

THE UNIVERSITY OF CALGARY

OBJECT ORIENTATION IN ROUTE GUIDANCE SYSTEMS

by

SHUI LIU

A THESIS

**SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN
PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE
OF MASTER OF SCIENCE**

DEPARTMENT OF GEOMATICS ENGINEERING

CALGARY, ALBERTA

SEPTEMBER, 1996

© SHUI LIU 1996



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced with the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-20876-1

ABSTRACT

The value of Route Guidance Systems (RGS) has been recognized since the nineteen seventies, but only recent technological advances in computer science, positional devices, and geographical information systems have made the practical implementation of RGS possible.

Two of the basic issues in route guidance systems, the objectives of this research, are road network modeling and optimal path searching.

Unlike many other kinds of data handled routinely by most information systems, geographical data in road networks are complicated by the fact that they are encapsulated with rich internal attributes and structural relations between components, which implies that higher requirements for data handling mechanism are necessary. The relational data model, which is the dominant data model used in commercial GIS, has proved to be inadequate for road network modeling. In this research, the applicability of the object-oriented data model is investigated, and the basic classes defined. As was expected, this model shows its efficiencies in terms of data manipulation, model extensibility, data semantics, and reducing data redundancy.

Optimal path searching algorithms are here studied. In particular, the searching spaces of D*(Dijkstra's algorithm), A* , and BA*(Bi-directional A*) algorithms are investigated. The relationship between the number of nodes involved in the searching process and the accuracy of the solution in A* algorithm are examined. Finally, a new optimal path searching algorithm is proposed and the results will demonstrate the reliability and efficiency of this algorithm.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Rongxing Li, for his guidance, support, and encouragement throughout my graduate studies.

Special thanks go to my friends Mr. Guangyu Zhang and his wife Xiaoqing Ye for their financial and emotional support during my study here. Their kindness and encouragement will forever be appreciated.

Dr. J. A. R. Blais and Dr. Richard Levy are acknowledged for their critiques of this thesis work.

Appreciation also goes to my fellow graduate students, Mr. Liming Qian, Mr. Haihao Li, Mr. Chuang Tao and Miss Ying Chen for their assistance through these two years.

Dedicated to my parents and my wife

TABLE OF CONTENTS

APPROVAL PAGE.....	ii
ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
DEDICATION.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x

CHAPTER	Page
1 INTRODUCTION.....	1
1.1 Route guidance systems.....	1
1.2 Conventional approach and its problems.....	4
1.3 Objectives of the research.....	8
1.4 Thesis overview.....	8
2 MODELING ROAD NETWORKS.....	10
2.1 Road networks.....	10
2.1.1 The elements of road networks.....	10
2.1.2 The problems with the currently used model.....	12

2.2 Data models.....	14
2.3 Relational data model and its limitations.....	16
2.3.1 Relational data models.....	16
2.3.2 Problems with relational data models.....	17
2.4 Object-oriented data models.....	20
2.4.1 Basic concepts in object orientation.....	20
2.4.2 Object-oriented models.....	24
2.5 Modeling road networks.....	28
2.5.1 Descriptions of currently used road network models...	28
2.5.2 Object-oriented model for road networks.....	29
3 OPTIMAL PATH SEARCHING ALGORITHMS.....	40
3.1 Introduction	40
3.2 Dijkstra's algorithm.....	41
3.3 Heuristically-informed searching.....	45
3.3.1 A* algorithm.....	45
3.3.2 Bi-directional search.....	49
3.3.3 Searching within a window.....	50
3.3.4 Hierarchical structure-based searching.....	53
3.4 Road-based searching.....	54

4 IMPLEMENTATION AND RESULT ANALYSIS.....	59
4.1 Data structure used in the existing road networks.....	59
4.2 Searching space in D* algorithm.....	61
4.3 Result analysis on A* algorithms.....	65
4.3.1 Searching space in A* algorithms.....	65
4.3.2 Relative error analysis on A* algorithms.....	67
4.4 Result analysis on road-based searching.....	70
5 CONCLUSIONS AND RECOMMENDATIONS.....	78
5.1 Conclusions.....	78
5.2 Recommendations.....	81
REFERENCES.....	82

LIST OF TABLES

No.	Page
4.1 Result comparison of route 1.....	72
4.2 Result comparison of route 2.....	73
4.3 Result comparison of route 3.....	73
4.4 Result comparison of route 4.....	74
4.5 Result comparison of route 5.....	76

LIST OF FIGURES

No.	Page
2.1 Elements of road network.....	11
2.2 Object-oriented vs. Record-oriented data models.....	25
2.3 Representation of object in object-oriented data model.....	30
2.4 Structure of object node.....	31
2.5 Structure of object arc.....	32
2.6 Road as a composite object of arcs.....	32
2.7 Object super_network consisting of Master_nodes and Master_edges.....	33
2.8 The basic structure of class arc.....	34
2.9 Hierarchy of fundamental objects in networks.....	38
3.1 Intermediate stage during the execution of D* algorithm.....	42
3.2 An example of D* algorithm.....	44
3.3 A* example network.....	47
3.4 An example of A* algorithm.....	48
3.5 An example of BA* algorithm.....	50
3.6 Searching within a window.....	52
3.7 Searching using “hierarchical structure”.....	54
3.8 Some of the factors considered in the process of road selecting.....	56

3.9 Road-based searching.....	57
4.1 Data structure for the storage of road networks.....	60
4.2 An example of D* algorithm.....	63
4.3 An example of D* algorithm.....	64
4.4 Searching space in D* algorithm.....	64
4.5 Relationship between the searching spaces in D* and BD* algorithms.....	65
4.6 An example of A* algorithm.....	66
4.7 Searching space in A* algorithm.....	66
4.8 Relationship between the searching spaces in A* and BA* algorithms.....	67
4.9 Relationship between the number of nodes and the average speed.....	69
4.10 Relationship between relative error and average travel speed.....	70
4.10 First simulated network.....	71
4.11 Second simulated network.....	75

CHAPTER 1

INTRODUCTION

The value of Route Guidance Information Systems (RGIS) has been recognized since the nineteen seventies. Since then, numerous navigation related technologies have been developed, but only the recent technological advances in computer science, positioning devices, and geographical information systems, have made the practical implementation of RGIS possible.

Some of the reasons for the high demand of route guidance information systems are for the protection of human life, economy, security, and travel time management. Some studies have shown the benefits gained by using such systems. For example, the Organization for Economic Cooperation and Development [OECD, 1988] published a study report on the benefits of route guidance systems which indicates that billions of dollars are lost annually by the industrialized nations due to the lack of appropriate route guidance systems.

A number of other studies have shown that drivers often take routes which are longer than the optimum routes, due to a lack of knowledge of those optimum routes. These studies conclude that on average, 6% of the total mileage driven and 12% of the total travel time are excess travel beyond the optimum route [Yuval, 1989]. In the light of such findings, it is clear that substantial reductions in travel can be achieved, if drivers are assisted in their route selection.

1.1 Route Guidance Information Systems

Route Guidance Information Systems can provide drivers with information regarding the road network. They produce a sequence of routing instructions and suggest the recommended routes to reach specific destinations.

Among many definitions of route guidance systems, Krakiwsky [Krakiwsky, 1987, Bullock, 1995] presented that a RGIS allows a driver to:

- (a) position a vehicle using signals from satellites and information from on-board positioning devices;
- (b) plot the position on a CRT or flat panel display;
- (c) call up a digitized electronic map of the area and see the vehicle's position relative to a desired location(s); and,
- (d) obtain instructions (visual and/or audio) using an expert system on how to proceed from the present location to the desired location.

Generally speaking, the RGIS consists of four primary modules, namely positioning and locating module, database on road network, best route calculation, and route guidance modules [Reginald, 1991, Lapalme, 1992].

Positioning and locating modules give answers to the question of "where am I?". Positioning and locating are the two key technologies behind any route guidance systems. Positioning technologies - such as GPS and dead reckoning, positioning (coordinates) in a defined coordinate space, and establishing a vehicle's position relative to a particular environment, in most cases the digitally defined street network.

In order to locate the vehicle on the digital map, two map-related functions, address matching and map matching, should be provided. Address matching determines a street address from given coordinates, or vice versa. This step is necessary because most people know the address of their destination rather than the coordinates. To navigate to an address the navigation system must convert the address to coordinates in navigation systems.

The purpose of map-matching algorithms is to find the nearest link and “snap” the vehicle onto that link when a positioning system outputs coordinates that are not exactly on a road link in a digital road map.

Databases on the road network contain all the necessary environmental information. The most common data models for the management of such data are hierarchical, network, relational and object-oriented models. The data model used for modeling road networks should be carefully considered because the data model, which will be discussed in detail later, has great effects on the efficiency of the database, and once implemented, is difficult to change. In other words, an improper data model would have long lasting effects on the performance of the RGIS. Selecting an appropriate data model has been one of the major concerns for the designers of route guidance information systems.

Best-route calculation modules can provide an optimal path based on the criteria such as shortest distance, minimum travel time, or minimum cost. Best-route calculation requires high levels of functions provided on the road network database. The database must at least know the directionality of each road link and turn restrictions so that the route selected does not include any impossible or illegal turn, or travel the wrong way on one-way roads. For the calculation of the best route, some other factors should also be considered, which include, but are not limited to, the classes of road, speed limits, number of lanes, and so on.

Route guidance modules provide turn-by-turn driving instructions from the initial location to the destination once the user inputs the destination of the trip through the user interface. As the vehicle travels, the driver knows where the vehicle is in the route and as a turn or maneuver approaches, the algorithm alerts the driver and indicates when the maneuver should be performed.

In brief, route guidance information systems provide the functions of positioning, locating, best-route calculation, and guide the driver to his/her destination in the most efficient way.

1.2 Conventional Approach and Problems

It is not the intention, in this research, to discuss all the problems related to route guidance information systems. Instead, only two issues are going to be studied. They are data models for road networks and optimal path searching algorithms, which are considered to be the basic problems in route guidance systems and are still open.

a) Modeling road networks

Unlike many other kinds of data handled routinely by most information systems, geographical data in road networks are complicated by the fact that they are encapsulated with rich internal attributes and structural relations between components. The relational data model is currently used for modeling road networks. This model has been widely accepted in GIS and other information management systems because of its simplicity and the availability of a standard language (the structured query language, SQL) for the manipulation of the database.

Relational data model and relational database management systems were originally developed for conventional business data processing applications, such as inventory control systems, hotel management, ticket reservation, accounting systems, and so on. They are found to be effective in managing structured data found in these cases. Attempts to make use of relational database in a variety of other applications, where data are not conceptualized or are logically complex, have quickly exposed several serious shortcomings of relational database technology. The reported shortcomings of relational data model include [Taylor, 1990; Worboys, 1990, 1994]:

- Inadequate support for the treatment of complex objects.

The number of data types provided by a relational data model is limited. It is however possible to represent complex objects by using this model. The First Normal Form (1NF), which is considered the fundamental requirement of this model, forces the complex objects to be decomposed into their component elements, until they can be described directly by the built-in data types, or when objects are queried from the database. Those

elements have to be reassembled to construct meaningful entities. This procedure of decomposing and reassembling complex objects to and from its elements introduces a large overhead in both construction and query processing.

- **Data redundancy**

To represent complex objects using the relational data model, two or more tables are needed. The relationships between tables are established by virtue of common data values contained in the corresponding tables. This approach introduces great data redundancy.

Data redundancy is one of the problems associated with relational databases. Not only does it waste storage space and slow down processing, but it can lead to serious data corruption. If the same details are stored in two different tables in a database, it may be possible for the item in one of those tables to be changed, while the corresponding item in the other table remains the same. This generates discrepancy, and there may be no way of telling which version is correct.

- **Unacceptable performance for various types of computing-intensive applications.**

A relational data model tends to split data into multiple tables or files in order to cut down on duplicated data, the tables being manually linked together using artificial keys. The relational design process of 'normalization' usually means that each table can only hold a small number of fields. Consequently, the information pertaining to a record may be spread out over a large number of tables. For this reason, response time for queries is weak, due to the need to access multiple tables and 'join' them in order to obtain information.

To overcome the problems mentioned above, two technologies have been proposed, namely, the extended relational data model and the object-oriented data model.

The fundamental differences between them are the basic model and the database language. The extended relational approach starts with the relational model of data and a

relational query language, and extends them in various ways to allow the modeling and manipulation of additional semantic relationships and database facilities.

The object-oriented approach starts with an object-oriented data model and a database language that captures it, and then extends them in various ways to allow additional capabilities.

One important point that must be recognized is that an object-oriented data model is a more natural basis than an extended relational model for addressing some of the deficiencies of the relational database technology. Object orientation has been adopted in system analysis/design, computer programming, and information systems. These applications have the same characteristics of being logically complex and requiring sophisticated data modeling [Garvey, 1989; Kim, 1990 and 1995; Hughes, 1991, Olajide, 1995]. Advantages of an OOM include rich typing, inheritance, object identity, and the encapsulation of data and operations which can be used to model road networks.

b) Optimal path searching algorithms

Finding the optimal path in a street network can be abstracted to the mathematical problem of finding the minimum-cost path in a weighted graph. There is extensive literature on graph-search algorithms and, in particular, algorithms to determine the optimal paths together with their applications (Gibbons, 1985; Kuipers, 1992, 1988; Lapalme *et al.*, 1992; Johnson, 1977, Frederickson, 1994, Gallo, 1984, Gopal and Smith, 1990).

Generally speaking, there are two kinds of algorithms used for finding the optimal route in a network. One is following strict mathematical steps by which the best route can be obtained. Dijkstra's algorithm is one of the most widely used algorithm for shortest path searching [Dijkstra, 1959]. The advantage of this approach is that the best solution can be obtained. The negative side of this algorithm is that the running time is unacceptable when applied to very large road networks. This is one reason why so many alternatives exist.

Other approaches use so-called heuristics. The basic idea is to try to simulate the methods people use when they are given the task of finding the best route. Humans can figure out the solution quickly using a very large and complex network based on common sense (knowledge). The characteristics of this method are that the solution obtained is not 100% guaranteed to be the best one, but close to it. However, the searching time can be greatly reduced because it reduces the number of nodes involved in the searching process.

A* algorithm, Bi-directional A* algorithm, window-based searching, and hierarchical structure-based searching algorithms are the typical heuristically-informed searching algorithms.

All the algorithms mentioned above involve arc-based searching. Although these algorithms have advantages over the Dijkstra's (D*) algorithm, the computation time is still a problem when these searching algorithms are applied to very large road networks. This is because during the searching process, each algorithm creates a tree which branches out, iteration by iteration, growing each time, extending the search from the source node to the destination node. The major difference between the D* algorithm and heuristically-informed algorithms is in the searching space. In heuristically-informed algorithms, the searching space becomes elliptical in shape rather than the circular shape associated with the D* algorithm.

Now let us examine how humans look for the best route. Generally, when a human is given the task to identify the best route between two points on a road map, one of the first steps performed is to look at roads which lie between the source node and destination node, neglecting all roads which seems to take you further away from the destination goal. Here at least two things can be learnt from this procedure: The searching is road-based searching. So firstly, people find the best road by examining and connecting different roads rather than different arcs or segments. Secondly, all the information, positional and non-positional such as the class of the road, and speed limits, is used in the searching process. This is why people can figure out the best solution in a very short time no matter how complex the road networks are. If there is one algorithm that performs this

way, it can be sure that the computation time for optimal path will be significantly reduced.

1.3 Objectives of the research

This research aims to solve some of the problems presented in the current route guidance information systems. In detail, the research objectives are:

- to investigate various existing data models of spatial information systems;
- to demonstrate the advantages of the object-oriented data model over relational or other data models when used in modeling road networks;
- to study the characteristics of the currently used optimal path searching algorithms, especially, the road-based searching algorithm, which is based on the object-oriented data model and knowledge-based reasoning, is proposed to reduce the computation time for optimal path searching.
- to analyze some experimental results with simulated data. The efficiency and accuracy of optimal path searching algorithms will be investigated.

1.4 Thesis Overview

There are five chapters in this thesis. The principal contents of each chapter are described as follow:

Chapter 1 gives an introduction to the entire research initiative. It provides some background information on the topic and development work covered in the later chapters.

Chapter 2 focuses on the principles and methodologies for modeling road networks. First, a brief introduction to data models is given, and then the relational data models and the problems of relational models when they are used for modeling road networks are examined. Following that, the object-oriented data model will be discussed,

explaining why it is more suitable. Finally, the objects involved in road networks are defined.

In Chapter 3, typical algorithms used for finding the optimal path in road networks will first be reviewed and compared, and then new algorithms will be proposed, which try to combine the advantages of existing methods and use knowledge-based reasoning to calculate the optimal path more efficiently.

In Chapter 4, the results obtained by different algorithms will be given and compared. Especially, the searching space of each algorithm will be studied, and the relationship between the number of nodes involved in the searching process and the accuracy of the “optimal path” discussed. Result analysis on road-based searching algorithm will be given.

Chapter 5 summarizes the main conclusions and contributions of the thesis and recommends directions for further work.

CHAPTER 2

MODELING ROAD NETWORKS

This chapter will discuss the data models used for RGIS. It proceeds with a brief introduction on road networks, followed by a review of the purpose and concepts of data models so as to provide context for the following discussion of the object-oriented data model. After a general discussion of the data models, the chapter will examine the disadvantages of the relational data model, the dominant data model currently used in RGIS. It then reviews the most representative approaches for improving the relational model. Following that is the introduction of the object-oriented technology. It concludes by describing the characteristics of the data model - object-oriented data model for road networks.

