User View of Spatial Networks in Spatial Database Systems

Virupaksha Kanjilal & Markus Schneider*

University of Florida, Gainesville, FL 32611, USA,
{vk4, mschneid}@cise.ufl.edu

Abstract. Spatial networks find application in the areas of transportation GIS, network analysis, city planning and others. To effectively use them, it is essential to store spatial networks inside database systems. This paper shows how a user would conceptually view and make use of spatial networks in a database. The discussion has been based on a spatial network data type called *snet* which can store the geometry and the semantics of a network as a single object.

1 Introduction

There is an increasing interest in spatial networks among the researchers in the areas of transportation GIS, network analysis, moving objects databases etc. Road networks, river networks, pipeline networks and other similar networks, which are characterized by a spatial embedding are examples of spatial networks. The increasing interest and use of spatial networks promises the generation of huge amounts of spatial network data which can be stored and handled efficiently only by a database system. Current implementations of spatial networks store the basic components of a spatial network in a database and create an in-memory graph data structure to represent the network. This approach cannot take advantage of the database features like concurrency control or transaction processing. More importantly this approach stores a network by putting it's components into various tables. This makes it impossible to formulate a number of queries. In order to efficiently store spatial network data and perform analysis such as querying, manipulation and other operations on it necessitates the creation of a spatial network data type and incorporating it in available spatial databases. The spatial network data type will enable natively storing spatial networks and running operations on them.

GIS applications demand that spatial database systems, which support GIS applications should store both thematic data as well as geometric data of spatial objects in addition to providing data management operations. So databases should be extended to support spatial network objects. An abstract model of spatial networks built on the concept of infinite point set has been described in [6, 5]. This model is a generic one which means it can represent any type of spatial network. By using *labels* to mark thematic attributes, it cannot only store

* This work was partially supported by the National Science Foundation under grant number NSF-IIS-0812194.

the geometry and topology of the network but also the semantics and the thematic information therein. The data model presents a careful abstract design of a spatial data type and the semantics of related operations. This model serves as a high-level specification for implementation of spatial networks on computers. As computers can only allow discrete representation, the abstract model cannot be directly implemented on computer systems. Based on the abstract specification of spatial networks, this paper shows how a spatial network data type can be elegantly modeled at a logical level by a spatial database type attribute called *snet*. We call this logical model the 'user view' or the 'database view' and the conveys how a user would conceptually perceive a network object in the database. A single data *snet* object encapsulates an entire spatial network. This conceptual view not only makes the modeling of spatial networks simpler, but also allows various types of queries to be formulated. We demonstrate an SQL-like *Spatial Networks Query Language* (*SNQL*) which may be used to create, access, query, and manipulate a spatial networks within a database system and run operations on it. In order to effectively capture both the spatial, topological configuration, and the semantics of a spatial network, a simple mapping of a network to a mathematical graph is not sufficient. Instead, it is conceptually broken down into simpler components which are finally combined to create a data type termed as *snet* representing a spatial network. *Snet* stores the entire spatial network as a single object so the data for a network need not be spread across several tables and queries can be answered by accessing only a single field in a database table. We specify *snet* as an abstract data type. An abstract data type hides its internal data structure. Retrieval and manipulation of its values can only be performed by high-level operations that conceal how tasks are accomplished. This strategy helps us provide an user interface for the spatial network data structures without specifying the internal structure.

Previous models to implement spatial networks in computer systems has been briefly described in Section 2. The user view of the spatial network in a database along with *Spatial Network Query Language* has been described in Section 3. Future improvements to this model has been discussed in Section 4.

