UNIVERSITY OF MINNESOTA

CENTER FOR TRANSPORTATION STUDIES

ITS INSTITUTE

EVALUATION OF ROAD-MAP DATA FOR ITS

Shashi Shekhar
Department of Computer Science and Engineering

CTS 00 - 05

HUMAN-CENTERED TECHNOLOGY TO ENHANCE SAFETY AND MOBILITY

Technical Report Documentation Page

1. Report No.	2.	3. Recipient's Accession No.
CTS 00-05		

4. Title and Subtitle	5. Report Date
EVALUATION OF ROAD-MAP DATA FOR ITS	December 2000
	6.
7. Author(s) Shashi Shekhar	8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Computer Science and Engineering University of Minnesota 4-192 EE/CSci Building 200 Union Street SE Minneapolis, MN 55455	10. Project/Task/Work Unit No. Final Report
	11. Contract (C) or Grant (G) No. (C) (G)
12. Sponsoring Organization Name and Address Center for Transportation Studies ITS Institute 200 Transportation and Safety Building 511 Washington Ave. SE Minneapolis, MN 55455	13. Type of Report and Period Covered
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract (Limit: 200 words)

Road-map databases contain a large amount of complex information, including street names, street address ranges, turn restrictions, one-way street information, etc. In addition, these databases are likely to contain a large amount of time-varying data, which need to be checked for accuracy on a periodic basis. More and more applications are based on map databases, so establishing quality evaluation procedures and assurance programs for these databases is of great importance.

In this project, the researchers designed a prototype user interface for displaying the shortest paths from source to destination. They used the JAVA language, which is platform-independent and allows the execution of interactive software across computer networks. They also evaluated other user interfaces available on web sites to identify desirable functionalities. Compared with the existing systems (e.g., Maponus), their system goes a step forward in interactivity by providing an interactive map, on which users can click and choose their sources and destinations.

17. Document Analysis/Descriptors	18. Availability Statement
Spatial Databases, Geographic Information Systems, Road map	No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161

19. Security Class (this report)	20. Security Class (this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	22	

Evaluation of Road-Map Data for ITS

Prepared by

Shashi Shekhar
Department Computer Science and Engineering
University of Minnesota
4-192 Electrical Engineering/Computer Science Building
200 Union St. S.E.
Minneapolis, MN 55455

Submitted to

Intelligent Transportation Systems (ITS)
Center for Transportation Studies
University of Minnesota
200 Transportation and Safety Building
511 Washington Ave. S.E.
Minneapolis, MN 55455

December 2000

CTS 00- 05

Acknowledgments

We would like to thank Maureen Jensen, Jim Aswegan, and Burke Hilden at Mn/DOT for their comments and patience. We would also like to thank Kristina McCarthy at the University of Minnesota for her help in the project.

Contents

List of Figures

List of Tables

Executive Summary

Road-map databases contain a large amount of complex information, including street names, street address ranges, turn restrictions, one-way street information, etc. In addition, these databases are likely to contain a large amount of time-varying data which need to be checked for accuracy on a periodic basis. More and more applications are based on map databases,so establishing quality evaluation procedures and assurance programs for these databases is of great importance.

In this project, we designed a prototype user interface for displaying the shortest paths from source to destination. We use the JAVA language, which is platform-independent and allows the execution of interactive software across computer networks. We also evaluated other user interfaces available on the web sites to identify desirable functionalities. Compared with the existing systems(e.g. Maponus), our system goes a step forward in interactivity by providing an interactive map, on which users can click and choose their sources and destinations.