2.1. Road networks

2.1.1 Road networks: elements and their attributes

A road network is a system of connected roads through which traffic flows. The rate of flow is controlled by several factors such as the type of street, speed limit, congestion, presence of traffic control devices, and so on. All of these factors can be added to networks to realistically model movement through a network.

a) Elements in a network:

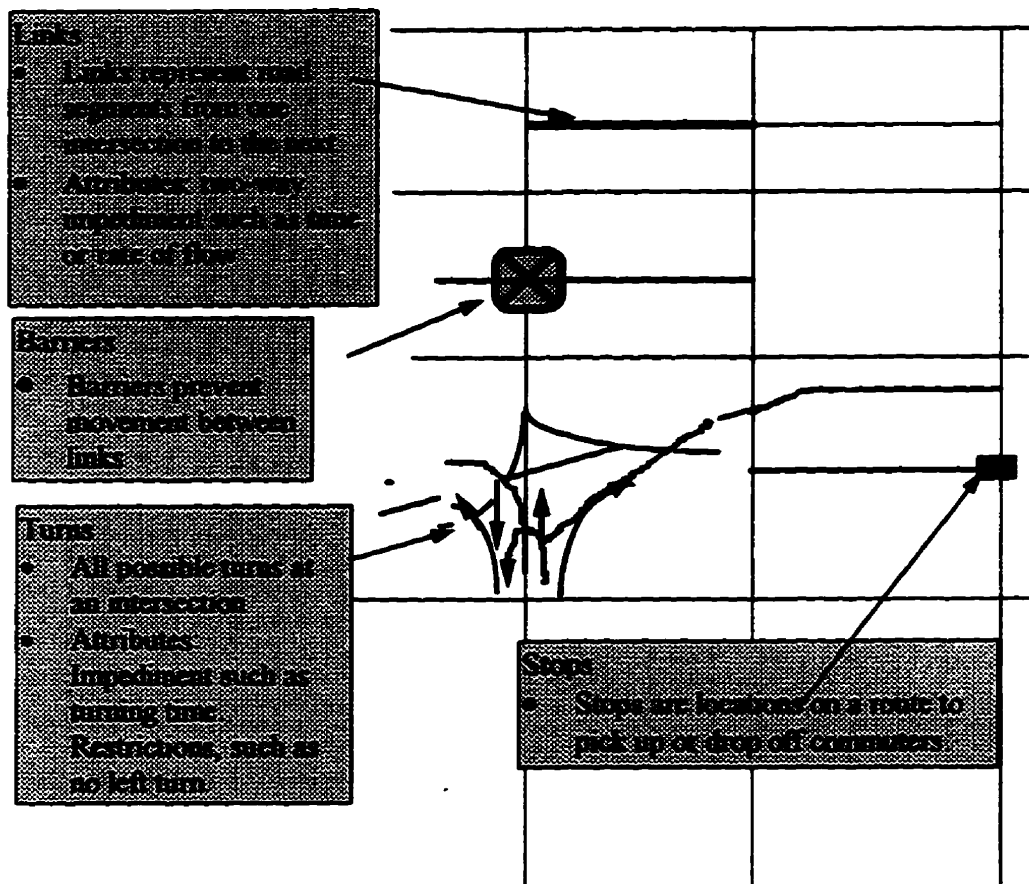


Figure 2.1: Elements of road network

As shown in Figure 2.1, each network contains a number of elements, among others, links, barriers, turns, and bus stops.

b) Attributes associated with network elements:

Most of the network elements have one or more characteristics that are an important part of the network. For example, in an urban street network, each street has a name, speed limit and width; turns may include overpasses with either unobstructed flow or traffic control devices; and a bus stop might have a number of people to be picked up or dropped off.

Impedance measures resistance to movement. Impedance is attributes of arcs and turns. Arc impedance is the amount of resistance required to traverse the arc from one end to the other. For example, the length of arcs can be used as the impedance; longer arcs having a larger impedance than shorter arcs. A larger impedance indicates more resistance to movement.

Turns also have impedance attributes. They measure the amount of resistance in moving from one arc, through a node, onto another arc. Turn impedance will vary according to the conditions at the intersection of the two arcs. For example, impedance of traffic flow passing through a stop sign and moving onto another arc is greater than if there were no stop sign.

The possible number of turns at any node is equal to the number of connecting arcs multiplied by itself. So if two arcs connect, there are four possible turns; if three arcs connect at a node, turns can be made in nine directions.

All arc and turn impedance in a network must always be in the same units for accurate calculation.

The purpose of impedance is to simulate the variable conditions along lines and turns in real networks. The results of a route will differ according to the impedance you assign to elements in a network data set. The optimum path is the path of least resistance.

2.1.2 The problems with the currently used data model for road networks

The database on road networks serves two purposes in Route Guidance Systems: providing information to calculate the optimal route and giving enough information - turn-by-turn instructions guiding the user in travel from a given point to another specified point.

The data models currently used for modeling road networks are the hybrid (relational) data models, which means that the relational data model is used to handle the thematic information, while a separate subsystem is included to store and retrieve

geometric information. The linkages between the thematic and geometric data regarding the same object are established by unique identifiers.

In this model, the network is composed of two basic elements, links and nodes. Nodes terminate links, links always join two nodes, and links that terminate at the same node can be considered connected.

The drawbacks of this representation can be summarized as follows:

No intelligence: Links in this model represent road segments from one intersection to the next. One critical characteristic of links in this model is that they are completely independent of each other. Each segment of State Street does not “know” that it is a part of the same street as the link connected to it. While these implicit relationships may be derived from the data in the model, such independence makes manipulation and analysis of the entire entity State Street less efficient. Intelligence is very important in spatial reasoning, especially in road guidance systems whose main tasks are to search for the optimal path and provide users with turn-by-turn instructions. In addition, this sort of representation necessitates redundancy in the attribute data and requires more explicit procedures to maintain integrity of the data over the entire road and network.

Architecture is not elegant: In this model, an object that has both a thematic and a spatial component, has parts in both subsystems. In order to retrieve an object, the two subsystems have to be queried and the answer composed. The existence of two storage subsystems implies that query optimization is not possible to the full extent, because the two storage managers each have their own locking protocol. Another drawback of this dual architecture is that integrity constraints of the system can be violated. For example, an entity’s geometric information can still exist in the geometric storage subsystem while its thematic information has been deleted from the relational DBMS.

Route Guidance Information Systems (RGIS) are being considered internationally as a means of managing traffic flow. The basis of this technology is a spatial database to support the applications on which RGIS are built. While there is ongoing work

progressing regarding the technical aspect of RGIS, such as the most effective technology for positioning, and transmitting positional or traffic flow information, there is little exploration of the most appropriate structure of the database.

This chapter will discuss the data models used for RGIS. It proceeds with a brief review of the purpose and concepts of data models in order to provide context for the following discussion of the object-oriented data model. After a general discussion of the data models, the chapter will examine the disadvantages of relational data model, the dominant data model currently used in RGIS. It then reviews the most representative approaches for improving the relational model. Following that is the introduction of the object-oriented technology. It concludes by describing the characteristics of the data model - object-oriented data model for road networks.

2.2 Data Models

The choice of an appropriate representation for the structure of a problem is perhaps the most important component of its solution. For database design, the means of representation is provided by the data model. A data model provides a tool for specifying the structural and behavioral properties of a database and ideally should provide a language which allows the user and database designer to express their requirements in ways that they find appropriate, while being capable of transformation to structures suitable for implementation in a database management system [Tsichrizis, 1982]. Data modeling is among the first stages of database design. The purpose of data modeling is to bring about the design of a database which performs efficiently; contains correct information (and which makes the entry of incorrect data as difficult as possible); whose logical structure is natural enough to be understood by users; and is as easy as possible to maintain and extend. Of course, different problems require different means of representation and a large number of data models is described in database literature [Chen, 1976, Data, 1990, Frank, 1989, Herring, 1992, Hull, 1987, Goodchild, 1992, Usery, 1993].

Among many definitions of a data model, Codd [1981] presented that a data model consists of three components:

1. A collection of data object types (essentially the entities that form the database structures).
2. A collection of general integrity rules, which constrain the set of instances of those object types that can legally appear in a database. A data model often requires integrity constraints because the components defined in the data model are usually application specific.
3. A collection of operators, which can be applied to such object instances for retrieval and other purposes.

A database system can be distinguished from others by the data structures and the operators it presents to the user. The user of a relational system sees the data as tables. Operators within and among tables are supported by relational algebra. Basic operations include table union, difference, intersection, selection, projection, etc. The user of non-relational systems sees other data structures. Those other structures, in turn, require other operators to manipulate them. For example, in IMS, which is a hierarchical system, the data are presented to the user in the form of a set of tree structures, and the operators provided for manipulating such structures include operators for traversing hierarchical paths up and down the tree.

The spatial database represents real world objects as seen by an application. Its design often evolves through the hierarchical processes of conceptualization of reality in a data model - conceptual model, which incorporates only these properties thought to be relevant to the application or applications at hand; the structuring of this model being in a computer-representable format - logical model; and the design of a file structure for the storage of the structured data - physical model.

While logical designs of databases depend very much on different application situations, actual designs have been dominated by a single model: the relational model.

During the past three decades, DBMS have evolved from hierarchical, network to relational databases [Lee, 1990]. Today's most popular commercial DBMS, namely, Oracle, Sybase, Ingres, Arc/Info, etc., are all based on relational data models [Ullman, 1988].

2.3 Relational data model and its limitations

2.3.1 Relational data model

The fundamentals of the relational model were presented by E.F. Codd in a classic paper [Codd, 1970]. The mathematical formalism underlying the relational model is based on set-theory.

From the point of view of a user or database programmer, a relational data model consists of named tables (relations). Rows of these tables are called tuples, while columns are called attributes. Entities are represented by tuples of attribute values. Associated entities have attribute values in common.

Tables are always normalized, that is, the construction of relational tables is guided by the so-called normalization rules, in order to restrict redundancy, so that each entry in a table is a single value, never a set of values. Thus information is represented in a simple, general, and uniform way, which greatly simplifies the tasks of updating, factoring out redundancy, finding inconsistencies, and generating reports.

The manipulation language of relational tables, called SQL, has been standardized and is now commonly used for specifying reports from relational databases. There are two different kinds of notations for expressing operations on relations:

a) Algebraic notation, called relational algebra, where queries are expressed in a procedural manner by applying specialized operations to relations. The relational algebra consists of a collection of eight operations that can be grouped in two categories [Data, 1990]: (i) the basic operations on sets that apply to relations: union, intersection,

difference, and cartesian product, and (ii) the special relation operations: select, project, join, and division.

b) Logical notation, called relational calculus, where queries are expressed in a declarative manner by writing logical formulas that the tuples in the answer must satisfy.

2.3.2 Problems with relational models

The relational model of representation has been most widely accepted because of its simplicity and the availability of the standard query language (SQL) for the manipulation of the database. Various commercial systems are also available in the market (e.g., DB2, Oracle, Ingres, dBase, etc), but this model is more suitable for the management of non-spatial data and conceptual information, such as business applications. However, when they were applied in type-rich applications, where data are not conceptualized or logically complex, they began to show insufficiencies in terms of manipulation efficiency, model extensibility, data semantics, and architecture [Lee, 1992].

Manipulation Efficiency

Efficiency refers to the speed by which data can be manipulated. There are several reasons why a relational database is slow under certain circumstances. One of them is due to the requirement that all relations must be in the 1NF: nested relations are not allowed and the value of any attribute must be atomic (simple, indivisible) and a single value.

The 1NF assumption is fundamental to relational models, and the separation of related data into different tables is dictated by rules for good design in relational databases, mainly to avoid problems in updating. As a result, complex objects with rich information have to be decomposed into elementary items in order to fit into the entries of a relational table. When these objects are queried from the database, the system has to reassemble those items to construct meaningful entities. This nature of operation can negatively affect the efficiency of the system. Complex spatial objects are encapsulated

with rich internal attributes and structural relationships between components. Decomposition introduces a large overhead in both construction and query processing.

Data Semantics

Semantics in a database refer to the meaning of data attributes and relations. A data model that can relate data together but does not maintain the meanings of these linkages lacks semantics.

A data model rich in semantics must distinguish between different types of relations, which is particularly important for providing useful integrity constraints and an environment for database browsing. In spatial data modeling, three relationships, namely classification, generalization, and aggregation, are of particular interest to us. Classification is a form of abstraction that groups several objects, according to their structural and behavioral properties, into a common class. Generalization is the grouping of several classes, which share common properties, into a higher order class. This process highlights the similarities of several classes and hides their differences. This mechanism is very useful when there is a need to store properties common to several classes. Aggregation is the forming of complex objects from its component objects.

The different types of relationships required in spatial data modeling are more than the above three. The relational data model cannot distinguish these relationships because it provides only two constructs for representing relationships; one within a table and the other across tables through common values. This leads to semantic overloading [Hull and King, 1987] indicating that a single construct must support several types of relationships thus causing an ambiguity in meaning.

Model Extension

A data model that provides a limited number of data types and does not allow the creation of new types by the user is called inflexible. In a relational database, several built-in data types such as integer, real numbers, and character strings, are usually provided. Above them are the relations, the only type-like constructs that a user can

define. They are however not true types because they cannot be used in the same way as the built-in types. For example, relation can reference an atomic value but cannot reference another relation.

The 1NF assumption is directly responsible for this deficiency because it does not allow nested relations. For this reason, aggregations are expressed in the form of queries which require the users to “navigate” or jump from one relation to another. The problem with this schema is that the user must possess complete knowledge of the relations in order to form the queries, a task often difficult for the general user.

Architecture

The relational model lacks appropriate mechanisms for data structuring so that data concerning a single object are not decomposed into different structures, e.g., thematic data in relational structure and geometric data in another structure, leading to the use of different database management systems for the same object, which, in turn, first makes the query optimization to the extent of the whole system impossible, and secondly, may violate the integrity constraints of the system.

It can be seen that spatial information systems exemplify the situations where rich object type and complex data items present challenges. The discovery of the shortcomings of conventional database has provided impetus for people to search out better models for improving expressive power and structural flexibility. The new data model must incorporate solutions to many of the problems outlined above in order to meet the requirements of current and newly emerging database applications.

To meet the requirements of new complex database applications, two approaches: evolutionary approach (extended relational data model) and revolutionary approach (object-oriented data model) have been suggested. The evolutionary approach is to extend the relational model with a set of fundamental OO concepts (complex objects, abstract data types, access methods, and the encapsulation of data with methods), found in most object-oriented programming languages. The database language that embodies the united

object-relational paradigm should be an extension to SQL. The database language should then be embodied in a wide variety of host programming languages. This type of OODBMS integrates well with existing relational database and provides a smooth flow of data between engineering and business applications.

The revolutionary approach is to extend object-oriented programming languages by allowing programming language objects to be persistent and sharable, that is, stored as a database, as well as permitting other database functions, such as transaction, management, and limited query facilities. The result is an object data model for which there exists no unique formal proposal but a variety of system-dependent data models.

Compared to extended relational data model, object-oriented data models, according to Won [Won, 1990], are more natural for addressing some of the deficiencies of the relational database technology previously outlined; for example, support for general data types, nested objects, and support for compute-intensive applications. The basic concepts of object-orientation will be discussed in detail in the following section.

2.4 Object-Oriented Data Models

Object-oriented approaches originated in programming languages such as Simula and Smalltalk. The application of object-oriented ideas to databases was spurred on by the apparent limitations of traditional technology when used in some of the newer applications. Typical examples are the applications of databases in computer-aided design (CAD), office information systems (OIS), software engineering and geographical information systems (GIS). A common difficulty in all of these application areas is the gulf between the richness of the knowledge structures in the application domains and the relative simplicity of the data model in which these structures can be expressed and manipulated. Object-oriented models have the facilities to express more readily the knowledge structure of the original application.

2.4.1 Basic Concepts in Object Orientation

Object orientation offers a number of new concepts and techniques, not available in conventional data models. Object orientation concepts such as object classes, functions encapsulation, inheritance and polymorphism greatly facilitate spatial modeling, including the relationships among the components, objects and tasks to be performed and conditions to be met [Blais *et al.*, 1996].

Objects

In object-oriented modeling, all conceptual entities are modeled as objects. An object has state, exhibits well-defined behavior, and has a unique identity. The state of an object is defined or described by properties or attributes, but unlike a relational structure, such properties are not restricted to non-decomposable data types and may in fact be objects themselves. The behavior of an object is defined as a set of operations.

The data and operations are encapsulated within an object. It is usually important to associate the appropriate functions with the entities to avoid logical complications in the information processing. Data encapsulation is also called data hiding because access to a given set of data structures is restricted to a list of functions explicitly specified by the designer [Blais *et al.*, 1996].

What makes objects powerful is that they can contain other objects as well. Objects that contain other objects are known as composite objects. Objects can be as simple or as complex as the application demands; more complex objects can be constructed from combinations of existing objects which can, in turn, be simple or complex objects.

Because objects can be composed of other objects, object-oriented languages can represent information in the way one naturally thinks about it. Even a complex, deeply nested structure can be treated as a single, unified object. Since that complex object can have its own behavior, other objects can use it with very little awareness of its internal complexity. This approach not only keeps things simple, it can also make complex things simple as well.

