To the Graduate Council:

I am submitting herewith a thesis written by Feng Lu entitled "A Spatiotemporal Indexing Method for Disaggregate Transportation Data." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Geography

Bruce Ralston, Major Professor

We have read this thesis
and recommend its acceptance·

Cheng Liu

Shih-Lung Shaw

Accepted for the Council

Intern Vice Provost and
Dean of the Graduate School

A SPATIOTEMPORAL INDEXING METHOD FOR DISAGGREGATE TRANSPORTATION DATA

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Feng Lu
August 2000

Acknowledgments

There are several people to whom I am grateful for helps during the nearly two years I have spent at the University of Tennessee. I would like to express my deepest gratitude to Dr. Bruce Ralston for being my great advisor. Dr. Ralston spent much more time to help me in my thesis-from topic selection, programming, to outline and writing. In addition, he was helpful to my family. I am also grateful to Drs. Cheng Liu and Shih-Lung Shaw. Dr. Liu originated the spatiotemporal indexing method in this thesis and helped me learn data structures and code writing. Without his help, it would have been impossible for me to finish my thesis so soon I learned basic GIS and transportation knowledge including the spatiotemporal data models and their applications in transportation from Dr Shaw. In fact, the spatiotemporal query problem for trip data was first identified by him Dr. Shaw and Dr. Liu also helped me personally.

There are a number of other people whose assistance should be recognized. I would like to thank Mike Schultze and Amy Rose for their generous help in using ArcInfo package to run my AML code. Amy also helped me debug the errors in the code.

Perhaps the greatest debt I owe is to my family. During the last three years I have been studying in the US, my family gave me support. I feel great to have my wife Cuiling and our daughter Rongrong's love.

Abstract

Time, location, and attributes are three elements of a GIS, but all commercial GIS packages can only handle location and attributes; they are in fact a static GIS. Spatiotemporal GIS has been a hot research topic recently. Spatiotemporal GIS and its application in transportation research are still premature. This thesis focuses on spatiotemporal query problems on travel data Specifically, It attempts to answer this question· during a time period, which trips pass through one or more specific streets? To speed up this spatiotemporal query for large data sets, a spatiotemporal index on the trip data is built by combining Avenue, AML, and C++. All the trip origin ends and those last destination ends for each individual on each day are geocoded using Avenue scripts The trip shortest path route system is created based on ArcInfo dynamic segmentation and network analysis functions. An array of 2-D tree structures based on each trip's beginning time and ending time and each street traversed are then created in C++ and Avenue. This array of 2-D tree structures is stored in memory. Finally, the spatiotemporal query function is performed by examining the array of 2-D tree structures for a given time window using Avenue and C++. A sample trip log data file in the Knoxville Metropolitan Area and Knox county street shape file are used to implement the spatiotemporal query. This thesis is concluded that efficient indexing methods must be developed to handle complicated spatiotemporal queries for large travel data sets

Table of Contents

Chapter		Page
1.	Introduction and Problem Descriptions	1
2.	Literature Review	7
	Temporal Non-spatial Databases and Temporal Queries	9
	Spatial-temporal Data Models and Queries	12
	Activity-based Transportation Research and Temporal GIS-T	28
3.	Data Description and Methodology	33
	Sample Data Description	33
	Trip Data Model (Representation of Trip Log Data Set)	36
	Comparison of Common GIS Approaches and the 2-D Trees Approach	41
	Methodology and Flowchart	45
4.	Creating the Trip Route System in Dynamic Segmentation	49
	Linear Referencing Systems and Dynamic Segmentation	49
	Geocode Trip Ends	53
	Create a Trip Route System	60
5	Temporal Indexing	70
	Generic Description of 2-D Tree Structure	70
	Create an Array of 2-D Trees	74
6	Spatiotemporal Query and Its Implementation	83
	Spatiotemporal Query Code Description	83
	Implementation and Examples	85
7.	Conclusions and Future Research	96
References		99
Appendix		105
Vita		132

List of Tables

Table		Page
2 1.	Temporal Indexes	11
3.1.	Name and description of main fields in the trip log file	35
3 2.	Trip path feature table description after using WritePathLong command	40

List of Figures

Figure		Page
1.1.	An example network on which trip1 and trip2 both pass through link AB	6
2.1.	Schematic representation of the space-time path	8
2.2	Schematic representation of the space-time prism	8
2.3.	The snapshot approach for representing spatiotemporal data	13
2.4.	A space-time composite of urban encroachment	15
2 5.	The ESTDM data structure	19
2 6	Spatial changes at time t_i displayed as a simplified map (a) and the corresponding event components (b) in the ESTDM model	19
2.7.	Framework of the TRIAD model	21
2.8.	The structure of the feature-based view	22
2.9.	The structure of the time-based view	23
2.10.	The structure of the location-based view	24
2.11.	The search spaces of four primitive geographic queries	26
2.12	A conceptual framework of the relationships between entities in a disaggregate travel data set	32
3.1.	Trip sample database of Knoxville	34
3.2.	Knox county network shape file	37
3 3.	A route system named BUS on a ROADS coverage	42
3.4.	Flowchart of building spatiotemporal index for disaggregate transportation data	46
4.1.	A route defined on a set of four arcs	50

Figure		Page
4.2.	An event database of traffic accidents on a route	51
4.3.	Pavement data contained in an event database	51
4 4.	Address geocoding object model in ArcView	54
4.5.	Trip chain characteristics of the sample data set	56
4.6	Geocoding result for the sample trip data set in Knoxville Metropolitan Area	61
4.7.	A shortest path trip route system named path1 based on STREETS network coverage and STREETS.STP stops info file in ArcInfo network analysis	62
4.8.	A sample trip shortest path route system on Knox network	68
4.9.	A sample text file exported from the section table of the trip route system	69
5.1.	A generic 2-D tree structure	71
5.2.	A sample trip 2-D tree based on starting time and ending time	73
5.3.	The structure of an array of 2-D trees	74
6.1.	The user interface of a spatiotemporal query function on a trip route system	87
6 2.	Create an array of 2-D tree structures by loading a DLL file	88
6 3.	The time window (8 00 am – 9·00 am) of the spatiotemporal query on one street (example one)	89
6 4.	The message box report showing the spatiotemporal query result on one street (example one)	90
6.5.	The selected paths showing the spatiotemporal query result on one street (example one)	91
6.6.	The time window (5:00 pm – 7:00 pm) of the spatiotemporal query on a set of streets (example two)	92

Figure Page

6.7. The message box report showing the spatiotemporal query result on a set of streets (example two) 93

6.8. The selected paths showing the spatiotemporal query result on a set of streets (example two) 94

Chapter 1

Introduction and Problem Descriptions

The study of travel demand analysis has evolved through several stages since 1950s. It evolved from analyzing aggregate data (based on traffic analysis zones) to analyzing disaggregate data (based on individuals or households). Pas (1990) classifies the development of travel demand analysis and modeling into five eras. The social physics era began in mid-1950s with the transportation studies in the major metropolitan areas of North America. Travel demand models (e.g. the four step transportation demand modeling) were formulated and calibrated at the level of the traffic analysis zone based on physics models such as the gravity model These models were also called 'aggregate models'. The econometric era started in 1960s with the development of the econometrics of discrete choice models which describe disaggregate behavior. In the 1970s, the psychological scaling techniques to measure consumer's perceptions, attitudes, and beliefs were used in disaggregate behavior models. This era was called the psychological era. The human activity analysis era started in mid-1970s. In the human activity analysis framework, travel is treated as a derived demand based on individuals' needs and desires to participate in activities at spatially separated locations. Human activities and travel patterns within spatio-temporal constraints became the focus of this era. The fifth era called the dynamic analysis era began in mid-1980s with focus on the longitudinal data analysis This era can be seen as the extension of the human activity analysis era (Shaw, 1999). Thus the activity-based approach has been a focus in travel demand analysis since mid-1970s.

The activity-based approach emphasizes travel and activity patterns, and the dynamics of travel behavior. How to handle dynamic travel behavior is a major challenge to the GIS-transportation community. Goodchild (2000) argues that there are three views of GIS-T: the map view, the navigation view, and the behavior view The map view is concerned about how to describe and represent a static transportation phenomenon. DIME and TIGER data models created by the Bureau of the Census are examples The link-node data model is a basic structure for representing static transportation networks. The navigation view is concerned about connectivity and planarity, and the storage of time dependent attributes. Routing problems are the main perspectives of the navigation view. The behavior view deals with the behavior of discrete objects – vehicles, people, trains, or boats – on and off the network. The behavior view of GIS-T is important because the traditional aggregate models cannot satisfy the needs of transportation analysis. For example, when dealing with travel demand modeling, we should be concerned about the characteristics of households and individual travelers. In the meantime, the short-term activity patterns of travelers also should be analyzed. Time therefore becomes an important element in GIS-T.

In fact, dynamic travel behavior and activity patterns are also important in Intelligent Transportation Systems (ITS). According to Japan's ITS system architecture, human, vehicle, roadside, and center are the four important subsystems of ITS. ITS is concerned with real time data. Human and vehicles are moving objects along networks. How to capture the characteristics of moving objects (attributes, locations) and find their activity patterns are important aspects of ITS. In order to capture these characteristics, continuous data on moving objects is a concern. How to create data models for the continuous

moving objects is a challenge to GIS-T researchers due to the complexity of dynamic traveler behavior and the large amount of data (Shaw, 1999).

Dangermond (1984) suggests that time, location, and attributes are three elements of a GIS A major research question concerns the handling of large amounts of historical spatial data. Many researchers have studied this problem. Hagerstrand's time geography provides the theoretical direction (Hagerstrand, 1970). He introduced time as the third dimension of the trajectory of objects. Thus, location, attribute, and time become basic elements of describing geographic features. Handling time component in GIS is a major challenge. How should we store the large sets of data? How can such data be retrieved efficiently? How can we perform spatial and temporal analysis?

There are several research topics for spatiotemporal GIS such as spatiotemporal data models, temporal overlay, and spatiotemporal query This thesis focuses on how to build a spatial temporal index function for trip log data.

Up to now, all commercial GIS packages only can handle one state of geographical phenomenon. For example, ArcView's shape file and ArcInfo's coverage are both basic organization units of spatial data, but they can only store static geographic features. When dealing with multiple states or dynamic geographic features, all these dynamic features have to be stored in different shape files or coverages. This approach for handling dynamic geographic features is called the snapshot approach. With this approach, there is data redundancy, because all geographic features at different states have to be stored, even though some features do not change. Other shortcomings of this approach are that there is no temporal overlay function, and it is difficult to trace changing patterns of geographic features.

GIS includes functions for spatial data input, manipulation, retrieval, analysis, display, and output. Users use GIS' query functions to retrieve geographic information. Spatiotemporal query is an important component in spatiotemporal GIS. Current commercial GIS have data models and algorithms for efficiently handling spatial query. However, temporal query is treated as a kind of attribute query. That is, all temporal information is treated as attributes for spatial features. For example, when feature class A exists during time bt and et, then bt and et can be attributes of A. When querying whether A exists between time period bt and et, we can query based on attribute fields related to time.

What is a spatiotemporal query? According to Langran (1992), a spatiotemporal query defines a search space within the data space composed of location, attributes, and times by constraining location, attributes, and times to points or segments along each axis

Spatiotemporal query is complicated and difficult to handle There is no generic spatiotemporal data model available, because time is more complex than the spatial domain, as time dimensions are non-homogeneous (Snodgrass, 1992). Without a spatiotemporal data model, querying features based on space and time is inefficient. Because all the current commercial GIS software can only use the snapshot approach to handle spatiotemporal data, it is difficult to conduct spatiotemporal queries efficiently.

Querying trip log data based on a space-time window is a spatiotemporal query. Trip log data analysis is useful for urban transportation planning and transportation behavioral analysis. It comprises the basic data for disaggregate transportation and activity-based modeling approaches.

Trip log data contains trip information for each household or each person on each day, and each trip has its own path. In order to study traffic flow and traffic congestion on a particular street, we might want to ask questions such as: during a time window, how many trips pass through this street? What are those trips? That is, we want to create spatiotemporal query windows.

There are different kinds of spatiotemporal queries for trip data. In this research, I attempt to solve the problem of querying what trips pass through a set of streets during a time window. For example (see Fig. 1.1), there are two trips (trip1, trip2) They both use link AB. We might ask how to search trips passing through AB during the time period 9.30 – 10:30 am? If the data set is small, there will be no need to build a spatiotemporal index. However, if the data set is quite large, it will be necessary to build a spatiotemporal index.

Trips can be regarded as moving objects (points) along a network, and each trip has a path (polyline). There is no generic data model to handle moving objects. Here, the trip data model is assumed to be a spatial data model plus an attribute table to handle time elements. A trip path is considered to be a basic spatial feature, and time elements (beginning time, and ending time) are considered to be two fields in a path attribute table.

I use the Knoxville area in Tennessee as a test area for building a spatiotemporal index for trips The purpose is to speed spatiotemporal query. There are two input data files: one is Knox County street network (Arcview shape file), and another is a trip log file (a text file). Based on these two files, a trip path system can be created using ArcInfo's dynamic segmentation functions and network analyst. Then a spatiotemporal index can be built for each street (arc) based on the starting and ending times trips pass

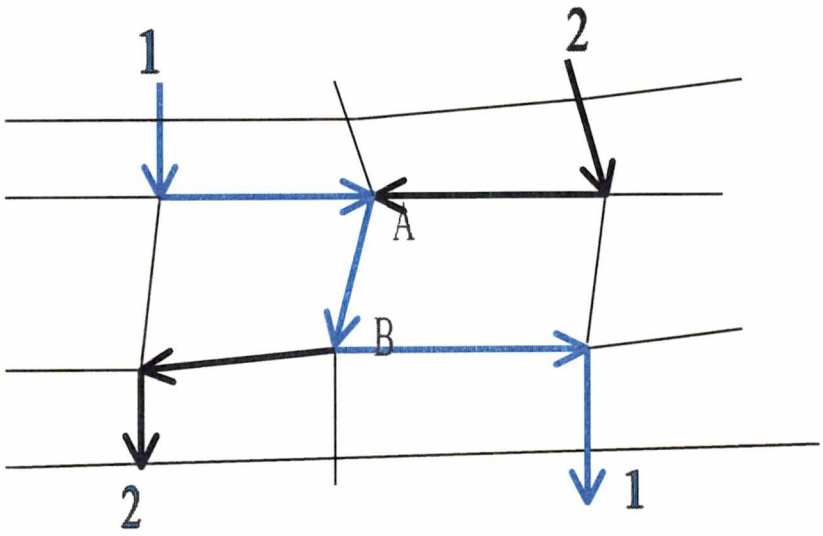


Figure 1.1. An example network on which trip1 and trip 2 both pass through link AB.

through this street using a two-dimensional tree. Combining ArcView spatial search function and two-dimensional tree structures, spatiotemporal query can be performed efficiently. The construction of a spatiotemporal index and the development of spatiotemporal query functions constitute the main contribution of this thesis.

Chapter 2

Literature Review

While spatiotemporal GIS is a new research area, its theoretical foundation can be traced to time geography initiated by Hägerstrand (1970). The time geographic framework focuses on the behavior possibilities of individuals within spatial and temporal constraints. In this framework, an individual can only participate in events or activities at a single location in space at a given time. Thus, the space-time path of an individual can be drawn using a three-dimensional coordinate system (Figure 2 1). X-Y coordinates stand for a planar space, while z stands for a time dimension A vertical line represents stationary activities (an individual stays at a location for some time); while a sloped line represents movements among different locations within a time period. The higher the slope degree, the slower the movements will be. The broken lines describe the spatial movements of the individual over a time frame. The space-time path can be rotated to form the space-time prism, which determines the feasible set of locations for an individual to participate in activities within space and time budgets (Figure 2.2). The space-time prism does not trace the observed movements of an individual, instead, it shows the feasible space for an individual to travel within a period of time.

Even though the time geographic framework was originally designed for describing behavior patterns of individuals, it can be applied to all kinds of spatial temporal phenomena. Geographic features might be points, lines, or polygons, and their movements (or transformation or modification) can be continuous or discontinuous. The time geographic (or space-time) framework provides theoretical foundation for these

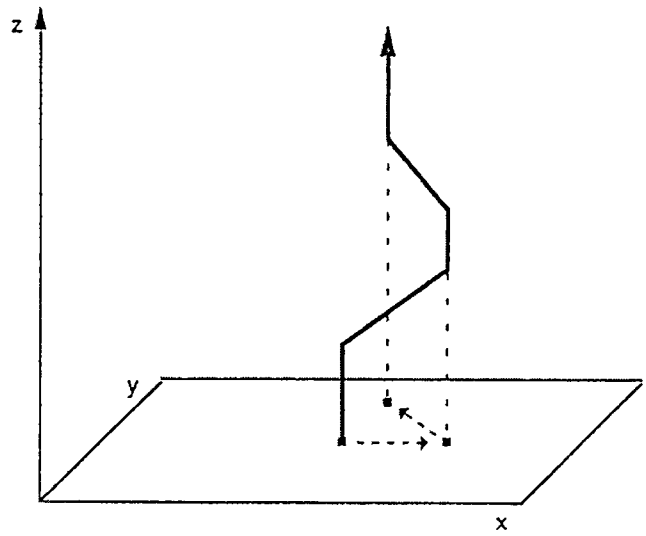


Figure 2.1. Schematic representation of the space-time path. (Source: Miller, H. J., 1991.)

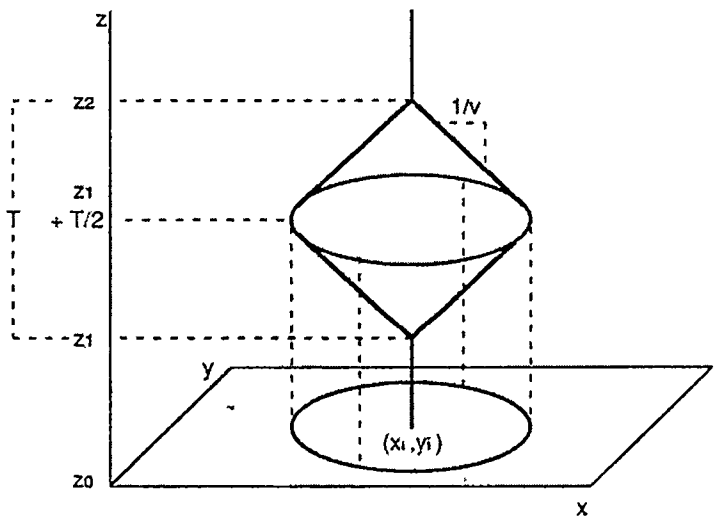


Figure 2.2. Schematic representation of the space-time prism. (Source: Miller, H. J., 1991.)

phenomena

Current research on time and data models for GIS has focused mainly on the representation of temporal geographical entities and implementation of temporal databases

Temporal Non-spatial Databases and Temporal Queries

Some researchers in computer science began to study temporal database in 1980s. They mainly focused on expanding the relational data model to incorporate some aspects of temporality. There are three main approaches. relation-level versioning, tuple-level versioning, and attribute-level versioning

Relation-level versioning creates a new snapshot of a table whenever any of its attribute change. This approach is conceptually simple; however, it is highly redundant. Ben-Zvi (1982) 's time relational model and Clifford and Warren (1983)'s historical data model are two representatives of this approach.

Tuple-level versioning is an approach where attribute changes cause new records (tuples) to be created or old records to be updated or deleted. Snodgrass and Ahn (1985)'s method is a representative. They introduce two important aspects of time: world time (or valid time) and system time (or transaction time). The world time traces the changes that occur in the real world, while the system time traces the changes recorded in the database. They use four time stamps to bracket intervals of system time (from system time to system time) and world time (from world time to world time). New tuples are added to the bottom of a relation (table). Tuples are deleted by changing their time field values. Tuples are altered by deleting the current version and adding the new version.

The tuple-level versioning approach is better than the relation-level versioning in that it reduces storage cost a lot, and relational theories and algebra can apply.

Attribute-level versioning requires variable-length fields of complex domain to hold lists of time-stamped attribute versions (Langran, 1992). This approach is compact but requires alternative relational algebra to handle Gadia (1986), and Clifford and Tansel (1985)'s methods are the representatives of this approach.

In addition to these temporal data models, many temporal query languages have been proposed Each temporal query language is related to one temporal data model. Snodgrass (1992) lists twenty-one kinds of temporal query languages such as TQuel, TSQL, Postquel, HQuel, and HSQL, which are based on conventional query languages – SQL (Structured Query Language, a tuple calculus-based language), Quel (the tuple calculus based query language), QBE (Query-by-Example, a domain calculus based query language), relational algebra (a procedural language with relations as objects), and DEAL (an extension of the relational algebra incorporating functions, recursion, and deduction). Snodgrass argues that the implementations of these temporal query languages are limited in scope and are unsystematic in their design Now an effort is underway within the research community to create a common generic Temporal Structured Query Language based on SQL (Snodgrass, 1992).

At the system level, many temporal index data structures have been proposed to speed up temporal queries. Snodgrass (1992) summarized these temporal index structures according to their foundation tree structure, temporal dimensions, temporal keys etc (see table 2.1). All these temporal index structures are based on B+ trees (indexing on values

Table 2.1 Temporal Indexes (Source Snodgrass, 1992)

Name	Citation	Based on	Primary/ Secondary	Temporal Dimensions	Temporal Key(s)	Non-Temporal Key(s)
Append-only Tree	Guandhi, et al, 1991	B+ Tree	Primary	transaction	event	0
Checkpoint Index	Leung, et al, 1992	B+ Tree	secondary	transaction	event	0
Lop-Sided B+ Tree	Kolovson, 1990	B+ Tree	Both	transaction	event	0
Monotonic B+ Tree	Elmasri, et al, 1992	Time Index	Both	transaction	interval	0
--	Lum, et al, 1984	B+ Tree or Hashing	primary	transaction	none	1
Time-Split B-Tree	Lomet, et al, 1990	B+ Tree	primary	transaction	interval	1
Mixed Media R-Tree	Kolovson, et al, 1989	R-Tree	Both	transaction, trans+valid	interval, pairs of intervals	k ranges, k >= 1
Time Index	Elmasri, et al, 1990	B+ Tree	Both	both	interval	0
Two-level Combined Attribute/Time Index	Elmasri, et al, 1991	B+ Tree + Time Index	Both	both	interval	1
--	Ahn, et al, 1988	B+ Tree, Hashing	various	various	interval	1
SR-Tree	Kolovson, et al, 1990	Segment Index + R-Tree	Both	both	interval, pairs of intervals	k ranges, k >= 1

of a single key) and R trees (indexing on ranges of multiple keys) Each temporal index structure has its own advantages and application areas However, there is little effort to compare them in space and time efficiency.

Spatial-temporal Data Models and Queries

A data model is an abstract representation of some real-world situation or domain of interest about which information will be stored in a database (Dictionary of Computing 1996) Data models are the cores of an information system; they define data object types, relations, operations, and rules to maintain database integrity (Date, 1995; Miller, H.J. and Shaw, S L , 2000) Temporal GIS includes time elements in conventional GIS; it aims to process, analyze, retrieve and manage spatiotemporal data. Spatiotemporal data models are the cores of a temporal GIS Without a good data model, it will be ineffectual for a temporal GIS to handle spatiotemporal queries.

Since 1980s, geographers began to conduct studies on the representation of temporal geographic entities, and several spatiotemporal data models have been proposed. Peuquet (1999) classifies spatiotemporal data models into four kinds location-based representation; entity-based representation, time-based representation; and a combined approach. Yuan (1999) groups these data models into two kinds· by time-stamping spatial objects, and by events or processes. Here, the author lists five kinds of spatiotemporal data models according to their historical developments.

The Snapshot Model

The snapshot model views spatiotemporal data as a sequent of snapshots at different time slices (Figure 2.3). In this model, every layer is a collection of temporally homogeneous units of one theme i.e. one layer holds information related to a single thematic domain at a time point. Spatiotemporal data are recorded by discrete temporal intervals. At each time point, all features (or phenomena) are included regardless of what has or has not changed since the previous snapshot. Temporal intervals are not necessarily the same. For example, at time point t_2, S_2 holds all the features even if some features don't change from S_1; and temporal interval (t_1-t_2) is not necessarily equal to other temporal intervals (t_2-t_3, etc). The snapshot model is the only data model available within current commercial GIS. The temporal urban mapping model for the Baltimore-Washington area and the San Francisco Bay area (USGS, 1998) are examples of this approach.

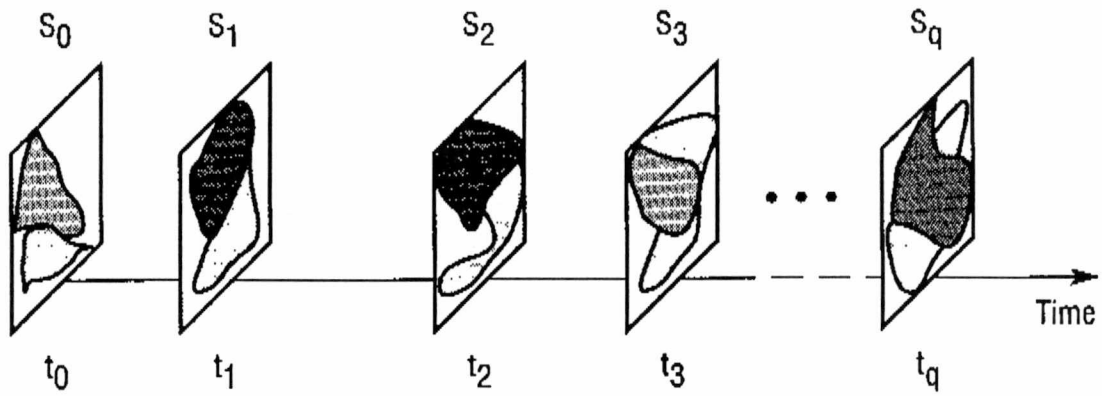


Figure 2.3. The snapshot approach for representing spatiotemporal data: each snapshot, Si represents the state for a given point in time Ti. (Source: Peuquet et al, 1995)

Langran (1992) and Peuquet (1999) summarize three main shortcomings of snapshot approach.

- Redundant storage. Even though during each temporal interval, only a small portion of features changes, a snapshot is a complete map of all the regions, which duplicates all the unchanged data. The data volume increases enormously when the number of temporal points (or temporal intervals) increases.
- No temporal topology. Temporal topology means temporal relationship for each spatial feature (i.e. its previous and next versions). In order to compare spatial entities to their previous and next versions to retrieve spatial changes, those temporal adjacent snapshots have to be compared exhaustively. The reason for this problem is that snapshots represent states but not the events that change from one state to another state.
- Undetermined change. Because snapshots only represent states, exactly when a spatial entity changes cannot be determined. In addition, some critical changes at some locations may occur between two consecutive snapshots, but may not be represented.

The Space-time Composite Model

The space-time composite model represents spatio-temporal data as a set of spatially homogeneous and temporally uniform objects, and can been viewed as overlays of temporal snapshots. This approach was proposed by Christman (1983), and Langran and Christman (1988) Space-time composite model begins with a base map or a layer that

represents the original state of geographic phenomena. Each change causes a portion of the geographic phenomena to become discrete objects, which have their own histories. After all temporal changes, the original layer is changed to a layer composed of basic spatiotemporal units (points, lines, or polygons), which are overlays or intersections of different temporal snapshots and have their distinct temporal attributes.

Figure 2 4 shows a space-time composite for urban encroachments. The land use decomposes over time into smaller fragments, which reference distinct temporal attribute sets The whole region was originally rural land use (at time T_1) At time point T_2, portion A changed to urban land use; at time point T_3, portion B changed to urban land use; at time point T_4, portion C changed to urban land use. The procedure of urban encroachments can be seen as overlays of three snapshots (T_1, T_2, T_3), and keeps all intersection points, lines, and polygons as basic spatio-temporal units, which have their own history.

