd \in D_{statedetail}\}$

2.4 Dynamic Graph Data Types

In SBDTN, transportation networks are modeled as dynamic graphs with every node or edge associated with a *temporal* attribute which describes its state history.

Definition 10 (dynamic node) A dynamic node can be considered as a normal graph node with a temporal attribute associated. The carrier set of the *dynnode* data type is defined as follows:

$D_{dynnode} = \{(nid, pos, tp) \mid nid \in D_{int}, pos \in D_{point}, tp \in D_{temporal}\}$

where *nid* is the identifier of the dynamic node which is isomorphic to integer, *pos* is a point value which describes the position of the node, and *tp* is the temporal attribute associated with the node.

Definition 11 (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.

$polyline = \{<p_1, p_2, ..p_n> \mid n \geq 2, \ \forall i \in \{1, \ldots n\} : p_i \in D_{point}\}$

Definition 12 (dynamic edge) A dynamic edge can be viewed as a normal graph edge with a temporal attribute associated. The carrier set of the *dynedge* data type is defined as follows:

$D_{dynedge} = \{(eid, nid_f, nid_t, route, tp) \mid eid, nid_f, nid_t \in D_{int}, route \in polyline, tp \in D_{temporal}\}$

where *eid* is the identifier of the edge, nid_f and nid_t are identifiers of the "from" node and the "to" node of the edge respectively, *route* is a polyline which describes the geographical shape of the edge, and *tp* is the temporal attribute associated with the edge.

In the above definition, we use a polyline instead of a *line* value (a *line* value is defined as a set of line segments [4] so that it can also be used to describe curves in the X×Y plane) to describe the route of an edge because we need the order of the line segments indicated in the polyline to transform the location expressed by a real number $p \in [0, 1]$ to the corresponding Euclidean coordinate value and vice versa.

From the above definition we can see that, since the *route* attribute is represented by a polyline, a dynamic edge can actually assume a shape of complicated curve in the X×Y plane instead of just a straight line.

Definition 13 (dynamic graph) A dynamic graph, *G*, is composed of a set of dynamic nodes and a set of dynamic edges. The carrier set of the *dyngraph* data type is defined as follows:

$D_{dyngraph} = \{(gid, N, E) \mid gid \in D_{int}, N \subseteq D_{dynnode}, E \subseteq D_{dynedge}$, and:

(1) $\forall e \in E: \exists v_1, v_2 \in N \ \wedge \ \text{from}(e) = v_1 \wedge \text{to}(e) = v_2$

(2) $\forall v \in N: \exists e \in E \ \wedge \ (\text{from}(e) = v \vee \text{to}(e) = v)\}$

In implementation, *N* and *E* can be implemented as relational tables so that in a *dyngraph* value only the relation names are kept (see Subsection 2.6).

In a dynamic graph system, since every node or edge has a temporal attribute associated, we can know its state at any given time instant. This is very useful in moving objects databases since a lot of queries can only be processed efficiently by accessing the states of the transportation networks. For instance, "please tell me all the edges which are currently blocked by traffic jams". Besides, through the temporal attribute, we can also know the life span of any node or edge of the graph system so that the

topology changes of the transportation networks can also be expressed and queried. For instance, "find the shortest path from *a* to *b* at time instant *t*".

2.5 Graph Spatial Data Types

Based on the above definitions for dynamic transportation networks, we can then define some useful data types, *gpoint*, *gpoints*, *gedgesect*, *gline*, and *gregion*, which form the basis for the modeling and querying of moving objects.

Definition 14 (graph point, graph points) The *gpoint* data type describes a point inside the graph system, and the *gpoints* data type describes a set of graph points. Their carrier sets are defined as follows:

$D_{gpoint} = \{(gid, eid, pos) \mid gid, eid \in D_{int},\ pos \in [0, 1]\}$

$D_{gpoints} = \{PS \mid PS \subseteq D_{gpoint}\}$

In the definition of graph point, *gid* and *eid* together decide a unique edge *e* of the graph system, while $pos \in [0, 1]$ indicates a position inside *e*. If $pos = 0$ or 1, then the graph point actually coincides with a node. Since node information can be retrieved from edges, we only need to represent edge identifiers here to make the model clean.