Messages

The way objects interact with each other is to send each other messages asking them to carry out their methods. A **message** is simply the name of an object followed by the name of a method the object knows how to execute. The object that initiates a message is called the sender and the object that receives the message is called the receiver.

From this point of view, an object-oriented system, then, consists of some number of objects interacting with each other by sending messages to one another. Since everything an object can do is expressed by its methods, this simple mechanism supports all possible interactions between objects.

Polymorphism is the ability to produce different responses by different objects to the same method. It is so important that it's considered one of the defining characteristics of object-oriented technology. The key benefits of polymorphism are that it makes objects more independent of each other and allows new objects to be added with minimal changes to existing objects. These benefits, in turn, lead to much similar systems that are far more capable of evolving over time to meet changing needs.

Classes

A class encapsulates the methods and variables to be included in a particular type of object. The descriptions of the methods and variables that support them are included only once, in the definition of the class. The objects that belong to a class, called instances of the class, contain only their particular values for the variables.

Inheritance is a mechanism whereby one class of objects can be defined as a special case of a more general class; automatically including the method and variable definitions of the general class. Special cases of a class are known as **subclasses** of that class; the more general class, in turn, is known as the **superclass** of its special cases. In addition to the methods and variables they inherit, subclasses may define their own methods and variables and may override any of the inherited characteristics.

Classes can be nested to any degree, and inheritance will be automatically accumulated down through all the levels. The resulting tree-like structure is known as a **class hierarchy**.

Data Abstraction

The process of creating new data types is known as data abstraction. Object-oriented technology provides extensive support for data abstraction. The technology not only allows a programmer to create new data types on the fly, it actually treats these new types as though they had been built into the language.

In a conventional data model, the entities of interest and their relationships are defined at the record or tuple level. For instance, a tuple is related to another tuple through a foreign key. These are not very useful concepts in describing geometric features. The object-oriented data model provides concepts closer to an application domain. The five basic concepts of abstraction upon which the object-oriented data model is built are: object definition, classification, generalization, association, and aggregation.

Object definition is the fundamental abstraction needed in the object-oriented data models. Each object has a type which may be referred to as object type or abstract data type. Defining objects is perhaps the most important task in object orientation data modeling, as everything else is based on this abstraction.

Classification is a form of abstraction that groups several objects, according to their structural and behavioral properties, into a common class. **Generalization** is the grouping of several classes, which share common properties and operations, into a higher order class called **superclass**. **Specialization** is the reverse of the generalization abstraction mechanism, and it specializes a class (superclass) into a **subclass**. Subclasses will have all the properties of their superclasses and can have their own properties. Inheritance could be single or multiple. Single inheritance is when a subclass is derived only from one superclass, while multiple inheritance is when a subclass is derived from more than one superclass.

Aggregation is the forming of complex objects from its component objects. Association describes the logical relationship between classes. In this case, the associated objects are referred to as members. An example of an association of objects which is not a classification of them might be: the set of all things which are red or the set of things which weigh more than one ton.

Utilizing the above techniques, complex problems in GIS can be modeled much easier and their relationships can be represented in a more natural way than can be done using conventional data models.

2.4.2 Object-oriented data model

Object-oriented data model in GIS is designed using some object-oriented methodology. It is an object-oriented DBMS with Abstract Data Types (ADT). Compared to relational data model, this approach is more complex but more flexible and extendible [Blais, 1996]. In this architecture, objects are the basic operating units which encapsulate both the spatial and non-spatial data, as well as all the functions related to the objects. Under this structure, spatial data and non-spatial data are directly handled by a general purpose DBMS. There is no question about the function of data storage, retrieval, integrity or maintenance. Developers of object-oriented GIS concentrate only on spatial data specific functions like graphics rendering, position-based query and interpretation, which are usually part of the user/development layer. While there is a lack of standards in general-purpose object-oriented information systems, which is partly responsible for the fact that standard OO query language (comparable to SQL) is not yet available; therefore making a true object-oriented implementation difficult at the moment; people have already started testing these systems for geographic applications. Two great characteristics in this model are summarized as follows:

- a) A real-world entity, no matter how complicated it is, can be represented by exactly one object; implying that no artificial decomposition into simpler parts should be necessary due to some technical restrictions.

As shown in Figure 2.2 the mapping of real world objects is 1:1 to objects in the object-oriented data model, whereas it is 1:N to conventional data models. This is to say, one object in real world corresponds directly to one object in the object-oriented model, while to represent the same real world object in conventional data models, several computer records and files may have to be used. In this way, the object-oriented data model captures the semantics of real world objects more directly in a database [Karimi and Lee, 1995].

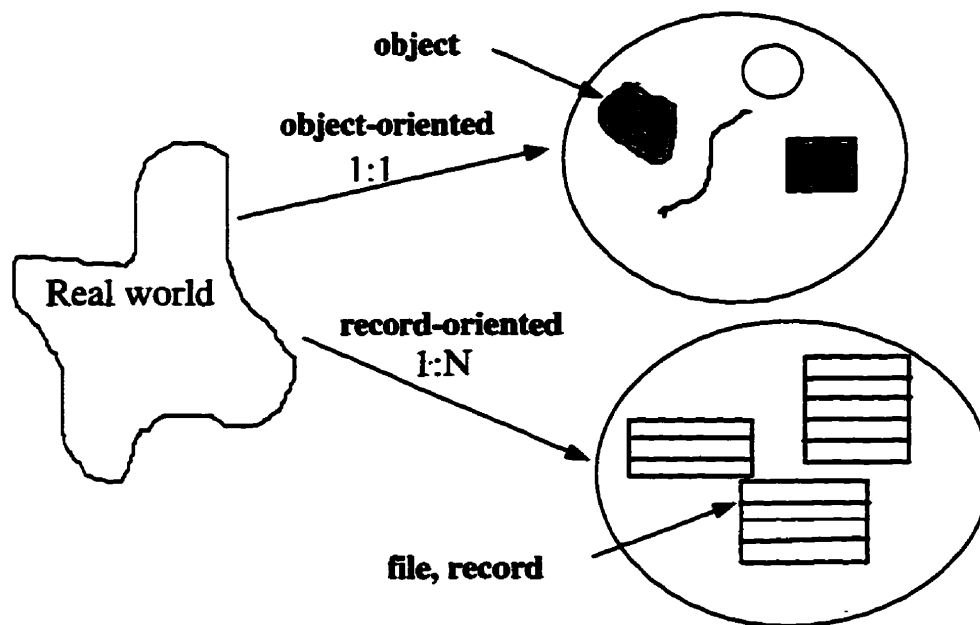


Figure 2.2: Object-oriented vs. Record-oriented data models

- b) Object-oriented data models consist of individual objects that collaborate, following well-defined patterns of interaction, to produce the desired behavior. It is analogous to a human organization, in which each member assumes certain responsibilities, and the members cooperate with each other to perform activities [Qian, 1996].

The key benefits of object technology are: a better representation of the user's real world, reuse, uniformity, and increased productivity with enhancement of software quality. These benefits can contribute significantly to the successful exploitation of GIS.

Elegant architecture

Objects can be used to describe the user's real world. This facilitates the understanding of the user requirements. The OO approach emphasizes a more intuitive or natural representation of the data, i.e. one that mirrors the user's view of their information. It helps to close the semantic gap between real world objects and concepts and their representation in the system. Through the use of composite objects, class hierarchies, and other structures, object-oriented technology can effectively represent the increasingly complex information. The fact that object-oriented structures reflect the way people naturally organize and understand the real world means that the richer structures make information more accessible, not less.

Faster development and high quality

Object-oriented technology can speed development because of the following three separate techniques: building software out of standard objects, reusing existing models of corporate processes, and replacing conventional development phases with rapid prototyping. High quality stems mostly from the fact that the programs are assembled out of existing, proven components rather than being written from scratch every time.

Better performance

Object-oriented systems generally give better performance when complex objects and complex relationships must be dealt with. This is because object-oriented technology supports the treatment of complex objects. In an object database, complex structures are represented by composite objects, that is, objects which contain other objects. These component objects can contain other objects in turn, and so on, allowing structures to be nested to any degree.

Composite objects do not literally contain other objects in the sense that one object is physically stored inside the other. Rather, composite objects contain the address of their component objects, allowing them to be accessed quickly when needed. There is no need to break up large objects for storage in normalized tables and then reassemble them at run time via slow joining operations. This is especially true when objects have complex geometry. It is claimed that the graphics performance can be 10 to 1000 times faster than its relational equivalents [Graham, 1991].

Compared with an object database, storing complex objects in a relational database is tedious at best. It is like having to disassemble your car each night rather than just putting it into the garage!

Potential concerns about object-orientation

While object-oriented technology promises many benefits, there are some valid concerns about its ability to deliver those benefits.

Even though it has been around for over twenty years, object-oriented programming is not yet stable technology. While some of the basic principles are clear - built for reusability, model real-world systems, and maximize modularity, to name three - the actual techniques and procedures for applying these principles are not well established. A body of rules comparable to structured programming will be required in order to provide guidance and discipline in object-oriented development efforts.

There is also a shortage of good tools for supporting object-oriented development efforts. These tools include programs to assist in the design of objects, manage libraries of reusable objects, and also design and maintain data-input forms and reports.

A standard OO query language comparable to SQL is not yet available, and current query languages for object-oriented systems are more complex than standard SQL, which makes true object-oriented implementation difficult at the moment. From a development point of view, C++ is definitely qualified to be a query language, but it is not appropriate for general users of the system. Object Management Group (OMG) is

targeting the establishment of Object SQL (OSQL), which is a database programming language that combines an expression-oriented procedure language with a high-level, declarative, and optimizable query language.

Other problems with object-oriented systems include concurrency, schema evolution and object integrity, etc.

2.5 Modeling Road Networks

This section describes the data models for a navigable database to support Route Guidance Systems. The basis of this technology is a spatial database that supports the applications on which RGIS are built. While there is ongoing work progressing regarding technical aspects of RGIS, there is little exploration of the most appropriate structure of the database.

This section proceeds with a brief review of the purpose and concepts of data models to provide the basis for the following general discussion of the data models, and then the object-oriented data model is introduced, and finally, the objects defined in road networks will be described in detail.

2.5.1 Description of the currently used data model

The data model is an abstraction of the real world, and it reflects decisions about what features and relationships are necessary to represent in a database. It must effectively replicate the way that users of the database conceptualize the road network. In this case, the database and data model refer to the geographic database that represents spatial entities, e.g. roads, and relationships between them, and how to relate non-spatial attribute information to the spatial objects.

The data models currently used for modeling road networks is the relational data model. In this model, the network is composed of two basic elements, links and nodes. Each link starts and ends at a node while nodes only occur at the ends of roads or road intersections. They can also represent a change in attributes between intersections. This

might occur, for example, if a road name changes along its route. Figure 2.1 illustrates the link-node structure.

One critical characteristic of links in this model is that they are completely independent of each other. Each segment of State Street does not “know” that it is a part of the same street as the link connected to it. While these implicit relationships may be derived from the data in the model, such independence makes manipulation and analysis of the entire entity State Street less efficient. In addition, it necessitates redundancy in the attribute data and requires more explicit procedures to maintain integrity of the data over the entire road length.

In summary, it is too simple, as a model, to represent road networks. This is because it is very difficult to describe the complex relationships between objects defined in the road networks without data redundancy. When the road network is very large, data redundancy is a major concern in the design of the data model. The second problem is that a lot of useful information for calculating the best route is not included in this model. That information is very important for improving the efficiency of an algorithm. It will be seen that in the object-oriented model, all the information and relationships between the objects, no matter how complex they are, can be expressed in a natural way, and therefore has a positive effect on the search time for the best route.

2.5.2 Object-oriented model for road networks

In an object-oriented system, every element of a feature can be perceived as an object. Objects must contain, embedded within the instance variables and methods, sufficient information, relations, and functionality to support spatial analysis and query.

Components of an object

In the object-oriented model, an object may contain the six elements represented in Figure 2.3 [Tang, 1992].

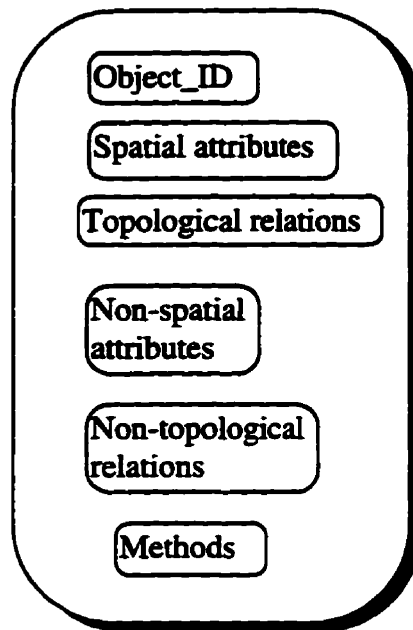


Figure 2.3: Representation of object in object-oriented data model

The inclusion of these six elements in an object can enhance the holistic representation schema for real world phenomena. The unique identifier is system-generated for each object in the database. Explicit object identifiers are not required. Object identifiers listed in tables and figures in this research are used for illustration and descriptive purposes. The positional or geometric information is usually represented by coordinates. Non-spatial attributes data is characteristic of features such as the name and number of lanes of a road. Topological relations are relations among geometric objects such as boundaries, neighbors, and interior. Non-topological relations are non-geometric links between features (Rugg, 1988) such as *is_a*, *a_kind_of*, *above* and *part_of* relations. For example, a highway is *above* a river; and a river is *part_of* the county boundary. Methods embedded in each object are used to perform actions such as creation of new object instances, queries, computation, and display.

Object types defined in road networks

Based on the analysis of the functions provided by Route Guidance Systems, six fundamental geometric objects have been defined in this data model. They are *point*, *node*, *arc*, *road*, *Master_node*, and *Master_edge*.

The data part of point is described by a single x, y coordinate and a sequence number or PointID.

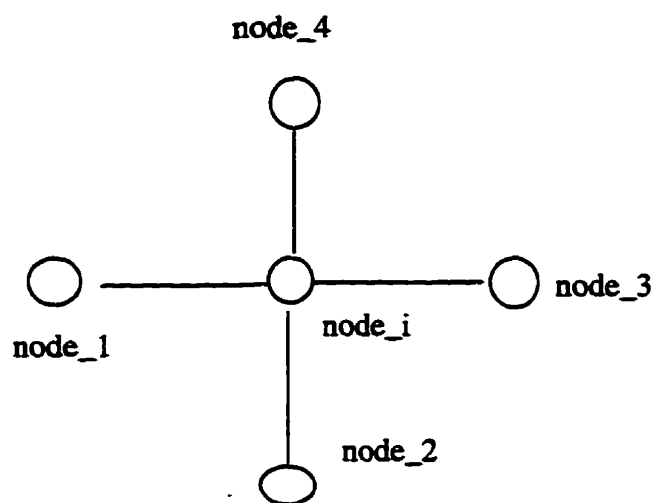


Figure 2.4: Structure of object node

Nodes indicate the endpoints and intersections of arcs which connect a sequence of segments (e.g., intersections connecting street segments). A node is an endpoint of an arc. The from-node is the first vertex in the arc, the to-node is the last vertex. Together they define the direction of the arc. A node is shared by the set of arcs which connect to each other at the node. The data part of the node object should include at least the following information: NodeID, Node_name, spatial data related to this node, relationships with other nodes (topology), the number of nodes connected to it, and their NodeIDs, classes and speed limits as well as the directionality of the road segments connected to node_i. It also should include the times taken to pass by this node from different directions; an important factor to be considered in searching out the optimal path

(minimum travel time), as well as the method part of the node object which includes all the functions that can be applied to the data. The graphic representation of the node object is drawn below.

Arcs form the basic unit from which route systems are built. An arc is a continuous string of x, y coordinate pairs beginning at one location(starting node) and ending at another(ending node).

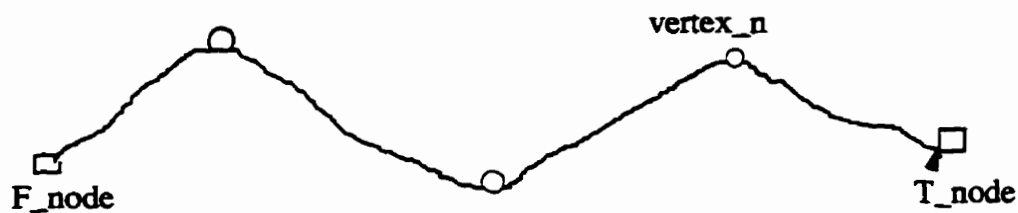


Figure 2.5: The structure of object arc

Arc-node topology defines connectivity-arcs that are connected to each other if they share a common node. This is the basis for many network tracing and path-finding operations. Object arc inherits some of the data and behaviors defined in the object node in addition to the data structure and behavior defined uniquely for itself. The arc object can be viewed as a complex object composed of objects point and object node. It is an aggregated object.

Road is a collection of arcs. It is a complex object constructed from the objects mentioned above. As an object, road has its own data structure and built-in methods.