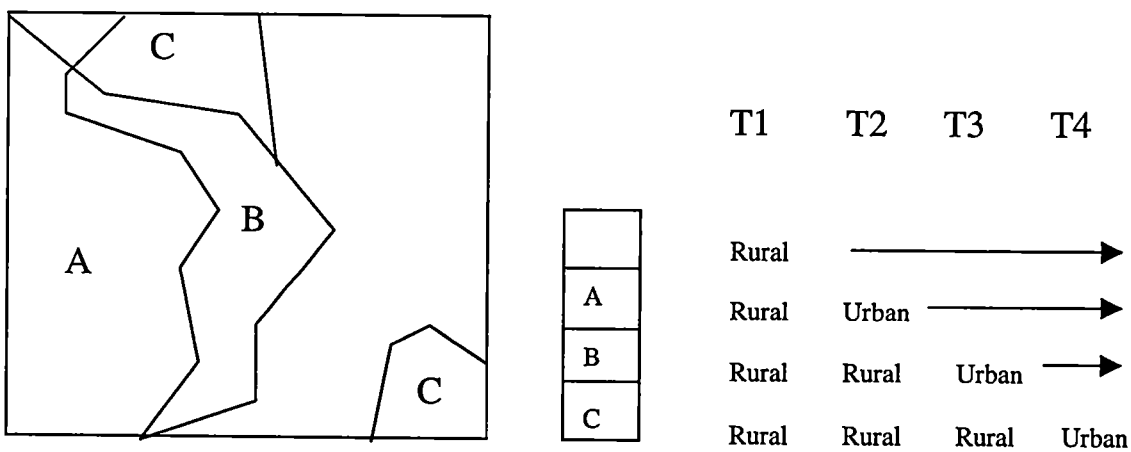


Figure 2.4 A space-time composite of urban encroachment. Each polygon has an attribute history from that of its neighbors. (Modified from Langran 1992, pp. 41)

Space-time composite employs the topological model, thus explicitly maintaining the integrity of individual entities and their changing topology through time. Compared to the snapshot approach, this approach has the advantage of less data redundancy. In addition, accessing temporal changing information in the space-time composite is conceptually straightforward because this approach contains temporal topology.

However, two shortcomings can be detected in space-time composite model (Langran, 1992; Yuan, 1999; Peuquet, 1999). As time progresses and temporal changes increase, the space-time topology becomes quite complex i.e. the representation decomposes into progressively smaller objects, and ultimately poses intolerable storage processing problems. There are many aspatial attributes that can change over time. If components that make up the spatial and aspatial aspects of any given entity are changing at different times and at different rates, maintaining the identity of individual entities becomes difficult (Peuquet, 1999).

Yuan (1999) points out that the space-time composite model is able to record temporality within the largest common units of attribute, space, and time, but it fails to capture temporality attributes across space (i.e. motion or movement) Disaggregate transportation or trip data can be described as moving objects, thus the space-time composite model cannot efficiently represent disaggregate transportation data.

The Spatiotemporal Object Model

The spatiotemporal object model uses an object-oriented approach. In the object-oriented approach, every entity is an object, which is the basic organizational unit. An object has properties and methods. Spatiotemporal object (ST-Object) model was

proposed by Worboys (1992, 1994). The approach taken by Worboys is to form classes of primitive spatiotemporal objects by associating two-dimensional temporal elements (i e world time and system time) with spatial objects Composite spatiotemporal classes are created based on primitive spatial object classes and primitive temporal object classes.

The spatial data model Worboys built is based on combinatorial topology. Spatial objects are represented as simplicial complexes. A simplex is either a single point, finite straight line segment or triangular area. A simplicial complex is a collection of non-overlapping simplexes.

A spatio-bitemporal object is created based on spatial and bitemporal (i e. world time and system time) extents. This object can be represented by attaching bitemporal elements as labels to components of simplicial complexes A ST-complex is a collection of ST-simplexes, which is an elemental object (simplex) to which is attached a bitemporal reference

According to Peuquet (1999), the spatiotemporal data model provides a cohesive representation that allows the identity of objects as well as complex interrelationships to be maintained through time. This model is able to record changes in space, time, and attributes. However, because the spatiotemporal object model represents the world as a set of discrete objects consisting of ST-simplexes, continuous or gradual changes in space through time cannot be represented.

The Time-based (or Event-based) Spatiotemporal Data Model

Spatiotemporal data models based on time or events mainly represent temporal change as a function of time. In this approach, there is a base map or starting point, from which, the sequence of events represents an ordered progression through time of known changes.

Peuquet et al (1995) propose an Event-based Spatiotemporal Data Model (ESTDM) based on time as its organization basis, and intend to analyze temporal relationships and change patterns within a pre-specified geographical area (Figure 2 5). Using doubly linked list structure, the ESTDM consists of a header, a base map that defines the initial geographic area, and an event list with set of components attached to each event. The header contains information about the thematic domain, a pointer to the base map, and pointers to the first and last events. The base map shows an initial snapshot of the entire geographic area using a raster run-length-encoded method. The event list contains spatial changes through temporal intervals. Each event contains a time-stamp, a list of pointers pointing to each event component, and a pair of pointers pointing to the previous and next events. An event component represents changes to a predefined location (raster cells) at a time point. All locations that have changed to the same value within a single thematic layer are members of the same component. Figure 2 6 shows the component structure: a component descriptor that shows the new value; and an array of locational elements called tokens. A single token represents a set of consecutive cells along a row using run-length encoding. It consists of row number, the first column number (left-most column with the same value), and the last column number (right-most column with the same

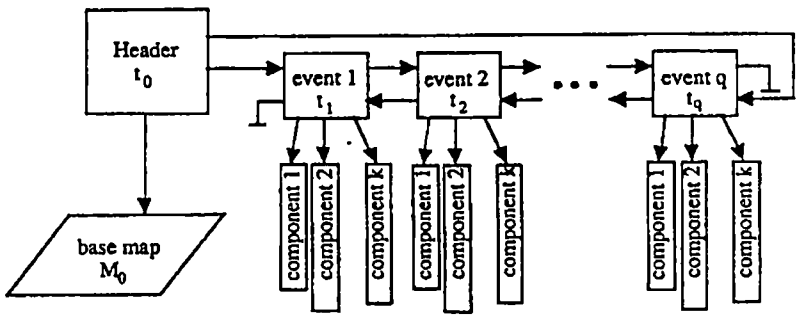


Figure 2.5 The ESTDM data structure (Source: Peuquet et al, 1995)

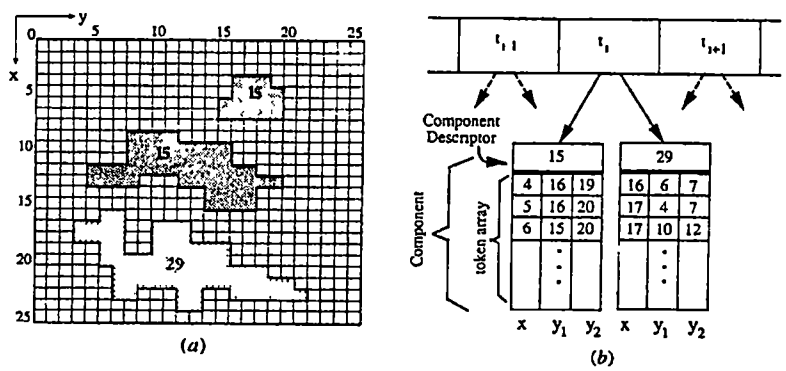


Figure 2.6 Spatial changes at time t_i displayed as a simplified map (a) and the corresponding event components (b) in the ESTDM model. (Source: Peuquet et al, 1995)

value).

Because time is a basic organization unit, ESTDM has the capabilities of performing temporal manipulations (temporally based queries) on spatiotemporal data. For example, ESTDM can easily retrieve all locations that changed to a given value at a given time; retrieve all locations that changed to a given value over a given temporal interval;

calculate the total change in area to a given value over a given temporal interval. Adding new events is straightforward (just adding to the end of the event list). ESTDM can also reduce data redundancy, since it only handles temporal changes.

ESTDM is a raster-based spatiotemporal data model. All temporal changes are associated with predefined locations or cells. However, the adoption of ESTDM model to a vector-based system requires redesign of event components. Spatial objects always change in geometrical properties and topological relationships, how to handle spatial objects' identities becomes a real challenge Using ESTDM approach in a vector-based system, predefined entities have to be created to handle temporal changes. If the entities are points, there is no problem of changing topology; but if the entities are lines or polygons, these entities will be fragmented if changes occur.

Comprehensive Models

A comprehensive model integrates different kinds of spatiotemporal data models together as one data model A feature-based data model (or vector data model) is more effective in retrieving information about spatial features or objects; a location-based data model (or raster data model) is more effective in retrieving information about locations; while a time-based data model is more effective in retrieving information about specific times or changes through time. Integrating these three data models would allow us to handle complicated spatiotemporal problems such as managing changes of spatial objects and maintaining object identities.

A prototype of spatiotemporal data model called TRIAD was proposed by Peuquet (1994) and implemented by Peuquet et al (1994). In TRIAD, there are three

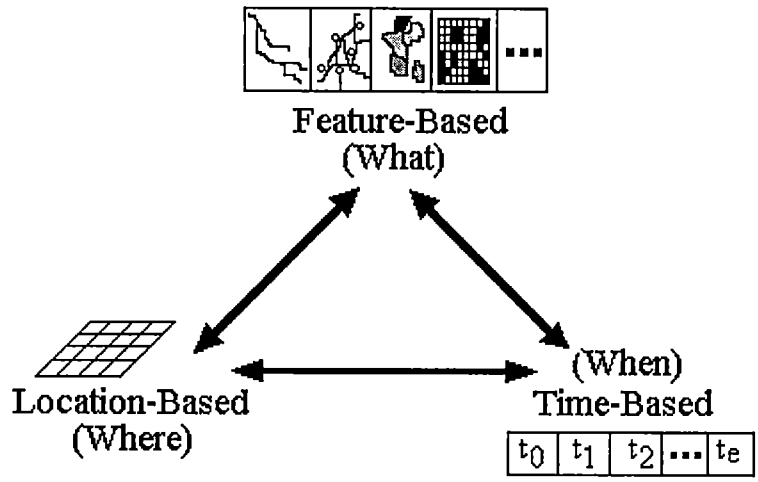


Figure 2.7 Framework of the TRIAD Model (Source. Peuquet et al, http //www.geog.psu.edu/tempest/)

interdependent representations: feature-based, location-based, and time-based views, which handle what, where, and when problems respectively (Figure 2.7).

The feature-based view maintains the integrity of geographic individual objects along with their spatial and temporal changes (Figure 2 8). It consists of feature inherent and non-inherent attributes (feature id, name, feature class, layer info, and other static attributes), as well as spatial delimiters and temporal delimiters. Spatial delimiters store feature's first and latest generalized locations (the bounding rectangle for an areal feature, two x-y coordinate pairs to denote the endpoints of a line, and a single x-y coordinate to denote a point feature). Spatial delimiters are used for linking the feature-based view with the location-based view to retrieve detailed location information. Temporal

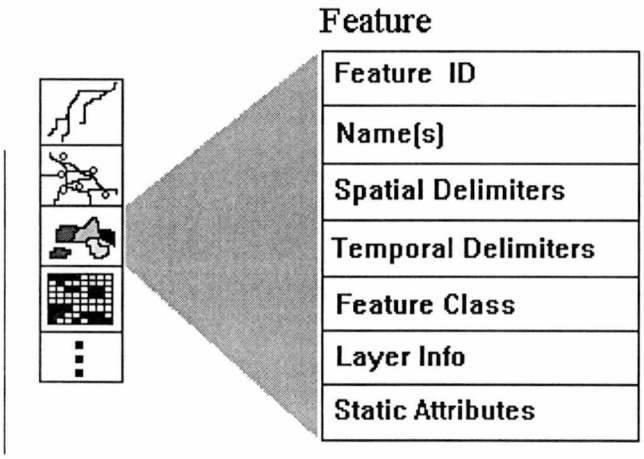


Figure 2.8 The Structure of the Feature-based View (Source: Peuquet et al, http://www.geog.psu.edu/tempest/)

delimiters store the times a specific feature begins to exist and ceases to exist. Temporal delimiters are used as the primary link to the time-based view to retrieve detailed events or temporal changes.

The time-based view is a modified version of ESTDM (Peuquet, 1995) (Figure 2.9). In this view, the basic organization unit is an event. Attributes associated with each event include one timestamp, feature changes and locational changes. The timestamp stores the time at which the event occurs. Feature changes record the specific changes occurring to one or a group of features during a temporal interval. Feature changes consist of a list of feature ids (FID List), their attributes and new values. Locational changes record the specific changes occurring to one or some locations during a temporal

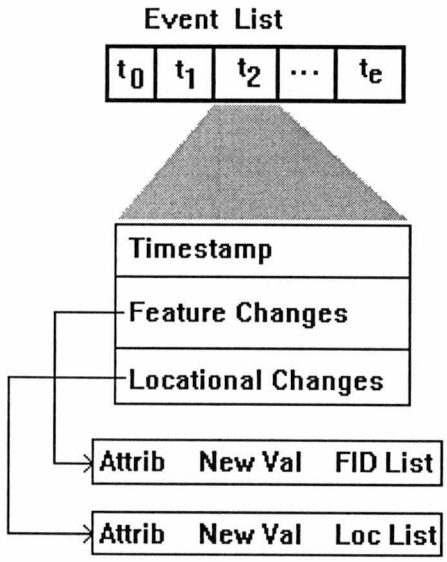


Figure 2.9 The Structure of Time-based View (Source: Peuquet et al, http://www.geog.psu.edu/tempest/)

interval. Locational changes consist of a list of location ids (Loc List, cells or pixels), attributes and new values associated with these locations.

The location-based view uses cells or pixels as the basic organizational units (Figure 2.10). Every cell is represented by a list of changes occurring at that location. There are two types of changes: feature and attribute changes. A feature change records whether a feature begins or ends its presence at that location, and the time of change. An attribute change records a change in an attribute value.

The TRIAD model unifies feature-based, location-based, and time-based views together and thus has the advantages of these three views or approaches. It's easy to store, retrieve, and manipulate spatiotemporal data using this model. For example,

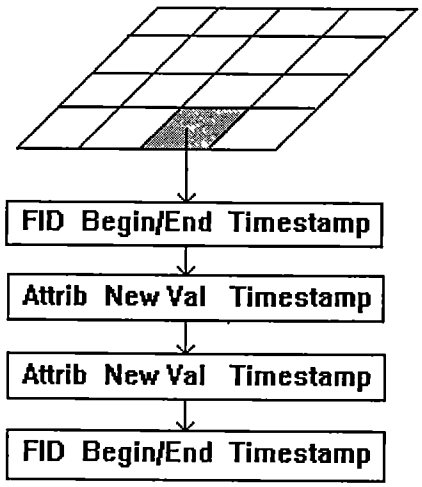


Figure 2.10 The Structure of the Location-based View (Source: Peuquet et al, http://www.geog.psu.edu/tempest/)

spatiotemporal queries can be launched from any combination of the three views: when, where, and/or what.

Yuan (1994) proposed a comprehensive data model called the three-domain model to analyze wildfire. Similar to the TRIAD model, the three-domain model defines semantical, temporal, and spatial objects in three interrelated domains. She argues that the major advantage of this model is that there is no pre-defined data object; rather the model dynamically links relevant objects from the three domains to represent a geographic entity or concept (Yuan, 1999). Therefore, this model can manage complicated changes of spatial objects and maintain object identities.

Spatiotemporal Queries

In the past, research on temporal GIS mainly focused on the design of spatiotemporal data models, which are the foundation of spatiotemporal data storage, manipulation, retrieval, and analysis Spatiotemporal queries are usually associated with spatiotemporal data models. For example, ESTDM is designed for handling temporal changes for locations; thus it can easily handle queries related to temporal intervals; while TRIAD is designed based on what (feature), where (location), and when (time), and thus can handle complicated spatiotemporal queries.

Langran (1992) listed a generalized set of potential queries for a temporal GIS to treat.

" 1. Examine a feature's lifespan.
2 Examine a single time slice.
3. Examine a feature's lifespan; when the feature meets some criteria, examine its time slice.
4. Examine a single time slice; examine the lifespans of features meeting some criteria.
5 Examine the lifespans of all features.
6. Examine all time slices. " (P 73).

She then summarized four primitive query types that are at the root of this list (Figure 2.11):

" 1. Simple temporal query, i e. what is the state of a feature at a time slice?
2. Temporal range query, i.e. what happens to a feature over a given period?
3. Simple spatiotemporal query, i.e. what is the state of a region at a time slice?
4. Spatiotemporal range query, i.e. what happens to a region over a period? " (P 97).

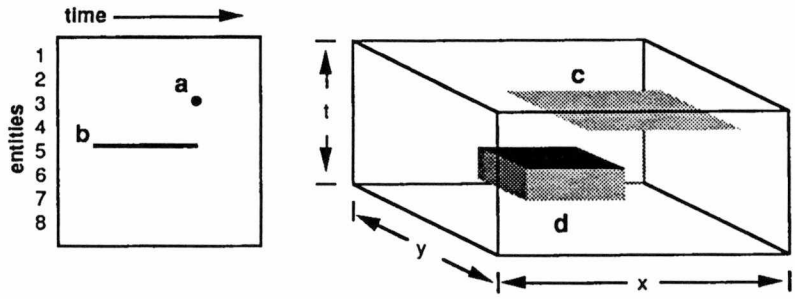


Figure 2.11 The search spaces of four primitive geographic queries. (a) A simple temporal query. (b) A temporal range query. (c) A simple spatiotemporal query. (d) A spatiotemporal range query. (Source: Langran, G., 1992. Pp 97)

According to Langran, a temporal query defines its data space by thematic attributes and time; while a spatiotemporal query defines a geometric data space within which to model cartographic objects.

Langran analyzed k-dimensional data structures and their applications in accessing spatiotemporal data. Zero-dimensional data structures (e.g. k-d-tree, k-d-b-tree, multikey hashing etc) were developed specifically to access dimensionless attribute data; One-dimensional data structures (e.g. strip tree) can only handle one-dimensional spatial object; while k-dimensional data structures (e.g. R-tree, R+ tree, Packed R-tree, Cell tree, Grid file etc) can be used to access three-dimensional objects in three dimensional data space. She tested four data structures (grid file, offset grid file, R-tree, and six-dimensional data structure) to access spatiotemporal data stored in a space-time composite in qualitative experiments and found R-tree and offset grid file data structures appear to be reasonable approaches to spatiotemporal data access.

Peuquet (1994) grouped spatiotemporal queries into three classes:

The first query class addresses changes in an object or feature (spatial objects relative to space and time or just to time), for example:

" (a) Has this object moved in the last two years?
(b) Where was this object two years ago?
(c) How has this object changed over the last five years?"

The second query class addresses changes in the spatial distribution of an object or a set of objects (locations relative to time), for example:

" (d) What areas of agriculture land use in January 1, 1980 have changed to residential land use as of December 31, 1989?
(e) Did any land use changes occur in this drainage basin between January 1, 1980 and December 31, 1989?
(f) What was the distribution of commercial land use 15 years ago?
(g) What areas have changed from predominantly agricultural land use to urban land use over the last 50 years?"

The third query class addresses the temporal relationships among multiple geographic phenomena (time relative to the attributes of specific locations or specific objects), for example:

" (h) Which areas experienced a landslide within one week of a major storm event?
(i) Which areas within one-half mile of the new urban bypass road have changed from agricultural land uses to other land uses since completion of that road?"

To measure all temporal relationships or temporal topology, Peuquet also outlined a set of temporal operators, which include before, equal, meets, overlaps, within, and start/end. These operators can be used to compare the temporal dimensions of different events, and thus support the third query class.

In the TRIAD model, Qian and Peuquet (1997) used quadtrees to achieve compact representation of location-based view and retrieve location-based changes. An extension of R-tree indexing that maintains an index for attributes changes over arbitrary time

intervals was used in addition to the conventional B-tree used for checkpoint-based event indexing for time-based view. Qian and Peuquet (1998) designed a visual query language called VSQL, which combined both data base query capabilities and geographic data presentation functionality.

Activity-based Transportation Research and Temporal GIS-T

Activity-based approaches became the focus of transportation demand analysis and modeling in the mid-1970s (Pas, 1990). In this approach, travel is treated as a derived demand to carry out individuals' social and economic activities at spatially separated locations. Goodwin (1983) defines the activity-based approach as "the consideration of revealed travel patterns in the context of a structure of activities, of the individual or household, with a framework emphasizing the importance of time and space coordinates". From this definition, several important characteristics can be identified. Transportation demand analysis focuses on activities but not trips. Trips are no longer considered as the basic units of observation. The activity-based approach attempts to reveal travel patterns within the context of individual or household, thus it is a disaggregate approach. Activities and travel can only be performed under the constraints of space and time. Each activity must be performed at a location and a time point.

The activity-based approach emphasizes travel activity patterns and the dynamics of travel behavior. GIS has not been used in activity-based transportation study until recently (Miller, 1991). In the activity-based approach, travel and activities have to be represented as a dynamic process that is referenced in both space and time dimensions. Spatiotemporal GIS-T data models have to be developed. However, up to now, there is

not an operational spatiotemporal GIS-Transportation (or temporal GIS-T) data model available. In fact, there are very few articles touching temporal GIS-T (Shaw, 1999; Shaw et al, 2000; Dueker, 1999; Goodchild, 2000; Moreira et al, 1999).

Each individual in each household makes trips every day. Trips are performed within specific spatiotemporal constraints Each trip has a starting location, an ending location, and a path. Trips can be treated as moving objects along a transportation network.

Dueker (1999) outlines three approaches to represent moving objects by incorporating a new dynamic or moving object class into GIS-T:

1. A static object with frequently changing positions;
2. A new object class with location as an attribute rather than part of the definition;
3. A moving object construct with starting location and attributes of direction, speed, and destination to define a moving object.

Moreira et al (1999) design a moving object data model similar to the third approach of Dueker's. In this model, the trajectory of moving objects is decomposed into sections that are described with variability functions. The basic data structure for each section contains the identification of each object, the valid time interval, the variability function, and initial state value (i e. each basic tuple stores time interval, initial value, and a behavior function). Then a linear approximation function to describe the movement within a valid time interval can be described. Superset and subset semantics are used to correct the imprecision.

Shaw et al (2000) propose an object-oriented conceptual framework for disaggregate travel data (i.e. trip data) (Figure 2.12) and use a relational database approach and the dynamic segmentation method to handle attributes, time, and spatial features based on a

sample travel log data set. The data associated with each trip is separated into four components to reduce data redundancy: spatial (trip ends and trip path), temporal (trip beginning time, trip ending time, and trip duration), actor (individual), and attributes (trip characteristics such as trip purpose, travel mode, etc.). With the origin and destination of each trip identified from the GIS geocoding function, a trip path (shortest path) is created based on dynamic segmentation, which can minimize spatial data redundancy. Temporal, actor, and attributes components are each associated with one relational table. These relational tables and spatial database are linked together by some key fields such as individual id and trip id. This trip data model separates spatial features and temporal components, and temporal components can be treated as attributes of spatial features. For example, trip path (line feature) stands for spatial feature for each trip; while temporal attributes (e.g. beginning time, ending time, duration etc.) can be attached as attributes for each trip path

Shaw (2000) suggests another temporal GIS-T data model called a temporal dynamic segmentation, which is based on dynamic segmentation. A temporal dynamic segmentation is different from a general dynamic segmentation in that the previous uses time as the basic measurement unit; while the next uses distance. At some locations on a trip path, there are significant speed changes. These locations can be termed as critical points. Assuming speed is constant between two adjacent critical points, temporal intervals can be calculated. A temporal route can be created based on these different temporal intervals. This data model combines spatial and temporal data together (i.e. spatial and temporal data determine a spatiotemporal object) and has a simple data structure, thus data manipulation and retrieval related to time become easy. In a class

project dealing with trucking inventory analysis using GPS data, this data model was implemented in ArcInfo*.

A spatiotemopral data model for disaggregate travel data should be able to answer various queries that are based on location, time, attribute, or a combination of these variables. Shaw et al (2000) propose various types of spatiotemporal database queries using the following examples.

"Time-based query

1. Where were the trips that occurred between 7 am and 8 am on March 21, 1999? *(control time to find location)*
2. What were the trip purposes of those trips that occurred between 7 am and 8 am on March 21, 1999? Who made those trips? *(control time to find attributes)*
3. What was the change of trip patterns from 7 am to 9 am on March 21, 1999? *(control time to find the change in location)*

Location-based query:

4. What were the trips that traveled on University Drive between 15th Street and 16th Street? *(control location to find attribute)*
5. In which time period on March 21, 1999 did University Drive between 15th Street and 16th Street have the most trips? *(control location to find time and attribute)*

Attribute-based query:

6. Where were the work trips? *(controls attribute value to find location)*
7. What is the temporal distribution of all work trips? *(control attribute value to find time)*

Combination query:

8. What was Joe Johnson's travel pattern on March 21, 1999? *(control date and attribute to find location and time)*
9. Did Joe Johnson show different daily travel patterns on March 21 and March 22 of 1999? *(control attribute and time to find change pattern)*
10. What were the trips made by Joe Johnson that had less than 20 minutes time lapse between consecutive trips? *(control time and attribute to find trip chaining behavior).* "

In these query types, combination query is the most difficult one, since two variables (among time, location, and attribute) have to be controlled to measure the other one. Without an efficient index method, it would be impractical to make a combination query

* This class project was finished by Mo Chatterjee, Feng Lu, David Ralston, Amy Rose, and Xiaohong Xin, May, 2000

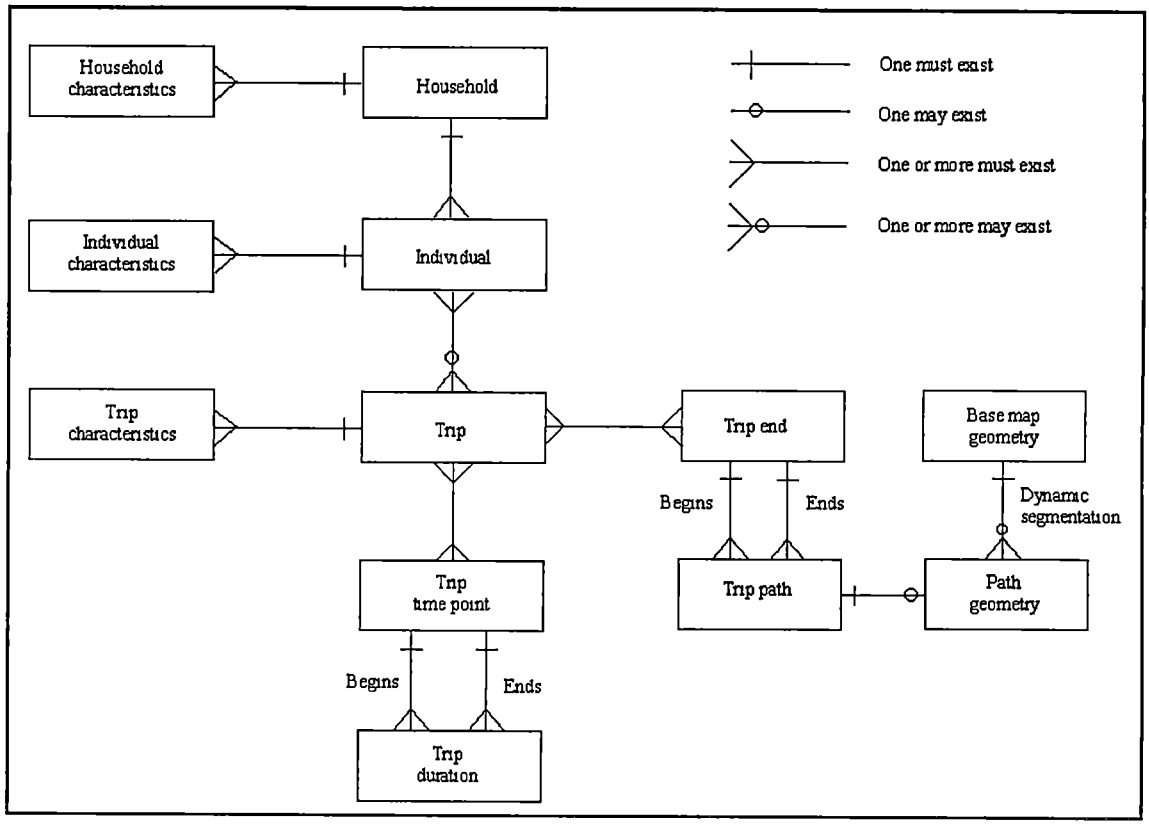


Figure 2.12 A conceptual framework of the relationships between entities in a disaggregate travel data set (source: Shaw, 2000)

for a large database. Unfortunately, there are almost no spatiotemporal indexing methods available. This is the reason for this study.