Definition 15 (graph edge section) The *gedgesect* data type represents a section of an edge. Its carrier set is defined as follows:

$D_{gedgesect} = \{(gid, eid, S) \mid gid, eid \in D_{int},\ S \in \text{cinterval}([0, 1])\}$

Definition 16 (graph line) A graph line is defined as a consecutive chain of edge sections inside the graph system. Its carrier set is defined as follows:

$D_{line} = \{< \omega_i >_{i=1}^{n} \mid n \geq 1, \omega_i = (gid_i, eid_i, S_i) \in D_{gedgesect}$ where:

(1) $\forall i \in \{2, \ldots n\text{-}1\} : S_i = [0, 1]$;

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

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

where $\text{adjacent}(\omega_i, \omega_{i+1})$ means that ω_i, ω_{i+1} meet with their end points spatially so that all edge sections in the graph line should form a chain.

Definition 17 (graph region) A graph region is defined as a set of graph edge sections. The carrier set of the *gregion* data type is defined as follows:

$D_{gregion} = \{ W \mid W \subseteq D_{edgesection}\}$

Different from graph line, graph region can be formed by a set of arbitrary graph edge sections.

2.6 An Application Example

In this subsection, we give an example to show how the data types defined above can be used as attribute types in defining database schemas. We suppose that the transportation networks for automobiles in the Verkehrsverbund Rhein-Ruhr (VRR) area of Germany are composed of one highway network and *N* city street networks. The highway network connects different cities while each city network corresponds to the transportation network inside a city. The highway network and the street networks can overlap each other and moving objects can transfer from one network to another via

geographically overlapped nodes (such as from the node “FernUni-HagenNet” to the node “FernUni-HighwayNet”), as illustrated in Figure 3.

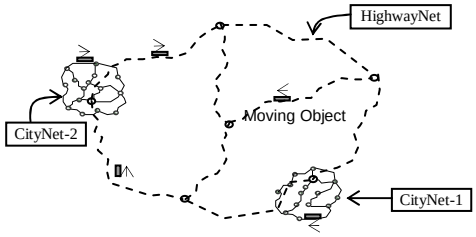


Fig. 3. Transportation networks

In the database system, all these transportation networks can be presented in a relation **DynGraphs** with each tuple of the relation describing an independent graph:

DynGraphs(gname: *string*, dgraph: *dyngraph*);

For each dynamic graph (say HagenNet), the corresponding dynamic nodes and dynamic edges are also implemented as independent relations so that they can be expressed by two relations, one for nodes and one for edges:

HagenNodes(dnode: *dynnode*); **HagenEdges**(dedge: *dynedge*);

As a result, in the dgraph attribute of the **DynGraphs** relation, only the relation names (for instance, HagenNodes and HagenEdges) are stored. Besides, we suppose that there are several auxiliary relational tables in the system, which allow us to translate logic node, edge, or graph names to the corresponding identifiers.

Table 2. Operations of the SBDTN model

Group	Class	Operations
1	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**
	Time type specific	**year, month, day, hour, minute, second, period**
2	Transformation	**euc_graph, graph_euc, node_edge, edge_nodefrom edge_nodeto, dyngraph, dynode, dynedge**
	Construction	**gedgesect, gpoint**
	Data Extraction	**getnodes, getedges, id, pos, route, temporal, atinstant statedetail, state, blockages, blocksel, blockpos**
	Truncation	**atperiods, present, at**
	When & Projection	**when, deftime**
	Graph General	**snapshot, shortestpath**
	Range Specific	**intervalnum, getinterval**

3 Operations of the SBDTN Model

3.1 Overview

Table 2 lists the operations of the SBDTN model. Operations in Group 1 have been defined and implemented in our earlier work [4, 2, 7, 5] so that we will use them directly without redefinition in this paper. In the following discussion, we will focus on the operations listed in Group 2.

3.2 Transformation Operations

The signatures of transformation operations are listed in Table 3. For the sake of readability, *gid*, *nid*, and *eid* are used instead of *int* in defining the signatures.

Table 3. Transformation operations

Operation	Signature		Operation	Signature		
euc_graph	*point*	→ *gpoints*	**node_edge**	*gid* × *nid*	→	*set*(*eid*)
	points	→ *gpoints*	**edge_nodefrom**	*gid* × *eid*	→	*nid*
	region	→ *gregion*	**edge_nodeto**	*gid* × *eid*	→	*nid*
graph_euc	*gpoint*	→ *point*	**dyngraph**	*gid*	→	*dyngraph*
	gpoints	→ *points*	**dynnode**	*gid* × *nid*	→	*dynnode*
	gline	→ *line*	**dynedge**	*gid* × *eid*	→	*dynedge*
	gregion	→ *line*				

As shown in Table 3, transformation operations can be further divided into three groups. The first group of operations, **graph_euc** and **euc_graph**, transform Euclidean information to the corresponding graphical representation and vice versa. Since every node or edge is spatially embedded, that is, its geographical information is known to the system, it is easy to implement these two transformation operations.