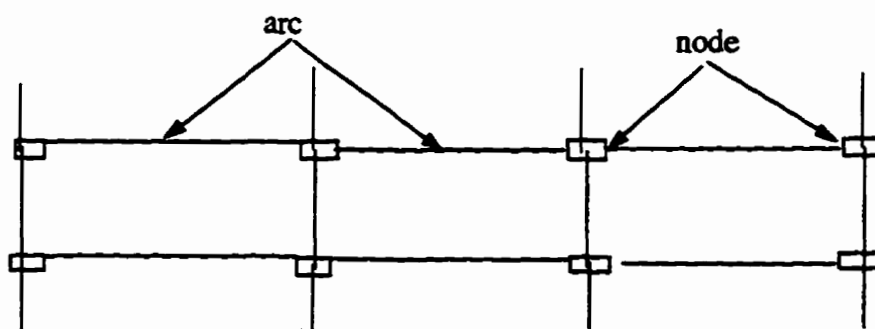


Figure 2.6: Road as a composite object of arcs

Master_node and Master_edge are objects which are collections of the roads related. They are actually the abstraction of a sub_networks. Master_nodes are connected by Master_edge. The super_network which consists of Master_nodes and Master_edges is represented graphically in Figure 2.7.

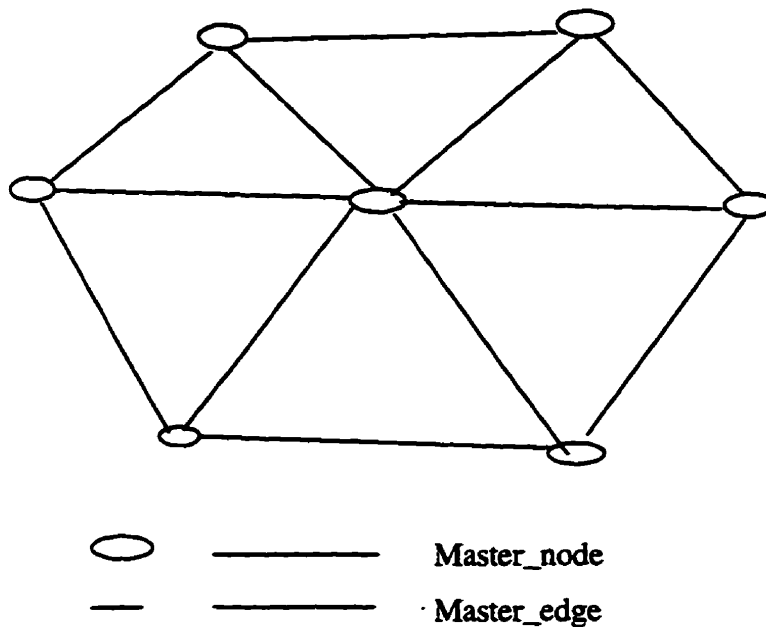


Figure 2.7: Object super_network consisting of Master_nodes and Master_edges

The definition for any class includes two parts: variables and methods, or functions. For example, an arc class includes the following variables: arcID, arcname, representing the ID, name, and number of lanes this arc has; startnodeID, endnodeID, representing the IDs of the two end nodes for this arc; arcbelongID, and arcindex_num, representing the road this arc belongs to, and the number of points included in this arc. The main functions include: Read(), Write() for reading and writing data about the arc; GetTdis(), GetTtime() for calculating the length of this arc and calculating the time taken to travel along this arc from the startnode to the endnode. The basic structure of class arc is represented in Figure 2.8.

```
class arc : public node
{
public:
```

```

int          arcID;
char         arcname[m];
int          arclanes;
int          arcarcindex_num;
int          arclist[mm];
int          arcbelongID;
int          startnodeID;
int          endnodeID;
.....

void         Display(void);
void         Read( );
void         Write( );
double      GetTdis( );
double      GetTime( );
.....
}

```

Figure 2.8: The basic structure of class arc.

Listed in the following are the basic structures for class point, node, road, master_node, and master_edge.

```

Class point
{
public:
    int          pointID;
    double       x_coor;
    double       y_coor;
    .....

    int          GetpointID(void);
    double       Getx(void);
    double       Gety(void);
    double       Getd(point * pt1);
    .....
}

```

```

Class node : public point
{
public:
    int          nodeID;

```

```

        char          nodename[m];
        double        x_coor;
        double        y_coor;
        int            index_num;
        int            index_list[n];
        int            arcclassID[n];
        int            Mtime[mm][nn];
        .....

        void          Read(void);
        void          Write(void);
        double        Getd( node * pt1);
        double        Getnpd( point * pt2);
        .....
    }

class road : public node
{
public:
    int            roadID;
    char          roadname[m];
    int            roadindex_num;
    int            nodelistID[n];
    int            arcindex_num;
    int            arclistID[mm];
    .....

    void          Display(void);
    void          Read(void);
    void          Write(void);
    double        Road_k( node * pt1, node * pt2);
    double        Road_alfa( node * pt1, node * pt2, node * pt3, node * pt4);
    double        Road_pd( node * pt1, node * pt2, node * pt3, node * pt4);
    double        Road_od( node * pt1, node * pt2, node * pt3, node * pt4);
    node *        Searchnode(node * ptr);
    double        Roadparameter(node * pt1, node * pt2);
    .....
}

```

```

class M_node : public road
{
    public:

```

```

int          M_nodeID;
char         M_nodename[m];
int          M_nodeindex;
int          M_nodelistID;
int          M_exitnodeindex;
int          M_exitodelist[mm];
int          M_roadindexnum;
int          M_roadindexlistID[n];
.....

void         Display(void);
void         Read(void);
void         Write(void);
void         D*algorithm(node * pt1, node * pt2);
void         A*algorithm(node * pt1, node * pt2);
void         BA*algorithm(node * pt1, node * pt2);
void         R*algorithm(node * pt1, node * pt2);
.....
}

```

class M_edge : public road

```

{
    public:
        int          M_edgeID;
        char         M_edgename[m];
        int          M_edgeindex;
        int          M_edgelistID[mm];
        int          M_exitnodeindex;
        int          M_startnodeID;
        int          M_endnodeID;
        .....

        void         Display(void);
        void         Read(void);
        void         Write(void);
        void         D*algorithm(node * pt1, node * pt2);
        void         A*algorithm(node * pt1, node * pt2);
        void         BA*algorithm(node * pt1, node * pt2);
        void         R*algorithm(node * pt1, node * pt2);
        .....
}

```

All the objects defined above communicate with each other to perform activities according to the patterns defined within the objects. From the description above it can be seen that any entity in the real world, no matter how complex it is, can be described by just one object in the object-oriented data model, which captures the semantics of the complex objects and also simplifies the query process of the complex objects. This is because any information related to the complex object can be obtained without carrying out the reassembling procedure which is a necessary step in the relational data model when complex object query problems have to be dealt with.

Hierarchy and relations of fundamental geometric objects

The hierarchy and representation of the primitive geometric vector objects in an object-oriented data model are shown in Figure 2.9.

A *Master_node*, which is a complex object, consists of a set of roads. A *road* is a sequence of non-intersecting arcs. Each contains a begin node, an end node, and a series of points. The begin_node or end_node is a *node*. A *node* is a topological junction specifying the geometric location of one or more arcs. In this model, a *node* has a node identifier and inherits the properties and methods of a *point*, where a *point* is a zero-dimensional object that specifies the geometric location and unique identifier within the map. This facilitates the construction of topological relations between objects. A *point* has a point identifier and stores the x and y coordinates of the point. The z coordinate of an object is optional depending on the type of object and application.

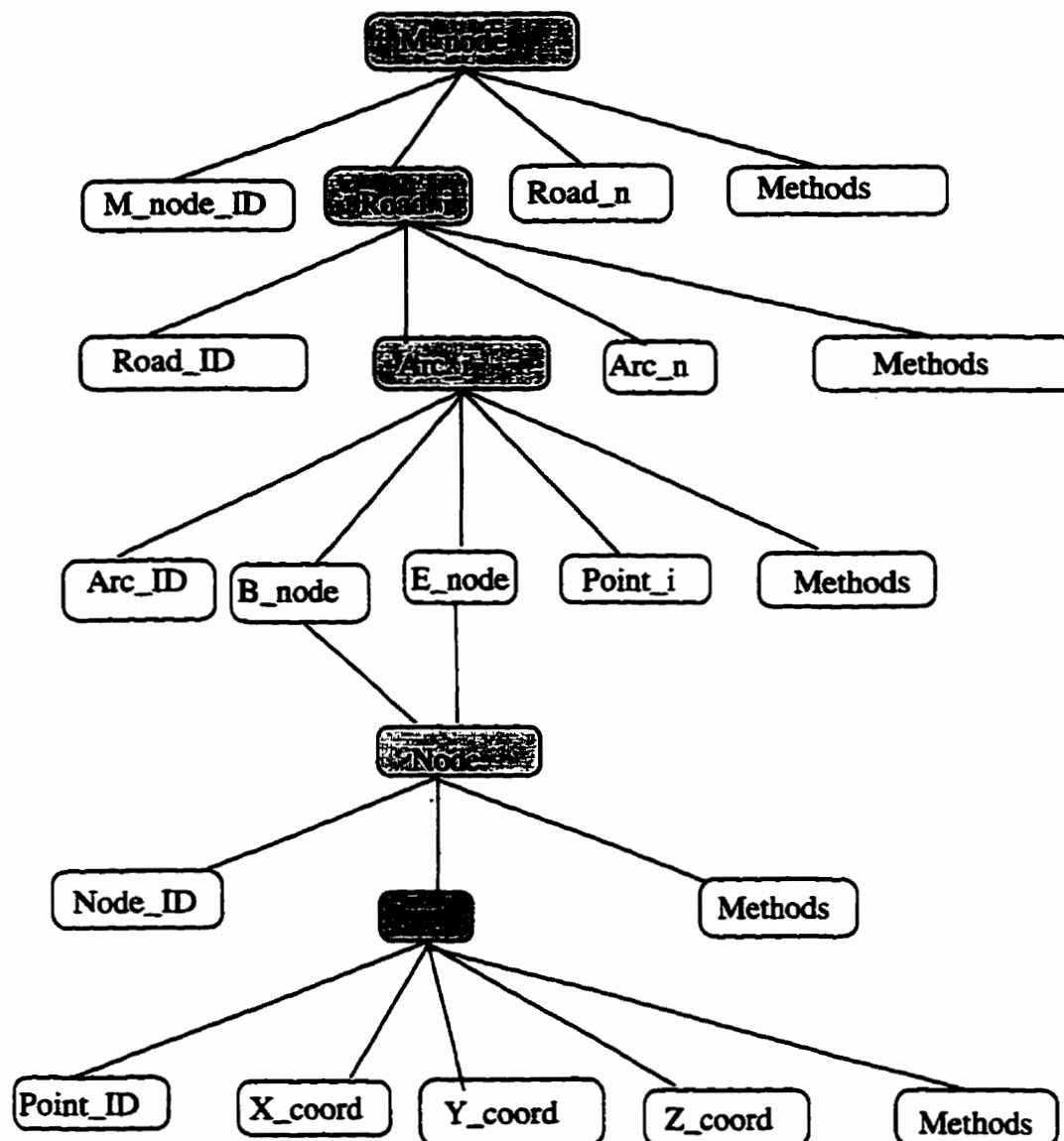


Figure 2.9: Hierarchy of fundamental objects in networks

Compared to the currently used data models, the object-oriented approach has positive advantages by providing a model for real world feature representation. Apart from the topological relations among geometric elements, *an object in the OO model also includes non-topological and semantic relations among objects which are missing in the traditional GIS model.* This model enhances the representation of features in a holistic

way since each object can describe the total information about a location and the relations with other features for a specific application. This enhancement can ensure a more complete digital representation and spatial description of geographical phenomena. More meanings of geographical features can be captured in this sort of database because the model allows incorporation of complexities of spatial data and relations in the database. With the use of an object-oriented approach, the design of user-defined types and encapsulation of data and functions within the object make the digital representation of complex spatial features possible and practical.

CHAPTER 3

OPTIMAL PATH SEARCHING ALGORITHMS

In recent years there has been a great deal of interest in minimum path algorithms. This is especially true in the field of in-vehicle route guidance systems (RGS) where there is a definite necessity to calculate the minimum path route from an origin to a destination quickly and accurately.

Due to the time constraints inherent in in-vehicle RGS, the minimum path algorithms used tend to be of a heuristic nature, having no guarantee of finding the optimal solution. The objective of this chapter is to illustrate the principles behind proposed RGS minimum path heuristics.

3.1 Introduction

The implementation of route guidance systems (RGS) has renewed interest in algorithms that identify minimum paths. One of the key components of RGS operational tests is the ability to calculate minimum path routes from an origin to a destination in an efficient and timely manner. Over the past thirty years, a number of researchers have studied minimum path algorithms for applications in such diverse fields as transportation models and circuit board design (Pohl, 1971, Van Vuren, 1988, Rilett, 1994]. Although all minimum path algorithms have the same basic structure, the wide variety of objectives and constraints for each application has resulted in a number of different algorithms and heuristics for solving the minimum path problem.

The issue faced by RGS developers is how to identify the best route from one origin to one destination given the underlying constraints of the traffic network. The definition of "best" is subject to interpretation, and this chapter addresses problems in

which travel links are defined exclusive of road type. As evidenced by current RGS field tests in North America, Europe and Japan, a number of researchers have recently investigated the theoretical and application issues involved in implementing computer algorithms in solving the above problem. However there has not been a complementary increase in the published research on the techniques used. Though there have been some papers discussing empirical results, no corresponding information has been provided on the algorithms themselves [Kuznetsov, 1992].

This chapter discusses the standard algorithms that are applicable to RGS and illustrates some of the trade-offs faced by the RGS path selection developers. The first section of this chapter examines the structure of the standard minimum path algorithm, that is, Dijkstra's algorithm, while the second part of this chapter examines the so-called heuristics-informed searching algorithms; considered the most suitable for RGS applications. This is followed by an introduction to a new algorithm - Road-base searching. Examples will also be given to show the features of different algorithms.

3.2 Dijkstra's algorithm (D* algorithm)

The traffic network through which minimum paths are to be found is defined by a directed graph $G(N, A)$ that consists of a set of nodes N (n elements) and a set of arcs A (m elements). Each directed arc connects a node i with a node j and has an associated cost C_{ij} . For traffic networks this arc cost is always positive and finite. In this research it will be assumed that arc (or link) cost refers to the arc travel time, although any generalized cost may be used.

Dijkstra's algorithm assumes that the graph is connected; that is, each origin-destination (O-D) pair has at least one directed path connecting the nodes, and this path is of finite length.

The main decision variables are the arcs that make up the minimum path for a particular O-D pair and the associated minimum route travel cost. The route cost from the origin to a particular node n is defined as $L(n)$ and this route cost is referred to as the

“label” of a particular node. The inputs to the algorithm include, among other attributes, the directed graph and the positive, finite travel time cost for each arc.

Dijkstra’s algorithms can be used to find the minimum distance from the origin node to all the nodes of the network. If there is a need to find the optimal path, one node u to another node v , the best way is to run Dijkstra’s algorithm with u as the origin node and stop when we deduce the distance to v .

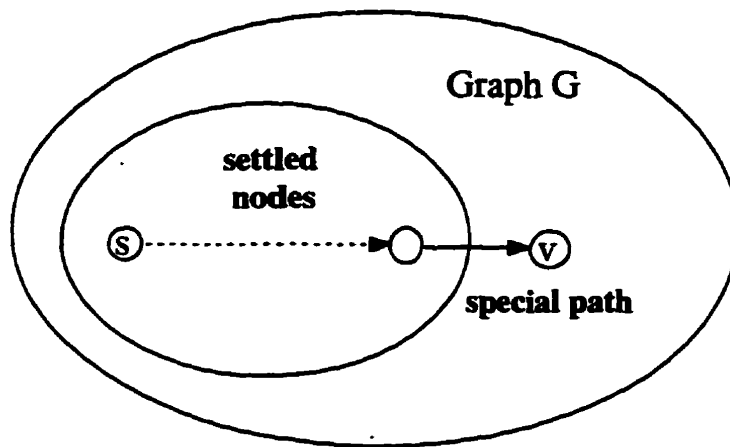


Figure 3.1: Intermediate stage during the execution of Dijkstra’s algorithm.

The essence of Dijkstra’s algorithm is that the minimum distance from the source to other nodes in the order of minimum distance, that is, closest nodes are discovered first. As Dijkstra’s algorithm proceeds, a situation like what is demonstrated in Figure 3.1 happens. In the graph G there are certain nodes that are settled, that is, their minimum distance is known; this set always includes s , the source node. For the unsettled nodes v , the length of the shortest special path is recorded, which is a path that starts at the source node, travels only through settled nodes, then at the last step jumps out of the settled region to v .

A value $L(u)$ for every node u is maintained. If u is a settled node, then $L(u)$ is the length of the shortest path from the source to u . If u is not settled, then $L(u)$ is the length

of the shortest special path from the source to u . Initially, only the source node s is settled, and $L(o) = 0$, since the path consisting of s alone surely has distance 0. If there is an arc from s to u , then $L(u)$ is the label of that arc. Notice that when s only is settled, the only special paths are the arcs out of s , so that $L(u)$ should be the label of the arc $o \rightarrow u$ if there is one. INFTY serves as an “infinite” value and indicates that no special paths have yet been discovered, that is, initially, if there is no arc $s \rightarrow u$, then $L(u) = \text{INFTY}$.