Chapter 3

Data Description and Methodology

As stated in Chapter 1, the purpose of this study is to build a spatiotemporal index for disaggregate travel data. Specifically, this thesis intends to query what trips pass through one or more streets during a time window This chapter explains why it's necessary to build a spatiotemporal index, and presents the methodology and flow chart for building one A sample data set and a trip data model are described first

Sample Data Description

The sample data used in this study is a dummy trip log data set in DBASE format that describe travel characteristics of some households and individuals in the Knoxville Metropolitan Area (Figure 3.1). This data set consists of 83 trips (records), and travel characteristics of people who make these trips In activity-based modeling, travel activities research is based on individuals and households Each trip is made by one individual in one household. In this trip log data set, there are attributes related to each trip such as starting address, ending address, starting time, and ending time. In addition, household and individual information is also recorded. Since the trip log data set involves the spatial, temporal, and other travel-related data items at the individual level, it can be used to study travel activity patterns and the dynamics of travel behavior in a GIS environment

Table 3 1 describes the main data fields in the trip log file There also are other fields in this file such as trip purpose and individual's attributes. Since the purpose of this

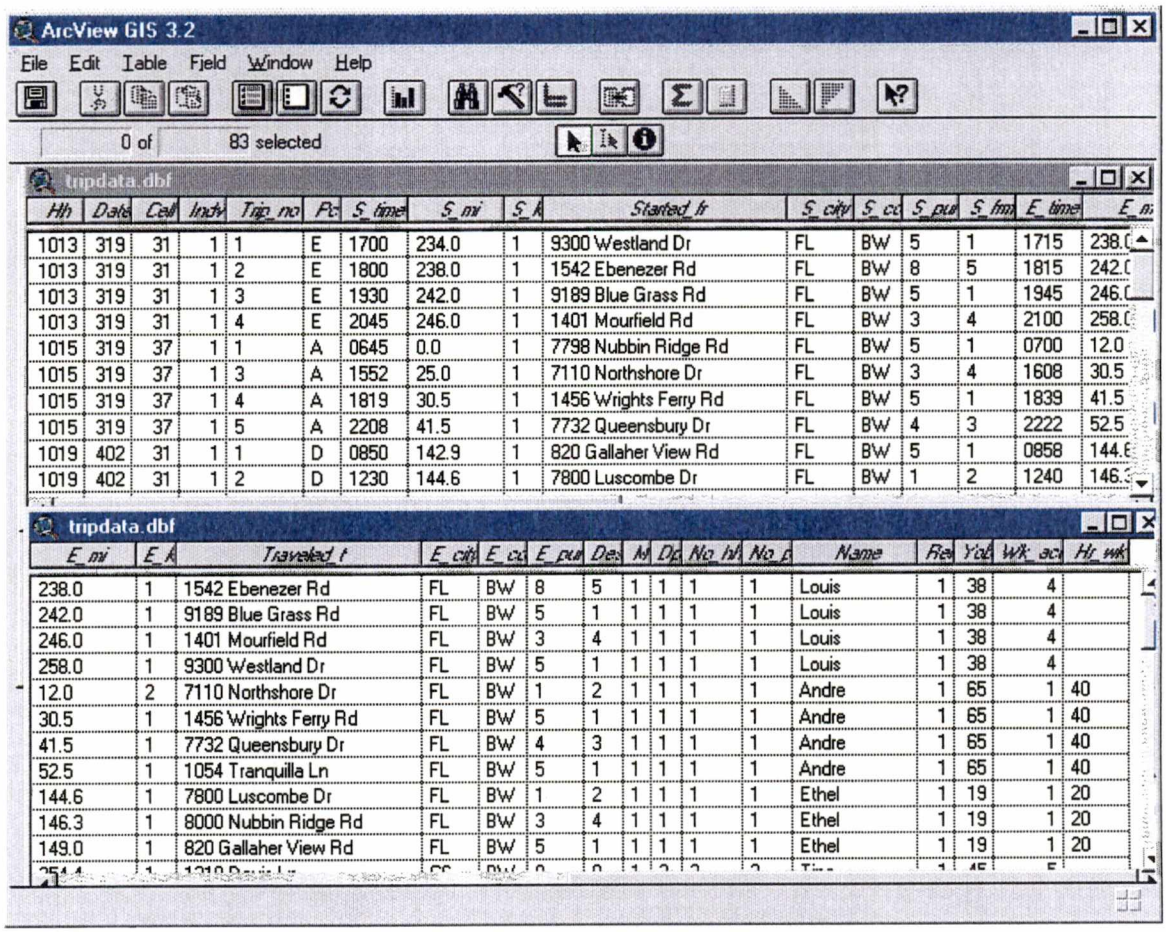


Figure 3.1 Trip sample database of Knoxville. (Notes: due to the length of table records, the original table is displayed as top and bottom tables here.)

Table 3 1 Name and description of main fields in the trip log file

Field Name	Description
Hh	identification number of household
Date	date of each trip, concatenation of month and day
Indv	identification number of individual in each household
Trip_no	identification number of trip made by each individual
S_time	trip starting time, concatenation of hours and minutes
Started_fr	starting address of each trip, in the format of US streets
S_city	starting city name of each trip
S_co	starting county name of each trip
E_time	trip ending time, concatenation of hours and minutes
Traveled_t	ending address of each trip, in the format of US streets
E_city	ending city name of each trip
E_co	ending county name of each trip
Name	name of individual

thesis is to query what trips pass through one or set of streets, only the main fields are listed and described *Hh* is the identification number of household represented by a unique number *Date* is the date of each trip, represented by concatenation of month and day. A *Date* value of 319 means March 19 i e. the trip was performed on March 19. *Indv* is the identification number of individual in a household. *Indv* is only unique in each household. In the trip table, each individual can make one or more trips. *Name* represents the name of an individual *Trip_no* is the identification number of the trip made by one individual Similarly, *Trip_no* is only unique for each individual Each trip can be uniquely identified by concatenation of *Hh, Date, Indv,* and *Trip_no S_time* and *E_time* are trip's origin time and destination time, represented by concatenation of hours and minutes. *Started_fr* and *Traveled_t* are trip's origin address and destination address in standard US streets format Thus a trip and its related travel information can be

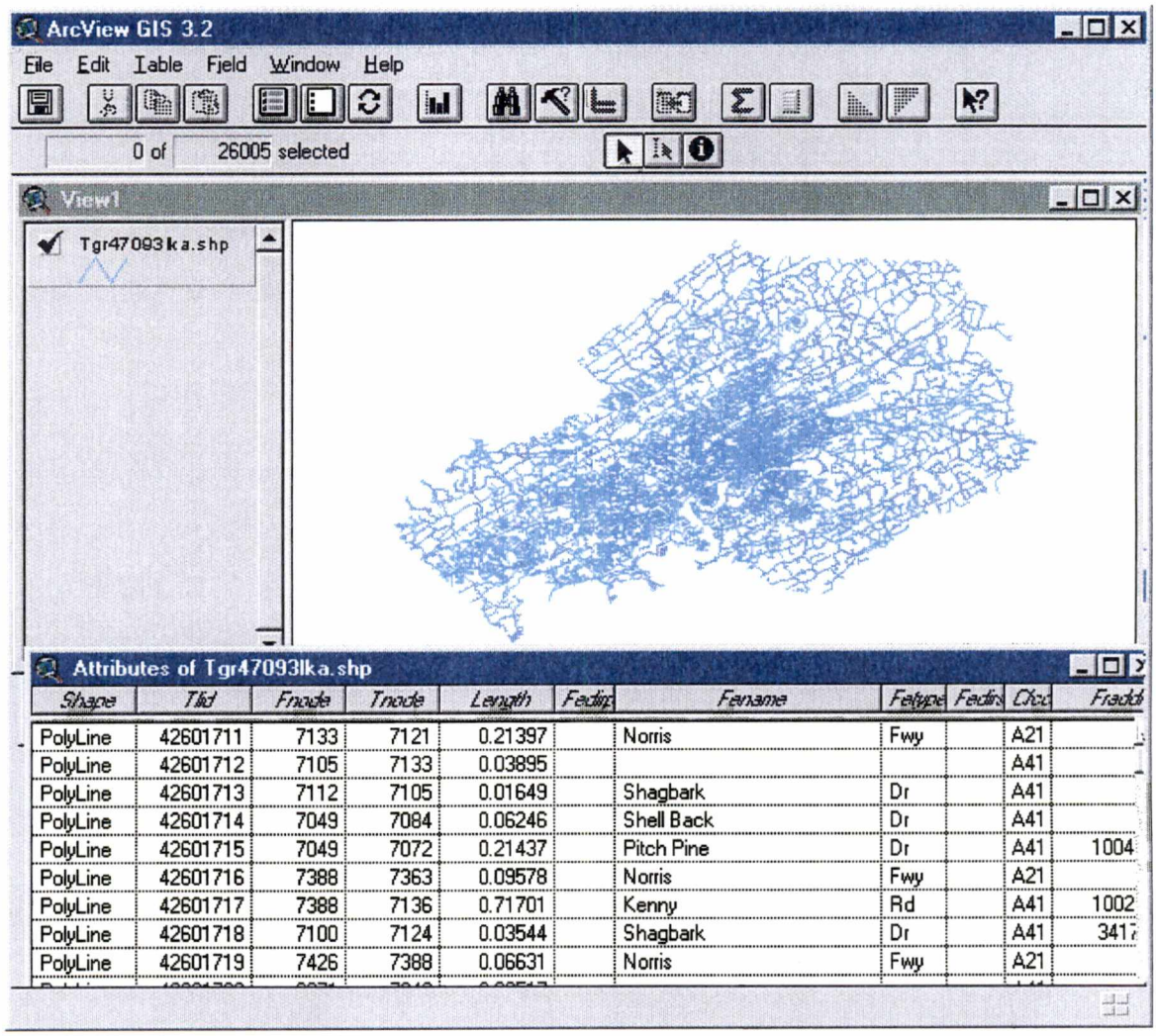
expressed For example (see Figure 3.1), the first trip in trip log table (first record) was performed on March 19 by individual 1 (*Indv*) of household 1013 (*Hh*), whose name is Louis (*Name*); it is the first trip (*Trip_no*) made by this individual on this day This trip started from 9300 Westland Dr (*Started_fr*) at 17:00 (*S_time*) and ended at 1542 Ebenezer Rd (*Traveled_t*) at 17 15 (*E_time*).

In this trip log file, there are some incorrect addresses This is true in the real world Sometimes, people don't know exactly where they are during travel or fill survey forms incorrectly These incorrect addresses can be neglected when building a trip data model. This trip log file is small in size (only 83 trips), but it's enough for this study, since we are only concerned about spatiotemporal query methodology

The base coverage is Knox County street shape file, converted from a TIGER file (Figure 3 2) There are totally 26005 street segments in Knox County. The trip log file can be mapped onto this network based on trips' starting addresses and ending addresses.

Trip Data Model (Representation of Trip Log Data Set)

In order to perform spatiotemporal queries, the trip log data set must be represented in a GIS environment (i.e a trip data model must be created) Shaw (2000) suggests that there are two kinds of representation issues for trip data· representation of trip locations, and representation of complex relationships among the different entities (households, individuals, trips, and their spatial, temporal and attribute data). As stated in Chapter 2, Shaw separates trip data into four components spatial (trip ends and trip path), temporal (trip time points and trip duration), actor (individual or individuals involved in



Shape	Tlid	Fnode	Tnode	Length	Fedirp	Fename	Fetype	Fedirs	Cfcc	Fradd
PolyLine	42601711	7133	7121	0.21397		Norris	Fwy		A21	
PolyLine	42601712	7105	7133	0.03895					A41	
PolyLine	42601713	7112	7105	0.01649		Shagbark	Dr		A41	
PolyLine	42601714	7049	7084	0.06246		Shell Back	Dr		A41	
PolyLine	42601715	7049	7072	0.21437		Pitch Pine	Dr		A41	1004
PolyLine	42601716	7388	7363	0.09578		Norris	Fwy		A21	
PolyLine	42601717	7388	7136	0.71701		Kenny	Rd		A41	1002
PolyLine	42601718	7100	7124	0.03544		Shagbark	Dr		A41	341
PolyLine	42601719	7426	7388	0.06631		Norris	Fwy		A21	

Figure 3.2 Knox County network shape file (converted from tiger file)

each trip), and attributes (trip characteristics such as trip purpose, travel mode, etc) If all the data related to trips are stored in one table (Figure 3.1), there is significant data redundancy.

In this study, the main concern is how to perform a spatiotemporal query for trips on a street network The representation of complex relationships among the different entities will not be explored i.e. the sample trip log table will not be separated into different tables However, representation of trip locations must be explored

Shaw et al (2000) states there are two approaches for representing the location of a trip. The point-based approach defines each trip as two point locations (i e origin and destination point). However, using this approach, it's difficult to visualize the spatial pattern of an individual trip, since there is no a path between each trip origin and destination The path-based approach defines each trip as a path (each trip traverses a path). The spatial pattern of an individual trip can be easily visualized using the path-based approach. Since it's difficult to get detailed trip path information, most travel surveys only collect trip origin and destination locations. In order to represent trip path in a GIS environment, a candidate path (usually the shortest path between trip origin and trip destination locations) must be created for each trip In this study, I also assume that trips always take shortest paths. This assumption is reasonable, as shortest paths can show spatial patterns of trips. The global positioning system (GPS) can be used to record location and time information of moving objects, thus it can record detailed path information (locations and times along trip path) for each trip, and it has been used in some transportation studies (Quiroga et al, 1998) In the future, GPS is likely to be commonly used in collecting travel survey data

In order to represent trip path in a GIS environment, trip ends (trip origins and destinations) must be geocoded first In this thesis, trip ends are geocoded in ArcView Once the locations of trip ends are known, trip paths must be created. There are two approaches to represent trip paths One approach is to create a trip path shape file, the other approach is to use dynamic segmentation method to represent trip paths The trip path shape file approach stores trip path locations and attributes in a new shape file. Since all the trips take place on a street network, trip path locations can be found on the network. Creating a new trip path shape file will duplicate trip path geometry Dynamic segmentation associates numerous feature attributes with a line feature, a collection of line features, or a portion of a line feature without changing the underlying geometry of the lines. The dynamic segmentation approach does not create a new file to store trip path locations i e it does not duplicate trip paths' geometry Trip path locations are still stored in the street network, and trip path attributes are associated with the street network.

Using the approach of creating a trip path shape file, the spatiotemporal query problem can be solved using ArcView and C++ together *. ArcView doesn't create route systems based on dynamic segmentation. In ArcView Network Analyst Extension, shortest paths can be written into a shape file (i e trip paths can be saved as a shape file) Since this spatiotemporal query is to query during a time window, which trips pass through one or more streets, even if there is a trip path shape file, this shape file should have relationships with the street network (i e what streets one trip pass through should be recognized)

* Personal communication with Dr Bruce Ralston, March, 2000

In ArcView Network Analyst Extension, there are two commands used to create shortest paths as a shape file WritePath and WritePathLong Using the WritePathLong command (aNetwork.WritePathLong(aFileName)), the relationship between a trip path shape file and a street network can be identified In fact, the Network.WritePathLong request writes a record to the result theme feature table for each line feature (street) traversed by a trip path segment (ArcView online help). Using this command, one trip path is segmented by each street, through which it passes, into trip segments.

Table 3.2 illustrates the fields written to the result theme feature table (i e trip path attribute table) *Path_id* stands for trip segment identification number, while *N_recnum* is the record number of the line feature (i e street's identification number). *N_recnum* links the trip path shape file to the street network Based on the trip path feature table, all trips that pass through a street can be identified Therefore, the spatiotemporal query can be performed

Table 3 2 Trip path feature table description after using WritePathLong command. (Source ArcView V 3.2 Online Help).

Field Name	Description
Path_id	Identification number of the trip path segment to which the line feature belongs.
F_label	The name (label) of the stop at the start of the segment to which the line feature belongs
T_label	The name (label) of the stop at the end of the segment to which the line feature belongs
F_cost	The cost of reaching the beginning of the line feature
T_cost	The cost of reaching the end of the line feature
N_recnum	The record number of the line feature.
N_travdir	The direction the line feature is traversed A value of FT means the line is traversed in the same direction it was digitized A value of TF means it is traversed in the opposite direction it was digitized

Creating trip path shape file in ArcView in this way results in data redundancy Since trips are assumed to take place on the street network, creating another trip path shape file will duplicate the geometric representation of each trip path (trip path geometry information is already contained in the street network) Representing trip path using dynamic segmentation can minimize data redundancy ArcInfo route systems are an implementation of dynamic segmentation. In dynamic segmentation, there is no need to duplicate trip path geometry or coordinates. Trip paths can be associated with a street, a collection of streets, or a portion of street of the network. Trip attributes such as starting time and ending time are attached to trip paths Since dynamic segmentation method can reduce data redundancy, trip paths are represented using the dynamic segmentation approach in this thesis Dynamic segmentation and its implementation in ArcInfo will be explored in Chapter 4

Comparison of Common GIS Approaches and the 2-D Trees Approach

After a trip path system is created, the spatiotemporal query can be performed with or without a spatiotemporal index. In this study, a 2-D trees approach is used to build a spatiotemporal index In order to compare the 2-D trees approach and common GIS approaches, the section table and Big O notation are discussed first.

The section table is the key table in a trip path system for the spatiotemporal query In the ArcInfo route system, there are three related tables: RAT (route attributes table), SEC (section) table, and AAT (arc attribute table) (Figure 3 3). The basic unit of a SEC table is section, which is a line feature or a portion of line feature A trip path is

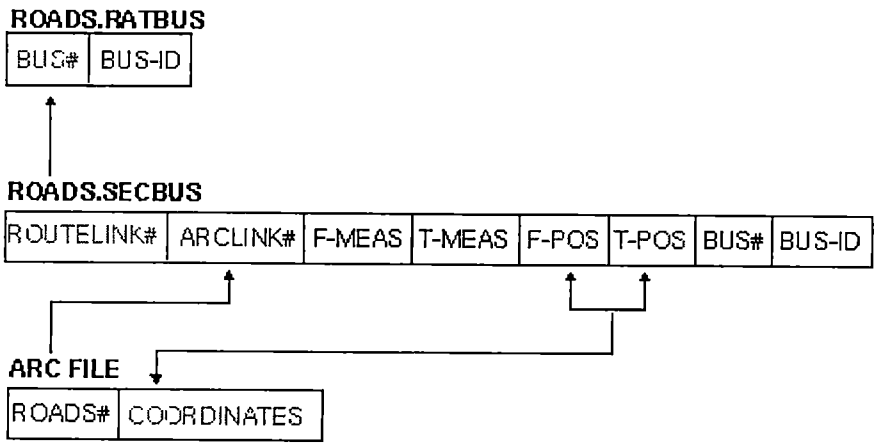


Figure 3 3 A route system named BUS on a ROADS coverage. This figure shows relationships between RAT, SEC, and ARC files. (Source: ArcInfo V. 7.2.1 Online Help)

composed of a set of sections In the SEC table, for each section, there are trip path (passing through this section) internal id, street (containing this section) internal id, and other attributes such as beginning time and ending time when one trip passes through a street attached to each record.

Big O notation is used to describe how complex an algorithm or a program is Instead of run-time analyses, big O notation derives a general assessment of the time (the number of operations) required for an algorithm or a program. Specifically, it provides an upper bound on the number of operations required. Big O notation can be defined by a mathematical formula:

Definition: $T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq c\ f(N)$ when $N \geq n_0$ T(N) or O(f(N)) is referred to big O notation.

Using a common GIS approach, the spatiotemporal query problem can be solved graphically select one street or a set of streets from the trip path system in ArcView (using SelectByPoint or SelectByRec commands) to retrieve selected streets, then related trips and sections are selected in the section table based on the selected streets' ids. From the set of selected trips and sections, a time range query can be made from starting time and ending time (when a trip passes through a street). Finally, trips passing through selected street(s) during a time window can be retrieved and shown on the map

Using the common GIS approach, there is an efficiency problem Even after graphically selecting one or more streets, the following temporal query still takes time of $O(k * n^2)$ Here k and n represent the total number of sections and number of selected sections upon selected streets in the section table, respectively. First, selecting trips and sections in the section table based on selected street(s) must search through the whole section table (linear search); then upon the selected trips, beginning time and ending time queries must be made (two linear searches) Thus temporal query takes $O(k * n^2)$ If the trip data set is small or few trips pass through each street, this is not a problem. However, for a metropolitan area, hundreds of thousands of trips are performed each day, and there are many trips passing through each street The time to make a spatiotemporal query using this common approach is prohibitive Thus an efficient spatiotemporal index or data structure must be built.

Bentley (1975) proposed a K-D-tree data structure, which handles K-dimensional query on discrete objects (point objects) A K-D-tree is useful in that it can perform range searching. A 2-D tree is a special case of a K-D tree

A 2-D tree data structure is used for two-dimensional range search. It's a binary tree and has the property that branching on odd levels is done with respect to the first key, and branching on even levels is done with respect to the second key. Weiss (1997) states that, for a perfectly balanced 2-D tree, a range query could take $O(M + N^{1/2})$ in the worst case, to report M matches for N nodes

Since 2-D tree structure is designed for a two-dimensional range query, it can be used to query starting time and ending time when trips pass through one or more streets. In this thesis, the 2-D tree approach is used to create an array of 2-D tree structures for the spatiotemporal query. From the section table of a trip path system, all streets where there are trips passing through are recognized. For each street, all trips passing through it are identified. Thus one 2-D tree can be built based on starting time and ending time of trips passing through a street for each street Therefore, an array of 2-D trees ordered by street internal ids is created, one for each street having trips passing over it. When performing a spatiotemporal query, users can first graphically select one or more streets from the trip path system in ArcView, then set a time window to retrieve trips from the array of 2-D trees. After users select the street(s), the temporal query using an array of 2-D trees could take $O(\log L * (M + N^{1/2}))$ in the worst case Here L , M, and N represent the number of streets on which there are trips traversing, the number of matched trips, and the number of trips traversing a specific street, respectively Since $L < k$ (the number of streets is always less than the number of sections), $N \leq n$ (the number of trips on a street is less or equal to the number of sections on a street), and $M < N$, $O(\log L * (M + N^{1/2})) << O(k * n^2)$. This means 2-D tree approach speeds up the spatiotemporal query greatly

Consider the following example Assume there are 100,000 trips in Knox Metropolitan Area each day, and these trips are segmented in 500,000 sections in the trip path system. There are 10,000 trips traversing Cumberland Ave., and 1000 trips are within 8 00 am - 9 00 am. There are totally 26005 streets in this area If users query during 8·00 am – 9·00 am, what trips pass through Cumberland Ave., the common GIS approach will take 500,000 * (10000 * 10000) = 5 * 10^{13} operations at the worst case, the 2-D tree approach will only take log 26005 * (1000 + 100) = 177386 operations at the worst case We can see for a large data set, the 2-D tree approach performs the spatiotemporal query much faster then the common GIS approach.

Methodology and Flowchart

Figure 3 4 shows the flowchart of building a spatiotemporal index From the trip attribute table, a trip end location shape file is created using ArcView's geocoding function. Arcinfo's dynamic segmentation modules are used to create shortest path trip system Based on the resulting trip path system, an array of 2-D trees (according to beginning time and ending time when trips pass through street segments) is built using a C++ program and an Avenue script written for this thesis. The 2-D trees are linked back to the trip path system to retrieve trips. The following shows the four main steps

Step 1 Geocode Origin and Destination Ends of Trips

ArcView is used to geocode trip ends In this step, all the origin addresses and those destination addresses that are each individual's last trip end on each day are geocoded Since each individual performs a trip or a chain of trips on each day,

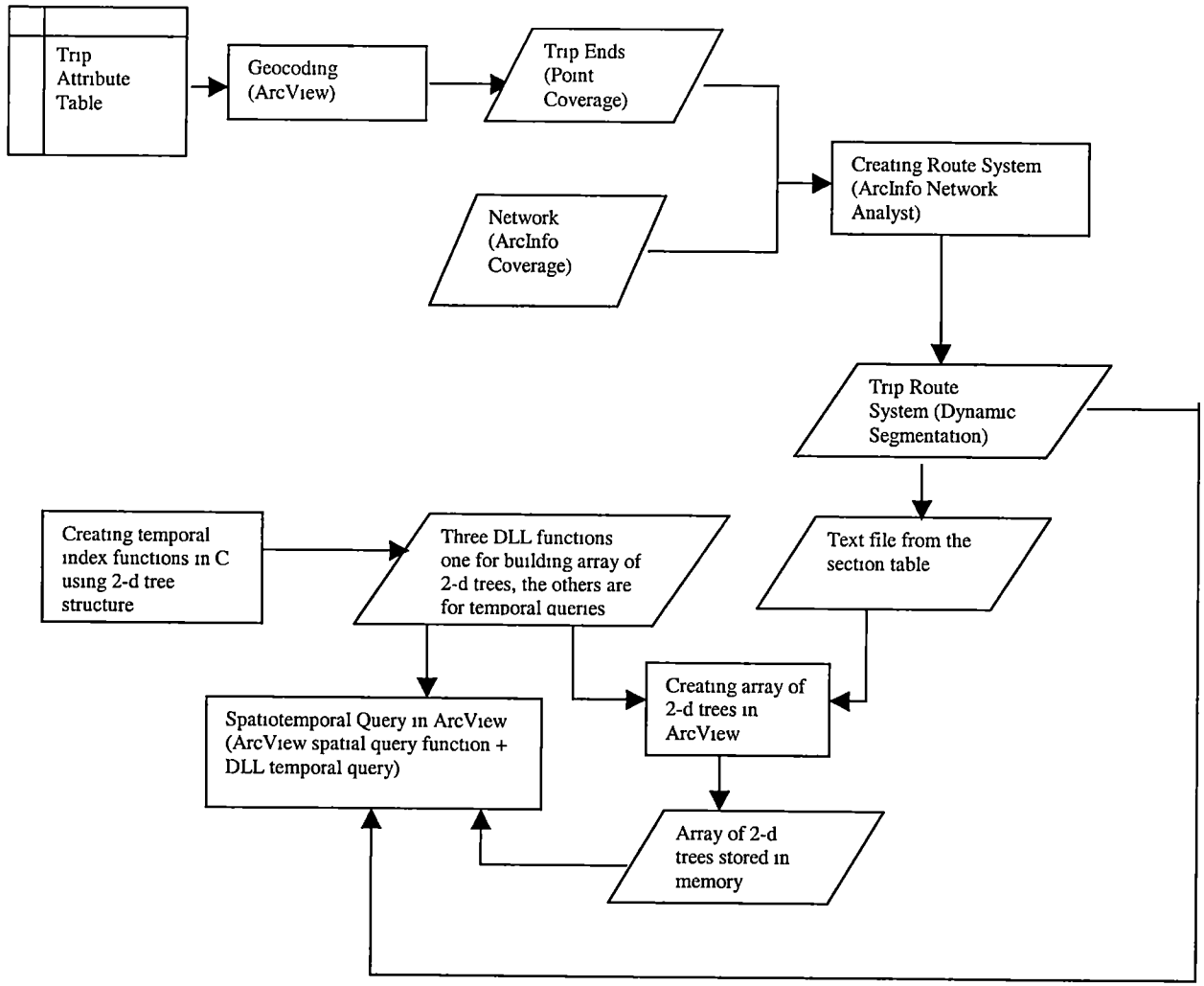


Figure 3 4 Flow chart of building spatiotemporal index for disaggregate transportation data

geocoding all trip origin addresses and the last destination address on each day for the individual can efficiently and sequentially describe the spatial patterns of the trip or trip chain, and easily create trip shortest paths

Step 2 Create Trip Path System in ArcInfo

From the trip ends point shape file (geo-coding results), a trip path system is created in ArcInfo network analysis's path-finding function After a trip path system is created, its RAT table is joined to the section table based on route (trip) internal id. Starting time and ending time fields are added to the section table Assuming speed is constant for each trip, starting time and ending time when a trip passes through a street (section) are linearly interpolated Then the section table is exported into a text file from which an array of 2-D trees is built.