For the next step, we would like to extend the work to large metro-area map. We plan to implement several display windows and to use zoom in and zoom out to control the map's hierarchical level. we would also enhance the system architecture by using Client/Server model.
Keywords: Spatial Databases, Geographic Information Systems, Road map

1 Introduction

1.1 Objectives

There is a need to establish quality evaluation procedures and a quality assurance program for various kinds of road-map data within MNDOT, and particularly within Guidestar. The objective of this project is to define quality measures of road-map data for projects in Intelligent Transportation Systems. Road map databases contain a large amount of complex information including street names, street address ranges, turn restrictions, one-way street information, etc. In addition, the database is likely to contain a large amount of time-varying data such as incidents, maintenance schedules, etc., which need to be checked for accuracy on a periodic basis. A variety of projects including Genesis and Travlink have developed software based on such road-map data. The software developed in these projects is extremely sensitive to any errors in the road-map data. Thus, reliable quality measures and quality assurance efforts are required within the MN Guidestar program.

Several MN Guidestar projects used road-maps of the Twin Cities metro area, and with the advent of ITS, more and more projects would be based on software using the road-map data. The road-maps are being acquired from different sources such as Etak and Census Bureau Tiger files. Even though the projects need an up-to-date and accurate road-map, it has been difficult to judge the quality of the maps being used with respect to the objectives of projects. An interesting example is the use of the Census Bureau Tiger files, which are known to be imperfect. Even when vendors (e.g. Etak) are hired to develop road-map data, it is often difficult for MNDOT to judge the quality of the delivered product, which has restricted licensing which prevents wide-spread use and evaluation of the data.

1.2 Literature Survey and our contributions

Research into map quality is relatively scarce, but of the articles that were reviewed, there are two major areas of interest: i) the need for quality control of data, and ii) techniques to ensure data quality or to quantitatively measure the quality.

Several articles identified a need for the quality control of data, but as of now have not addressed these issues. One looked at the issue from the angle of policy concerns and of the cost involved in implementing a data quality-control system[11]. Several other articles summarized data collection efforts and a potential need for quality control efforts[5]. Another article looked at the potential need for quality control in order to meet the requirements of the Intermodal Surface Transportation Act of 1991[9].

Quality control efforts are already underway in certain projects, which the literature describes. The Federal Aviation Administration is using an independent verification and validation agent to both ensure quality control and to quantitatively measure data quality[5]. Data quality assurance checks were a common way to ensure a high standard of data accuracy[1, 4, 2]. One conference paper concentrated on finding accuracy estimators for data and for navigation fixes[10].

The requirements for road-maps for ITS have been discussed with some references to quality attributes. An electronic guide for ITS, comprised of a digital map of the roads and streets, including the properties of the traffic, the geometry of the street network and their attributes[6]. Other reports deal with how to efficiently use digital maps and spatial databases, with what users of map data should be aware of when selecting a digital map database[7], with the different functional roles of digital road map databases in an overall ITS environment[13], and with a new software approach for capturing geometrical and attribute data from two digital maps in order to produce a high quality merged map[3].

Our contributions in this project cover several areas of interested concern: First we make a classification of road-map errors, which can serve as a source to develop corresponding strategies. Second we discuss several alternate methods to minimize the errors. We also developed a webRoute on a map evaluation.

1.3 Research tasks

The project focused on classifying the kinds of errors in road-map data and on developing validation procedures via the following tasks:

Task1: Identifying Error Sources

Task 1 is concerned with cataloging and classifying error sources in road-map databases. Example error sources include the data entry operator, out-of-date information, and faulty sensors and transmission media. The characterization of error sources will be useful in developing procedures for identifying errors.

Task2: Classification of Errors

A classification scheme is developed to categorize the errors in the road-map database. Example categories include omission (missing roads), topological errors (e.g. incorrect turn restriction), geometric error (e.g. in the length of road-segment), spelling mistakes (e.g. the road name), inconsistencies, etc. The goal of categorization is to facilitate accessing the impact of errors on different applications and projects.

Task 3 : Evaluation and Validation Methods

We also explore validation procedures to ensure the high quality of road-map data. Example methods include (i) range checks on sensor measurements, (ii)cross entry (e.g. comparing road names against a database of valid road names or comparing incidents against current sensor measurements), (iii) road network checking (e.g. checking that benchmark routes are attached to the network), (iv) path consistency checking (e.g. checking that the road names match for different segments, except for the suffix), (v) peer review by engineers, and (vi) field testing. The task is to define and compare promising validation methods.