Now let us suppose we have some settled and some unsettled nodes, as suggested by Fig. 3.1. We find the node v that is unsettled, but has the smallest $L(v)$ value of any unsettled node. v will be settled by:

- accepting $L(v)$ as the minimum distance from s to v .
- adjusting the value of $L(u)$, for all nodes u that remain unsettled, to account for the fact that v is now settled.

The adjustment required by step(2) is done as follows. The old value of $L(u)$ with the sum of $L(v)$ and the label of the arc $v \rightarrow u$ will be compared, and if the latter sum is smaller, we replace $L(u)$ by that sum. If there is no arc $v \rightarrow u$, then we do not adjust $L(u)$. For the convenience of programming, listed below are the details for each step in finding the shortest path by using this algorithm. The algorithm proceeds in the following manner:

- **STEP 1:** Set $L_o = 0$, $L_n = \text{infinity}$ (for all nodes rather than the origin)
- **STEP 2:** Place the nodes that have been examined (finite labels) in ascending order.
- **STEP 3:** Select the node with the lowest label (travel time). This is node i . Note that by definition there cannot be a faster route to get to node i from the origin node o . Set the node i .
- **STEP 4:** Identify the arcs emanating from node i identified in step 3.
- **STEP 5:** For each arc ij identified in step 4, calculate the cost of traveling to node j from node i .

$$L_j' = L_i + C_{ij}$$

This algorithm was run on a simulated network which consisted of 75 links and 56 nodes . As seen in Figure 3.2, the origin is located at node 12 and the destination is located at node 32. The dark nodes represent the nodes where their minimum path from the origin to them have been found before the minimum path to node 32 was found, and the red line connecting the origin and the destinations represents the optimal route. There are a number of points worth noting about the algorithm.

The algorithm, although guaranteed to identify the optimal route, is somewhat inefficient in that the minimum paths to many nodes are calculated which are not on the minimum path from node 12 to node 32.

The logic of the node selection step in which the node with the lowest estimated travel time is chosen results in the search fanning out equally from the origin in all directions. Techniques to increase the efficiency of the algorithm and its speed will be the focus of the next section.

3.3 Heuristically-informed searching

The application of heuristics implies using additional knowledge which will aid in solving a problem. The goal in many instances is to speed up the time it takes to arrive at a solution as compared to using strict algorithmic approaches. This of course is of paramount importance, considering the patience levels of most potential users.

3.3.1 A* algorithm

The Dijkstra's algorithm is inefficient when operating in a one-to-one mode, due to having to search outwards in all directions from the origin. This inefficiency was recognized very early by a number of researchers who developed techniques to help constrain the search.[Hart, Nilsson and Raphael, 1968, Nicholson, 1966]. The general strategy was to change the order in which nodes were examined such that the nodes that had a higher "likelihood" of being on the minimum path were given priority over those with a lower "likelihood". The algorithm developers attempted to quantify the likelihood

of a particular node being on the minimum path by “not only considering how far the node was from the origin but also its direction from the origin”.

As an example of this last point, consider the previous problem of finding the best path from node 12 to node 32. Intuitively, a node that is 3 km east of node 12 should be examined before a node that is 2.5 km west of node 12. The challenge is to develop a method whereby this likelihood is automatically reflected in the labels that the nodes are assigned during the minimum path search.

The A* algorithms have basically the same structures as the D* algorithm. The difference lies in step 5 where the label for a particular node is calculated. The modified step is illustrated below.

STEP 5: For each ij identified in step 4, calculate the estimated cost of traveling to the destination node k from node i using node j .

$$L_j' = L_i + C_{ij} + e(j,k)$$

If $L_j' < L_j$ then place arc ij on the minimum path to node j and update the label of node j ($L_j = L_j'$).

where $e(j, k)$ = the estimated cost of travel from node j to destination node k .

These A* algorithms require a good estimate of the cost from a particular node j to the destination node k . This projected cost is then combined with the calculated cost of traveling from the origin node i to node j . The result is an overall estimate of the minimum travel time from the origin node i to the destination node k using the intermediate node j . Intuitively, nodes that lie between the origin and destination would then be examined before any nodes that do not. The general result is that the search area becomes elliptical in shape rather than the circular shape associated with the D* algorithm.

There are a number of ways to estimate the travel cost in step 5 above. It can be shown that if the estimated cost of travel calculated within the A* algorithm is a lower

bound (i.e. it always equals or underestimates the true cost of travel), then the A* algorithm will always find the minimum path.





The logic of the A* algorithm is best demonstrated with respect to Euclidean networks (those where the arc lengths are equal to the Euclidean distance between the two nodes) and where the objective is minimum travel distance. As discussed above, the label of a node calculated in step 5 is comprised of the calculated distance from the origin to the end node and an estimated distance from the node to the destination node. In the A* algorithm the estimated distance from node j to the destination node k is set to the Euclidean distance from node j to the destination node k as shown in equation [1] below.

$$E(j,k) = \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2} \quad \dots\dots\dots (1)$$

where: $E(i, j)$ is the cost of travel from node j to destination node k (the Euclidean distance), x_j, x_k are the x coordinates of nodes j and k ; y_j, y_k are the y coordinates of nodes j and k , respectively.

Note that by definition $e(j, k)$ is a lower bound (there will not be a path from node j to the destination node k that can be shorter) and therefore the algorithm is guaranteed to converge.

Objective: Identify minimum path from node A to node B
Given: Link distance, minimum path from A to C and from A to D

Node	D	A	C	B
				
Distance	0	10	22	40
Label(for D*)	10*	0	12	infinity
Label (for A*)	50	0	30*	infinity

* Next node to examine

Figure 3.3: A* example network

Figure 3.3 demonstrates this step graphically. The objective of the search is to find the minimum path from node A to node B. It may be seen that the label of node D has been identified as 10 km (distance from A to D) and the label of node C (distance from A to C) has been identified as 12 km. In the standard D* algorithm the label of nodes C and D would be 12 and 10 km, respectively. Node D would be therefore be examined next because it has the smaller label.

However, in the A* algorithm the label of node D would be 50 km, which is calculated by adding the distance of 10 km from node A to node C and the estimated distance of 40 km from node D to node B. The label of node C would be 30 km (12 + 18). In the A* algorithm node C would be examined next even though it is farther from the origin than node D. Selecting node C over node D intuitively makes sense because node C would appear to have a higher likelihood than D of being on the minimum path from A to B. The end result is that the algorithm tends to search outward from the origin in a more elliptical, rather than circular manner, where the major axis of the ellipse is the straight line between A and B.

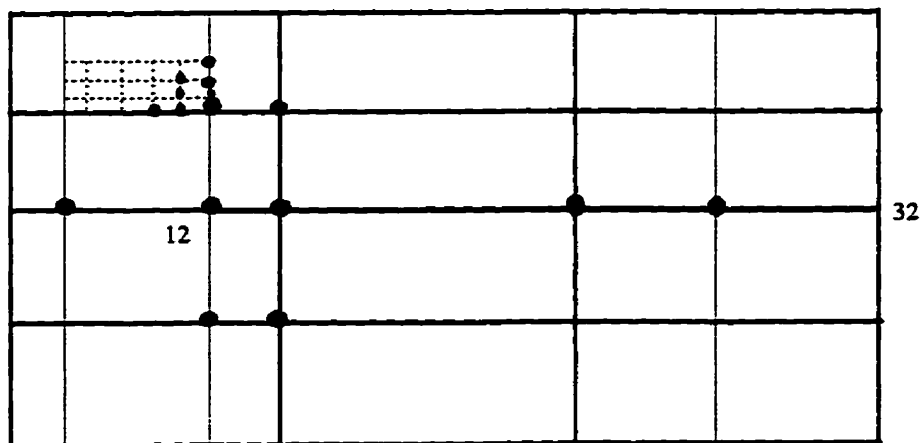


Figure 3.4: Example of A* algorithm

Figure 3.4 represents the results of the A* algorithm for the same O-D pair (origin 12 to destination 32) in the simulated road network that was shown in Figure 3.2. It may

equally between both searches would be the simplest method, it is not the most efficient. The best strategy involves identifying the minimum path search that has the fewest nodes which have been examined but has not had the minimum path identified. That is to say, the computational effort is concentrated on the search having the least nodes to examine and sort during each iteration. Intuitively, the search that is in a sparse area of a network will get priority.

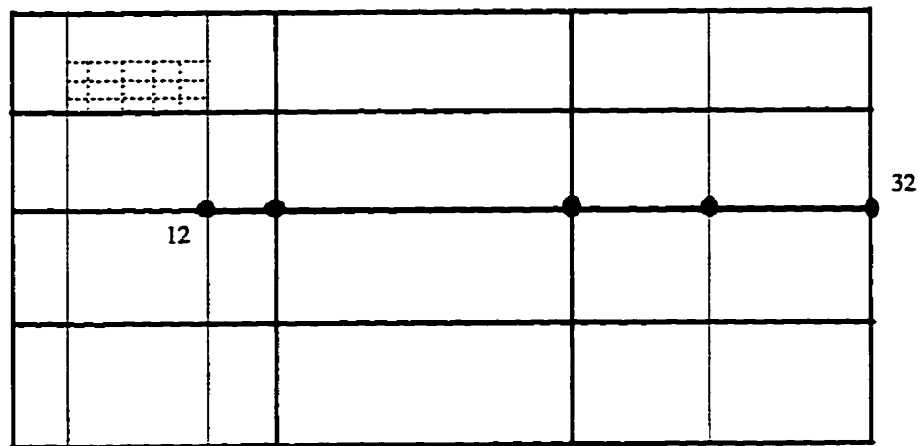


Figure 3.5: Example of BA* algorithm

Figure 3.5 represents the results of the Bi-directional A* algorithm for the same O-D pair (origin 12 to destination 32) in the simulated road network that was shown in Figure 3.1. It may be seen in Figure 3.5 that the number of nodes involved in the searching process is greatly reduced.

3.3.3 Searching within a window

This method is conceived by imagining how humans might attempt to solve a particular problem based on the common sense. A good example related to route determination is utilizing the additional information of how the crow flies (i.e. Euclidean distance) between s and t . Generally, when a human is given the task to identify the best route between two points on a road map, one of the first steps performed is to look at

roads which lie between s and t , neglecting all roads which seem to take you further away from the destination.

Dijkstra's algorithm, as described above, creates a tree which branches out, iteration by iteration, growing each time, branch by branch, extending to the next closest node to s and then the next closest and so on. As described above, it is strictly algorithmic. In the context of graph search methodology, it is considered as a breadth first searching procedure. In a directive sense, if one could narrow the search down to a particular corridor (buffer) of possibilities, which has a high likelihood of containing the optimal route, then the solution could be arrived at much more swiftly.

The idea of searching within a window was proposed in 1991 by Karimi [1991]. This method when applied will first examine the number of nodes in the road network. If the number is less than, say 3000 (the time taken for searching the best route by using Dijkstra's algorithm, in this case less than 45 seconds, which is thought to be acceptable), Dijkstra's algorithm will be applied to the whole network; otherwise, a window will be defined according to the distance and the direction of the two end points. Then, the D* algorithm will be applied in the defined window.

The steps used to calculate the optimal path in window-based searching are as follows:

1. Get n the total number of nodes
2. If $n < H_value$
 - {
 - Call D* algorithm; exit
 - }
- else
 - {
 - Compute the straight line distance between the two given nodes (d);
 - Create a rectangular window (H_Window) with: width = $0.5d$ and length = $1.5d$ (the straight line between the two nodes lie on a line joining the mid points of the widths).
 - }
3. Extract the sub_network nodes within the rectangle.
4. Create an an adjacency list of the sub_network.

5. Call D* algorithm.

Window_based searching starts with comparing the size of the underlying network with the H_value . (The H_value is a predetermined value of the number of nodes. The network sizes below this number result in acceptable running times.) If n is smaller than the H_value , the D* algorithm will be invoked. Otherwise, the H_window will be defined according to the two given nodes.

The next step is to extract the sub_network within the H_window and its adjacency list. In the extraction process, only the links which entirely lie within the H_window will be considered. This method ensures that a smaller network is used in real-time computation. The size of the H_window should be fixed in such a way that the total number of intersection nodes inside the window does not exceed the H_value .

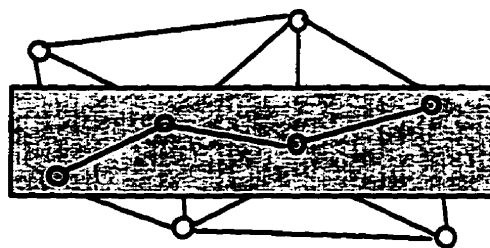


Figure 3.6: Searching within a window.

In window_based searching, the number of nodes involved in the searching process can be reduced by ignoring the nodes outside the defined window, and the searching process is executed within the window. Window_based searching, like other heuristic algorithms, cannot guarantee the best solution but it ensures a good solution within an acceptable computation time [Karimi, 1991].

When using this approach, three problems should be considered and solved. The first one is how to ensure that the total number of intersection nodes inside the H_window does not exceed the H_value . The number of nodes within the H_window depends on two factors: the distance between the two given nodes and the density of the roads in the road

network. It is very difficult to control the number of nodes covered by the H_window if no further measure is taken. Another concern is the relative error of the solutions of this approach. The relative error of the solutions depends on whether the fixed H_window size of $0.5d \times 1.5d$ contains an adequate subnetwork. Generally, the larger the window, the better the solutions.

Finally, when this method is applied to the network each time, a temporal database needs to be generated, so the total time taken to get the solution is the sum of the time to generate the temporal database and the time to search for the best route. For larger and more complex road networks, the process to generate the temporal database which includes extracting the subnetwork and creating the adjacency list, will take a considerable time.

Basically the principle of window-based searching is the same as that of D^* algorithms. The only difference is in the searching area. The searching area in window-based searching is a rectangular window - a subnetwork of the original road network.

3.3.4 Hierarchical structure-based searching

The hierarchical structure is built by selecting a set of connected edges in the graph, and then having them form a connected sub-graph representing the next higher level [Car and Frank, 1993(a) and 1993(b)]. This process can be repeatedly applied to form a multi-level hierarchy. The selection is based on the classification of the streets according to levels like interstate highways, freeways, and local roads, etc.

Let us suppose the objective of the search is to find the optimal path between start node **a** and end node **b**; the searching process is as follows:

First, the road network is organized into different levels. Adjacent levels have nodes in common. These nodes are exits and entrances to the higher or lower level. The first level is the subnetwork composed of the highways represented in Figure 3.7 by the segments in bold lines, and the second level is the subnetwork composed of state

highways and freeways represented by solid segments, while the third level is the network itself, which means that all the segments in this graph are included.

To start with, determine the level k . This is the highest level; a and b are in ($k = \min(a, b)$). Find the nearest node of node a and b in this level, which is a and c . Find path $a \rightarrow c$ by Dijkstra's algorithm, and apply the D^* to a lower level network to find the optimal path from node b to node c . Finally, the best route from a to b can be obtained, which is the combination of the optimal paths from $a \rightarrow c$ and $c \rightarrow b$.

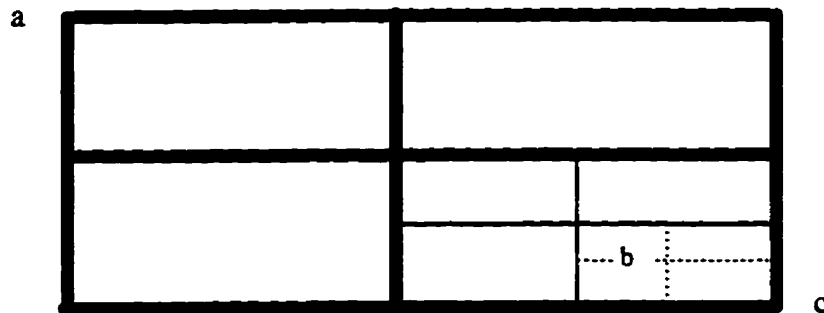


Figure 3.7. Searching using “Hierarchical Structure”.

To search for the optimal path between the two given nodes using this approach, D^* algorithm will be applied to different road networks. The searching time for the optimal paths will be reduced by using the “hierarchical structure” approach because of the classification of the road networks. The larger the networks, the more advantageous this approach will be shown. The negative side of this approach is that different databases corresponding to different kinds of road levels should be established, which is definitely not an appropriate way from the point of view of saving storage space, especially when road networks are quite large.

3.4 Road-based searching

A new approach, named road-based searching, has been proposed recently in this research. This approach is based on the object-oriented data model of the network and

knowledge-based searching. In this model, several kinds of classes (objects) which include point, node, segment, road, Master_node, and Master_edge for modeling the road network have been defined. The major difference between this method and the methods mentioned above is that in the searching process much more knowledge is used, which is something badly needed for improving the efficiency of any kind of route searching algorithms. The information which is used in the searching process includes, but is not limited to, spatial related information such as position, direction, non-spatial information of the roads (such as speed limits, number of lanes, classes of the roads) as well as all kinds of relationships among nodes, segments and roads. The largest unit considered when moving the search from the current node to the next node is “a road” or “part of a road” instead of “segment”, which is the largest unit in segment-based searching. It is the first time a road is treated as a whole unit in the optimal path searching process. The searching steps are summarized as follows:

1. **Road selecting:** The algorithm will first examine the positions of the given nodes (relatively origin node and destination node), calculating the distance between the two nodes and its direction. Secondly, it examines some of the roads in order to locate a road (or part of a road) which best fits the distance and the direction of the given nodes. The selecting process is based on the properties of the roads and the information of the given end nodes.