Step 3 Build Temporal Index in C++ and ArcView

In C++, three DLL functions are created. The first builds an array of 2-D trees for each street in the route system The resulting 2-D trees are stored in memory and a pointer to them is passed back to ArcView. The other two functions are for temporal query functions (query for one street, query for a set of streets respectively)

Step 4 Spatiotemporal Query in ArcView

In ArcView/Avenue, the spatiotemporal query problem is separated into spatial query and temporal query Using ArcView spatial query function, streets are graphically selected; then given a time window, one of the two temporal query DLL functions is

called to retrieves trips that pass through the selected streets The query results are shown on the map, and in a message box

Chapter 4

Creating the Trip Route System in Dynamic Segmentation

Linear Referencing Systems and Dynamic Segmentation

Different kinds of transportation features share the same digitized road network If these transportation features are represented separately from the road network, there will be much data redundancy On the other hand, many transportation data are recorded using location referencing systems such as milepost and latitude-longitude. Efficient data models must be developed to handle these various location schemes. In the late 1980s, the concepts of dynamic segmentation for linear features and linear referencing were proposed

Linear referenced data are those data located on a linear transportation feature using an offset distance from a known point on the feature and following the feature's path to the desired location (Dueker et al, 1997) Linear referencing systems are used in GIS-T to integrate linearly referenced data and geographic locational data. A linear referencing system consists of three main components a transportation network, a location referencing method; and a datum (Vonderohe et al, 1995, Dueker et al, 1997) The transportation network contains node-arc topological relationships The linear referencing method is a way to identify a specific location with respect to a known point along a defined path The datum is a set of objects that serve as the basis for locating the linear referencing system in the real world

Linking linear referenced data based on a linear referencing method to the transportation network requires a segmentation scheme. Variable–length or dynamic segmentation is such a segmentation scheme for controlling the attributes of linear referenced data and measures the locations where this attribute exhibits a specific value (Miller and Shaw, 2000). For example, the bus fares along a bus route on a street network might show different values on different parts of the route, which vary in length. These different parts can be considered as dynamic segments.

ArcInfo uses a relational data schema to support a dynamic segmentation model based on a three level of structure of sections, routes, and events. Routes are linear features such as a river, a highway or a pipeline. Each route has a "mile-post" type of measure defined along it. As partial arcs or whole arcs, sections are the basic organization units to integrate routes, events and the transportation network. A route is composed of sequences of sections (Figure 4.1).

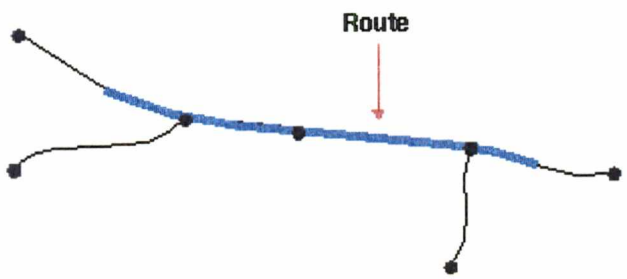


Figure 4.1. A route defined on a set of four arcs. The start and end points of the route do not have to coincide with the start and endpoints (nodes) of the arcs.

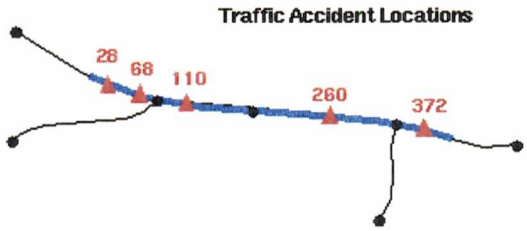


Figure 4.2. An event database of traffic accidents on a route. Each event is recorded in terms of a linear measure along the route and graphically represented with a point maker. (Source: ArcInfo online help).



Figure 4.3. Pavement data contained in an event database. The pavement events are defined in terms of a linear measure along the route. (Source: ArcInfo online help).

Events are transportation phenomena that occur on the routes. There are three event classes: point events, line events, and continuous events. Point events take place at a single measure along a route (Figure 4.2). Line events take place between two measures along a route (Figure 4.3). Line events can be discontinuous (have gaps) or continuous (no gaps). Continuous events take place between two measures along a route, and have no gaps. Events are a portion of a route or a single point on the route.

In ArcInfo, dynamic segmentation function is implemented in a route system. A route system is composed of a group of routes such as a bus route system Two related tables a section table (or SEC table) and a route attribute table (or RAT) together define a route system (there is one SEC table and one RAT in a route system) SEC table and RAT are tied to the network arc attribute table (AAT) through some key fields. Figure 3 3 shows the interrelationships among AAT, SEC, and RAT.

An AAT table contains fields such as Arc#, Arc user id, from node, to node, left poly, right poly, length etc Arc# is the internal id of each arc An AAT is connected with a SEC table by Arc# of the AAT and Arclink# of the SEC table Arclink# in a SEC table is the internal id of an arc with which a section is associated. Routelink# in a SEC table is the internal id of a route to which a section belongs. The Subclass# is the internal id of a route in the AAT SEC table is connected with the RAT by Routelink# of the SEC and Subclass# of the RAT Therefore, RAT, SEC, and AAT are linked together

In addition to Routelink# and Arclink#, a SEC table also contains F-meas, T-meas, F-pos, and T-pos F-meas and T-meas are the starting and ending measures of a section. The measure item can be defined to satisfy different needs. For example, the measure item might be length, time, or travelling cost The F-pos and T-pos are the starting and ending positions along an arc for one section. The F-pos and T-pos are expressed as percentages For example, if a section is located from 40% to 60% of an arc, then F-pos and T-pos equal to 40 and 60 respectively. The F-pos and T-pos are used to associate one section to partial or a whole arc

In RAT, in addition to subclass# (route internal id), there is also a field called subclass-id (route user-id) This route user-id is defined by users, and can be used to link

with route attributes and events. Events along a route system are usually stored in an event table (Info, DBASE or Text file). Events are tied to a route system by route user-id and off-distance measures on a route.

Geocode Trip Ends

Address geocoding in ArcView is a process to create a shape file based on an address data file in tabular form and a reference feature theme The reference feature theme contains address attributes associated with geographic features The address data file or event table contains an address field for each record to match against the reference feature theme ArcView compares addresses in the event table with the address attributes in the reference feature theme (or matchable theme) Based on some standards and rules, ArcView can decide whether addresses match. When a match is found, locational coordinates are derived from the matched feature in the matchable theme and assigned to the address in the event table

Figure 4 4 shows the geocoding object model in ArcView. In the center of this model, the MatchSource class performs indexing and searching on the matchable theme Each matchable theme has a MatchSource associated with it. A MatchSource can be created by using a predefined AddressStyle (such as "US Streets With Zones") as a template The MatchKey class performs street address standardization for the event table. The MatchKey is applied to the MatchSource to a collection of possible match candidates (MatchCand) called MatchCase The MatchCand's best candidate's score is compared with a minimum match score to determine if it's an acceptable address to be made The GeoName as a feature theme can store the final geocoding result

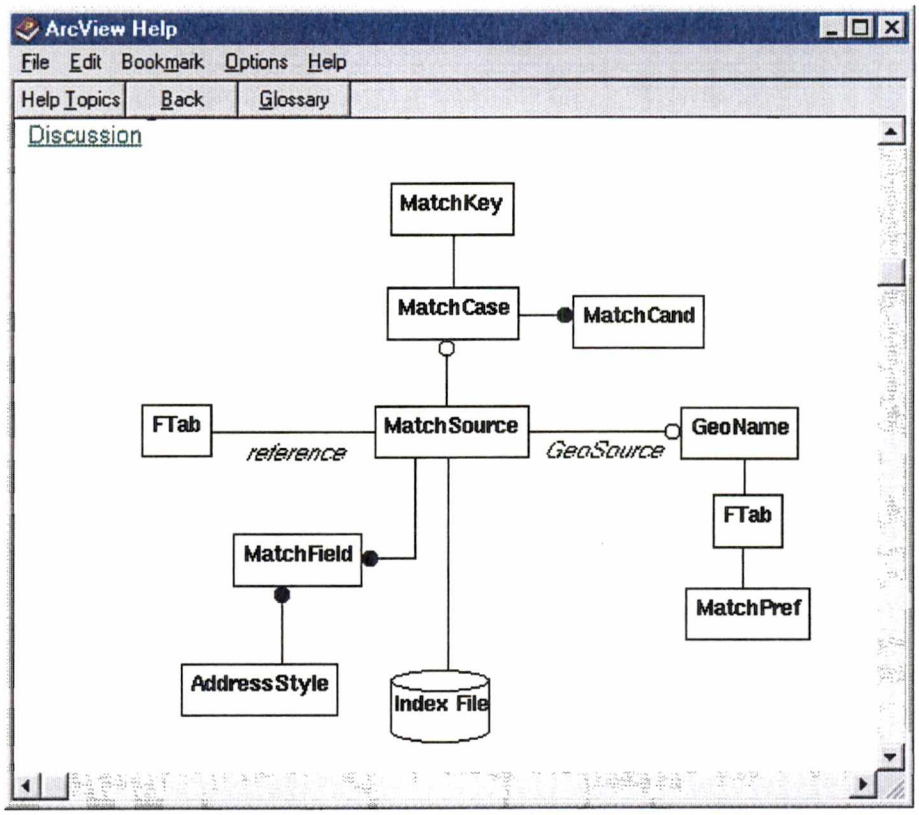


Figure 4.4. Address geocoding object model in ArcView. (Source: ArcView V. 3.2 Online Help)

As stated in Chapter 3, in order to create a trip route system, all trip origin addresses will be geocoded; in addition, those trip destination addresses that are the last ending addresses for each person on each day also will be geocoded. In the sample data set, each person on each day must have one trip or a chain of trips. The sample data set is organized by ordered travel behaviors for each person in each household (i.e. trips are recorded sequentially for each person). If one person only makes one trip, then the origin and destination addresses are geocoded to make a trip path. If one person makes a chain of trips, then all the origin addresses and the last destination address are geocoded to

make trip chains Only geocoding origin addresses will miss some trips ends, while geocoding all origin and destination addresses will result in much data redundancy, since most of trip destination addresses can be found in the next trip's origin address field. Even the geocoding approach used here has some data redundancy Some trips might share the same origin addresses, which will be geocoded multiple times However, using this approach, trip paths can be conveniently created in ArcInfo, and trip attributes can be attached to these trip paths. The next section will show this advantage.

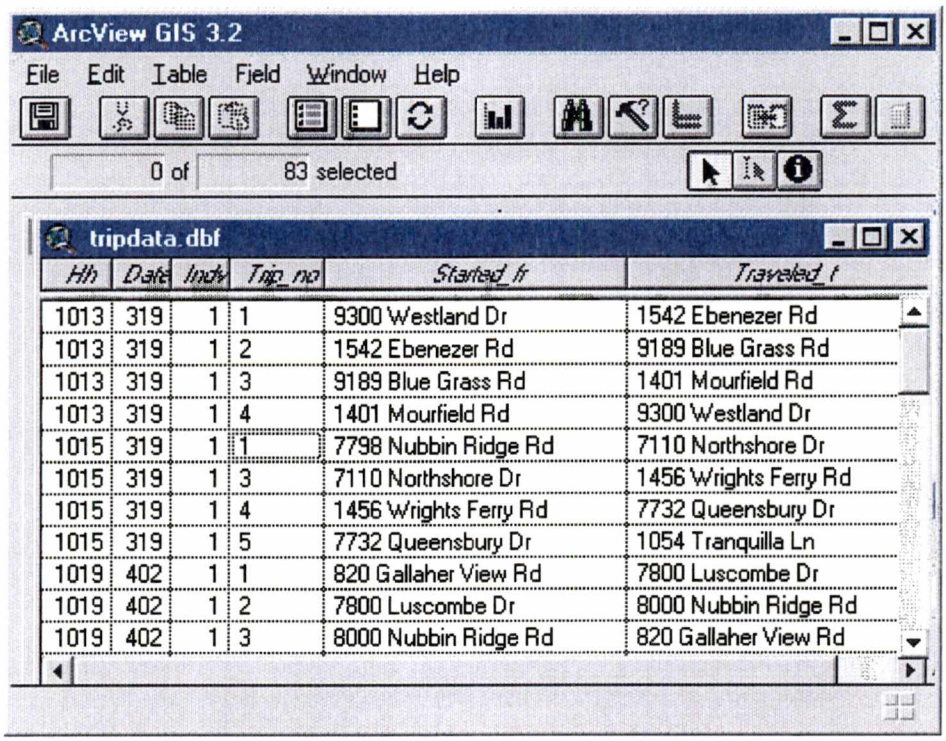
Figure 4 5 shows a subset of the sample data set. Person 31910131 (concatenation of Date, Hh, and Indv) made 4 trips on March 19. The trip chain of this person pass through these addresses 9300 Westland Dr – 1542 Ebenezer Rd – 9189 Bluse Grass Rd – 1401 Mourfield Rd – 9300 Westland Dr. Thus geocoding all trip origin addresses and the last destination address (i e 9300 Westland Dr) can sufficiently describe trip chain behavior of this person

There are two Avenue scripts to perform trip ends geocoding processes. trip.match, and trip geocode. The trip match script makes the Knox network coverage matchable, while the trip geocode script geocodes addresses for the sample data set. The source code used in this study is attached as an appendix. In the next section, pseudo codes are used to describe these scripts

trip.match script: making a street theme matchable:

Input: a street theme (ArcView shape file or ArcInfo coverage)

Output: a matchable street theme



Hh	Date	Indv	Trip_no	Started_fr	Traveled_t
1013	319	1	1	9300 Westland Dr	1542 Ebenezer Rd
1013	319	1	2	1542 Ebenezer Rd	9189 Blue Grass Rd
1013	319	1	3	9189 Blue Grass Rd	1401 Mourfield Rd
1013	319	1	4	1401 Mourfield Rd	9300 Westland Dr
1015	319	1	1	7798 Nubbin Ridge Rd	7110 Northshore Dr
1015	319	1	3	7110 Northshore Dr	1456 Wrights Ferry Rd
1015	319	1	4	1456 Wrights Ferry Rd	7732 Queensbury Dr
1015	319	1	5	7732 Queensbury Dr	1054 Tranquilla Ln
1019	402	1	1	820 Gallaher View Rd	7800 Luscombe Dr
1019	402	1	2	7800 Luscombe Dr	8000 Nubbin Ridge Rd
1019	402	1	3	8000 Nubbin Ridge Rd	820 Gallaher View Rd

Figure 4.5. Trip chain characteristics of the sample data set. Only origination address (Started_fr) and destination address (Traveled_t) fields, and those fields (Hh, Date, Indv, Trip_no) to determine a person and his/her trips are displayed.

Procedure:

If the theme is matchable, then exit.

Get an address style file name object corresponding to the style object database.

Get the list of styles from the address style file name.

Get the desired style "US Streets with Zone", give it a name called addrStyle.

Set up components list for the US Streets with Zone style.

nameList = {"Fraddl", "Toaddl",

"Fraddr", "Toaddr", "NONE", "NONE",

"Fename", "Fetype", "NONE", "Zipl", "Zipr"}

Search each field in nameList from the theme

Insert found fields into a new list called attlist

Create a MatchSource object using the address style named addrStyle, the theme, and attlist

Assign the MatchSource object to the theme

trip.geocode script geocode addresses for the sample data set (event table), based on a matchable theme Descriptions of fields in the event table can be found in Chapter 3

Input: a matchable theme (street network), an event table.

Output: a geocoding result shape file

Procedure:

1 Preprocess the event table, generate one unique id field for each individual. add the last destination address for each individual on each day to the bottom of the origin address field, and generate one unique id for each trip (Concatenation of Date, Hh, Indv, and Trip_no).

Get the Vtab for the event table. The Vtab is named as addressvtab

Get the matchable theme called theTheme

Create a field called nonuniid to uniquely identify each individual on each day and add it to addressvtab.

Calculate nonuniid as concatenation of Date, Hh, Indv

Create a field called trip_id to uniquely identify each trip

Calculate trip_id as concatenation of Date, Hh, Indv, Trip_no.

For each record in the addressvtab

If this nonuniid <> next record's nonuniid (this means this record is the last record for an individual on each day), then

Append one record on the bottom of addressvtab, set the Started_fr value as this individual's Travelled_t value, and set the nonuniid as this individual's nonuniid

Loop

2. Geocode the origin address field (Started_fr) in the event table, create a point shape file

Set the matchabel feature theme (thetheme)'s MatchSource as aMatchSource

Specify the output point shape file for the geocoding result, using

GeoName Make, set the name of this shape file

Create a match key based on the standardization rules for the MatchSource.

Use the aMatchKey AllowIntersections request to support street intersection standardization.

Create a new match case consisting of a list of candidate records and information describing how well the candidates match the key

Create a new match preference used to access geocoding preferences such as spelling weight, minimum acceptable score, etc

Create a new theme feature table using aMatchSource InitGeoTheme command

For every address record there will be a record in the ftab These are currently unmatched.

For each record in the new theme feature table

Get an address for this record using aMatchKey SetKey

Find candidates for the address using aMatchSource.Search

If there are no candidates, then mark unmatched for this record and continue

Else if the best candidate exceeds the minimum specific match score, then

Mark matched for this record

Else write unmatched for this record.

Loop

3 Make marks on trip_id field of the unmatched records and their previous record In order to avoid mismatch when creating a trip route system, the unmatched record and its previous record for the same person on each day will not be considered as stops These records' trip_id is set by 99999999. Convert trip beginning time (S_time) and ending time (E_time) into seconds and store them in two new fields (startt and endt) respectively.

Add fields startt and endt to the geocoded shape file's ftab.

Convert S_time into seconds and store it in startt.

Convert E_time into seconds and store it in endt

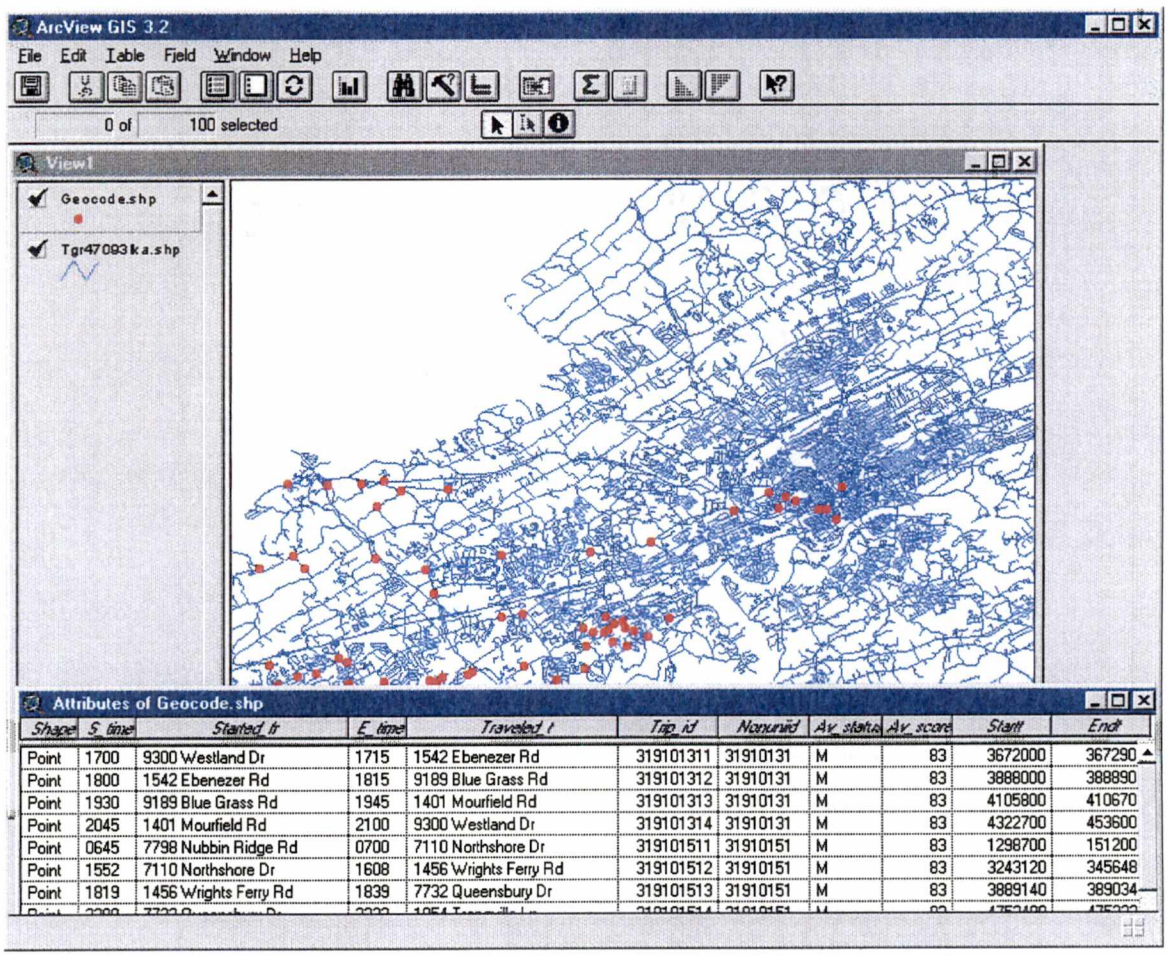
Create a unique value list for the nonunid field called valuelist

```
For each value in the valuelist
  Select records that nonuniid = value (i e  select one individual's trip chains on
  each day).
  For each record in each individual's trip chains (i e  the selected records)
    If this record is unmatched ("Av_status = U") then set trip_id = 99999999
    If this record is unmatched and has previous record, then set previous
    record's trip_id  = 99999999
  Loop
Loop
```

The sample data set's trip ends were geocoded by running these two Avenue scripts. There were 95 records geocoded successfully among 100 records The geocoding result was saved as geocode.shp (Figure 4.6)

Create a Trip Route System

Assuming each trip always take the shortest path, then a trip route system can be created using ArcInfo network analysis functions In ArcInfo's Arcplot, the NETCOVER command is used for creating a route system based on distance, time, cost, or other measures Stops represent trip ends of a route. In this case, stops are the origin and destination points of a trip ArcInfo assumes all stops are at nodes of the base network. Stops are stored in an info file, in which node-ids are recorded. The stops info file is connected with the base network by node-ids. Figure 4 7 shows a shortest path route system data file structure STREETS is the base network coverage In the stop file



Shape	S_time	Started_fr	E_time	Traveled_t	Trip_id	Nonunid	Av_state	Av_score	Startf	Endf
Point	1700	9300 Westland Dr	1715	1542 Ebenezer Rd	319101311	31910131	M	83	3672000	367290
Point	1800	1542 Ebenezer Rd	1815	9189 Blue Grass Rd	319101312	31910131	M	83	3888000	388890
Point	1930	9189 Blue Grass Rd	1945	1401 Mourfield Rd	319101313	31910131	M	83	4105800	410670
Point	2045	1401 Mourfield Rd	2100	9300 Westland Dr	319101314	31910131	M	83	4322700	453600
Point	0645	7798 Nubbin Ridge Rd	0700	7110 Northshore Dr	319101511	31910151	M	83	1298700	151200
Point	1552	7110 Northshore Dr	1608	1456 Wrights Ferry Rd	319101512	31910151	M	83	3243120	345648
Point	1819	1456 Wrights Ferry Rd	1839	7732 Queensbury Dr	319101513	31910151	M	83	3889140	389034

Figure 4.6. Geocoding result (Geocode.shp) for the sample trip data set in Knoxville Metropolitan Area. Tgr47093lka.shp is the Knox street network.

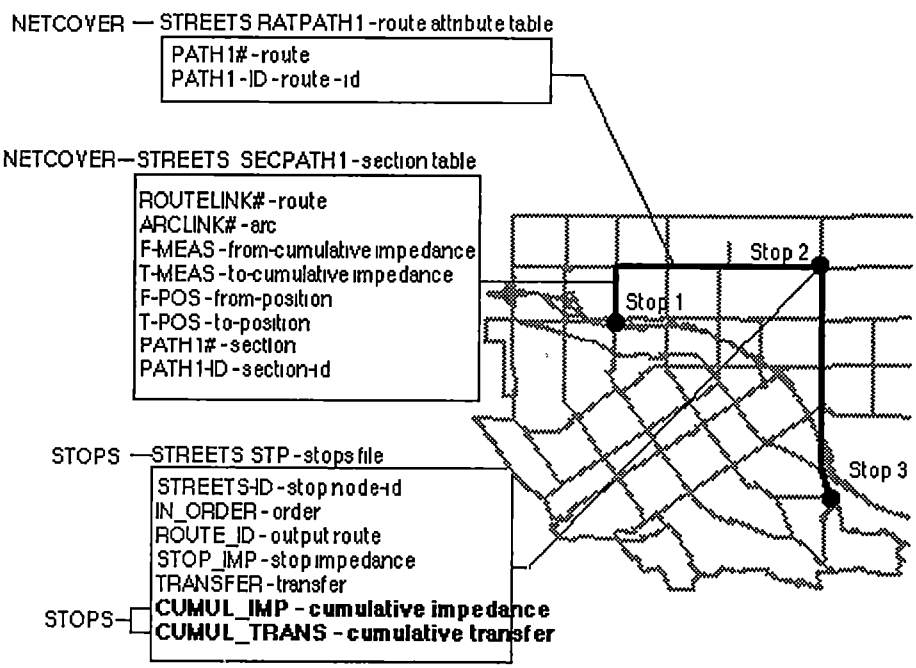


Figure 4.7. A shortest path trip route system named path1 based on STREETS network coverage and STREET STP stops info file in ArcInfo network analysis. (Source: ArcInfo V. 7.2.1 Online Help).

named STREETS.STP, STREETS-ID, the stop node-id in STREETS coverage, is a required field. Other fields in STREETS.STP are optional. For example, IN_ORDER means the creation of a route system by an order of stops. If there is no IN_ORDER field, then routes are created by the sequence of stops (i.e. according to the stop sequence in the Info table) ROUTE_ID is used to classify the stops into groups to make sequential routes among each group. STOPS_IMP represents the impedance when a trip passes through an intersection or stop. For example, travelers' speed might slow down when passing through an intersection STOPS_IMP can be set a value to represent blockade

time or cost TRANSFER represents a cost when travelers turn directions (to the right or to the left) at the intersections

Some trip ends (origin and destination points) might not be at nodes. To overcome the shortcoming of all stops being at nodes in ArcInfo, pseudo nodes must be created based on these trip ends using the split command in ArcEdit After that step is completed, network analysis functions can be used to create a route system Splitting a network using trip ends that are not at nodes destroys the topology of the network The unsplit command can be used to restore the network topology after creating the route system. Unsplit not only updates the network coverage, it also updates route systems. Before creating a shortest path route system, the geocode shp (Geocoding result from ArcView/Avenue) and the Tgr47093lka shp (the network shape file) must be converted into ArcInfo coverages

An AML code was used to automatically create a trip shortest path system After creating the trip route system, the section table is exported to a text file that is used to create an array of 2-D trees for the streets. The following shows main steps to create the shortest path system

Input: an arc coverage (network including nodes) and a point coverage (geocoding result).