Operations in the second group, **node_edge**, **edge_nodefrom**, and **edge_nodeto**, transform node identifiers to the associated edge identifiers and vice versa. These two operations are also not hard to be implemented since the needed information is contained in the corresponding *dynedge* values.

The third group of operations, **dyngraph**, **dynnode**, and **dynedge**, transform identifier representations to the corresponding graphical entities.

3.3 Construction Operations

Construction operations are used to construct *gpoint* and *gedgesect* values, which can be used in further queries of moving objects. Their signatures are listed in Table 4.

As shown in Table 4, there can be two ways to indicate the position in a dynamic edge when constructing *gpoint* and *gedgesect* values. One way is to give the 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.

Table 4. Construction operations

Operation	Signature	
gpoint	*gid* × *eid* × *real*	→ *gpoint*
	gid × *eid* × *point*	→ *gpoint*
gedgesect	*gid* × *eid* × *range*(*real*)	→ *gedgesect*
	gid × *eid* × *point*× *point*	→ *gedgesect*

3.4 Data Extraction Operations

Date extraction operations are relatively simple and their only functionality is to extract detailed information from a given object. Their signatures are listed in Table 5.

Table 5. Data extraction operations

Operation	Signature	
getnodes	*dyngraph*	→ *set*(*dynnode*)
getedges	*dyngraph*	→ *set*(*dynedge*)
id	*dynnode*	→ *nid*
	dynedge	→ *eid*
pos	*dynnode*	→ *point*
route	*dynedge*	→ *line*
temporal	*dynnode*	→ *temporal*
	dynedge	→ *temporal*
atinstant	*temporal* × *instant*	→ *intimestate*
statedetail	*intimestate*	→ *statedetail*
state	*statedetail*	→ *state*
blockages	*statedetail*	→ *blockages*
blocksel	*blockages* × *bloackreason*	→ *blockages*
blockpos	*blockages* × *bloackreason*	→ *blockages*

As stated earlier, since node or edge sets can be implemented as relational tables, the outputs of the operations **getnodes** and **getedges** are actually relation names.

3.5 Truncation and Projection Operations

Conceptually, the temporal attribute associated with a node or an edge is similar to a "moving statedetail" value with the exception that the statedetail value can only change discretely. The operations in the Truncation, When, and Projection classes are designed because of this similarity. The signatures of these operations are listed in Table 6.

The **atperiods** operator returns part of the temporal value which corresponds to the given time period. The **present** operation decides whether the temporal attribute is defined or not at a certain time instant. The **at** operation returns part of the temporal

value which corresponds to the indicated state. The **when** operation returns part of the temporal value which satisfies a certain condition. The **deftime** operation returns the defined time of the temporal value.

Table 6. Truncation, When, and Projection operations

Operation	Signature	
atperiods	*temporal* × *periods*	→ *temporal*
present	*temporal* × *instant*	→ *bool*
at	*temporal* × *state*	→ *temporal*
when	*temporal* × (*statedetail*→*bool*)	→ *temporal*
deftime	*temporal*	→ *periods*

3.6 Graph General Operations

Graph general operations include the **snapshot** operation and the **shortestpath** operation. Their signatures are defined in Table 7.

Table 7. Graph general operations

Operation	Signature	
snapshot	*dyngraph* × *instant*	→ *line*
	dynnode × *instant*	→ *point*
	dynedge × *instant*	→ *line*
shortestpath	*gpoint* × *gpoint* × *instant*	→ *gline*

The **snapshot** operation returns the snapshot of a node, an edge, or a graph at a certain instant of time. For a certain node or edge ζ, this operation needs to check the defined time of the temporal attribute first. If the temporal attribute is defined at the given time instant *t*, then the state of ζ at time *t* will be further checked. If ζ is opened, then its geometry will be output. If ζ is blocked (in this case, ζ must be an edge), then only the unblocked part of its geometry will be output while the blocked part will be taken away from the output, as illustrated in Figure 4.

a) a blocked edge b) the corresponding snapshot

Fig. 4. Snapshot of a blocked edge

If the temporal attribute of ζ is not defined at time *t*, or if ζ is in a closed state at time *t*, then the geometry of ζ will not be output. The snapshot of a graph can be obtained through the union of the snapshots of its component nodes and edges.