2 Related Work

Graph based approachs to model spatial network [2] simplifies a network to a graph. These approaches completely loses the geometric information in the spatial networks and only maintains the topology of the network. Moreover this solution does not allow the spatial networks to have attributes of their own. Though effective, it has been shown that this method of modeling a spatial network is neither elegant nor robust [9]. [4] provides an elaborate graph-like network data model with three external, user-accessible, representations of the transportation infrastructure, namely the kilometer post, link-node, and geographic representations. Oracle Spatial [7] has a graph-based implementation of a single generic data model for storing spatial networks where the actual network data is separated as links and nodes and are spread across various tables. It has a set of Java API which acts as the middleware layer and provides the functions to access and manage the network data. This kind of data model does not

recognize the network as a single entity and hence various parts of the network have to be scattered into a number of tables when stored in a database. This is highly undesirable as common operations (for example, "Do two networks geometrically intersects") that involve more than one network require the reading of more than one set of database tables. Relational algebra, which is a formal system for manipulating relations or tables in a database, cannot model spatial network queries since it does not have any spatial operators built into it. Moreover, spatial operations cannot be implemented using relational algebra since it is not a full-fledged computer language. Thus it is impossible to formulate SQL queries on them. In order to be useful, these approaches use a *middleware* layer which performs an additional step of interrelating the various representations which means that there needs to be the ability to translate among the different external infrastructure representations. As the middleware layer is outside the database, all the features of the database, for example, transaction processing, concurrency control, querying, etc.are lost. Additionally, the interface they expose and the platforms they work on are very specific, making application development using them difficult. ESRI's ArcGIS software has specialized network data models [1] for different industries like pipeline, transportation etc. The data is managed by *Geodatabase* [8], which is a middle ware layer like object-relational construct for managing geographic data as features within an industry-standard relational database management system (RDBMS). Another approach taken by some (e.g.[3]) is using routes which correspond to roads or highways in real life and to paths in graphs. They are important conceptual entities as addresses are given relative to routes. Moreover, it is easy to relate network positions to routes.

In order to overcome the disadvantages, our model takes a single entity view of the network, which means all the primitive components and representations of a spatial network are combined into a single data type. This allows a spatial network to be stored as a single entity in a database as a *first class citizen* and all the features of the database can be taken advantage of. Our emphasis in this paper is on an increase of functionality and elegance at the logical level, the propagation of the abstract data type approach in a database context, and the possibility of query support.

3 Database View of Spatial Networks

This section describes how a spatial network data type looks in a spatial database. First we describe what spatial networks are and how they can be modeled in a database using a *snet* data type in Section 3.1. Later we describe a query language called *Spatial Network Query Language* which may be used to create, access and manipulate them in a database (Section 3.2).

3.1 Description of Snet Objects

Spatial networks including transportation networks like road networks for cars, buses, and taxis or railway networks, water pipelines and power networks are a ubiquitous spatial concept. The primary purpose of spatial networks is to provide a spatially constrained environment for materials (in the broadest sense)

to move or flow through them. We will first familiarize ourselves with a set of components and features characteristic of all networks. Networks have linear component called *channels* through which flow of material takes place. These are the roads in a highway network or the pipes in a pipeline network. Each channel has a geometry as well as some thematic information attached to it. Examples of thematic information are the name of the road or the speed limit of a road in a highway network. It is assumed that a particular network has a set of attributes associated with it and the set defines the type of the network it is. These channels can intersect among themselves to create *junction points*. Intersection of two roads can be considered as a junction point. Sometimes, the channels cross over or under one another. These points are called *crossover points*. Bridges over a road are examples of crossover points. Since a network can have at a minimum of one channel, a single channel can also be considered a spatial network.

Spatial networks are directly stored in databases as a stand-alone entity as objects of the table attribute type *snet*. An entire *snet* object including the geometry, connectivity and semantics is stored as a single object in the database as a binary large object (blob). The blob has an internal structure which helps quick access and efficient execution of operations on the *snet* object. This mechanism avoids the need for making objects of *snet* data type dependent on multiple tables which in turn makes the data type *snet* a truly first class citizen of the database. The *snet* data type is described as an abstract data type in this paper so as to provide a user's view of spatial networks in databases without specifying how they are implemented. All user access to a spatial network has to be performed using the operators defined in the interface of *snet*. This strategy prevents the user from directly accessing and modifying the internals of a spatial networks object thus potentially rendering it inconsistent.