Some of the factors that should be considered or the rules that should be followed in the process of road selecting include:

Direction of the road: The smaller the difference of the directions between the selected road and the straight line formed by linking the two given nodes, the more priority the road will be given.

Average speed of the road: The larger the speed, the more priority will be given to the road.

Distance covered by the road: If two roads have the same direction and average speed, the one with a longer projection on the line formed by the origin to the destination node will be given more priority.

Straightness of the road: This value could be the ratio of the actual length of the road to the Euclidean distance between the two selected points (which are closest to the origin and destination node respectively) on the road.

Average distance between the selected road and the line formed by the two given nodes.

Figure 3.8 gives an example explaining why road No.2 is chosen instead of No. 1 if one road must be selected from the given two roads as a part of the optimal path from node A to node B.

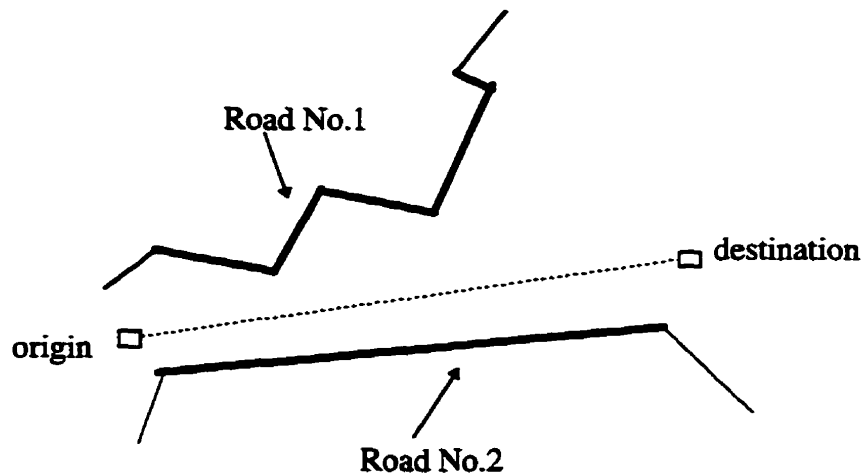


Figure 3.8: Some of the factors considered in the process of road selecting

2. **Bridging the gaps:** After the first step, a number of roads (or a number of parts of roads) are selected. All these roads are not required to be connected to each other, which means that a gap between neighboring roads is permitted. The gaps between

the roads are bridged by the optimal paths obtained by applying D* algorithm or A* algorithm to the road network.

3. **Combination:** The resulting route is the combination of the selected roads and the optimal paths connecting them.

When this algorithm is used to find the optimal route between two given points, the “intelligence” and “better space reasoning” of this searching method will help to get the right solution in much less time.

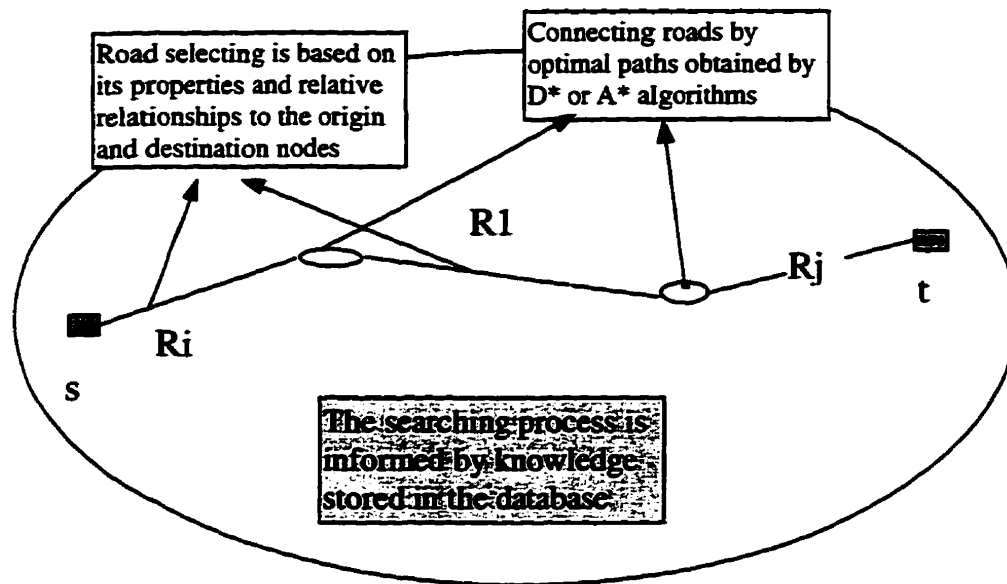


Figure 3.9. Road-based searching

An example is given below just to illustrate the efficiency of this approach. To find the optimal path between node 0, and node 34, 42 iteration times are required by using A* algorithm while only two iteration times are needed by using the road-based searching method.

A number of examples will be given to show the features of the different algorithms mentioned above and the results of the selected “optimal paths” are also

compared with each other to demonstrate the relationship between accuracy and the searching time.

CHAPTER 4

IMPLEMENTATION AND RESULT ANALYSIS

The purpose of this section is to summarize the basic features for each algorithm, and analyze their advantages and disadvantages. First, the data structure used in the existing RGS systems to store the road network data will be illustrated, and then the features of different algorithms will be summarized. The results of representative algorithms for optimal path searching will be given and compared.

4.1 Data structure used in the existing road networks

Right now, the data structures used to store the road network data are rather simple. Under this structure, only node and segment-related data are stored in the database. Data used to calculate the optimal path include spatial and non-spatial data such as the coordinates of the node, the ID and type of road the segments belonging to, as well as the topology of the nodes. Some data such as turn impediments and average speed of the road, which are related to the calculation of the optimal paths - paths with the minimum travel time are not included in this data structure. In other words, this kind of data structure can only be used to calculate the optimal path - path with the minimum distance.

The data structure used in the existing RGS system to store all the Calgary road network data is illustrated in Figure 4.1. In this structure, feature head, feature segment, and feature node are singly linked list structures with the following declarations [Karimi, 1991]:

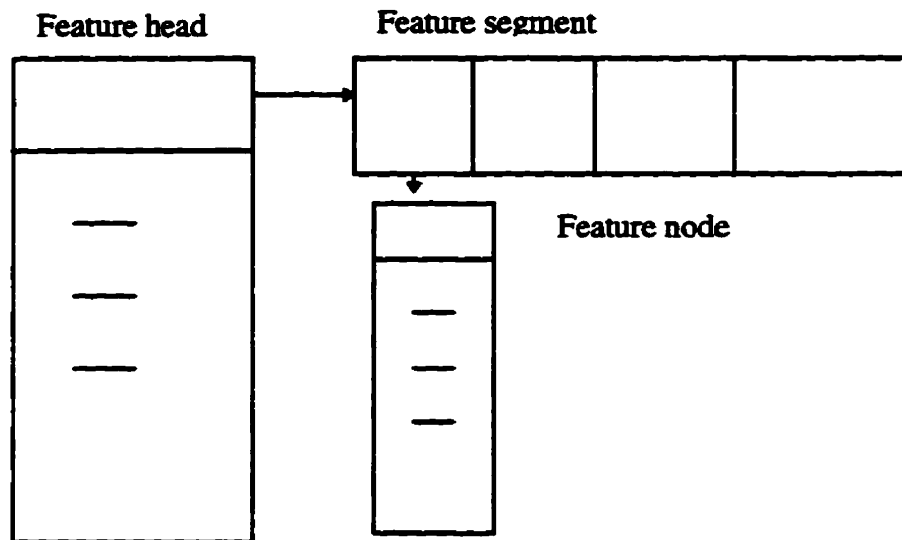


Figure 4.1: The data structure for the storage of the road network RGS system

```

structure Feature_Segment
{
    int          Feature_code;
    char         Feature_type;
    char         Sub_feature_type;
    char         Name[m];
    char         Street_type[n];
    char         Feature_direction[n];
    Feature_Segment * Next_Segment;
}

```

```

structure Feature_node
{
    int          Feature_code;
    int          Sequence_number;
    int          Section_number;
    int          Node_number;
    char         Node_type[m];
    int          X_UTM;
    int          Y_UTM;
    int          Address_before_left;
    int          Address_before_right;
}

```

```

        int          Address_after_left;
        int          Address_after_right;
        int          Cross_reference_feature_code;
        int          Cross_reference_sequence_number;
        Feature_node *Next_node;
    }

    structure Feature_head
    {
        Feature_Segment    * Next_Segment;
        Feature_node       * Next_node;
    }

```

It is clearly seen from this structure that data related to the turn impediment are not included. Actually, the type of intersection and the number of intersections of the road are some of the major factors that should be considered in the shortest path (path with the minimum travel time) searching.

Considering that the objective of this research is to search for the optimal path - path with minimum travel time rather than the path with minimum distance, more information such as turn impediment, average speed and the number of lanes of each road should be added to the database. Two simulated road networks are used here to demonstrate the features of different optimal path searching algorithms.

4.2 Searching space in D* algorithm

As mentioned before, the D* algorithm can be used to find the optimal paths from the origin node to any other nodes of the network. This section will point out the major features of this algorithm by summarizing the results obtained in this research.

Let us first review the general structure followed by the standard D* minimum path algorithms.

- **STEP1 - initialization**

Set the label of the origin node to 0. The label of all the other nodes is set to infinity.

$L_n = \text{infinity}; L_0 = 0$

- **STEP2 - decision logic**

Choose a node i . For all arcs ij emanating from node i :

Check arc ij , if

$L_i + c_{ij} < L_j$ then $L_j = L_i + c_{ij}$.

Step 2 examines all of the arcs emanating from node i . If using these arcs allows the vehicle to travel to any node j faster than the current “best” route to node j , then arc ij is placed on the minimum path from the origin node o to node j and the label of node j is updated.

- **STEP 3 - stopping criterion**

Repeat step 2 until no arcs satisfy the inequality.

The major variation between the different algorithms pertains to the manner in which the nodes in step 2 are identified and selected for examination. It is this step selection process that gives rise to the different algorithms and their respective attributes.

The basic premise of this algorithm is that during each iteration the “set” section of the minimum path tree expands by one node. The reliance on selecting the node with the lowest label is the reason that this algorithm is sometimes referred to as **the shortest first algorithm**.

There are two important points to be noted regarding the operation of this algorithm. The first point is that if only a particular minimum path route between an origin and a single destination is required, the algorithm can be stopped as soon as the label of that destination node is set. This type of operation is often referred to as **one-to-one mode**. Consequently, this algorithm may be used in RGS where the objective is to find a minimum path from a given origin to a single destination for an individual vehicle.

Figure 4.3 shows the searching area and the nodes examined before the optimal path (which is defined as the path with the minimum travel time represented by the red line) from node 12 to node 22 was found. All the nodes with the travel time of the optimal path less than the travel time of the optimal path from node 12 to node 22 are searched before the solution has been found.

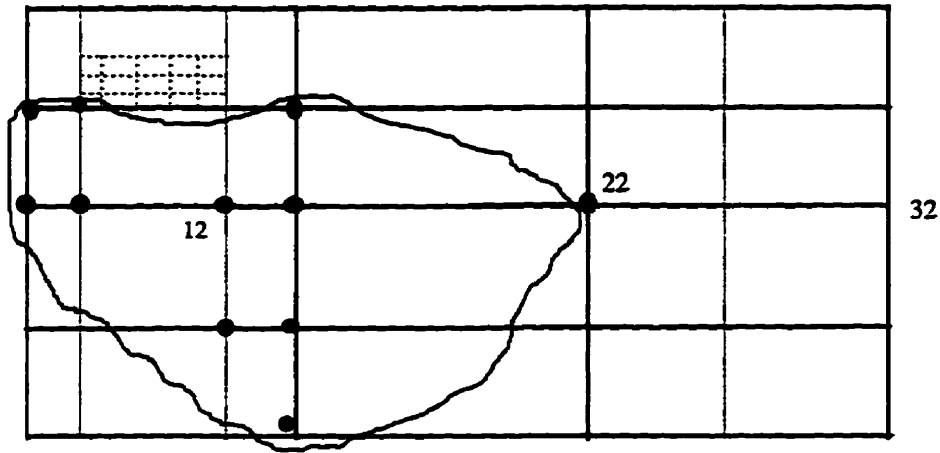


Figure 4.3: Example of D* algorithm

A lot of computation has demonstrated that when this algorithm is applied to a road network for optimal path searching, the searching area is “a circle” centered at the origin node with the radius (L) equal to the cost of the optimal path from the source node to the destination node, which means that all the optimal paths of the nodes with the cost less than L will be calculated.

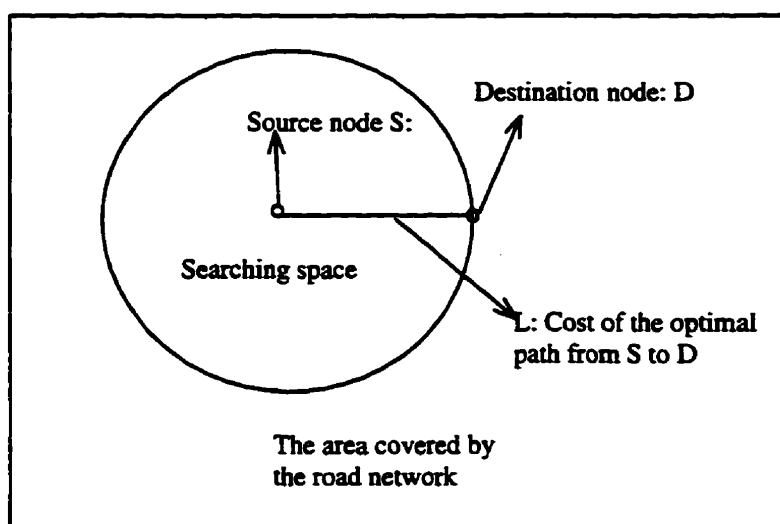


Figure 4.4: Searching space in D* algorithm

Figure 4.4 demonstrates the feature of this algorithm graphically. It shows the relationship between the searching area and the area covered by the road network. One point that should be noted is that the cost of the optimal path could be the minimum distance, minimum travel time or whatever cost defined by the user.

If the bi-directional searching method is applied to the network, the searching space will be limited to the smaller area shown in Figure 4.5, which suggests that the efficiency has been improved a little bit by this approach.

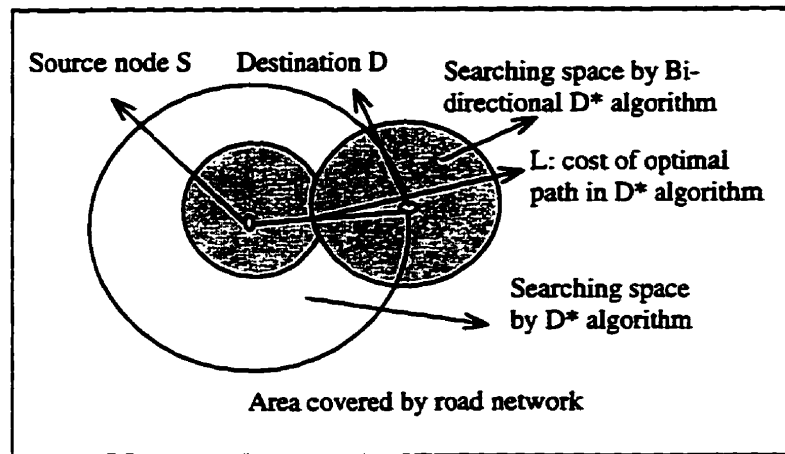


Figure 4.5: Relationship between the searching spaces in D* and Bi-D* algorithms

4.3 Result analysis on A* algorithms

4.3.1 Searching space in A* algorithm

A* algorithm is an improved approach to the D* algorithm, in which the searching space is greatly reduced because of the fact that additional information is used during the searching process. From the studies on A* algorithms, it can be concluded that by using this algorithm the searching area becomes elliptical in shape rather than the circular shape associated with the D* algorithm.

Figure 4.6 is one of the typical examples showing this feature, where the objective is to find the optimal path from node 12 to node 32. The searching space will be the whole area covered by the network in the case where D* algorithm is used for the optimal path. However, the searching space is constrained within the elliptical area (approximately) in A* algorithm.