The **shortestpath** operation computes the shortest path between two graph points at a certain instant of time. The result of the operation depends on the topology of the graph system and also the states of the nodes and edges at the given time instant. Blocked edges are handled in a similar way as stated in the **snapshot** operation.

3.7 Range Specific Operations

A range value can be viewed as a set of disjoint intervals over a certain data type. We assume that $\delta \in \text{BASE} \cup \text{TIME}$ is a type variable. The signatures of the range specific operations **intervalnum** and **getinterval** can be defined as follows.

Table 8. Range specific operations

Operation	Signature	
intervalnum	*range*(δ)	→ *int*
getinterval	*range*(δ) × *int*	→ *range*(δ)

The **intervalnum** operation returns the number of intervals contained in a range value. The **getinterval** operation returns the *i*th interval of a range value. If *i* is a negative number, then the backward-counted *i*th interval will be returned.

4 Query Examples

In this section, we give some query examples which are based on the database schema described in Subsection 2.6. Since node, edge, and graph names can be translated into the corresponding identifiers through the auxiliary relations, we suppose that the identifiers are already known and put them in the brackets behind the logic names.

Example 1. "What did the Hagen street network (*gid*) look like at time instant *t*1?"

LET HagenAtT1=**snapshot**(**dyngraph**(gid))

This query returns the snapshot of the graph at the given time instant. The result is a *line* value which describes the topology of the graph at the indicated time instant.

Example 2. "Find all road sections in the Hagen street network (*gid*) which are currently blocked by traffic jams"

SELECT **id**(dedge) FROM **getedges**(**dyngraph**(gid)) WHERE **not**(**isempty** (**blockagesel** (**blockages**(**statedetail**(**atinstant**(**temporal**(dedge), NOW))),traffic-jam)));

This example shows how the information contained in a graph can be extracted. As stated in Subsection 2.6, we suppose that node sets and edge sets are implemented as relations so that the result of **getedges**(**graph**(gid)) is actually a relation name. NOW is a real number variable whose value equals to the current time instant.

Example 3. "When was the Berliner street (*gid*, *eid*) closed for the second time?"

SELECT **min**(**getinterval**(**deftime**(**at**(**temporal**(dedge),closed)),2))
FROM **getedges**(**dyngraph**(gid)) WHERE **id**(dedge)=eid;

In this query, the **at** operation returns part of the temporal attribute which corresponds to the closed state. The result is then projected to a *periods* value by the **deftime** operation. The operation **getinterval** returns the indicated time interval and the beginning time of the interval is returned as the final result.

Example 4. "For how many times was the Berliner street (*gid*, *eid*) in a closed state in the year 2000?"

SELECT **intervalnum**(**deftime**(**atperiods**(**temporal**(dedge),**year**(2000))
when[**state**(.)=closed])) FROM **getedges**(**dyngraph**(gid)) WHERE **id**(dedge)=eid;

This query first selects part of the temporal attribute according to the indicated year and the indicated state. Then the number of intervals in the result is output as the final result.

Example 5. "What is the state of the Hagener street (*gid*, *eid*) at time instant *t*1?"

SELECT **statedetail**(**atinstant**(**temporal**(dedge),t1))

FROM **getedges**(**dyngraph**(gid)) WHERE **id**(dedge)=eid;

In this query, the detailed state information of the street at the given time instant will be returned as the final result.

Example 6. "Which roads were deleted from the Hagen street network (*gid*) in 2002?"

SELECT **id**(dedge) FROM **getedges**(**dyngraph**(gid))

WHERE **max**(**deftime**(**temporal**(dedge))) **inside year**(2002)

This query shows how to decide the deletion time according to the temporal attribute. For nodes or edges which are now still active, the maximum defined time of the temporal attribute will be the undefined value (⊥).

Example 7. "Find all road sections in the Hagen street network (*gid*) which intersects with region *R* and were added in the year 2000"

SELECT **id**(dedge) FROM **getedges**(**dyngraph**(gid)) WHERE **min**(**deftime**(**temporal** (dedge))) **inside** year(2000) AND **intersects**(**route**(dedge), R);

This query is very similar to Example 6. Through the temporal attribute we can decide the insertion time. Besides, we can extract the geographical information from dynamic edges and the information can be used for further computations.

Example 8. "Which road sections in the Hagen street network (*gid*) were blocked at 2 or more places and when?"