Snet is a table attribute type, which means we can declare tables having fields of type *snet*. Consider the case of the national power grids in the US, the grids themselves are a type of spatial network and consequently they may be represented as objects of *snet*. The national power grid in the US is categorized into three sectors: Eastern sector, Western sector and Texas sector. These network grids work independently of each other. The actual grids (including the geometry and the thematic data) are stored as objects of *snet* in a table *NatPowerGrids* which has the following schema: *NatPowerGrids(sectorName: String, Grid: snet)*. The sector name is stored as a string under *sectorName* and the power grids (the geometry and the thematic data) are stored as *snet* under the attribute 'Grid'. The rows from the table *NatPowerGrids* are of the form:

("Eastern Sector", Grid1)
("Western Sector", Grid2)
("Texas Sector", Grid3)

Even though we consider a spatial network as a single entity, it is conceptually composed of a number of basic components (for example, channels) and in some real life applications it is interesting to access these parts. A set of operators return one or more channels in a network as an object of *snet*. Since the returned objects are of type *snet* too, all the operators defined for *snet* work on them

too. The most basic of these operators is *GetChannel*. This operator takes as argument a string which is the id_attr of the channel and returns the channel with the same id_attr. The signature of this operator is $GetChannel : snet \times id\_attr \rightarrow snet$. The *GetAllChannels* operation returns a set of *snet* objects each of which contains a single channel each corresponding to a channel in the original network.

The *snet* data type has a rich set of operations which may be used in *Spatial Network Query Language* introduced in Section 3.2. Below is a small set of operations defined on *snet* data type which has been taken from [5]. These operations take one or more spatial networks as argument as indicated by their signature. The meaning of the operators is intuitively described below.

$Length:$	$snet \rightarrow real$
$shortestPath:$	$snet \times point \times point \rightarrow snet$
$getGeometry:$	$snet \rightarrow line$
$getGeometry:$	$snet \times id\_attr \rightarrow sline$
$getAttribute:$	$snet \times string \times point \rightarrow value$
$GetChannel:$	$snet \times id\_attr \rightarrow snet$
$GetAllChannels:$	$snet \rightarrow 2^{snet}$
$Union:$	$snet \times snet \rightarrow snet$
$Intersect:$	$snet \times snet \rightarrow snet$
$Window:$	$snet \times region \rightarrow snet$
$Clipping:$	$snet \times region \rightarrow snet$

The *Length* operator returns the length of a channel in an snet as a numeric value. There are two versions of *getGeometry*() - the one with only one argument, returns the entire network as a spatial line represented as a *sline*, the second version of this operator takes also an id_attr as argument and returns a spatial line representing the channel which has the same id_attr. Attribute values at a particular point on the network can be obtained by the function *getAttribute*(). This operator takes as arguments the attribute whose value is being sought and the network point at which it should report the value. The operations *Union* and *Intersect* are geometric set operations which take two networks and produce a resulting network which is the geometric union and geometric intersection of the given networks respectively. Given a region and a spatial network, the operations *Window* and *Clipping* extract those parts of the network which intersect with the region. The operation *Window* allows a user to retrieve those *complete* channels of a spatial network whose intersection with a given (region) window is not empty. The *Clipping* operation is similar to the *Window* operation, but it returns only those parts of channels which actually intersect with the specified region.

3.2 Spatial Network Query Language

In this section we show how users can access, manipulate and operate on *snet* object in the database. We first discuss the type of queries possible in a spatial network without assuming that is is a part of a database and then show how SQL like constructs can be extended to provide querying capabilities.

Unlike traditional spatial data objects, which only contain the geometry, spatial networks contain geometry as well as thematic information. Thus a spatial

network query may not be purely based on the geometry. Four type of queries have been identified for spatial networks; First, they may be based on the spatial network geometry (*network queries*). This type include queries like "does the bus route 20 intersect with bus route 21". Second, queries may be based on components in a spatial network (*component queries*). These include queries like "How many distinct roads are there in Gainesville" or "Does the 13th Street intersect with the 1st avenue". Third, queries may be based on attributes associated with the spatial network components (*component attribute queries*). These include "What is the capacity of the oil pipe". Lastly, queries may be based on attributes associated with entire spatial network (*network attribute queries*). Example of this type of queries is "Which department administer the highways".