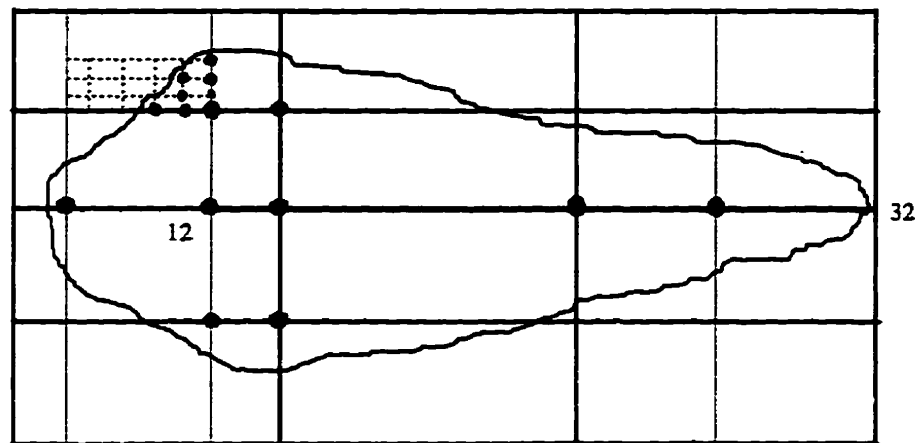


Figure 4.6: Example of A* algorithm

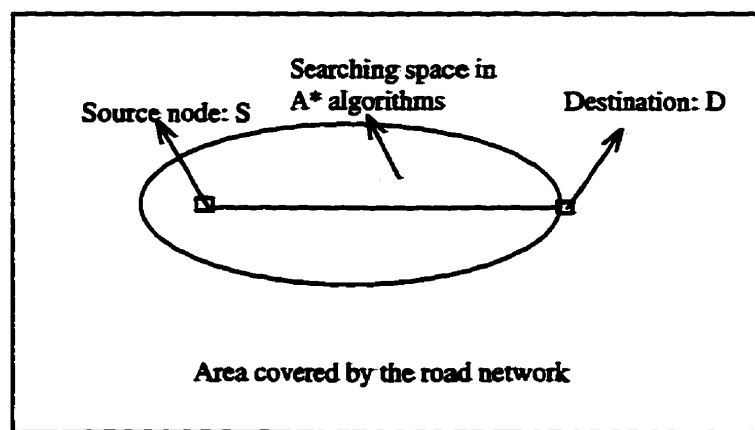


Figure 4.7: Searching space in A* algorithm

The studies have shown that when A* algorithm is applied to road networks, it tends to search outward from the source node in more of an elliptical rather than circular manner where the major axis of the ellipse is the straight line between the two end nodes.

The relationship between the searching space and the area covered by the road network is represented by Figure 4.7.

The relationship between the searching spaces in A* and BA* (Bi-directional A*) algorithm is represented by Figure 4.8.

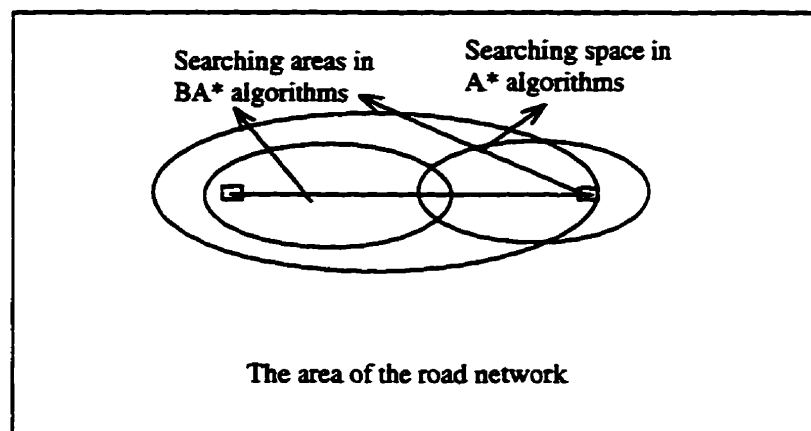


Figure 4.8: Relationship between the searching spaces in BA* algorithm and A*

These two figures show graphically the advantages of A* algorithms over the corresponding D* algorithms when used in optimal path searching.

4.3.2 Relative error analysis on A* algorithm

When A* algorithm is used, the search space is much more limited than that used in D* algorithm. One of the concerns of this approach is if it is possible to get the right solution by searching the limited area using A* algorithm. A series of sensitivity analyses were performed to illustrate some of the issues involved using A* and B_A* algorithms to calculate optimal paths for RGS applications. In particular, the relationship between the number of nodes involved in the searching process and the accuracy of the solution was examined for A* algorithms as compared to the D* algorithm.

The challenge for RGS developers is to identify the appropriate functions to estimate travel cost for their particular applications. As stated previously, the minimum

paths required by most RGS are based on travel time. In this research the estimated cost of travel is represented by equation [2] where the estimated travel cost to node k from node j is in time units.

$$E(j, k) = \frac{\sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}}{v_e} \quad \dots\dots\dots (2)$$

where: v_e = estimated speed.

Note that because link travel times in traffic networks are non-Euclidean (i.e. a person can travel further on one route but arrive earlier than if a person had chosen an alternative route) there is no guarantee that equation [2] gives a lower bound. However, as long as v_e is greater than any link travel speed on the network, then the estimated travel time will be a lower bound -- it may take longer but never shorter than the estimated straight line travel time. This implies that the algorithm will be guaranteed to find the optimal solution.

To formalize this, suppose we denote by L the actual cost from the source node S to the destination node D and denote the estimated cost of the optimal path by L'; then there is the theorem saying that "if $L \leq L'$ for every node n, the A* algorithm will always find an optimal path.

The appropriate value of v_e is probably best estimated based on empirical tests for each network. One could choose an unrealistic speed (i.e. 200km/h) that would guarantee convergence but this could add significantly to the computation time. It should be noted that as the value of v_e approaches infinity, the estimated travel time tends toward 0 and the A* algorithm is effectively the same as the D* algorithm. As the value of v_e approaches 0, the reverse occurs and the estimated travel time takes on greater importance. The end result is that the solution space is significantly constrained and the probability of finding the optimal solution is reduced. In the latter case, however, the algorithm is computationally faster because of the constrained search space.

Two figures are given below to show the relationships between the number of nodes involved in the searching process, the average speed used in A* algorithm, as well as the accuracy of the solution.

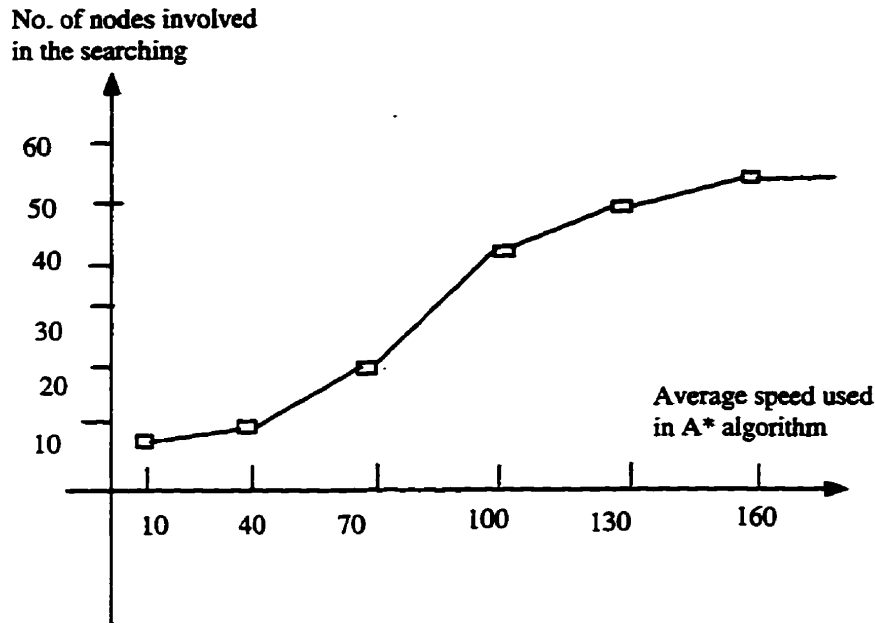


Figure 4.9: Relationship between the number of nodes involved in the searching process and the average speed used in A* algorithm

Figure 4.9 shows a graph of the number of nodes involved in the searching process as a function of the average travel speed used in the estimated travel time. The number of nodes involved in the optimal path searching process by D* algorithm is 56, which is the special case where the average speed is equal to infinity. It can be seen from this figure that the number of nodes increases as the travel speed used to estimate travel time increases. This pattern would be expected because as the value of average speed increases, so does the search area.

Although the A* algorithm heuristics are superior to the D* algorithm in terms of computation time, it should be remembered that they are not guaranteed to find the optimal solution.

Figure 4.10 shows a graph of relative error versus average travel speed used in A* algorithm. In this context, the relative error is defined as the increase in O-D route travel time, as identified by the A* algorithm, as compared to the optimal O-D route travel time, as identified by the D* algorithm.

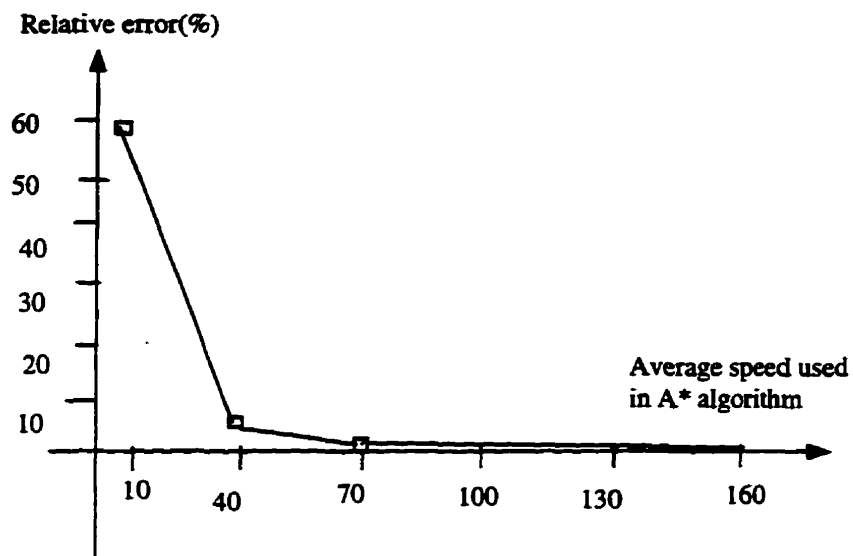


Figure 4.10: The relationship between relative error and average travel speed used in A*

The relative error in travel time ranges from 0 to 58%. As would be expected, the relative error decreases as v_e increases and this decrease seems subject to exponential function. The reduction in relative error is due to the fact that the search area increases as average speed increases and therefore, the probability of finding the optimal route also increases.

From an analysis of Figures 4.9 and 4.10, it may be seen that for the simulated road network an average speed of 70 km/h is appropriate for the A* algorithm. This would result in an approximate 60% decrease in computation time, as compared to the D* algorithm, and a relative error of approximately 1.0%.

4.4 Result analysis on road-based searching

In this section, the road-based searching algorithm is applied to the road network shown in Figure 4.10 to calculate the optimal paths. The relationship between the iteration times and relative error was examined for the road-based searching algorithm as compared to the D*, A*, and BA* algorithms. Considering that window-based searching and hierarchical structure-based searching methods, which are derived from D* algorithm, have the same characteristics as that of D* algorithm in terms of searching space and efficiency, the results of these two methods are not listed in the following tables.

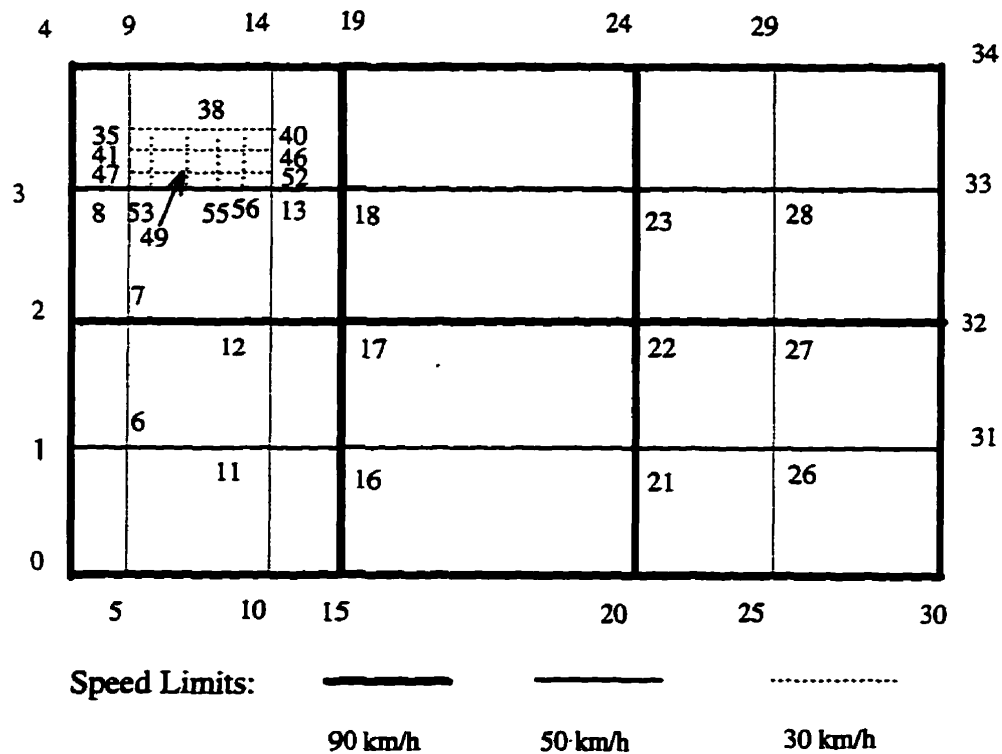


Figure 4.10: First simulated network

Listed in the following four tables are the results by D*, A*, BA* and road-base searching algorithms for different routes. One thing that should be mentioned here is that the average speed used in A* and BA* algorithm is 70 km/h, which is considered the most appropriate speed making A* algorithm get to the right solution with minimum examined nodes. The relative error of the result for A* algorithm is dependent on the selected average speed.

Table 4.1: Result comparison for route from 10 to 29

Algorithms	List of nodes in "the optimal path"	Iteration times	Travel time (hours)	Relative error
D* algorithm	10, 15, 16, 17, 18, 23, 24, 29	52	0.21667	0%
A* algorithm	Same as above	24	0.21667	0%
BA* algorithm	10, 15, 20, 21, 22, 23, 24, 29	8	0.21667	0%
Road-based searching	Same as the result of D* algorithm	3	0.21667	0%

Table 4.2: Result comparison for route from 6 to 28

Algorithms	List of nodes in "the optimal path"	Iteration times	Travel time (hours)	Relative error
D* algorithm	6, 7, 12, 17, 22, 23, 28	52	0.21750	0%
A* algorithm	Same as above	17	0.21750	0%
BA* algorithm	Same as above	8	0.21750	0%
Road-based searching	6, 7, 12, 17, 22, 27, 28	3	0.22583	3.8%

Table 4.3: Result comparison for route from 2 to 34

Algorithms	List of nodes in "the optimal path"	Iteration times	Travel time (hours)	Relative error
D* algorithm	2, 3, 4, 9, 14, 19, 24, 29, 34	50	0.21944	0%
A* algorithm	Same as above	19	0.21944	0%
BA* algorithm	Same as above	10	0.21944	0%
Road-based searching	2, 7, 12, 17, 22, 27, 32, 33, 34	2	0.22778	3.8%

Table 4.4: Result comparison for route from 46 to 32

Algorithms	List of nodes in "the optimal path"	Iteration times	Travel time (hours)	Accuracy (Relative error)
D* algorithm	46, 52, 13, 18, 17, 22, 27, 32	51	0.18478	0%
A* algorithm	Same as above	16	0.18478	0%
BA* algorithm	Same as above	9	0.18478	0%
Road-based searching	46, 52, 13, 12, 17, 22, 27, 32	2	0.20200	9.3%

It can be seen from these examples that in some cases the "optimal path" using different algorithms is the same; the only difference is in the iteration times for different algorithms. Iteration times for D* algorithm are much more than those of A* and BA* algorithm.

The BA* algorithm is more effective than A* algorithm. This is evidenced by the number of nodes examined in the searching process. A lot of calculations have shown that the computation time in BA* algorithms is usually reduced up to 50% when compared to the A* algorithm. Although these examples show that the relative error for BA* algorithm is 0%, it does not mean that the optimal paths are always 100% guaranteed using this algorithm. An example shown in Table 4.5 is given to show this case.