Output: a shortest path route system, and a text file.

Procedure:

1 Splitting the network coverage using the point coverage and creating a stop info file The stop file contains two fields, <network>-id, and route_id. <network>-id

stores node-ids, while route_id stores associated trip_ids started from this stop route_id is used to classify stops into different groups to create shortest paths. This creates one shortest path for each trip For each individual, the starting point and ending point on each day will be added to the stop file once, all other points will be added twice Thus a trip chains can be generated for each individual (each shortest path is created by two adjacent nodes in the stop file)

In ArcEdit:

Create an info file that contains the unique values from the field of person id (nonuniid) in the point coverage (this info file contains all individual ids)

Create a stop info file including <network>-id (node-id from the network), and route_id in order to store stop node-ids and trip-ids.

For each record in the info file that contains nonuniid unique values.

From the point coverage, select all records that nonuniid = this info file's nonuniid value (i e select all the trip ends for one individual on each day)

For each record in the selected set of the point coverage

Get this record's x, y coordinates, and select arcs using these coordinates from the network coverage

If this record (point) is at either the from node or the to node of the selected arc, then

Set the node's id to a variable ni,

Else if this point (record) is not at the from node, nor the to node of the selected arc, then

Split the network using the coordinates of this point (x, y).

Set the new node's id to a variable n1.

If this record is the first or last one in the selected set of the point coverage, or this record's trip_id = 99999999, then

Add one record to the stop info file, set the value of <network>-id = n1, the value of route_id = previous selected point's trip_id (for the first selected record, its previous selected point's trip_id equals to its own trip_id)

Else if the previous selected point's trip_id = 99999999, then

Add one record to the stop info file, set the value of <network>-id = n1, the value of route_id = this point's trip_id

Else

Add one record to the stop info file, set the value of <network>-id = n1, the value of route_id = previous selected point's trip_id,

Add another record to the stop info file, set the value of <network>-id = n1, the value of route_id = this point's trip_id

Loop

Loop

2 Create a shortest path route system using network analyst functions. Each shortest path is created based on each route_id After a trip route system is created, unsplit the network to restore network's topology and update the whole trip route system

In Arcplot·

Initialize a trip route system based on the network.

Assign stops to the stop info file, assign route_id to this file's route_id, other options are default.

Finish the trip route system based on stops

Go to Arcedit:

Select all arcs from the network coverage.

Unsplit them.

3 Create a text file containing information from the SEC table of the trip route system. The text file contains Trip_id (trip's id from the geocode.shp), Arclink# (street internal number), beginning time (when the trip enter the street), and ending time (when the trip leaves the street) The point coverage's attribute table (PAT)'s startt (starting time for each trip) and endt (ending time for each trip) are already in the unit of seconds, so they can be used to interpolate the beginning time and ending time when this trip passes through a street. The RAT will be joined by the point coverage's PAT based on the <subclass>-id (user id) in the RAT and Trip_id in the point coverage's PAT. The SEC table must be joined by RAT based on the SEC table's RouteLink# and the RAT's <subclass>#. Assuming speed is constant for each whole trip, beginning time and ending time when a trip passes through a street are calculated by linear interpolation Finally, export Trip_id, Arc#, beginning time, and ending time to a text file

In Arc:

Build relationship between RAT and the point coverage's PAT (using relate function) based on RAT's <subclass>-id and the PAT's Trip_id.

Build relationship between SEC table and RAT (using relate function) based on SEC table's RouteLink# and RAT's <subclass>#

Go to ArcEdit.

Add beginning time and ending time fields to the SEC table.

For each route (trip)

Select from the SEC all sections that are parts of this route (trip)

Set the f-meas of the first selected section to variable f.

Set the t-meas of the last selected section to variable t.

Calculate beginning time = startt + (endt – startt) * (f-meas – f) / (t – f).

Calculate ending time = startt + (endt – startt) * (t-meas – f) / (t – f)

Loop

Go to Tables

Unload the section table's Trip_id, ArcLink#, beginning time, and ending time fields to a text file

For the sample problem, there are 72 trip shortest paths created. Figure 4 8 shows the trip route system (called paths)'s structure AAT (Tgr47093lka.aat), RAT (Tgr47093lka ratpaths), SEC (Tgr47093lka.secpaths), and their interrelationships Figure 4 9 shows the output text file

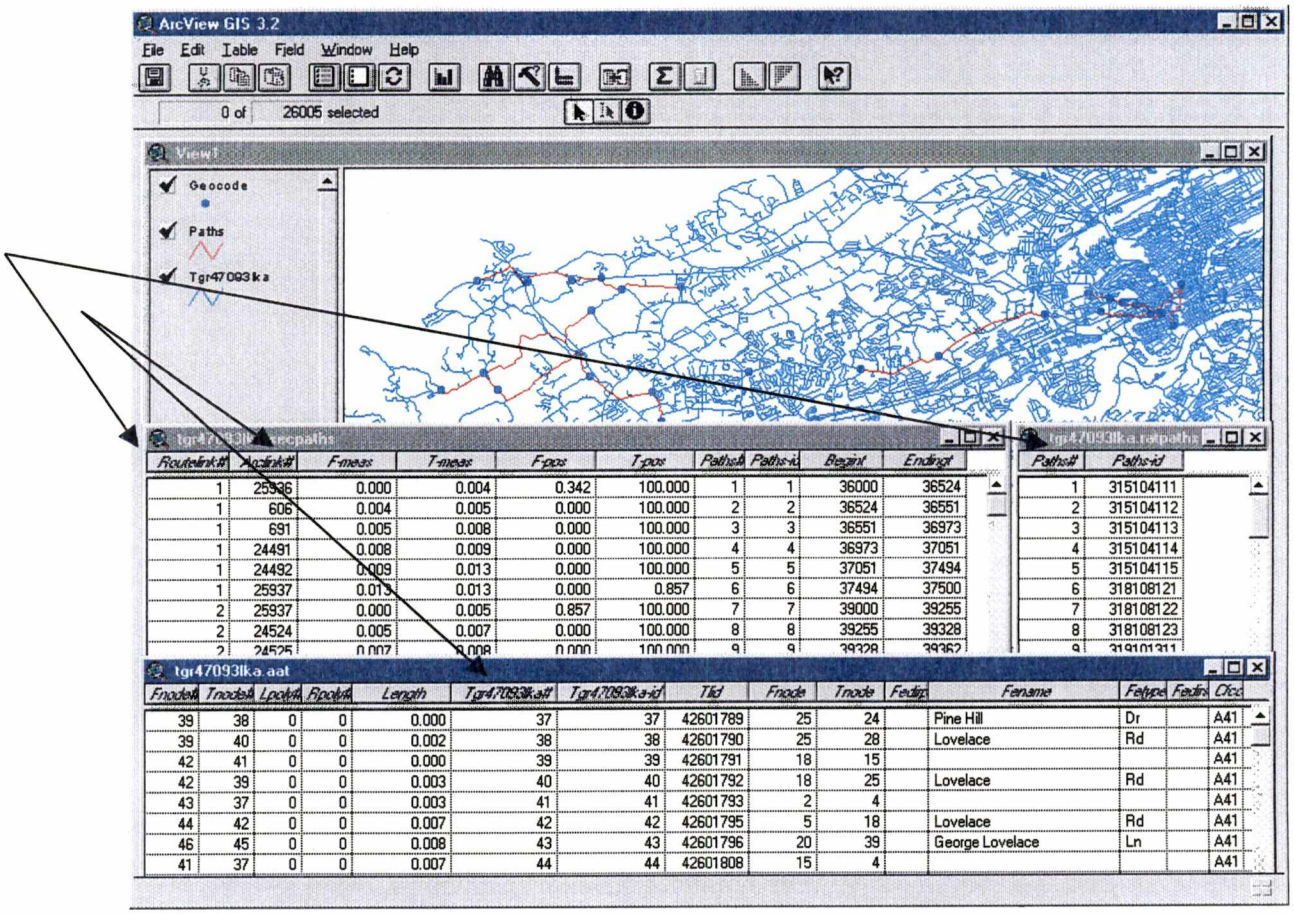


Figure 4.8. A sample trip shortest path route system on Knox network. The Arrows show the interrelationship between AAT, RAT, and SEC table.

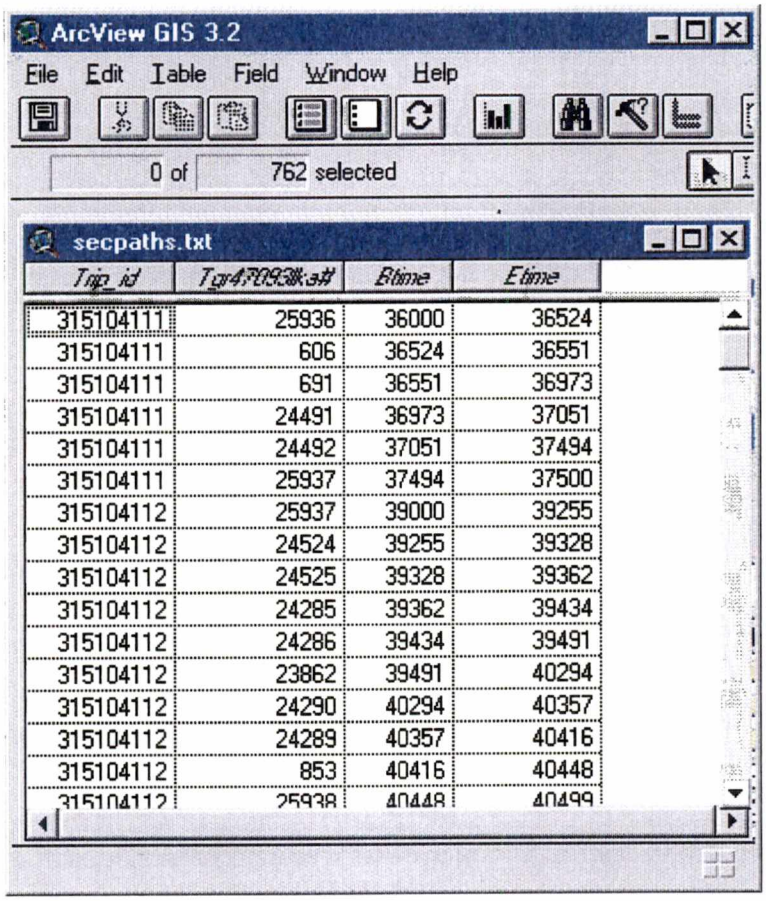


Figure 4.9. A sample text file exported from the section table of the trip route system. Trip_id is the unique id of each trip; Tgr47093lka# is the street internal id; Btime and Etime are starting and ending times (in seconds) when a trip traverses a street.

Chapter 5

Temporal Indexing

In this chapter, an array of 2-d trees for those streets on which trips pass will be built to speed up the spatiotemporal query discussed in Chapter 3. The 2-D tree structure will be described first, then the C++ program that contains the three DLL functions (one for building an array of 2-D trees, the other two are for temporal queries) is discussed Finally, the Avenue script used to call the DLL function to create an array of 2-d trees is presented.

Generic Description of 2-D Tree Structure

The 2-D tree structure is in fact a "2-dimensional binary search tree". This structure is a natural generalization of the standard one-dimensional binary search tree; it's also a special case of the K-D tree (k-dimensional binary search tree) structure The 2-D tree and the K-D tree were proposed by Bentley (1975) In a 2-D tree, each node can have at most two children (left child, right child) There are two keys for comparison to decide the position of a node in the tree For a node, one of the two keys of its left child is always less than its own; one of the two keys of its right child is greater than its own. The 2-D tree has the property that branching at an even level (assuming the root is at depth 0) is done with respect to the first key, and branching at an odd level is done with respect to the second key. For temporal querying, the first key is the beginning time and the second key is the ending time

Figure 5 1 shows a general 2-D tree structure X, Y are two keys of a node, Left, and Right represent pointing to the left child and right child respectively. Data stands for the additional data fields of a node. In order to build a 2-D tree, nodes are compared with X at even depths and Y at odd depths (the root is at depth 0). The first point (node) is the root; the second point (node) is the right child of the first point, since X of the second point is greater than X of the first point, the third point is the left child of the second point; since Y of the thrid point is less than Y of the second point and X of the third point is greater than X of the first point.

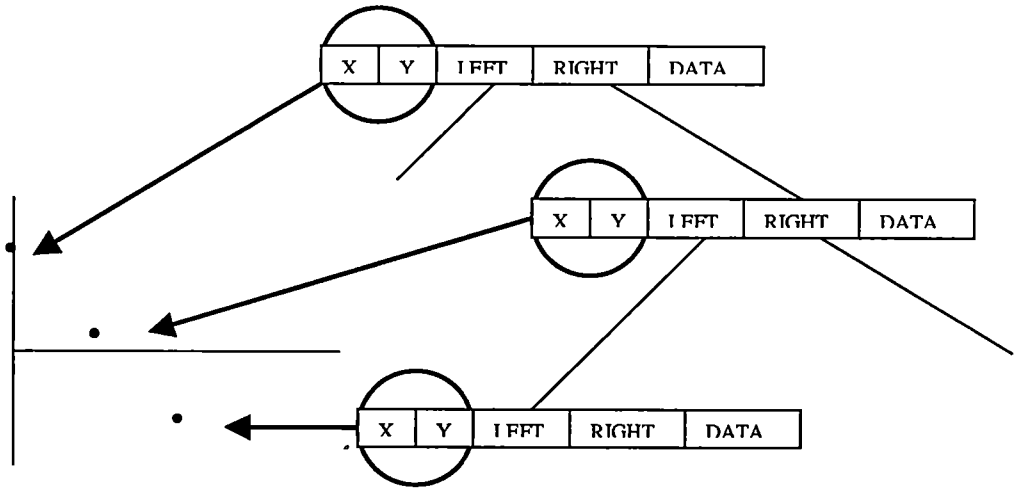


Figure 5 1. A generic 2-D tree structure. (Source: Worboys, 1995)

Figure 5.2 shows an example of building a 2-D tree for trips based on trip starting time and ending time The first, second, and third columns of each trip record represent trip id, beginning time, and ending time respectively. All trips are sequentially inserted to a 2-D tree Trip 1 is inserted to a null 2-D tree first, and becomes the root Since trip 1 is at even level (depth = 0), trip 2 is compared to trip 1 with respect to beginning time and becomes the right child of trip 1 (beginning time of trip 2 is greater than that of trip 1). When inserting trip 3, trip 3 is compared to trip 1 to decide on which side (left or right) trip 3 ought to go Since beginning time of trip3 is greater than that of trip 1, trip 3 will go to trip 1's right child Then trip 3 is compared to trip 2 with respect to ending time (trip 2 is at odd level) and becomes the left child of trip 2 (the ending time of trip 3 is less than that of trip 2) Similarly, trip 4 becomes the left child of trip 1, trip 5 becomes the right child of trip 2, and trip 6 becomes the left child of trip 3

Several kinds of queries are possible on a 2-D tree (Weiss, 1997). A range query searches nodes whose first key is between a specified set of values and whose second key is between another specified set of values An exact match searches for a node whose first key and second key are exactly equal to predefined values. A partial match query searches nodes based on one of the two keys (i e one of the two keys equal to a value) The exact match and partial match queries are both special cases of range query

Weiss argues that the running time of a range query depends on whether or not the tree is balanced or a partial match is requested, and how many items are found For a balanced tree, the range query takes $O(M + N^{1/2})$ time in the worst case. M and N represent the number of nodes found and total number of nodes, respectively

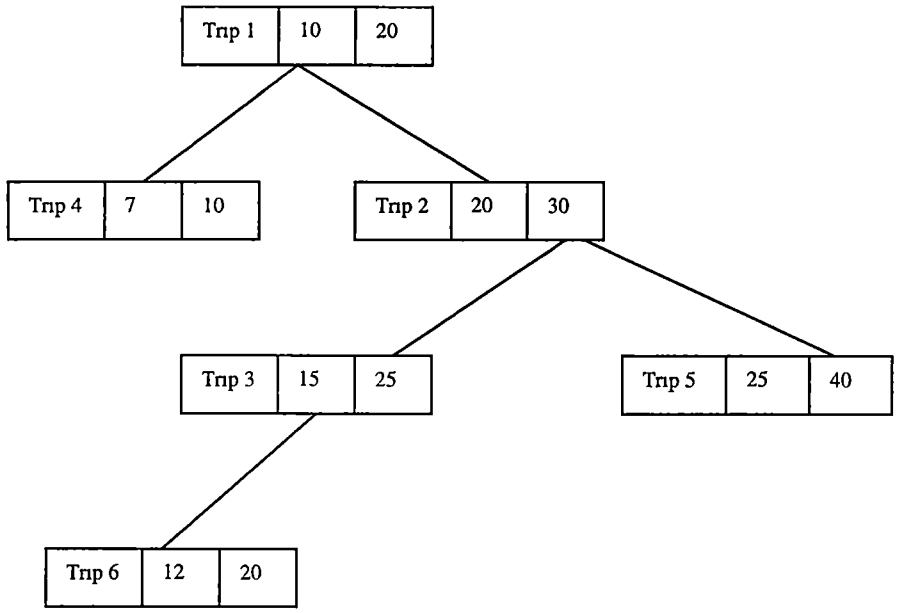


Figure 5 2 A sample trip 2-d tree based on starting time and ending time. The first column of each data is trip id, the second and the third columns are starting time and ending time Trips are inserted into the 2-d tree sequentially (i.e. by trip1, trip2, trip3 .)

A 2-D tree is a simple data structure for range query. In GIS, the two keys of 2-D tree can be seen as x and y coordinates of points Therefore, 2-D tree range query can be fit for points However, 2-D tree suffers from the problem of its structure depending on the order in which nodes are inserted (Worboys, 1995). Different node insertion orders result in different 2-D trees. In the worst case, the 2-D tree has the height of the total number of nodes.

Create an Array of 2-D Trees

From the trip route system, a text file containing route_id, arc#, starting time, and ending time is created This text file contains information about each trip, its related arcs, and the time window when this trip passes through those related arcs. Based on this text file, all trips (including time windows) passing over each specific street can be identified. Next, one 2-D tree recording the trips' ids and time attributes will be built for each specific street segment To speed up spatiotemporal query, streets' internal ids (Arc#) are stored in a sorted array, and each data item in this array has a pointer pointing to the 2-D tree structure (Figure 5 3).

C++ and Avenue together are used to build this array of 2-D trees. C++ is used to create some dynamic-link library (DLL) functions for an Avenue script to call. A DLL is an executable module that contains functions that other applications can use to perform

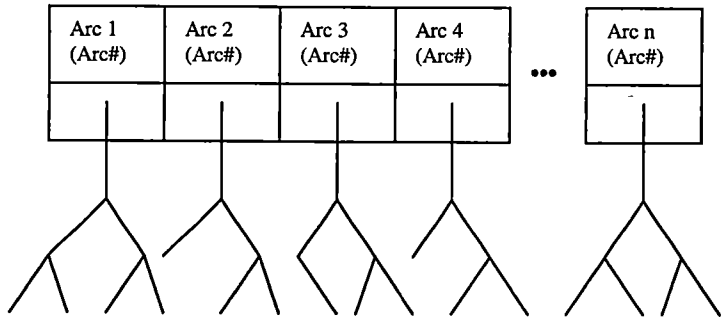


Figure 5.3 The structure of an array of 2-D trees. An ordered array of arc#. For each arc#, there is a pointer pointing to a 2-D tree based on starting time and ending time.

tasks DLLs are linked to an application at run-time. Three C++ files (twod.h, twod cpp, and BuildIndex.cpp) are used to create three DLL functions (two query functions, and one function for building an array of 2-D trees). One Avenue script is used to call the DLL function that creates an array of 2-D trees based on streets and stores them in memory. These C++ files are now described in detail.

1. **twod.h** the header file for creating a 2-D tree structure. This file defines a 2-D tree structure and functions or procedures in C++ Each 2-D tree has a root node A node is composed of one time array that contains two elements (i.e beginning time and ending time), one trip id, and two pointers pointing the left child and the right child Node insertion and range searching procedures for a 2-D tree are defined.

2. **twod.cpp:** the file used to build a 2-D tree, and perform a range query. Building a 2-D tree (inserting nodes into a 2-D tree) is composed of two procedures: recursively insert one item or trip (including trip id, beginning time, and ending time), and insert one node to a 2-D tree.

Recursively insert one item: recursively comparing this item to 2-D tree nodes to decide its position in the tree.

Input: an item array (item []) which includes three elements: beginning time, ending time, and trip id, a pre-known node called tnode, and one binary integer

called level to measure whether this pre-known node is at even or odd level. 0 and 1 represent even level and odd level respectively

Return: a 2-D tree node.

Procedure:

If tnode is null, then

Create a new tnode, and set the fields or values of this tnode by the item[] This tnode's pointers pointing to its left child and right child are set to null.

Else if item [level] less than time [level] of the tnode, then

Recursively insert this item to the tnode's left child, level will become !level (if previous level is 0, then this level will be 1, vice verse)

Else

Recursively insert this item to the tnode's right child, level will become !level.

Insert one node to a 2-D tree:

Input: a 2-D tree T, and an item array item [].

Procedure:

If T's root is null, then

Create a new node, set the fields and values of this node by item[], and set T's root as this node.

Else

Call the recursively inserting procedure. The current node is T's root, and level is 0 item[] will begin to compare and insert to T from T's root

Range query in a 2-D tree is also composed of two procedures RecPrintRange() and PrintRange() RecPrintRange() recursively query if nodes of the 2-D trees satisfying the range query condition, and output query results into a text file PrintRange() is used to query from the 2-D tree with a range of rectangle for beginning time and ending time

RecPrintRange():

Input: one array of two elements called low [] (low ends of beginning time and ending time), one array of two elements called high [] (high ends of beginning time and ending time), one 2-D tree node called tnode, and one binary integer level (0 or 1, represents even or odd level of tnode)

Output: a text file storing the query results

Procedure:

If tnode is not null, then

If tnode's time elements (beginning time and ending time) are between the range of low [] and high [], then output this node into a predefined text file

If low [level] less than time [level] of tnode, then

Recursively apply this procedure· RecPrintRange() to tnode's left child, set the level to !level.

If high [level] is greater or equal than time [level] of tnode, then

Recursively apply this procedure RecPrintRange() to tnode's right child, set the level to !level

PrintRange():

Input: one array of two elements called low [] (low ends of beginning time and ending time), one array of two elements called high [] (high ends of beginning time and ending time), and a 2-D tree T.

Procedure:

Apply RecPrintRange() procedure to T's root, and set level to 0.

2. **BuildIndex.cpp:** build a temporal index. There are three DLL functions: BuildIndex(), QueryTime1(), and QueryTime2().

BuildIndex(): create an array of 2-D tree structures for streets based on time elements (see Figure 5 3) A 2-D tree structure called arcStruc is composed of arcid (integer) and one 2-D tree.

Input: a text file containing trip_id, arc#, beginning time, and ending time

Return: a pointer to an array of 2-D tree structures

Procedure:

Read all values of arc# field from the text file and insert arc#s to a set (a set is composed of unique and ordered elements)

Allocate memory for an array of arcStruc (i e. create an array of 2-D tree structures space), assign the values of arcid fields by the set.

Read in each record from the text file including trip_id, arc#, beginning time, and ending time

Get the middle element from the array of 2-D tree structures' arcid field,

If this record's arc# equal to this element, then insert this record into the 2-D tree related to the element by calling 2-D tree insertion procedure.

Else if this record's arc# less than this element, then apply the same strategy to the left parts of the array of 2-D tree structures from this element

Else if this record's arc# is greater than this element, then apply the same strategy to the right parts of the array of 2-D tree structures from this element

Loop

Return this array of 2-D tree structures.

QueryTime1(): query the array of 2-D tree structures based on one arc (street) id and range of beginning time and ending time This procedure is used to query one arc

Input: one pointer (an array of 2-D tree structures), arc id, lower end of time range, and higher end of time range

Output: a text file storing the query result.

Procedure:

Get the middle element from the array of 2-D tree structures' arcid field,

If this arc id equal to this element, then call PrintRange() procedure.

Else if this arc id less than this element, then apply the same strategy to the left part of the array of 2-D tree structures from this element.

Else if this arc id is greater than this element, then apply the same strategy to the right part of the array of 2-D tree structures from this element.

QueryTime2(): query the array of 2-D tree structures for a set of streets based on range of beginning time and ending time.

Input: one pointer (an array of 2-D tree structures), one input text file containing arc ids, lower end and higher end of time range.

Output: a text file storing the query result

Procedure:

Read in each arc id from the input text file

Get the middle element from the array of 2-D tree structures' arcid field,

If this arc id equal to this element, then call PrintRange() procedure.

Else if this arc id less than this element, then apply the same strategy to the left part of the array of 2-D tree structures from this element.

Else if this arc id is greater than this element, then apply the same strategy to the right part of the array of 2-D tree structures from this element.

Loop

4. BuildTrees: an Avenue script used to create an array of 2-D tree structures from a text file, and store this structure as a pointer in memory in the form of a global variable The benefit of storing the address of an array of 2-D tree structures as a global variable is that a global variable remains in one ArcView application until this application is closed or until the ClearGlobals request executes Therefore, different spatiotemporal queries can be performed based on this structure

There are three steps to call a DLL function or procedure in Avenue. First, create a DLL object, then create a DLLProc corresponding to a procedure in the DLL. Finally, execute the call request on the DLLProc (ArcView V. 3.2 Online Help).

Input: a text file from the SEC table of a trip route system

Return: a pointer to an array of 2-D tree structures

Procedure:

Clear all global variables.

Create a DLL object from BuildIndexDLL.dll file (a DLL file containing the three DLL functions)

Make a DLLProc from the procedure of BuildIndex() in the DLL

Call this DLLProc by using the input text file and store the calling result as a global variable

Make a DLLProc from the procedure of QueryTime1() in the DLL and store it as a global variable.

Make a DLLProc from the procedure of QueryTime2() in the DLL and store it as a global variable

BuildTrees script is executed by clicking the Button C in an ArcView View GUI (graphic user interface, see Figure 6.1) The sample text file exported from SEC table of the trip path system is called secpaths.txt. When BuildTrees script is executed, the BuildIndex() function is called to create an array of 2-D tree structures based on the

secpaths txt. This array of 2-D tree structures is stored in memory for a spatiotemporal query script to call In the meantime, the two temporal query functions (QueryTime1() and QueryTime2()) are also stored as two global variables for the spatiotemporal query script to call. Next chapter will explain this spatiotemporal query script and the implementation of the spatiotemporal query.

This chapter has discussed the 2-D tree and an array of 2-D tree structures based on starting and ending times for trips when traversing each street The C++ program that includes three DLL functions for building temporal index and temporal queries are described. Finally, the temporal index (an array of 2-D tree structures) is created for the sample data set

Chapter 6

Spatiotemporal Query and Its Implementation

After an array of 2-D tree structures is created, an Avenue script is used to perform the spatiotemporal query. In this chapter, the spatiotemporal query Avenue script is described first, then some query examples are implemented

Spatiotemporal Query Code Description

One Avenue script called stquery.tool is used to perform the spatiotemporal query (i e which trips pass through a set of streets within one time window). In this script, an ArcView spatial query function is applied on the trip route system's network coverage to get Arc#s (i e street internal ids), then temporal query functions (DLLs created in C++) are used to search the array of 2-D tree structures to identify those trips that meet the time constraints Finally the query results will be shown on the map and in a report box. The following shows the contents of this script

There must be a trip route system and an array of 2-D tree structures of trips based on time ranges for the streets for this procedure to be run.

Input: Users select streets, and a time window.