Even though we treat *snet* as an abstract data type object implying that we make no assumptions about how it is implemented, in order to explain the *SNQL*, we lay down some constrains on how the thematic information of a spatial network is stored. The thematic information of any channel in a network is contained in a *label* associated with the channel. These labels are of type *net_label_type* and is essentially a list with each item in the list associated with a *network label attribute*. The network label attribute identifies an attribute in the label. Assuming T = *set of all valid component datatypes for labels*, and A = *set of all possible network label attributes*, formally, a label type is a tuple of pairs : $LS = (a_1 : b_1, a_2 : b_2, \ldots, a_n : b_n)$ where $a_i \in T$ and $b_i \in A$. The first item in any network label is a required attribute called *id_attr* which is a string denoting the name of the channel.

The CREATE LABEL statement is used to create label types in the following manner, CREATE LABEL $L(a_1 : b_1, a_2 : b_2, \ldots, a_n : b_n)$ where $a_i \in T$ and $b_i \in A$. This statement creates a new *net_label_type* L. A label of type L is defined as a tuple $l = (c_1 \in a_1, c_2 \in a_2, \ldots, c_n \in a_n)$. As an example, the label type for the road network containing the speed limits and the number of lanes may be created in the following manner :

CREATE LABEL roadLabel(string: id_attr, real:speed_limit, integer: lanecount)

This statement creates a new *net_label_type* called 'roadLabel' containing three network label attributes: the id_attr of type string,'speedlimit' of type real, and number of 'lanes' of type integer.

In order to create spatial networks of the newly created type *roadLabel* the CREATE SNET NetworkName(net_label_type) statement is used. The statement is written in the following manner :

CREATE SNET road(roadLabel)

The above statement creates an empty network called 'road' which is of type 'roadLabel'. The 'road' network has to be populated with channels and the corresponding label information using the ADD statement. The ADD statement is used to add channels to an already created *snet* object. In order to demonstrate how insertion and querying spatial networks in a database work, we consider a table:

RoadNetworks(RoadType:string, roadNet:snet, AdministeredBy:String)

Before we can discuss the ADD statement, we need the concept of channel chunks. Each channel in a network are conceptually subdivided into elementary parts called *channel-chunks*. *Channel-chunks* are contiguous part of channels where all the points of the channel have identical values for all the thematic attributes. It may be viewed as collecting contiguous points of a channel for which the thematic data are exactly the same. For example, consider a street represented as a channel C_1. This channel is labeled with two attributes *speed_limit* and *laneCount*. Figure 1 shows how these values change over the channel. Thus this channel consist of 4 channel-chunks : s_1(speed_limit = 40, laneCount = 2), s_2(speed_limit = 50, laneCount = 2), s_3(speed_limit = 50, laneCount = 3), and s_4(speed_limit = 40, laneCount = 3).

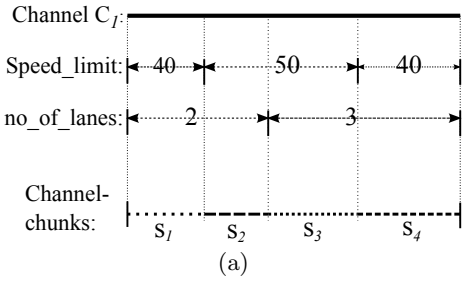

Fig. 1. An Example of a Spatial Network

All these channel chunks have to be inserted into an empty *snet* object individually using the ADD statement. Let us assume that there are two roads in a road network called '13th Street' and 'University Avenue'. Depending on the speed limit, "13th Street" consist of two channel chunks whose geometry is represented by the two *spatial line* objects l_1 and l_2 having speed limits 35 mph and 45 mph respectively. The line objects l_1 and l_2 are internally generated from their textual representation $l1$ and $l2$ respectively by the ADD statement. Similarly, on "University Ave" the speed limits vary from 35 mph and 45 mph at various portions. These portions are channel chunks and are represented by the two *spatial line* objects l_3 and l_4 (generated from their textual representation $l3$ and $l4$). Both these streets have a lane count of 4. The ADD statement takes a network in which to add the channel chunk, a simple line representing the geometry of the channel, and a label consisting of the label values. In this case the label attributes consist of (id_attr, speed limit, lane count). We use the ADD statements to add these roads in the network in the following manner :