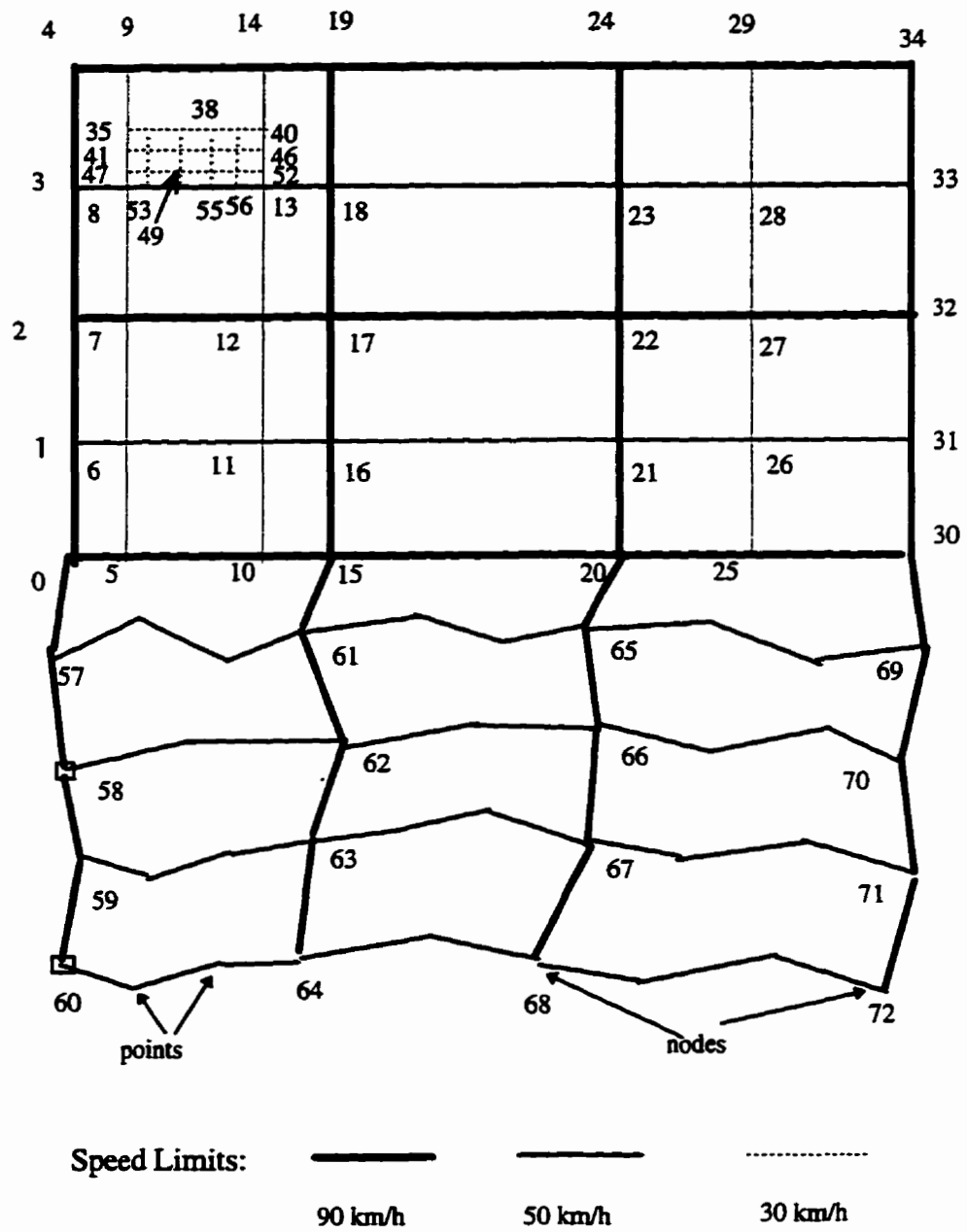


Figure 4.11: Second simulated network

Table 4.5: Result comparison for route from 60 to 29

Algorithms	List of nodes in "the optimal path"	Iteration times	Travel time (hours)	Relative error
D* algorithm	60, 59, 58, 57, 0, 1, 2, 7, 12, 17, 18, 19, 24, 29	69	0.3472	0%
A* algorithm	Same as above	26	0.3472	0%
BA* algorithm	60, 59, 58, 57, 0, 5, 10, 15, 20, 21, 22, 23, 24, 29	16	0.3680	6.0%
Road-based searching	60, 59, 58, 57, 0, 1, 2, 3, 4, 9, 14, 19, 24, 29	2	0.3597	3.6%

Compared with the results obtained by using other algorithms, road-based searching algorithm always has the least iteration numbers. The larger the network, the larger the difference of iteration times will be. Here the number of iteration times is the measurement of efficiency for different algorithms. The D*, A* , and BA* algorithms will examine all the nodes involved in the road network once during each iteration and the road-based algorithm will examine all the roads included in the road network. The computation time will approximately be the same in both situations. Compared to D*, A*, and BA* algorithms, road-based searching algorithm is more efficient in terms of searching time.

The optimal paths are not 100% guaranteed in using the road-based searching algorithm. However, the solutions are always close to the best ones. This is evidenced by the accuracy of the road-based searching, which ranges from 0% to 9.3%.

Road-based searching process is guided by spatial and non-spatial information, which is the major feature that distinguishes it from others. It integrates all the beauties of other methods. During the searching process it examines "roads" rather than "segments"

because of the adoption of object-oriented data model, which significantly reduces the number of nodes involved in the searching process, and thus reduces the searching time.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

The main purpose of this research is to investigate the applicability of object-orientation in road network modeling and to examine the relative errors and efficiency of the A* algorithm, BA* algorithm, and the newly proposed algorithm - the road-based searching algorithm. This chapter summarizes the main conclusions and recommendations obtained from this research.

5.1 Conclusions

Road network modeling benefits from the object-oriented technology in terms of elegant architecture, faster development and high quality, better performance as well as reducing data redundancy.

An object-oriented data model is appropriate in modeling road networks where the data is complicated by the fact that it is encapsulated with rich internal attributes and structural relations between components. Conventional approaches separate spatial and non-spatial information, which reduce efficiency and endanger data integrity. Object-oriented models show their suitability by providing important features such as data abstraction and specifications in constructing complex objects.

1. As a basic unit in object-oriented data modeling, an object encapsulates both state and behavior features. Objects can be as simple or as complex as the application demands; more complex objects can be constructed from combinations of existing objects which can, in turn, be simple or complex objects. The mapping of real world objects is 1:1 to objects in the object-oriented data model, whereas it is 1:N in conventional data models. In this way, the object-oriented data model captures the semantics of real world objects more directly in a database.

2. The support for the treatment of complex objects and the mechanisms for data structuring such that data concerning a single spatial is not decomposed into different structures in object-oriented data model, makes the representation of complex objects and the query to them much easier and efficient. There is no need to break up large objects for storage in normalized tables and reassemble them at run time via slow joining operations. Information access is usually achieved by navigation through pointers and no physical operations are needed to reconstruct the information from pieces as in the relational model. So information access become more direct and straightforward both conceptually and physically, and manipulating efficiency is improved.
3. The definition of classes and objects in the data model for road networks facilitates the implementation of the road-based searching algorithm. With the object-oriented data modeling, organization of the data related to the road network becomes much more efficient with less storage space being used and much more information being provided. All the optimal path searching algorithms, especially the road-based searching algorithm, can benefit from this modeling.

From the implementation and testing of optimal path searching algorithms, the following conclusions can be given:

1. D* (Dijkstra's) algorithm is a strict and standard algorithm for optimal path searching upon which almost all the heuristics-based searches are built on. This algorithm can be used to calculate the optimal path from source node to all the nodes of the road network. The searching space is "a circle" centered at the source node with the radius equal to the cost of the optimal path from source node to the destination node. Computationally, this algorithm in the worst case, requires $O(N^2)$ operations. This quantity is directly related to algorithm run time. For larger networks, the number of operations is increased tremendously. The $O(N^2)$ running time for larger networks is not appropriate and not acceptable.

2. The A* algorithm improves the D* algorithm by limiting the searching space into an elliptical rather than circular shape associated with the D* algorithm. As expected, the A* algorithm gave better results. Although not guaranteed to find the optimal route when the average speed used in this algorithm is less than the real travel speed, the relative error in route travel time was relatively small and the computation time was faster than the D* algorithm. From an analysis of Figures 12 and 13, it may be seen that for the simulated road network an average speed of 70 km/h is appropriate for the A* algorithm. This would result in an approximate 60% decrease in computation time, as compared to the D* algorithm, and a relative error of approximately 1.0%.
3. BA* (Bi-directional A*) algorithm also performs greatly in the simulated network in terms of computation time and relative error. Theoretically, the searching space of BA* algorithm should be smaller than that associated with A* algorithm. The only problem with this approach, when applied to larger networks, could be the relative error, because there is no guarantee of finding the optimal path. It may be beneficial for the proposed RGS to have a number of algorithms available that could be used in different situations.
4. Usually, the “optimal path” obtained from the road-based searching algorithm is acceptable, with accuracy ranging from 0% to 10%. The advantage of this algorithm over D*, A*, or BA* algorithms is that the computation time is greatly reduced; only 10% - 20% computation time is required as compared to A* algorithms.
5. The data set used in an actual RGS will be significantly more extensive than the simulated network. Consequently, the optimal path calculation time will be higher. In addition, the operating platform, and other factors relating to the implementation of the algorithm, will all affect the results. However, the basic algorithms, issues and relationships discussed here will remain the same.

5.2 Recommendations

For this work to be integrated in real road guidance systems and provide users with all the functions needed for route guidance, some further work needs to be done. This includes: defining more object types such as place of interest, shopping center, and park, improving the interface for information retrieval, and simplifying the query procedure.

Currently, the input data for optimal path searching are the names or the IDs of the source node and the destination node. For general users, what they have in mind is the name or address of a specific place or an area, such as Chinatown. If this software is used in situations like that, more functions should be developed and provided in addition to the nodeID-based searching functions.

Query languages used in the RGS systems should be more friendly and natural [Leung, 1993, Wang, 1994]. In any database systems, data retrieval is requested by a query, and a query is coded in a query language. The currently used query languages are artificially defined, and they have shortcomings when applied to retrieving geographical data in RGS. Many concepts related to geographical data are vague whereas the query languages were designed to express precise concepts only. For example, a user may want to find a motel in the downtown area which is *reasonably priced* and is not *very far* from the *city center*. However, currently used query languages are unable to express the vague selection conditions 'reasonable', 'very far', and 'city center'. They can specify precise values and value ranges only. In expressing vague concepts the artificially-defined query languages are inferior to a natural language. The applications of RGS can be facilitated if natural query languages can be used.

REFERENCES

1. Blais J.A.R., W. Zhou and A.W. Colijn, 1996, "On the optimal design of an environmental information system for the crown of the continent", Geomatics 1996, Proceedings on CD, National Defence, Ottawa.
2. Blais, J.A.R., 1996, "Design and implementation of geospatial information systems", Lecture notes, Department of Geomatics Engineering, The University of Calgary, Canada.
3. Blum, Y., 1989, "Comprehensive approach to route guidance using the Q-route prototype", M. Sc. thesis, Queen's University, Kingston, Ontario, Canada.
4. Bullock, 1995, "A prototype portable vehicle navigation system utilizing map aided GPS", M.Sc. thesis, Department of Geomatics Engineering, The University of Calgary, Canada.
5. Bondy and U. S. R. Murty, 1977, "Graph Theory With Applications", The Gresham Press, Old Working, Surrey, pp.1-41.
6. Car, A. and A.U. Frank, 1993, "Hierarchical street networks as a conceptual model for efficient way finding", Proceedings of the EGIS'93 in Geneva, Italy, pp.134-139.
7. Car, A. and A. U. Frank. 1993, "Lecture notes in computer science No.884", pp.15-24.
8. Chen, P P S., 1976, "The entity relationship model: towards a unified view of data. ACM Trans on Database Systems. 1, pp.9-36.
9. Codd, E. F., 1981, "Database Management", ACM SIGMOD Record, 11(2).
10. Data, C J., 1990, "An Introduction to Database Systems", Vol.1, Addison-Wesley, Reading, 845 pp.

11. Dijkstra, E., 1959, "Numerische Mathematik, 1, pp.267-271.
12. Frank, A. U. and M. Egenhofer, 1989, "Object-oriented database technology for GIS: Seminar Workbook", National center for Geographic Information and Analysis, Seminar presented at GIS/LIS'89, pp.1-56.
13. Frederickson, C.N. 1994, "Fast algorithms for shortest paths in planar graphs, with applications", SIAM Journal on computing Vol.16, No.6, pp.1004-1022.
14. Gallo, G. and S. Pallotino, 1984, "Shortest path methods in transportation models", Transportation planning models, M. Florian Editor, North Holland-Elsevier Science Publishers.
15. Gibbons, A. 1976, "Algorithmic graph theory", Cambridge University Press.
16. Golledge, R.G. 1991, "Designing a personal guidance systems to aid navigation without sight: progress on the GIS component", Int. J. Geographical Information Systems, Vol.5, No.4, pp.373-395.
17. Goodchild, M F. 1992. Geographical data modeling, Computers and Geosciences 18, 4, pp. 401-408.
18. Gopal, S. and T. R. Smith, 1990, "Human way-finding in an Urban Environment: a Performance Analysis of a Computational Process Model", Environment and Planning A 22, pp.169-191.
19. Graham, I. 1991, "Object-oriented methods", Addison-Wesley Publishing Company, pp.120-220.
20. Hart, P. N. Nilsson, and B. Raphael, 1968, "A formal basis for the heuristic determination of minimum cost paths", IEEE Transportation System Science and Cybernetics, 4.

21. Herring, J. R., 1992. "TIGRIS: A data model for an object-oriented geographic information system", *Computers & GeoScience*, Vol.18, No.4, pp.443-452.
22. Hughes, J G. 1991. "Object-oriented Databases", Prentice Hall, New York, 268 pp.
23. Hull, R. & R. King, 1987, "Semantic database modeling: survey, applications, and research issues.", *ACM Computing Surveys*, 19(3), pp.201-260.
24. Hung, M. S., and J. J. Divoky, 1988, "A computational study of efficient shortest path algorithms", *Computer operation research*, Vol.15, No.6, pp.567-576.
25. Johnson, D. B., 1977, "Efficient algorithms for shortest paths in sparse networks.", *Journal of the Association for computing machinery* 24, pp.1-13.
26. Karimi, H.A. 1991, "Design and implementation of automatic route guidance systems for land based applications", M.Sc. thesis, Department of Geomatics Engineering, The University of Calgary, Canada. Jin Y. Yen, 1975. "Shortest Path Network Problems", *Verlan Anton Hain KG*, pp.1-21.
27. Karimi, H. A. and Y. C. Lee, 1995. "Semantic data models in GIS and object orientation", *Proceedings of GIS'95*, Vol. 1, pp. 407-412.
28. Khoshafian, S. And A. Abnous, 1990. "Object orientation: concepts, languages, databases, users interfaces", *Wiley & Son Press*, New York, pp.120-145.
29. Krakiwsky, E. J., H.A. Karimi, C. Harris and J. George, 1987, "Research into electronic maps and automatic vehicle location (AVL) systems", *Eighth international symposium on computer-assisted cartography, AUTO-CARTO 8*, Baltimore, Maryland, U.S.A., March29-April 3.
30. Kufoniyyi, O. 1995, An introduction to object-oriented data structures, *ITC Journal*, Vol. 1, 1995, pp.1-6.
31. Kuipers, B., 1992, "Spatial reasoning", *Tutorial materials*.

32. Kuipers, B , 1978, "Modeling spatial knowledge.", *Cognitive Science* 2, pp.129-154.
33. Kuznetsov, T., 1992. "High performance routing for IVHS", preprint from IVHS America 3rd Annual Meeting.
34. Lapalme, G., 1992, "GeoRoute" *Communications of the ACM* 35, No.1, pp.80-88.
35. Lee, Y. C., 1990, "A comparison of relational and object-oriented models for spatial data", *Lecture Notes*, University of New Brunswick, pp.1-6.
36. Leung, Y. 1993, "An intelligent expert system shell for knowledge-based Geographical Information Systems: 1. The tools", *INT. J. Geographical Information Systems*, Vol.7, No.3, pp.189-199.
37. Liming, Q., 1996, "Building 3D GIS by object-orientation", M. Sc. thesis, Department of Geomatics Engineering, The University of Calgary, Canada.
38. Litton, G. M., 1987. "Introduction to database management, a practical approach", Wm. C. Brown Publishers, pp.230-250.
39. Nicholson, T.A.J., 1966, "Finding the shortest route between two points in a network", *The computer Journal*, Vol.9, No.3, pp.275-280.
40. OECD, 1988. "Route Guidance and In-Car Communication Systems", Road Transport Research. Report Prepared by an OECD Scientific Experts Group, Paris.
41. Palmer, E.S.. 1977, "Hierarchical structure in perceptual representation.", *Cognitive psychology*, pp.441-474.
42. Pohl, I., 1971, "Bi-directional Search", *Machine Intelligence*, No. 6, pp.127-140.
43. Rilett, L.R., 1994, "Minimum path algorithms for in-vehicle route guidance systems", *The Proceedings of the 1994 Annual Meeting of IVHS America*, Vol.1, pp.27-36.

44. Tang, A. Y., 1996, "A spatial data model design for feature-based geographical information systems", *Int. J. Geographical Information Systems*, Vol.10, No.5, pp.643-659.
45. Taylor, D. A., 1990, "Object-oriented technology", Addison-Wesley publishing company, pp.15-76.
46. Tkach, D., 1995, "Object technology", Addison-Wesley, pp.1-23.
47. Tom V., and V.O. Peter, 1992., "The GEO++ system: an extensible GIS", *International Journal of Geographical Information Systems*, Vol.8, No. 2, pp.143-162.
48. Tsichrizzis, D.C. and F. H. Lochovsky, 1982, "Data models", Prentice-Hall Inc, pp.80-120.
49. Ullman, J D., 1988. "Principles of Database and Knowledge-base Systems", Vol.1, Computer Science Press, pp.120-140.
50. Usery, E. L., 1993, "Category theory and the structure of features in geographic information systems", *Cartography and Geographic Information Systems*, 20, pp.5-12.
51. Van Vuren, T. and G. Jansen, 1988., "Recent developments in path finding algorithms: A review", *Transportation planning and technology*, Vol.12, pp.57-71.
52. Wang, F., 1994, "Towards a natural user interface: an approach of fuzzy query", *International Journal of Geographical Information Systems*, Vol8, No. 2, pp.143-162.
53. Won K., 1990. "Introduction to Object-Oriented Databases", The MIT Press, pp.1-23.
54. Worboys, M. F., 1994, "Object-oriented approaches to geo-referenced information", *International Journal of Geographical Information Systems*, Vol.8, pp.385-399.
55. Worboys, M. F., 1990, "Object-oriented data modeling for spatial databases", *International Journal of Geographical Information Systems*, Vol.4, pp.369-383

56. Won K., 1995, "Modern database systems", Addison-Wesley publishing company, pp.18-41.