Output: a report window containing query results (i e. selected trips, and their beginning time and ending time pass through those specific streets), and the query results are shown on the map (trip route system)

Procedure:

Use aView.ReturnUserRect command to get a rectangle from the user on the map

(View) in order to select street(s) from the network coverage

If this rectangle is null, then

Use aView GetDisplay ReturnUserPoint command to get the location of the mouse on the map display in order to select street(s) from the network coverage

Use anFTheme SelectByPoint command to select features (streets) of the anFTheme (the network coverage) at the mouse location

Else

Use anFTheme SelectByRect command to select features (streets) of the anFTheme (the network coverage) within the rectangle area

If the number of selected street(s) is greater than or equal to 1, then

Get the time range: lower end and higher end, and convert them into seconds.

If the number of selected street(s) equals to 1, then

Get the selected street's internal id.

Call the DLLProc that contains QueryTime1() function to perform temporal query on the array of 2-D tree structures based on the street internal id and time range, and store the result trips in a text file.

Read in the text file

For each record (trip) of the text file

Use aFTab Query command to query the trip route attribute table by the trip_id of this record

Loop

Display the text file as a message box report

Close the text file

Else if the number of selected streets is greater than 1, then

Add those street internal ids into a list

Create a text file

Write those street internal ids from the list into this text file.

Call the DLLProc that contains QueryTime2() function to perform temporal query on the array of 2-D tree structures based on the text file and time range, and store the results in an output text file

Read in the output text file

For each record (trip) of the output text file

Use aFTab Query command to query the trip route attribute table by the trip_id of this record

Loop

Display the output text file as a message box report.

Close the output text file

Implementation and Examples

The spatiotemporal query function is performed in an ArcView application. The trip route system is loaded into a View (Figure 6.1). A geocoded theme consisting of trip origin ends and last destination ends for each individual on each day is created using the methods described in previous chapters. The Tgr47093lka theme is the Knox street network coverage Paths theme is the trip route system

One button and one tool have been added to ArcView for performing the spatiotemporal query. The button containing icon C is used to create an array of 2-D tree structures of trips based on starting time and ending time for streets. In an ArcView application, this button should be clicked only once, since the array of 2-D tree structures are stored in memory as long as the ArcView application exists. Figure 6.2 shows how to create an array of 2-D tree structures by loading the BuildIndexDLL dll file. The tool containing icon Q is used to perform the spatiotemporal query. After the array of 2-D tree structures is created, users can use the tool to repeatedly perform spatiotemporal queries In the sample project, the Tgr47093lka theme should be active. Users select street(s) by drawing a rectangle box or clicking one point on the view Then a time window message box appears, and users can specify a time range during which trips pass through the specific street(s) (Figure 6.3, 6.6) The low and high ends of a time range are both in the format of HHMM (concatenation of hours and minutes) Finally all selected trips will be shown on the map and in a message box report.

Figure 6 3 to Figure 6 8 show two examples of the spatiotemporal query on one street and a set of streets respectively.

Example one: during 8 00 am – 9:00 am, which trips pass through the specific street? After keying the spatiotemporal tool down, the user selects the specific street by drawing a rectangle box or clicking on that street. The user then inputs the time range in the message box (Figure 6.3). Finally, the trip 402117011 (Trip_id) is selected and shown in a message box report (Figure 6.4) and on the map (Figure 6 5)

Example two: during 17:00 – 19·00, which trips pass through a set of streets? If the spatiotemporal tool is down, the user can select a set of streets by drawing a rectangle

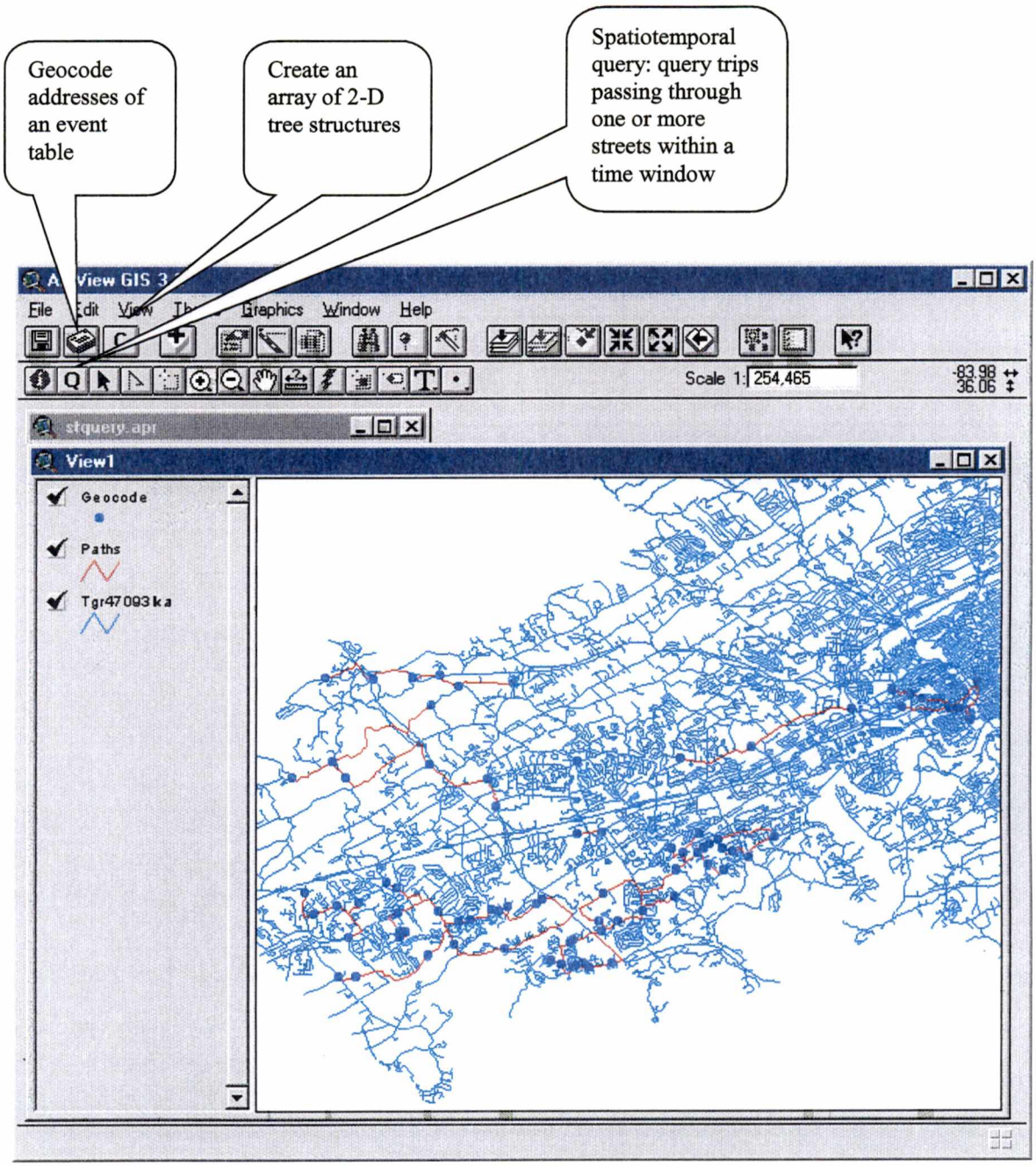


Figure 6.1. The user interface of a spatiotemporal query function on a trip route system.

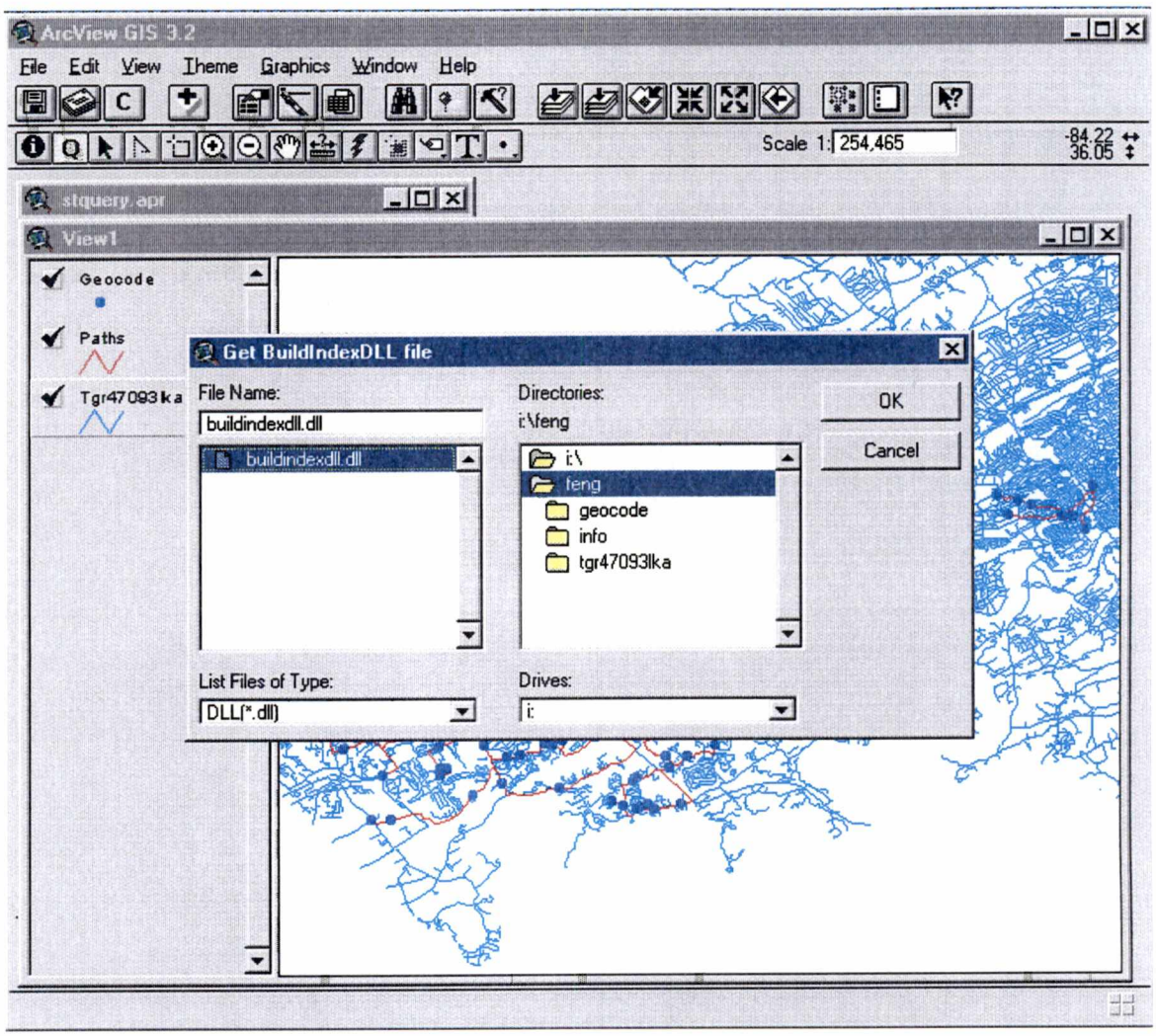


Figure 6.2. Create an array of 2-D tree structures by loading a DLL file.

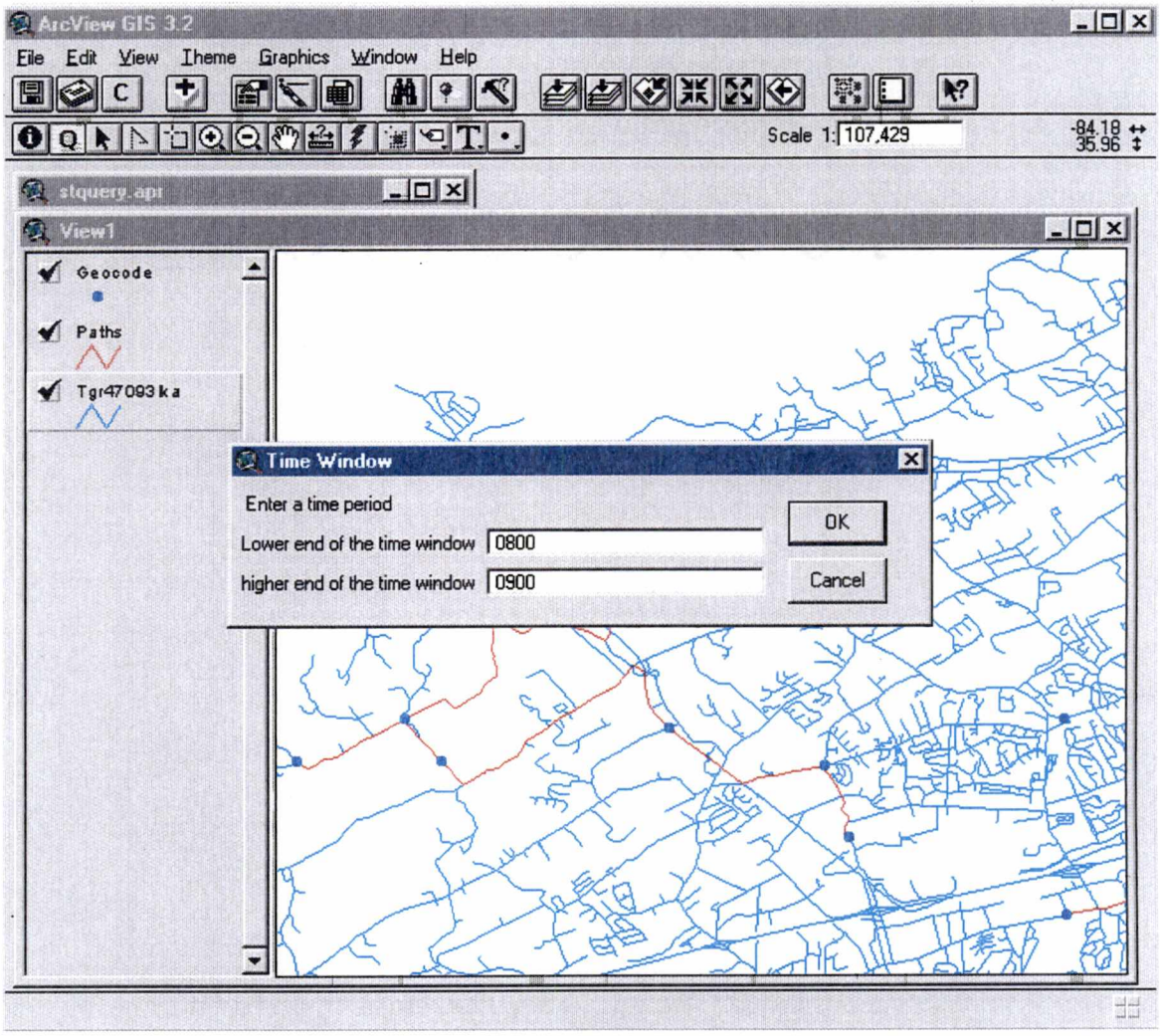


Figure 6.3. the time window (8:00 am – 9:00 am) of the spatiotemporal query on one street (example one).

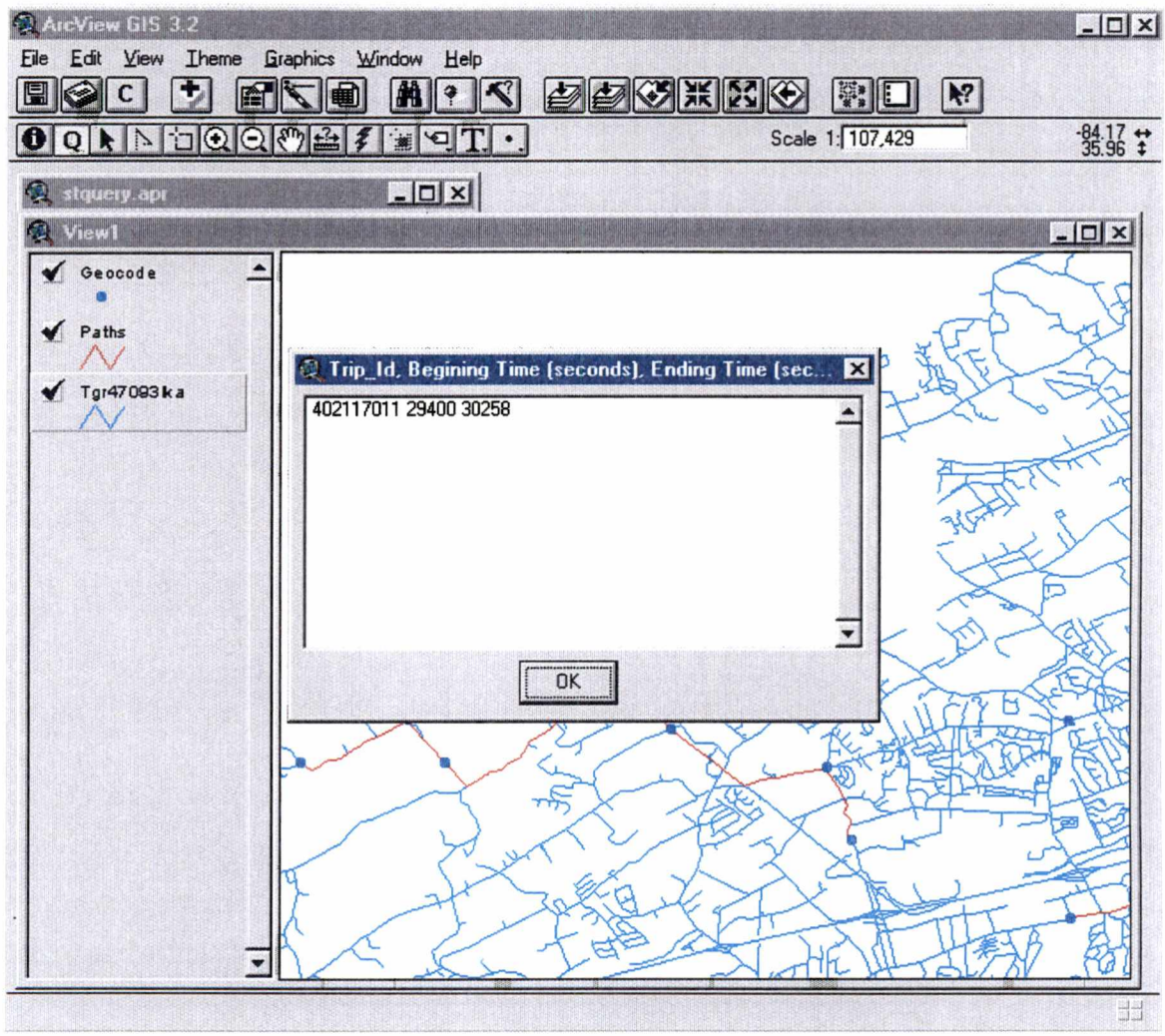


Figure 6.4. The message box report showing the spatiotemporal query result on one street (example one).

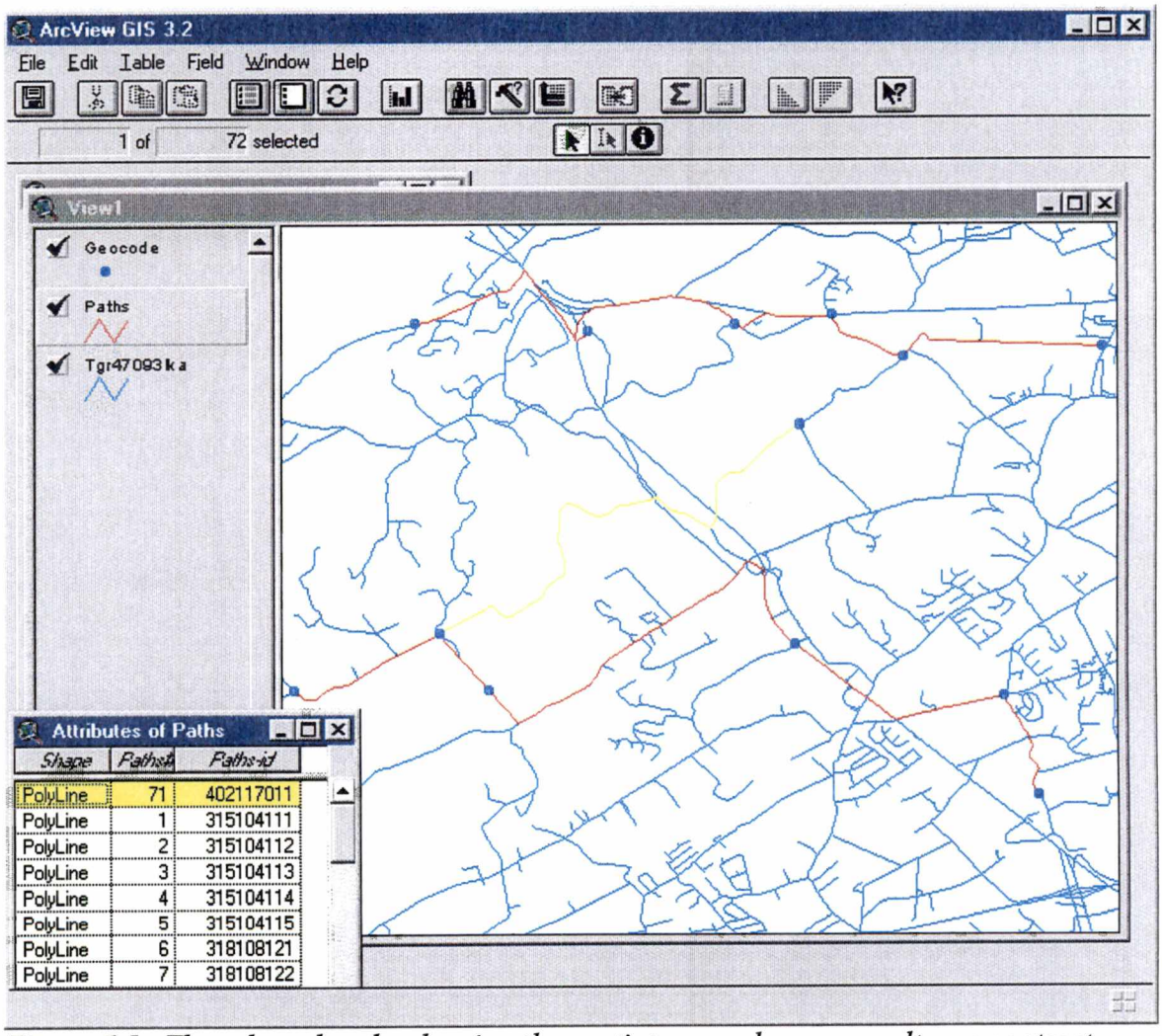


Figure 6.5. The selected paths showing the spatiotemporal query result on one street (example one).

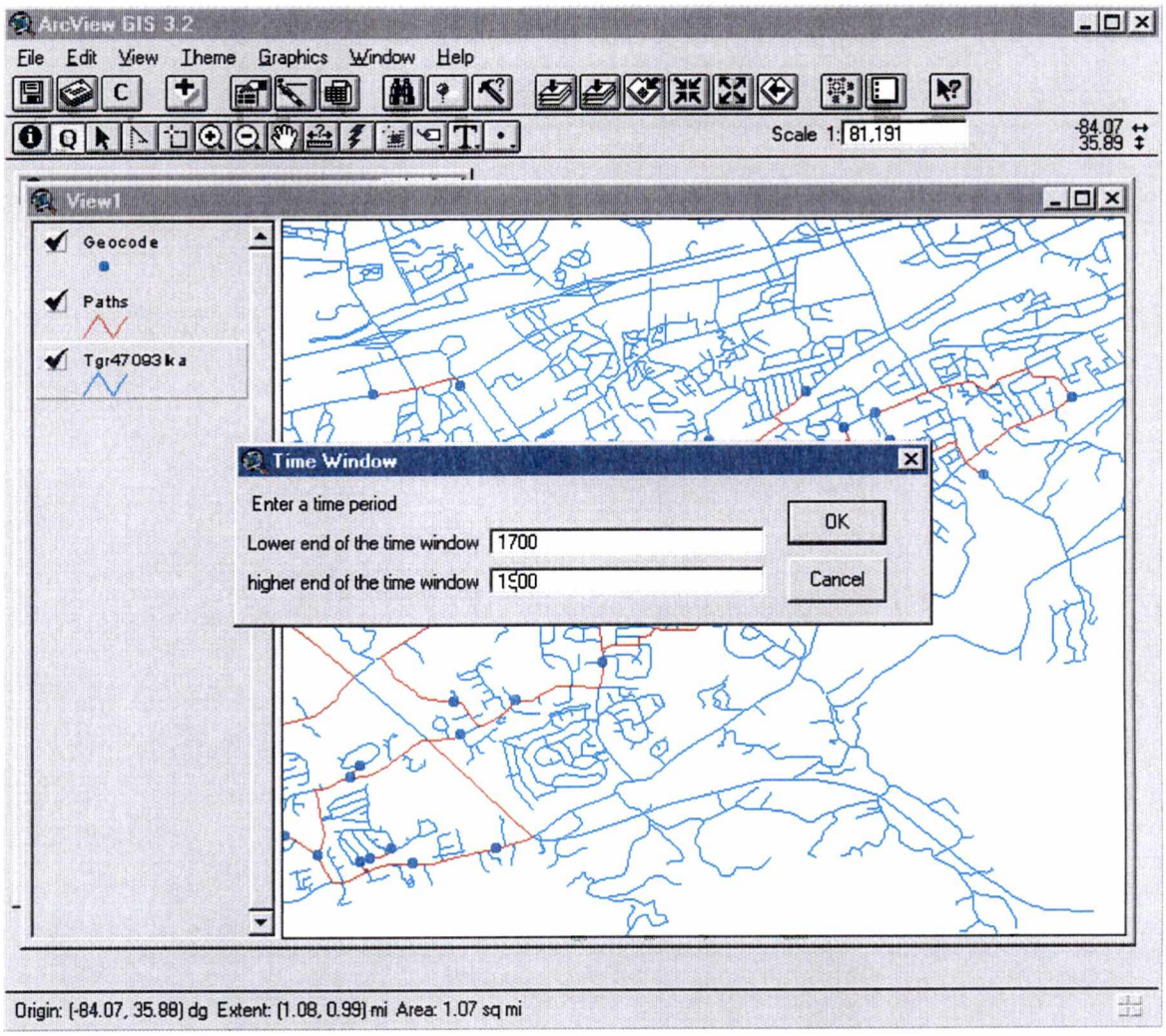


Figure 6.6. The time window (5:00 pm – 7:00 pm) of the spatiotemporal query on a set of streets (example two).