ADD(road, $l1$, ('13th Street', 35, 4))
ADD(road, $l2$, ('13th Street', 45, 4))
ADD(road, $l3$, ('University Ave', 35, 4))
ADD(road, $l4$, ('University Ave', 45, 4))

There can be one or more ADD statement with the same channel name (id_attr) indicating that all the channel chunks are from the same channel. A

channel as defined by the abstract model is a continuous linear feature without any breaks or branches. So as to maintain consistency with the abstract definition of channels, the geometry of the chunks of the same channel has to be a continuous sequence. This is known as the *geometric integrity* which states that channels are continuous and cannot have branches. This is ensured by the SNQL's geometric integrity check which ensures that channels are consistent and prevents the user from entering inconsistent data. Whenever a new channel chunk is added using the ADD statement, the SNQL internally checks to see whether chunks of the same id_attr has already been added. If there exists channel chunks with the same id_attr, then the geometric integrity check is performed. The check involves the following : first, the geometry of the newly added chunk should *meet* with the geometry of some other chunk with the same id_attr. *Meet* is a topological predicate which asserts that only the boundary points or the ends of the spatial lines intersect. Secondly, more than two channel chunks of the same id_attr should not *meet* at the same point thus preventing branches in a channel. If any one of the integrity checks fails, the geometric integrity does not hold and the ADD statement results in a failure.

After a spatial network object has been populated, we may insert it in a database table using the regular SQL INSERT statement. Assuming a spatial network called 'interstate_hwy' has been already created, we can insert it into the *RoadNetworks* table in the following manner :

INSERT INTO RoadNetworks VALUES('Interstate', interstate_hwy, 'Federal')

Similarly, all the other records have to be inserted in the table.

Based on the *snet* data type we now define constructs to pose queries in *SNQL*. An SNQL statement has the following clauses: the SELECT clause says what will be returned in the query, the FROM clause indicates a list of tables which may contain some *snet* attribute and the WHERE clause which contains a boolean expression that is evaluated over all the records in a table. Using this scheme, we provide sample queries belonging to the four query types discussed earlier. We assume that the table *RoadNetworks* contains the following tuples:

("Interstate", interstate_hwy, 'Federal')
("Country Roads', cr_road, 'State')
("Limited Access Highway", laHwys, 'Federal')
("Single Carriage Way", sig_way, 'State')
("Winter Roads', win_roads, 'State')

The database also has the table *NatPowerGrids* as discussed in Section 3.

Network queries deal with the the entire network as a whole. They include queries like 'which part of each of the networks intersect with the Interstate'. In formulating this query, we use the spatial network operator *intersect* which returns the intersection between two networks. The SNQL statement for the above query is as follows:

select Intersect(N.roadNet, M.roadNet)
from RoadNetworks **as** N, RoadNetworks **as** M
where M.roadNet = 'Interstate'

This query performs a self join, then uses the *intersect* operator pairwise with each road network type and the interstate network.

Component queries deal with the components in a network. Channels are the components of a network, and these queries extract part of a channel or complete channel and operate on them. The SNQL can handle these operations since a channel or part of a channel is also an *snet* object. A traditional query in transportation is the shortest path finding which may be of the form 'find the shortest route from Miami to Atlanta avoiding Alachua county'. This is a restricted form of shortest path since it has to avoid all the roads passing through Alachua county. Assuming that Alachua county is given as a region with the name 'alachua', and the initial and the final points as p_A and p_B respectively, we write the query in SNQL as

select ShortestRoute(sn, p_A, p_B)
from RoadNetworks **as** N
where sn **in** Difference(N.roadNet, window(N.roadNet, alachua))

The *window* operation returns the part of the network which intersects with Alachua. This portion of the network is subtracted from the original network (using the *difference* operator) and the shortest path is computed on the truncated network.