SELECT **id**(dedge), **deftime**(**temporal**(dedge) **when** [**intervalnum**(**blockpos** (**blockages**(.)))≥ 2]) FROM **getedges**(**dyngraph**(gid)) WHERE **not**(**isempty**(**temporal**(dedge) **when** [**intervalnum**(**blockpos**(**blockages**(.)))≥ 2]));

This query shows the functionality of the **when** operation. Through this operation, the query can deal with complicated situations by specifying flexible conditions.

Example 9. "Find all building road sections in the Hagen street network (*gid*). For how long have they been in the building state for the last time?"

SELECT **id**(dedge), **duration**(**getinterval**(**deftime**(**temporal**(dedge) **when** [**not**(**isempty** (blockpos(**blocksel**(**blockages**(.), temporal-construction))))])),-1))

FROM **getedges**(**dyngraph**(gid)) WHERE **not**(**isempty**(**blockagesel**(**blockages** (**statedetail**(**atinstant**(**temporal** (dedge), NOW))),temporal-construction)));

This query is fairly complicated. Through the data extraction operations and the **when** operation, the information contained in the dynamic edges can be extracted and analyzed. Besides, the range specific operations enable us to take just one specific interval value for processing.

Example 10. "What is the shortest path from point *p*1 in the Hagener street (*gid*1, *eid*1) to point *p*2 in the Düsseldorfer street (*gid*2, *eid*2) currently?"

LET H=**gpoint**(gid1,eid1,p1); LET D=**gpoint**(gid2,eid2,p2);

LET ShortestHD=**shortestpath**(H, D, NOW)

In this query, the time instant NOW must be specified for the **shortestpath** operation since the result can vary from time to time because of state changes and topology changes of the graph system.

5 Conclusion

In this paper, we have presented a State-Based Dynamic Transportation Network (SBDTN) model which can be used to describe the spatio-temporal aspect of temporally variable transportation networks. In this model, every node or edge of the graph system is associated with a temporal attribute which is composed of a series of temporal units. Each temporal unit describes the state of the node or edge during a certain period of time. In this way, the whole state and topology history of the graph system can be presented. The model is aimed to better support the next step modeling of moving objects in transportation networks. During its whole life cycle, every node or edge can keep a constant identifier. Besides, state changes such as closures and blockages can be expressed and queried.

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. These data types and operations are designed as a discrete model which offers a precise basis for the implementation of data structures in an extensible DBMS such as Secondo. Secondo is a new generic environment supporting the implementation of database systems for a wide range of data models and query languages. In the Secondo system, the data types and operations defined above can be implemented in C++ as two algebra modules, spatial algebra and dynamic graph algebra, so that the database system is extended to support the modeling and querying of temporally variable transportation networks.

References

1. Dieker S., Güting R.H., Plug and Play with Query Algebras: SECONDO. A Generic DBMS Development Environment. Proc. of the Int. Database Engineering and Applications Symp. (IDEAS 2000), September 2000.
2. Forlizzi L., Güting R.H., Nardelli E., Schneider M., A Data Model and Data Structures for Moving Objects Databases. Proc. ACM SIGMOD Conf. (Dallas, Texas) May 2000.
3. Güting R. H., Second-Order Signature: A Tool for Specifying Data Models, Query Processing, and Optimization. In: Proc. ACM SIGMOD Conference. Washington, USA, 1993.
4. Güting R. H., Böhlen M. H., Erwig M., Jensen C. S., Lorentzos N. A., Schneider M., Vazirgiannis M.. A Foundation for Representing and Querying Moving Objects. ACM Transactions on Database Systems, 25(1), 2000.
5. Güting R. H., Schneider M.. Moving Objects Databases. FernUniversität Hagen, Course, 2003.
6. Hamre T., Development of semantic spatio-temporal data models for integration of remote sensing and in situ data in a marine information system. PhD thesis, University of Bergen, Norway, 1995. Available at ftp://fritjof.nrsc.no/pub/publications/Hamre/nersc_tr106.tar.gz
7. Lema J. A. C., Forlizzi L., Güting R. H., Nardelli E., Schneider M., Algorithms for Moving Objects Databases. The Computer Journal, 46(6), 2003
8. Peuquet D. J., Duan N., An Event-based Spatiotemporal Data Model for Temporal Analysis of Geographic Data. International Journal of Geographic Information Systems 9, 1995.
9. Rasinmäki J., Modelling spatio-temporal environmental data, 5th AGILE Conference on Geographic Information Science, Palma (Balearic Islands, Spain) April, 2002.