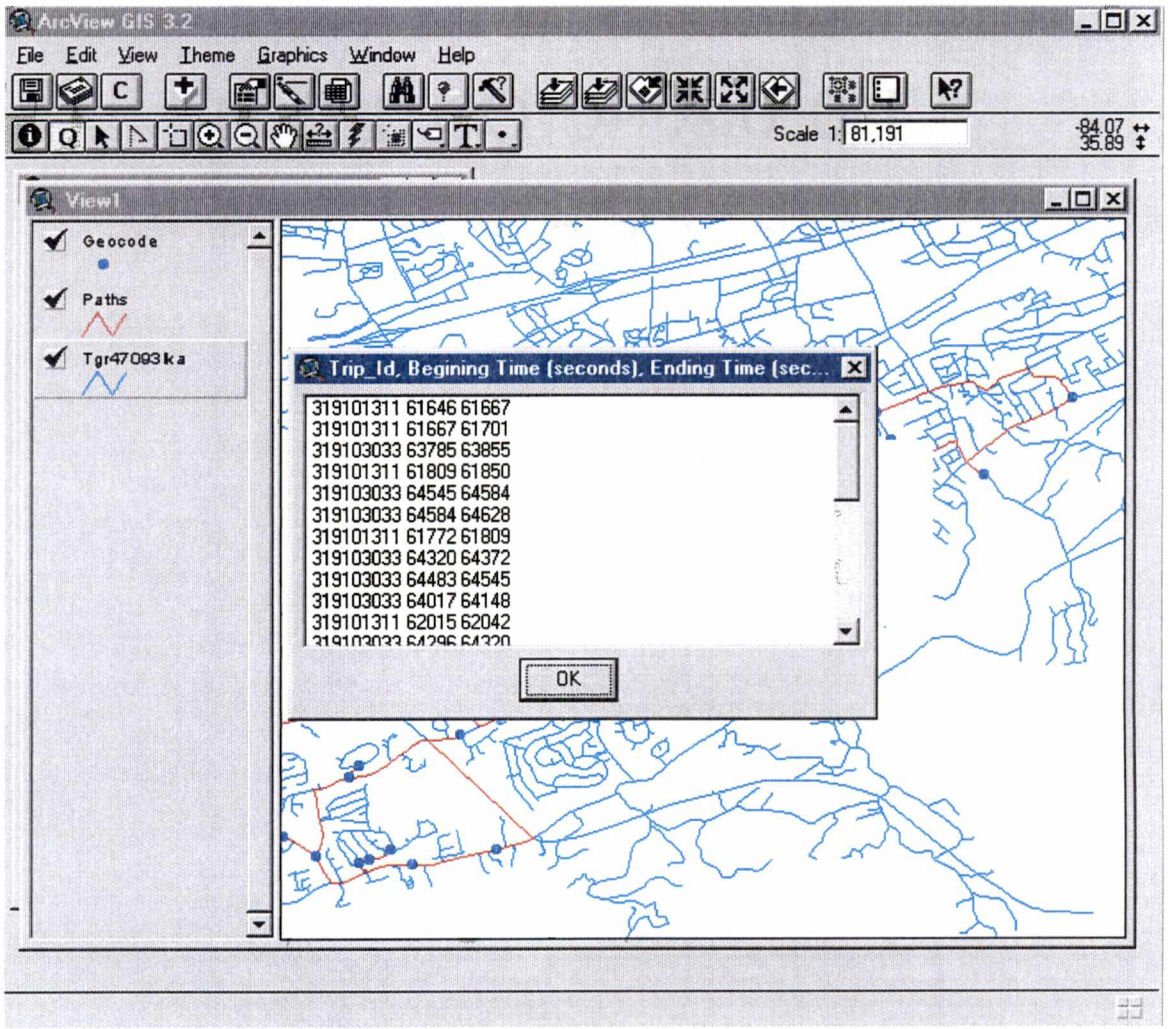


Figure 6.7. The message box report showing the spatiotemporal query result on a set of streets (example two).

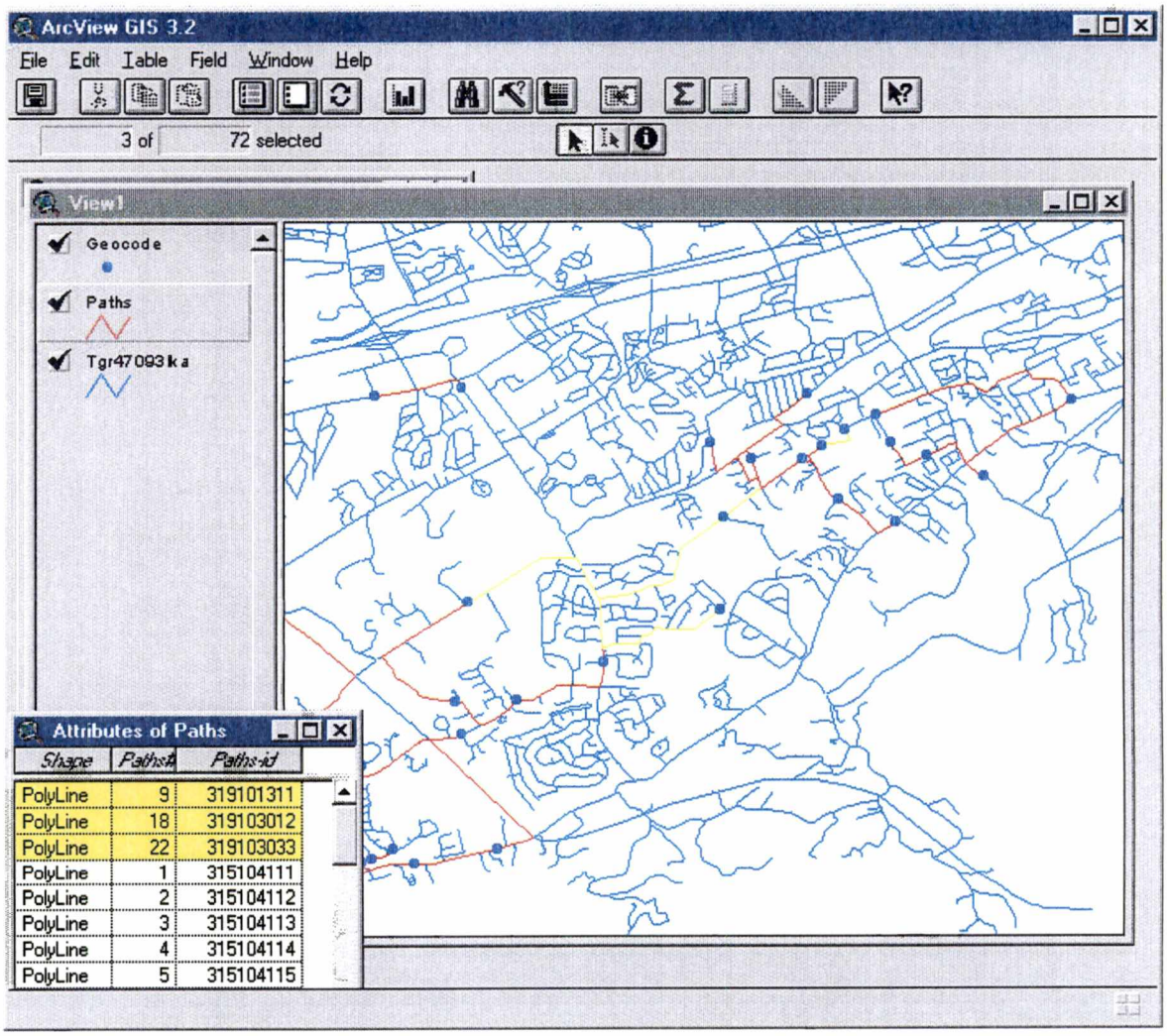


Figure 6.8. The selected paths showing the spatiotemporal query result on a set of streets (example two).

box, then input the time range into the time window (Figure 6 6) Finally the trip 319101311, 319103012, and 319103033 (Trip_id) are selected and shown in a message box report (Figure 6 7) and on the map (Figure 6.8)

Chapter 7

Conclusions and Future Research

In this study, spatiotemporal index on the trip data is built by combining ArcView/Avenue, ArcInfo/AML, and C++. This spatiotemporal index is used to answer the question about which trips pass through one or more specific streets during a time period All the trip origin ends and those last destination ends for each individual on each day are geocoded using Avenue scripts. The trip shortest path route system is created based on the geocoded locations, and ArcInfo dynamic segmentation and network analysis functions. An array of 2-D tree structures based on each trip's beginning and ending times and each street segment traversed is then created in C++ and Avenue This array of 2-D tree structures is stored in memory. Finally, the spatiotemporal query function is performed by examining an array of 2-D tree structures for a given time window using Avenue and C++

This study explores the spatiotemporal query problems for trip data. The spatiotemporal index developed is used to solve the spatiotemporal query problem for large trip data sets. If the size of a trip data set is small, there will be no need to develop such an index. Even though this study can only solve one kind of spatiotemporal query problem, it shows that efficient data structures can be created to solve complex spatiotemporal query problems for large data sets. An efficient trip data model (i e. how to represent disaggregate transportation data in a GIS environment) is the foundation for spatiotemporal queries and other data manipulation and analysis.

However, some shortcomings can be identified in this research. First, trips are assumed to take shortest paths. In fact, travel behavior is complicated. Thus, trips do not always take the shortest paths. Second, the spatiotemporal index can only solve one type of spatiotemporal problem, while there are several kinds of spatiotemporal query problems for disaggregate transportation data. The spatiotemporal index developed in this thesis can only solve the problem about which trips pass through one or more street(s) during a time window It cannot solve a similar problem about which trips pass through point locations or partial streets during a time window. Third, this spatiotemporal index is in fact the combination of spatial index and temporal index. Since a 2-D tree structure is only fit for point or non-dimensional search (Worboy, 1995), this structure cannot be used to perform the spatiotemporal query for trip data itself. The spatial query function in ArcView is used first to get selected streets, then 2-D tree structures are used to query trips passing through these streets during a time window. In fact, the array of 2-D tree structures is only a temporal index, since it is built based on beginning time and ending time. Finally, this spatiotemporal index can only be used to query historical trip data Assuming trip information and streets do not change, it is a static data structure If trips or streets change, the array of 2-D tree structures will have to be rebuilt If the data set is very large, this can be quite time consuming.

In future studies on spatiotemporal queries for disaggregate transportation data, there are at least three aspects to be considered.

Data collection: traditionally, disaggregate transportation data or trip data are collected by travel surveys, which usually record trip origin and destination addresses, and neglect other intermediate locations. Since GPS techniques can identify the location

of moving objects, they should be used in travel surveys. Trip intermediate points will be recorded by GPS; therefore, trip paths can be clearly identified The inexactness of the shortest path as a trip path will be overcome. In addition, after the spatiotemporal index is created, a large trip data set should be used to implement the spatiotemporal query.

Data model: A trip data model is the foundation of data retrieval, manipulation, and analysis. In this study, trip spatial information (trip path) is represented by dynamic segmentation, while trip temporal information is considered as an attribute of trip spatial information Shaw (2000) suggests a new trip data model combining trip spatial and temporal information together using the dynamic segmentation approach. Chapter 2 introduces this data model. Since trip spatial and temporal data are integrated parts of this data model, spatiotemporal manipulation, retrieval, and analysis can be easily handled. In the future, this approach can be adopted as a base to build spatiotemporal indexes

Index structure: If the 2-D tree structures are used to build a spatiotemporal index, these trees should be balanced first to speed up query. On the other hand, 2-D trees should be saved in a disk file Since 2-D tree is a static structure, it is difficult to handle dynamic trip data (i.e trips or streets change) Robinson (1981) proposed KDB-tree (combination of the KD-tree structure with the B-tree idea) to handle dynamic data. The KDB-tree structure might be a replacement of the 2-D (or K-D) tree structure. In addition, trip paths are one-dimensional linear objects, some multi-dimensional spatial data structures such as R tree and R+ trees should be explored in spatiotemporal queries for trip data.

REFERENCES

References

Ahn, I and Snodgrass, R. (1988) Partitioned Storage for Temporal Databases, *Information Systems,* 13(4), pp. 369-391.

Arc/Info Online Help Documents, Dynamic Segmentation.

Arc/Info Online Help Documents, Network Analyst.

ArcView Online Help Documents, Geocoding.

Bentley, J L (1975) Multidimensional Binary Search Trees Used for Associate Searching, *Comm. ACM,* 18, pp 509-517

Ben-Zvi, J (1982). The Time Relational Model, Ph.D. Dissertation, Computer Science Department, UCLA.

Chrisman, N R (1983). The Role of Quality Information in the Long-term Functioning of a Geographic Information System, in *Proceedings of Auto Carto* 6, Vol. 2, (Ottawa: Steering Committee of the Sixth International Symposium on Automated Cartography), pp. 303-321

Clifford, J and Warren, D S (1983) Formal Semantics for Time in Databases, *ACM Transactions on Database Systems,* 8 (2), pp 214-254.

Clifford, J , and Tansel, AU (1985) On an Algebra for Historical Relational Databases, in *Proceedings of the SIGMOD'85 Conference* (New York. ACM), pp 247-265.

Dangermond, J (1984). A Classification of Software Components Commonly Used in Geographic Information Systems, in *Proceedings of the U.S./Australia Workshop on Design and Implementation of Computer-Based Geographic Information Systems,* (Amherst: IGU Commission on Geographical Data Sensing and Processing), pp. 70-91

Date, C J. (1995) *An Introduction to Database Systems,* 6th edition (Reading: Addison-Wesley Publishing Company).

Dictionary of Computing, (1996). 4th edition (New York: Oxford University Press)

Dueker, K.J and Butler, J A. (1997). GIS-T Enterprise Data Model with Suggested Implementation Choices, Center for Urban Studies, Portland State University.

Dueker, K. J (1999) GIS-T Data Sharing Issues, Draft Discussion Paper 99-02, Center for Urban Studies, Portland State University.

Elmasri, R et al (1990). The Time Index – An Access Structure for Temporal Data, in *Proceedings of the Conference on Very Large Databases*. Brisbane, Australia.

Elmasri, R. et al (1991). Efficient Implementation Techniques for the Time Index, in *Proceedings of the Seventh International Conference on Data Engineering.*

Elmasri, R , et al (1992) Partitioning of Time Index for Optical Disks, in *Proceedings of the International Conference on Data Engineering*. Golshani, F. (Ed), IEEE, Phoenix, AZ, pp 574-583

Gadia, S K. (1986) Toward a Multihomogeneous Model for a Temporal Database, in *Proceedings of the International Conference on Data Engineering*, (New York IEEE), pp. 390-397.

Goodchild, M F (2000). GIS and Transportation Status and Challenges, *GeoInformatica*, forthcoming

Goodwin P (1983) Some Problems in Activity Approaches to Travel Demand, In Carpenter, S. and Stopher P (Eds) *Recent Advances in Travel Demand Analysis*, Aldershot, UK. Gower Publishing, pp. 470-474.

Gunadhi, H and Segev, A (1991). Efficient Indexing Methods for Temporal Relations, *IEEE Transactions on Knowledge and Data Engineering*

Hagerstrand T. (1970). What about People in Regional Science? *Papers of Regional Science Association*, 24, pp. 7-21.

Japan Ministry of Construction, Comprehensive Plan for ITS in Japan. www its go jp/ITS/5Ministries/index html

Japan Ministry of Construction, System Architecture. www its go jp/ITS/index/indexSA html

Kolovson, D. and Stonebraker, M (1989). Indexing Techniques for Historical Databases, in *Proceedings of the Fifth International Conference on Data Engineering*. Los Angeles, CA, pp. 127-137

Kolovson, C. and Stonebraker, M (1990) S-Trees Database Indexing Techniques for Multi-Dimensional Interval Data, Technical Report UCB/ERL M90/35, University of California.

Kolovson, C.P. (1990). Indexing Techniques for Multi-Dimensional Spatial Data and Historical Data in Database Management Systems, Ph.D. Dissertation, University of California, Berkeley.

Langran, G and Chrisman, N R (1988) A Framework for Temporal Geographic Information, *Cartographica*, 25 (3), pp. 1-14.

Langran, G (1992). *Time in Geographic Information Systems* (London: Taylor & Francis), pp. 37-44, 55-67, 95-157

Leung, T.Y. and Muntz, R. (1992). Generalized Data Stream Indexing and Temporal Query Processing, in *Second International Workshop on Research Issues in Data Engineering: Transaction and Query Processing.*

Lomet, D. and Salzberg, B. (1990) The Performance of a Multiversion Access Method, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Atlantic City, pp. 353-363.

Lum, V. et al (1984) Designing DBMS Support for the Temporal Dimension, in *Proceedings of ACM SIGMOD International Conference on Management of Data* Yormark, B (Ed.), Association for Computing Machinery, Boston, MA, pp 115-130.

Miller, H.J. (1991). Modeling Accessibility Using Space-time Prism Concepts Within Geographic Information Systems, *International Journal of Geographical Information Systems*, 5, 287-301.

Miller, H.J. and Shaw, S.L. *Geographic Information Systems for Transportation – Principles and Applications.* forthcoming

Moreira, J. et al (1999). Representation and Manipulation of Moving Points: An Extended Data Model for Location Estimation, *Cartography and Geographic Information Science*, 26 (2), pp 109-123.

Pas, E (1990). Is Travel Demand Analysis and Modeling in the Doldrums? In Jones, P. (Ed.) *Developments in Dynamic and Activity-based Approaches to Travel Analysis*, Aldershot, UK. Gower Publishing, pp. 3-27.

Peuquet, D. (1994). It's About Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems, *Annals of the Association of American Geographers*, 84 (3), pp. 441- 461

Peuquet, D and Niu, D. (1995). An Event-based Spatiotemporal Data Model (ESTDM) for Temporal Analysis of Geographical Data, *International Journal of Geographical information systems*, 9 (1), pp.7-24.

Peuquet, D. and Qian, L.J. (1996) An Integrated Database Model for Spatiotemporal GIS, *7th International Symposium on Spatial Data Handling*, pp 21-31.

Peuquet, D. (1999). Time in GIS and Geographical Databases, *Geographical Information Systems*, edited by Longley, P. et al. (New York: John Wiley & Sons, Inc.), pp. 91-103.

Qian, L.J. and Peuquet, D. (1998). Design of a Visual Query Language for GIS, *The 8th International Symposium on Spatial Data Handling*, Vancouver

Quiroga, C et al (1998) A GIS-GPS (Dynamic Segmentation) Methodology for Conducting Travel Time Studies, http //www rsip lsu edu/projects/ttg/ttg_main html

Robinson, J.T. (1981) The K-D-B-Tree A Search Structure for Large Multidimensional Dynamic Indexes, *Association for Computing Machinery SIGMOD*, 10, pp. 10-18.

Shaw, S.L. (1999). Activity-Based Travel Demand Modeling Using Geographic Information Systems, Department of Geography, University of Tennessee, Knoxville. Report of UTK Faculty Research Grant.

Shaw, S.L., and Wang, D.M. (2000) Handling Disaggregate Spatiotemporal Travel Data with GIS, *GeoInformatica*, 4 (2), pp 161-178.

Shaw, S L., (2000). Moving Toward Spatiotemporal GIS for Transportation Applications, Proceedings of ESRI User Conference, http //www esri com

Snodgrass, R., and Ahn, I (1985). A Taxonomy of Time in Databases, in *Proceedings of the SIGMOD' 85 Conference*, (New York: ACM), pp 236-245.

Snodgrass, R T. (1992) Temporal Databases, in A. U Frank, I Campari and U. Formentini (eds.) *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, Berlin: Springer-Verlag, pp. 22-64.

US Department of the Interior, US Geological Survey, National Mapping Division (1998). http //edcwww2 cr usgs gov/umap/umap html

Vonderohe, A et al (1995). On the Results of a Workshop on Generic Data Model for Linear Referencing Systems, *GIS-T 95 Proceedings*

Weiss, M A. (1997). *Data Structures and Algorithms Analysis in C.* 2nd Edition (Addison-Wesley), pp. 485-488.

Worboys, M F. (1992). A Model for Spatio-temporal Information, in Corwin, E. and Cowen, D (Eds) *Proceedings of the 5th International Symposium on Spatial Data Handling*, Columbus, OH: International Geographical Union, pp 606-611

Worboys, M.F. (1994). A Unified Model of Spatial and Temporal Information, *Computer Journal*, 37 (1), pp. 26-34

Worboys, M F. (1995). *GIS A Computing Perspective* (London Taylor & Francis), pp. 264-267

Yuan, M. (1994). Wildfire Conceptual Modeling for Building GIS Space-time Models, *Proceedings GIS/LIS'94*, pp 860-869

Yuan, M (1999). Temporal GIS and Spatio-Temporal Modeling
http://ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/yuan_may/may.html