Next we provide examples for *component attribute queries*. These queries are based on the thematic information attached to the components of a spatial network. As discussed earlier, each channel of a network has certain attributes attached to it in the form of labels. These values can be extracted by the operator *GetAttribute* which takes a spatial network and a label attribute and returns the value of that attribute corresponding to the network. For example an interesting query is 'find the average capacity in each of the power grids'. This query asks for the average transmission capacity of each of the power lines in each of the grid. We assume that the network label attribute 'capacity' is in the labels for the power grid networks in the table *NatPowerGrids*. We again use the *GetAttribute* operator to find the capacity of each of the power lines in a network and then average the value. This query demonstrates how SQL constructs like GROUP BY may be used in a traditional manner in SNQL to group by the average capacity for each power grid. The SNQL statement is as follows:

select G.sectorName, AVG(GetAttribute(sn, capacity))
from RoadNetworks **as** G, GetAllChannels(G.Grid) **as** sn
group by G.sectorName

The GROUP BY clause groups the average capacities by each network grid as each network grid has a unique sectorName which the SELECT clause returns.

The final type of queries termed as *network attribute queries* query the attributes attached to an entire network. These attributes are not attached to the *snet* object, but are part of the record containing an *snet* object. For example, 'which government entity maintains the road network' can be answered by a simple SQL query which selects out the 'administered by' field from the *RoadNetworks* table :

select N.RoadType, N.administered_by
from RoadNetworks **as** N

4 Conclusions and Future Work

This paper describes the user view of spatial networks in a spatial database. The single data type *snet* captures the geometry, connectivity and the thematic information in a spatial network and can be used as a table attribute type. Using the newly introduced data type *snet*, spatial networks can be stored in a database as a single entity which prevents it from being spread over a number of tables, thus making a spatial network a first class citizen of the spatial database. In order to access and manipulate the *snet* objects in a database, we described an SQL-like query language called SNQL. SNQL supports the creation, insertion, and manipulation of *snet* data objects in a database table as well as the execution of operations on them.

This paper is part of a complete spatial networks package called *Spatial Networks Algebra* (*SNA*) that can model a broad range of spatial networks and will have a comprehensive collection of operations and predicates defined on them and is aimed at incorporation in the database system.

References

1. http://support.esri.com/en/downloads/datamodel.
2. M.R.T. Dale and M.-J. Fortin. From Graphs to Spatial Graphs. *Annual Review of Ecology, Evolution, and Systematics*, 41(1):21–38, 2010.
3. Hartmut Guting, Teixeira de Almeida, and Zhiming Ding. Modeling and Querying Moving Objects in Networks. *The VLDB Journal*, 15(2):165–190, 2006.
4. Christian Jensen, Torben Pedersen, Laurynas Speiys, and Igor Timko. Data Modeling for Mobile Services in the Real World. In *Advances in Spatial and Temporal Databases*, volume 2750, pages 1–9. 2003.
5. Virupaksha Kanjilal and Markus Schneider. Modeling and Querying Spatial Networks in Databases. *Journal of Multimedia Processing and Technologies*, 1(3):142–159, 2010.
6. Virupaksha Kanjilal and Markus Schneider. Spatial Network Modeling for Databases. In *ACM Symposium on Applied Computing*. ACM, 2010.
7. Kothuri, Ravikanth V. and Godfrind, Albert and Beinat, Euro. *Pro Oracle Spatial for Oracle Database 11g (Expert's Voice in Oracle)*. Apress, Berkely, CA, USA, 2007.
8. Michael Zeiler. *Modeling Our World: The ESRI Guide to Geodatabase Design*. Environmental Systems Research Institute, 1999.
9. H. J. Miller and S.-L. Shaw. *Geographic Information Systems for Transportation*. Oxford University Press, 2001.