Appendix

```
/*****************************************/
/* File name    twod.h                   */
/* Description  this is the header file for 2-d tree    */
/*****************************************/

#ifndef _TWOD_H_
#define _TWOD_H_

typedef struct twodnode
{
        long int time[2],
        long int tripid,
        struct twodnode *left,
        struct twodnode *right,

} TwoDNode,

typedef struct {
 TwoDNode *root,
} TwoDTree,


/* create a new empty 2-d tree */

TwoDTree *new_TwoDTree(),

/* free the 2-d tree */

void Free_TwoDTree(TwoDTree *),

/* insert a array of two long int items into the 2-d tree */

TwoDNode *TwoDTree_Insert_Node(TwoDTree *, long int []),

/* print the 2-d tree from the node in order */

void  TwoDTree_Print_Tree(TwoDNode *),

/* print the 2-d tree from the node preorder */

print_tree(TwoDNode *),

/* search a particular node in the 2-d tree */

TwoDNode *TwoDTree_Search(TwoDTree *, long int []),

/* search nodes from the 2-d tree in a range */

PrintRange(long int [], long int [], TwoDTree *),

#endif
```

```
/*****************************************************************/
/* File name  twod cpp                                           */
/* Description  this is a file for creating and querying basic 2-d tree structure    */
/*****************************************************************/

#include <stdio h>
#include <stdlib h>
#include <malloc h>
#include "twod h"

/* create a new empty 2-d tree */

TwoDTree *new_TwoDTree()
{
 TwoDTree *t,
 t = (TwoDTree *) malloc(sizeof(TwoDTree)),
 t->root = NULL,
 return t,
}

extern FILE *Ofp,

/*
 recursive function used for inserting node into 2-d tree

*/

static TwoDNode *RecursiveInsert(long int item[], TwoDNode *tnode, int level)
{
        if (tnode == NULL) {
                tnode = (TwoDNode *) malloc(sizeof(TwoDNode)),
                if (tnode == NULL) {printf("out of space\n"), exit(1),}
                tnode->tripid = item[2],
                tnode->time[0] = item[0],
                tnode->time[1] = item[1],
                tnode->left = NULL,
                tnode->right = NULL,
   }
        else if (item[level] < tnode->time[level])
                tnode->left = RecursiveInsert(item, tnode->left, !level),
        else
                tnode->right = RecursiveInsert(item, tnode->right, !level),
        return tnode,

}

/*
insert a node into the 2-d tree

*/

TwoDNode *TwoDTree_Insert_Node(TwoDTree *f, long int item[])
```

```
{
        TwoDNode *fd,
        if (f->root == NULL){
    fd = (TwoDNode *) malloc(sizeof(TwoDNode)),
    fd->time[0] = item[0],
         fd->time[1] = item[1],
         fd->tripid = item[2],
         fd->left = NULL,
         fd->right = NULL,
         f->root = fd,
         return fd,
        } else
  return RecursiveInsert(item, f->root, 0),
}


/*
recursive function used for finding node
*/

static TwoDNode *RecursiveSearch(long int item[], TwoDNode *tnode, int level)
{

        if (tnode == NULL) {printf("not found\n"), exit(1),}
        else if ((item[0] == tnode->time[0]) && (item[1] == tnode->time[1])
                && (item[2] == tnode->tripid))
     return tnode,
        else if (item[level] < tnode->time[level])
     return  RecursiveSearch(item, tnode->left, !level),
        else if (item[level] >= tnode->time[level])
           return RecursiveSearch(item, tnode->right, !level),
        return (NULL),
}


/*
searching node in the 2-d tree
*/

TwoDNode *TwoDTree_Search(TwoDTree *t, long int item[])
{
 return RecursiveSearch(item, t->root, 0),

}


/* in order  print the 2-d tree */

void
 TwoDTree_Print_Tree(TwoDNode *header)
{
 if ( header == NULL ) return,
 TwoDTree_Print_Tree(header->left),
 fprintf(Ofp, " %ld %ld %ld\n", header->time[0], header->time[1], header->tripid),
 TwoDTree_Print_Tree(header->right),
```

```
}


/* pre order  print the 2-d tree */

print_tree (TwoDNode * root)
{
        if (root != NULL)
        {
                fprintf (Ofp, "%ld %ld %ld\n", root->time[0], root->time[1], root->tripid),
                if (root->left != NULL)
                {
                        fprintf (Ofp, "left child\n"),
                        print_tree(root->left),
                }
                if (root->right != NULL)
                {
                        fprintf (Ofp, "right child\n"),
                        print_tree(root->right),
                }
        }
}



/* this is for recursively searching for the nodes based on a range */

static void RecPrintRange(long int low[], long int high[], TwoDNode *t, int level)
{
 if (t != NULL)
 {
  if (low[0] <= t->time[0] && t->time[0] <= high[0] && low[1] <= t->time[1] && t->time[1] <= high[1])
     fprintf(Ofp, "%ld %ld %ld\n", t->tripid, t->time[0], t->time[1]),
        if (low[level] < t->time[level])
                RecPrintRange(low, high, t->left, !level),
        if (high[level] >= t->time[level])
                RecPrintRange(low, high, t->right, !level),

 }
}


/* searching for the nodes from the 2-d tree based on  a range */

PrintRange(long int low[], long int high[], TwoDTree *t)
{

 RecPrintRange(low, high, t->root, 0),
}


/* recursively free TwoDNode */
```

```
static void Recursive_Free_TwoDTree(TwoDNode *fn)
{
 if (fn == NULL) return,
 if (fn->left != NULL) Recursive_Free_TwoDTree(fn->left),
 if (fn->right != NULL) Recursive_Free_TwoDTree(fn->right),
 free(fn),
 return,
}

/* free the 2-d tree */

void  Free_TwoDTree(TwoDTree *f)
{
 Recursive_Free_TwoDTree(f->root),
 free(f),
 return,
}
```

```
/*************************************************************/
/* Program name  BuildIndex cpp                                */
/* Description  Building a spatial temporal index using 2-d tree based on */
/*              starting time and ending time for each arc      */
/*************************************************************/
#include <stdio h>
#include <stdlib h>
#include <iostream>
#include <set>
#include <string>
#include <algorithm>
#include "twod h"
using namespace std ,
typedef set<long int> SET_INT,

// define a struct for each arc including arcid, and 2-d tree

typedef struct{
        long int arcid,
        TwoDTree *tt,
} arcStruc,

FILE    *Ofp,
long int size,

extern "C"
__declspec( dllexport )

/* Query the array of 2-d trees based on arcids, begining time, and ending time
  arcid, result will be written into ofname
*/

int QueryTime1(arcStruc ** pt, long int arcid,  long int bt, long int et, char * ofname)
{
                long int lo,
                long int hi,
                long int mi,
                long int low[2],
                long int high[2],
                low[0] = bt,
                low[1] = bt,
                high[0] = et,
                high[1] = et,

//      check if there are problems with the input and output files

        if ((Ofp = fopen (ofname, "w")) == NULL)
        {
                printf("erro in open output file\n"),
                exit(1),
        }
```

```
/* read in arcids from the input file and do binary search to find the query result,
   then write into output file
*/

        lo = 0,
        hi = size - 1,
        while(lo <= hi)
                {
                        mi = (lo + hi) / 2,
                        if (pt[mi]->arcid < arcid) lo = mi + 1,
                        else if (pt[mi]->arcid > arcid) hi = mi - 1,
                        else {
                                PrintRange(low, high, pt[mi]->tt),
                                break,
                        }
                }

        fclose(Ofp),
        return (1),
}

extern "C"
__declspec( dllexport )

/* int QueryTime(arcStruc ** npt, long int arcid, long int bt,long int et)
  Query the array of 2-d trees based on arcids, beginning time, and ending time
  arcids are in file fn, result will be written into outfn
*/

 int QueryTime2(arcStruc ** npt, char * fn, char * ofn, long int bt, long int et)
{
                long int lo,
                long int hi,
                long int mi,
                long int low[2],
                long int high[2],
                FILE *nfp,
                long int arcid,
                low[0] = bt,
                low[1] = bt,
                high[0] = et,
                high[1] = et,

//      check if there are problems with the input and output files

        if ((Ofp = fopen (ofn, "w")) == NULL)
        {
                printf("erro in open input file\n"),
                exit(1),
        }
        if ((nfp = fopen (fn, "r")) == NULL)
        {
                printf("erro in open output file\n"),
                exit(1),
        }
```

```
/* read in arcids from the input file and do binary search to find the query result,
   then write into output file
*/

   while (fscanf (nfp, "%ld", &arcid) != EOF)
          {
                        lo = 0,
                        hi = size - 1,
                        while(lo <= hi)
                                {
                                        mi = (lo + hi) / 2,
                                        if (npt[mi]->arcid < arcid) lo = mi + 1,
                                        else if (npt[mi]->arcid > arcid) hi = mi - 1,
                                        else {
                                                PrintRange(low, high, npt[mi]->tt),
                                                break,
                                        }
                                }

          }
          fclose(nfp),
   fclose (Ofp),
          return (0),
}


extern "C"
__declspec( dllexport )

/* build an array of arcStruc  For each arc, build a 2-d tree
   user should input section file name which includes necessary information
*/

arcStruc **
BuildIndex (char * infile )
{
   SET_INT                arcIDs,
          long int                arcID,
          long int                routID,
          long int                bt,
          long int                et,
          long int ar[3],
          arcStruc **pt,
          int lo, hi, mi, i,
          FILE     *ifp,

          // check if there is a problem with input file

          if ((ifp = fopen (infile, "r")) == NULL)
          {
                  printf("erro in input file"),
                  exit(1),
          }
```

```
        // read in input file, and insert arcids into set arcIDs

        while (fscanf(ifp,"%ld,%ld,%ld,%ld", &routID, &arcID, &bt, &et) != EOF)
        {
                arcIDs insert(arcID),
        }

        fclose(ifp),

        SET_INT  iterator it,
    i = 0,

        // malloc an array of arcStruc, and assign values

        pt = (arcStruc **) malloc (arcIDs size() * sizeof (arcStruc *)),
        for (it = arcIDs begin(), it != arcIDs end(), it++){
                pt[i] = (arcStruc *) malloc (sizeof (arcStruc)),
                pt[i]->arcid = *it,
                pt[i]->tt = new_TwoDTree(),
                i++,
        }

    size = arcIDs size(),
        arcIDs clear(),

        if ((ifp = fopen (infile, "r")) == NULL)
        {
                printf("erro in open input file\n"),
                exit(1),
        }

        // read in input file, do binary search, and insert nodes into 2-d trees

        while (fscanf(ifp, "%ld, %ld, %ld, %ld", &routID, &arcID, &bt, &et) != EOF)
        {
         ar[0] = bt,
         ar[1] = et,
         ar[2] = routID,
          lo = 0,
          hi = size - 1,
          while (lo <= hi)
          {
                  mi = (lo + hi) / 2,
                  if (pt[mi]->arcid < arcID) lo = mi + 1,
                  else if (pt[mi]->arcid > arcID) hi = mi - 1,
                  else {
                          (void *) TwoDTree_Insert_Node(pt[mi]->tt, ar),
                          break,
                  }
          }
        }
        return pt,
}
```

```
/* ************************************************************************************
/* File name  tripdyna aml
/* Description
/*   this file creates a shortest path route system  Assume users have a point coverage
/*   (which can be a geocoding result for disaggregate trip dat), and the trips are all
/*   sequential  The point coverage only has points for each starting address, in addition,
/*   ending addresses for each individual at each day
/*   In arc/info, in order to create shortest path route system, all stops have to be at
/*   nodes  To overcome this shortcoming, the point coverage is first used for splitting the
/*   network, creating pseudo nodes  Using these pseudo nodes, shortest path system can be
/*   created  Then using unsplit command to delete all the pseudo nodes, which doesn't affect
/*   the final route system, because all the route system will be updated
/*
/* input  arc coverage (network including nodes), point coverage (used as stops to
/*       create shortest paths based on each individual)
/*
/* output  a shortest paths route system and a text file
/*
************************************************************************************

/* check if the input arguments are correct

&args arc_cover point_cover
&if [null %arc_cover%] = true  or [null %point_cover%] = true  &then
 &do
  &type USAGE  final aml <arc_cover> <point_cover>
  &message &on
  &stop
 &end

/* create an info file to store nonuniId field values from the point file

&s non [LISTUNIQUE %point_cover% -POINT nonuniId nonuniid txt]
TABLES
&if [EXISTS nonuni dat -info] &then
  &do
    sel nonuni dat
    purge
    y
  &end
&else
  &do
    define nonuni dat
    nonuniid 10 10 C
    ~
  &end
sel nonuni dat
ADD FROM nonuniid txt
q

&if [show program] = ARC &then
 &do
```

```
 display 0
 ae
 graphics off
&end
coord key
weedtolerance 0 00001

/* create a stop file to store pseudo node id and route_id

create stps info
%arc_cover%-id, 4, 5, b
route_id, 4, 12, b
~
/* get the total number of this info file, and based on each record, to select
/* points in the point coverage to split arc coverage

EDIT nonuni dat INFO
&S NON [SHOW NUMBER TOTAL]
&s cnt 1
&DO CNT  = 1 &TO %NON%
SEL $RECNO = %CNT%
&S TE [SHOW INFO [SHOW SELECT 1] ITEM nonuniid]

ec %point_cover%
ef point
SEL NONUNIID = [QUOTE %TE%]
&s end [show number SELECTED]

/* for each individual, get the points and split the arc coverage

&s count 1
 &do COUNT  = 1 &to %end%
 &s x [extract 1 [show label [show select %count%] coordinate]]
 &s y [extract 2 [show label [show select %count%] coordinate]]
 &s rid [show point [show select %count%] item trip_id]
 &if %count% = 1 &then
  &s rid_pre = %rid%
 &if %count% <> 1 or %rid% <> 99999999 &then
  &do
   ec %arc_cover%
   ef arc
   sel
   %x%,%y%

   &if [show number selected] = 1 &then
    &do
     &s xy_node [show arc [show select 1] nodes]
     &s x_1 [extract 1 %xy_node%]
     &s y_1 [extract 2 %xy_node%]
     &s x_2 [extract 3 %xy_node%]
     &s y_2 [extract 4 %xy_node%]
     &s z1 = %x% - %x_1%
     &s z2 = %y% - %y_1%
     &s k1 = %x% - %x_2%
     &s k2 = %y% - %y_2%
```

```
    &s z = %z1% * %z1% + %z2% * %z2%
    &s k = %k1% * %k1% + %k2% * %k2%

    /* check if these points are at nodes, if so, then write down node ids,
    /* else, split

    &if [SQRT %z%] <= 0 00001 &then
      &s n1 = [show node [show arc [show select 1] fnode#] id]

    &else &if [SQRT %k%] <= 0 00001 &then
      &s n1 = [show node [show arc [show select 1] tnode#] id]
    &else
     &do
      split
      %x%,%y%
     &end

    /* after split, store each pseudo node's id, and add to a stop file

    &if [show number selected] = 2 &then
     &do
     &if [show arc [show select 1] fnode# ] = [show arc [show select 2] tnode#] &then
       &s n1 = [show node [show arc [show select 1] fnode#] id]
     &else
       &s n1 = [show node [show arc [show select 1] tnode#] id]
     &end

  /* add nodes or pseudo nodes into the stop file

  edit stps info
  &if %count% = 1 or %count% = %end% or %rid% = 99999999 &then
   &do
   add
   calc %arc_cover%-id = %n1%
   calc route_id = %rid_pre%
   &end
  &else &if %rid_pre% = 99999999 &then
   &do
   add
   calc %arc_cover%-id = %n1%
   calc route_id = %rid%
  &end
  &else
   &do
   add
   calc %arc_cover%-id = %n1%
   calc route_id = %rid_pre%
   add
   calc %arc_cover%-id = %n1%
   calc route_id = %rid%
   &end
  &end
&end
 &s rid_pre %rid%
 ec %point_cover%
```

```
 ef point
 sel nonunuid = [quote %te%]
 &s count %count% + 1
&end
EDIT nonuni.dat INFO
&S CNT %CNT% + 1
&END
 quit
 y
 y

/* create shortest path route system based on the stop file  Create shortest path
/* based on each route_id

 ap
 netcover %arc_cover% paths ini
 stops stps # route_id
  path stops
 quit
 tables
 select %arc_cover% ratpaths
 alter
 paths-id
 paths-id
 12
 ~
 ~
 ~
 q

 /* unsplit the network

 ae
 ec %arc_cover%
 ef arc
 sel all
 unsplit
 quit
 y
 y

/* build relations between section attribute table, route attribute table, and point pat

relate drop
$all
~
relate add
route
%arc_cover% ratpaths
info
routelink#
paths#
linear
rw
path
```

```
%point_cover% pat
info
paths-id
trip_id
linear
rw
~

/* add beginning time and ending time fields into the section attribute table

ae
ec %arc_cover%
ef route paths
&s tot [show number total]
ef section paths
additem begint 4 10 b
additem endingt 4 10 b
save

/* calculate beginning time and ending time for the section attribute table
/* based on interpolation

&s count 1
&do count  = 1 &to %tot%
sel routelink# = %count%
&s f [show section paths [show select 1] item f-meas]
&s t [show section paths [show select [show number selected]] item t-meas]
&s t %t% - %f%
calc begint = route//path//startt + ( route//path//endt - route//path//startt ) * ( f-meas - %f% ) * 1000 0 / (
1000 0 * %t% )
calc endingt = route//path//startt + ( route//path//endt - route//path//startt ) * ( t-meas - %f% ) * 1000 0 / (
1000 0 * %t% )
save
&s count = %count% + 1
&end
quit
y
y

/* output some selected fields from section attribute table to a ascii file

tables
select %arc_cover% secpaths
unload secpaths txt route//paths-id arclink# begint endingt
q
&return
```

```
'*********************************************************************
'File Name  trip geocode
'
'Description  geocode addresses for an event table based on a matchable
'            theme
'Input  a matchable theme (street network), an event table
'
'Output  a geocoding point shape file
'*********************************************************************

'The matchable theme is active, and the event table is a dbase file
'called tripdata dbf

theView = av GetActiveDoc
theThemes = theView GetACTiveThemes
if (theThemes = nil) then
  exit
end
theTheme = theView GetActiveThemes Get(0)

'load an event table, and make a copy of this table
'All editing will be conducted on this copy

workDir = av getproject getworkdir asstring
theFname = filedialog show("* dbf", "Table(* dbf)", "Load the tripdata table")
if (nil <> thefname) then
  thefile = (workDir + "\tripcopy dbf") asfilename
  file copy(thefname, thefile)
else
  exit
end
isOK = vtab canmake(thefile)
if (isOK not) then
  msgbox error("Invalid file", "")
  exit
end
addressVtab = vtab make(thefile, false, false)
attributestable = table make(addressVtab)
attributestable setname("Trip-Log table")

'Edit addressVtab  create two fields  trip_id and nonunuid
'trip_id is the concatenation of Date, Hh, Indv, and Trip_no, trip_id can
'uniquely identify each trip
'nonunuid is the concatenation of Date, Hh, and Indv, it's used to uniquely identify
'each indivicual on each day

if (addressVtab CanEdit) then
  addressVtab SetEditable(true)
end
startField = addressVTab FindField("Started_fr")
endField = addressVTab Findfield("traveled_t")
nonfield = addressvtab findfield("nonuniId")
```

```
tripfield = addressvtab findfield("trip_id")
if (nonfield <> nil) then
  addressvtab removefields({nonfield})
end
if (tripfield <> nil) then
  addressvtab removefields({tripfield})
end
nonuniId = Field Make("nonuniId", #FIELD_CHAR, 10, 0)
tripid = Field Make("trip_id", #FIELD_LONG, 12, 0)
addressVTab AddFields({tripid, nonuniId})
addressvtab Calculate("([Date] AsString Trim+[Hh] AsString Trim+[Indv] AsString Trim+[Trip_no] AsStr
ing Trim) asnumber", tripid)
addressvtab Calculate("[Date] AsString Trim+[Hh] AsString Trim+[Indv] AsString Trim", nonuniId)

'To geocode all starting addresses and some ending addresses for creating trip paths,
'for each individual on each day, appending one record on the bottom of the table, replacing
'value of Start_fr on this record with the value of Traveled_t from the last trip on each day

num = addressVTab getnumrecords
for each i in 0  (num - 1)
  if (addressvtab ReturnValueString(nonuniId, i) trim <> addressvtab ReturnValueString(nonuniId,
i+1) trim) then
    a = addressVTab ReturnValueString(endField, i)
    b = addressvtab returnvaluestring(nonuniId, i)
    newRec = addressVTab AddRecord
    addressVTab SetValueString(startField, newRec, a)
    addressVTab SetValueString(nonuniId, newRec, b)
  end
end

addressField = addressVTab FindField("Started_fr")
zipField = addressVTab FindField("Zip")

if ((theTheme IsMatchable) Not) then
  av Run("trip match", nil)
end

'Get the matchable feature source and double check that it is valid

aMatchSource  = theTheme GetMatchSource
if (aMatchSource = Nil) then
  MsgBox Error("Theme"++theTheme GetName++"is not matchable ","")
  exit
end


'Specify the output point shapefile that will be created from the
'matched addresses

'Specify the output point shapefile for the geocoding result

aGeoName = GeoName Make(aMatchSource, addressVTab, addressField, zipField)

fnOutFile = FileDialog Put(av GetProject MakeFileName("theme", "shp"),
  "* shp","Output GeoCoding Shapefile")
```

```
if (fnOutFile = nil) then
  exit
else
  aGeoName SetFileName(fnOutFile)
end

'Create a match key based on the standardization rules for the
'MatchSource  Use the aMatchKey AllowIntersections request to
'supporting street intersection standardization

aMatchKey  = MatchKey Make(aMatchSource GetStanRules)
aMatchKey AllowIntersections(aMatchSource GetXStanRules, aMatchSource GetXDelimiter)

'Create a new match case  A match case is comprised of a list of

'candidate records and information describing how well the candidates
'match the key  The MatchCase will be populated with candidates later

aMatchCase = MatchCase Make( aMatchSource, aMatchKey )
aMatchCase AllowIntersections(aMatchSource, aMatchKey)

'Create a new match preference which will be used to access various
'geocoding preferences, such as spelling weight, minimum acceptable
'score, etc

aMatchPref = MatchPref Make

'Create a new theme feature table  For every address record there

'will be a record in the FTab  These are currently unmatched, i e
'the shape field is empty and the status is 'U' for unmatched  We
'must match an address to populate the shape field and toggle the status
'to 'M' for matched, something we do in the next step

'aMatchSource InitGeoTheme request will open the geocoding index

aGeoTheme = aMatchSource InitGeoTheme( aGeoName )
geoThemeVTab = aGeoTheme
addrGeoThemeField = geoThemeVTab FindField("Started_fr")

zipGeoThemeField = geoThemeVTab FindField("Zip")

'Populate the feature table by matching addresses to the matchable
'Theme  This places a point in the FTab's shape field for and sets
'the status to M (matched) for each match

numrecs   = geoThemeVTab GetNumRecords
numMatched = 0

av ShowMsg("Matching Addresses  ")

for each i in geoThemeVTab GetDefBitMap
  av SetStatus((i / numrecs) * 100)

  'Get an address
```

```
aMatchKey SetKey( geoThemeVTab ReturnValueString(addrGeoThemeField, 1 ))
if (zipGeoThemeField <> Nil) then
  aMatchKey SetZoneKey( geoThemeVTab ReturnValueString(zipGeoThemeField, 1))
end

'Find candidates for the address

numCand = aMatchSource Search( aMatchKey, 70, aMatchCase)

'If there are no candidates, continue on to the next addresss  This
'will be an unmatched record  If candidates are found, take the best
'candidate and see if it exceeds the minimum specified match score
'If it does write it, otherwise write unmatch

'If there are no candidates, WriteUnMatch and continue to the next addresss

if (numCand = 0) then   'No candidates
  aMatchSource WriteUnMatch(1, aMatchKey)
else                    'We have at least one candidate

  aMatchCase ScoreCandidates
  cand     = aMatchCase GetBestCand
  candScore = cand GetScore
  minScore  = aMatchPref GetPrefVal( #MATCHPREF_MINMATCHSCORE )

  'If the min required match score is exceeded - write it'
  'If the min is not met then the cand is not written and the record
  'will remain unmatched

  if ( candScore >= minScore ) then
    aMatchSource WriteMatch( 1, aMatchKey, cand )
    numMatched = numMatched + 1
  end

 end
end

av ClearMsg
av ClearStatus
aMatchSource EndMatch

'Report the results of the geocoding

MsgBox Info("Total records processed "++numrecs AsString+NL+
  "Total addresses matched "++numMatched AsString, "Geocoding Results")

'Add the new theme to the view and draw it

newTheme = Theme Make( aGeoName )
theFTab = newTheme GetFTab
tripIdField = theFTab findField("trip_id")
newTheme SetLabelField(tripIdField)
theView AddTheme( newTheme )
newTheme SetVisible( true )
```

```
addressVTab RemoveFields({nonunIId})
obitmap = addressvtab getselection
addressvtab query("[trip_id] IsNull", obitmap, #VTAB_SELTYPE_NEW)
addressvtab removerecords(obitmap)
obitmap clearall
addressVTAb SetEditable(false)

'Create Startt and Endt fields in the geocoding attribute table,
'and store starting time and ending time in seconds

theftab seteditable(true)

starttfld = theftab findfield("Startt")
endtfld = theftab findfield("Endt")
if (starttfld <> nil) then
  theftab removefields({starttfld})
end
if (endtfld <> nil) then
  theftab removefields({endtfld})
end
startt = field make("Startt", #FIELD_LONG, 10, 0)
endt = field make("Endt", #FIELD_LONG, 10, 0)
theftab addfields({startt, endt})
stimefield = theftab findfield("S_time")
etimefield = theftab findfield("E_time")
if ((stimefield = nil) or (etimefield = nil)) then
 exit
end
theftab Calculate("[S_time] Trim left(2) Asnumber * 3600 + [S_time] Trim right(2) Asnumber * 60",
startt)
theftab Calculate("[E_time] Trim left(2) asnumber * 3600 + [E_time] Trim right(2) asnumber * 60", endt)

,
'update geocoding shape file's trip_id field If not matched,
'trip_id is set to 99999999, and it's previous record's trip_id is set by 99999999
'it's convenient for shortest path creation
,
valuelist = list make
nonunIId = theftab findfield("nonunIId")
tripid = theftab findfield("trip_id")
avfld = theftab findfield("Av_status")
for each i in theftab
  tmpStr = theftab ReturnValueString(nonunIId, i)
  if (tmpStr <> nil) then
    valuelist Add(tmpStr)
  end
end
valuelist removeduplicates
theftab seteditable(true)
thebitmap = theftab getselection
thebitmap clearall

for each j in 0  (valuelist count - 1)
  expr = "([" + nonunIId getname + "]= " + valuelist get(j) quote + ")"
  theftab query(expr, thebitmap, #VTAB_SELTYPE_NEW)
```

```
 theftab updateselection

 for each rec in thebitmap
   if (theftab returnvalue(avfld, rec) = "U") then
    theftab setvaluestring(tripid, rec, "99999999")
    if (thebitmap GetPrevSet(rec) <> -1) then
      theftab setvaluestring(tripid, thebitmap getprevset(rec), "99999999")
    end
  end
 end
end
thebitmap clearall
theftab updateselection
theftab seteditable(false)
```

```
'*****************************************************
'File Name  trip.match
'
'Description  make one street network matchable
'
'*****************************************************

'Prepare for address matching by making a theme matchable

theProject = av.GetProject
theTheme = theProject FindDoc("View1") FindTheme("Tgr47093lka shp")
if (nil = theTheme) then
  MsgBox Error ("Unable to access Street theme in view1", "")
  exit
end

'Verify whether the theme is already matchable

if (theTheme isMatchable) then
  MsgBox Warning ("Street is already matchable", "")
  exit
end

'The GetDefStylesODB request to the AddressStyle class returns
'a file name object corresponding to the style object database
'supplied with ArcView

addrStyleFilename = AddressStyle GetDefStylesODB
if (nil = addrStyleFilename) then
 MsgBox Error ("Unable to find the default address style ODB", "")
  exit
end

'The list of styles can be extracted from the address style
'file name, and the desired style can be found within that list

addrStyleList = AddressStyle GetStyles
(addrStyleFilename)
addrStyle = AddressStyle FindStyle
("US Streets with Zone")
'Associate the known fields to the address components
theVTab = theTheme GetFTab
attList = { }

'Setup for the US Streets with Zone style

nameList = {"Fraddl", "Toaddl",
"Fraddr", "Toaddr", "NONE", "NONE",
"Fename", "Fetype", "NONE", "Zipl", "Zipr"}
for each fldName in nameList
  if (fldName = "NONE") then
```

```
    'NONE indicates that there is no field to match

    attList Add ("") 'Add a null value
    continue
  end

  'Get the field object

  aField = theVTab FindField (fldName)
  if (nil = aField) then
    MsgBox Error ("Unable to access required field"++ fldName,"")
    exit
  end
  attList Add (aField)
end

'Create a MatchSource object

aMatchSource = MatchSource Make (addrStyle, theTheme, attList)

'Now assign the MatchSource object to the theme

theTheme SetMatchSource (aMatchSource)

'Verify whether the theme is matchable

if (theTheme IsMatchable) then
  MsgBox Info (theTheme GetName++"is matchable", "")
else
  MsgBox Warning (theTheme GetName++"Is not matchable", "")
end
```

```
'*****************************************************************
'File Name  buildtrees
'
'Description  This program builds array of two-d trees  After building, the array of two-d trees
'             will be stored in memory  When users make a query, it will go to query this array
'
'*****************************************************************

av clearglobals
theFname = filedialog show("* dll", "DLL(* dll)", "Get BuildIndexDLL file")
if (nil = theFname) then
 msgbox info("Please select DLL file", "ERROR")
 exit
end
myDLL = DLL Make(theFname)
twoD = DLLProc Make(myDLL, "BuildIndex",
#DLLPROC_TYPE_POINTER,{#DLLPROC_TYPE_STR})
if (twoD = nil) then
 msgbox error("Error", "")
 exit
end
infile = "secpaths txt"
_mytwod = twoD Call({infile})
_querytime1 = DLLProc Make(myDLL, "QueryTime1", #DLLPROC_TYPE_INT32,
        {#DLLPROC_TYPE_POINTER, #DLLPROC_TYPE_INT32, #DLLPROC_TYPE_INT32,
#DLLPROC_TYPE_INT32,
        #DLLPROC_TYPE_STR})
_querytime2 = DLLProc Make(myDLL, "QueryTime2", #DLLPROC_TYPE_INT32,
        {#DLLPROC_TYPE_POINTER, #DLLPROC_TYPE_STR, #DLLPROC_TYPE_STR,
#DLLPROC_TYPE_INT32,
        #DLLPROC_TYPE_INT32})
```

```
'*****************************************************************************
'File Name  stquery tool
'
'Description  perform a spatiotemporal query (i e  which trips pass through one
'          or more specific streets within one time window
'
'Input  Users selecting streets, and a time window
'
'Output  a report box containing query results, the query result is also
'       shown on the map
'*****************************************************************************

'create an arclist list, get network (Tgr47093lka)'s ftab and trip path (Paths)'s ftab

arclist = list make
theView = av GetActiveDoc
r = theView ReturnUserRect
theTheme = theView GetActiveThemes get(0)
pathftab = theView findtheme("Paths") getftab
pathfld = pathftab findfield("Paths-id")
theftab = theTheme getftab
arcfld = theftab findfield("Tgr47093lka#")

'the user can either draw a box or a point to query features

thebitmap = theftab getselection
thebitmap clearall
if (r IsNull) then
  p = theView GetDisplay ReturnUserPoint
  if (System IsShiftKeyDown) then
    op = #VTAB_SELTYPE_XOR
  else
    op = #VTAB_SELTYPE_NEW
  end
  if (theTheme CanSelect) then
    theTheme SelectByPoint(p, op)
    thebitmap = theftab getselection
    theftab updateselection
    pathftab getselection clearall
  end
else
  if (System IsShiftKeyDown) then
    op = #VTAB_SELTYPE_OR
  else
    op = #VTAB_SELTYPE_NEW
  end
  if (theTheme CanSelect) then
    theTheme SelectByRect(r, op)
    thebitmap = theftab getselection
    theftab updateselection
    pathftab getselection clearall
  end
end
```

```
'Given a time window

if (thebitmap count > 0) then
 labels = {"Lower end of the time window", "higher end of the time window"}
 defaults = {"0800", "1000"}
 timewindow = msgbox multiinput("Enter a time period", "Time Window", labels, defaults)
 if ((timewindow = nil) or (timewindow isempty)) then
  msgbox error("You need to input a time window", "ERROR")
  return nil
 end

'convert time range into seconds

 bt = (timewindow get(0) trim left(2) asnumber * 3600) + (timewindow get(0) trim right(2) asnumber * 60)
 et = (timewindow get(1) trim left(2) asnumber * 3600) + (timewindow get(1) trim right(2) asnumber * 60)
end

'if selecting an arc, then use querytime1 function

if (thebitmap count = 1) then
 for each i in thebitmap
  arcid = theftab returnvalue(arcfld, i)
 end
 query1 = _querytime1 call({_mytwod, arcid, bt, et, "r txt"})

 ' get the query result from r txt and query route feature table

 f = linefile make("r txt" Asfilename, #FILE_PERM_READ)
 tf = textfile make("r txt" asfilename, #FILE_PERM_READ)
 if (tf getsize = 0) then
  msgbox info("no trips found", "")
  thebitmap clearall
  return nil
 end
 resultlist = tf read(tf getsize)
 tf close
 ok = true
 thebitmap clearall
 pathftab getselection clearall
 pathbitmap = pathftab getselection

 'repeatedly query pathftab

 while (ok)
  s = f readelt
  if (nil = s) then
   ok = false
  else
   expr = "([Paths-id] = " + s aslist get(0) + ")"
   pathftab Query(expr, pathbitmap, #VTAB_SELTYPE_OR)
  end
 end

 'display query result in a message box
```

```
  msgbox report(resultlist asstring, "Trip_Id, Beginning Time (seconds), Ending Time (seconds)")
  f close

'if selecting more than one arc, then use querytime2 function

elseif (thebitmap count > 1) then

  'get the selected arcs, put into arclist, then output to infile txt

  arclist empty
  for each i in thebitmap
   arclist add(theftab returnvalue(arcfld, i))
  end
  infile = linefile make("infile txt" asfilename, #FILE_PERM_WRITE)
  for each i in 0   (arclist count - 1)
   infile writeelt(arclist get(i) asstring)
  end
  infile close

  'use querytime2 function to get query result

  query2 = _querytime2 call({_mytwod, "infile txt", "r txt", bt, et})
  f = linefile make("r txt" Asfilename, #FILE_PERM_READ)
  tf = textfile make("r txt" asfilename, #FILE_PERM_READ)
  if (tf getsize = 0) then
   Msgbox info("no trips found", "")
   thebitmap clearall
   return nil
  end
  resultlist = tf read(tf getsize)
  tf close
  ok = true
  thebitmap clearall
  pathftab getselection clearall
  pathbitmap = pathftab getselection

  'repeatedly query path ftab

  while (ok)
   s = f readelt
   if (nil = s) then
    ok = false
   else
    expr = "([Paths-id] = " + s aslist get(0) + ")"
    pathftab Query(expr, pathbitmap, #VTAB_SELTYPE_OR)
   end
  end

  'display query result into a message box

  msgbox report(resultlist asstring, "Trip_Id, Beginning Time (seconds), Ending Time (seconds)")
  f close
end
av GetProject SetModified(true)
```

Vita

Feng Lu was born in Hunchun, Jilin Province, P.R.China, on August 26, 1967 He attended No. 2 Yanbian High School in Yanji, where he graduated in July 1986 He entered Peking University in September 1986, and graduated in July 1990 with a Bachelor of Science degree majoring in Economic Geography. From July 1990 until August 1994, he worked as a research assistant in Changchun Institute of Geography, Academia Sinica, and got a Master's degree in Cartography and Remote Sensing in July 1994. From September 1994 to July 1997, he worked as a research assistant in the Department of Urban and Environmental Sciences at Peking University. He entered Ph D. program in urban studies at the University of New Orleans in August 1997, then transferred to the Department of Geography at the University of Tennessee in August 1998 Upon successful defense of this thesis, Feng Lu will receive his M.S. in Geography with an emphasis in GIS and Transportation.