Task4:Testing the validation procedures

Alternative validation methods are tested on road-map data of the Twin Cities metro area as well as on data from other areas. We use the Census Bureau Tiger data and Etak's data for testing.

2 Source of Errors

The widespread availability of geographic information systems has enabled many users to integrate geographic information from a wide range of sources. The inevitable consequence of data integration is that errors will be generalized. Thus, it is important to recognize the source, type, severity and implications of the type errors that are inherent in the GIS data, e.g. MAP.

2.1 General error sources

The first major cause of error is derived from inherent properties of nature. Unlike geographic information data structures, the real world is not always distinct and clear, but is frequently gradual and fuzzy. For example, highways shift a little bit after construction due to settling and other soil movement, and this may change their locations slightly.

The second major source of likely error stems from the nature of measurement. Any measurements that are acquired with instruments inevitably introduce error. The capability of the person using the measuring device can also clearly affect the amount of error introduced. The scale on which the measurements are made and the frequency of sampling will also introduce potential errors.

The third source of error is due to the data models that we use to communicate our measurements. The very structure of a geographic model can be a source of error. In the case of a vector, the representation of a line or an edge implies a level of certainty or precision that may not be discernible in the real world. In the case of a raster, the aggregation or averaging of conditions by imaginary cells is also susceptible to error. The medium that we use to communicate measurements itself might also introduce errors. Data processing and transformation is also an important source of introducing errors. The more times a set of measurements is transformed through one process or another, the more likely new errors are to be introduced. The process of digitizing entities also inevitably introduces errors in digital geographic data.

2.2 Errors in map construction

A map overlay is an important operation in map construction. A map overlay involves the superimposition of two or more input maps or data layers, with the aim of producing a composite map showing the intersection of the mapping units on the individual data layers. The effects of propagating errors in the data layers differ with different error types. Cartographic error is error in positional features such as points and lines, and thematic error is error in the values of the thematic attribute.

The main consequence of applying a map overlay to data layers containing cartographic errors is the creating of spurious polygons on the composite map. Spurious polygons are artifacts of the map overlay operation that do not represent features in the real world. They arise from positional discrepancies between representations of the same line on alternate data layers.

Errors may also be attributed to the thematic components of these data. Error modeling may be handled differently for numerical and categorical data. For numerical data, the accuracy of a given layer might be defined in terms of the deviations between the actual and estimated values associated with each pixel or cell. For categorical data, errors in the individual data layers might be measured using the contingency table approach often applied in remote sensing.

3 Classification of error

Error and uncertainty have always been a feature of cartographic information and these aspects are also present in digital versions of analog maps. No map-related spatial data exist which are error free. To have a good classification of all the error sources helps in finding good methods for dealing with errors. We can classify the map errors based on two categories: Spatial and Nonspatial errors.

3.1 Model of Space

For different space models, there are different sources from which errors are introduced.

Set-based model

Set-based Space provides only elements, sets of elements and the membership relation between an element and the set in which it occurs. But it is fundamental to the modeling of any spatial information system, and is especially appropriate for modeling hierarchical relationships. Based on the basic constructs of element, set and membership, a large number of modeling tools may be constructs, such as Equality, Subset, Power set, Empty set, Cardinality, Intersection, Union, Difference, Complement.

One possible error in the set-based model is an inconsistent name for different segments of a road, or the road-classification may be in error, e.g. a highway is shown as a side-road.

Topological space

Topology is the study of topological transformations and the properties that are left invariant by them. Based on general pointset topology, a topological space is a collection of subsets of set S, called neighborhoods, that satisfy two conditions: 1. Every point in S is in some neighborhood. 2. The intersection of any two neighborhoods of any point x in S contains a neighborhood of x. The basic concepts in topological space include adjacency, overlap and intersect.

Possible errors in topological space include errors in lane adjacency and lane divider relationships. Two lanes may be shown as adjacent, even though there is a solid barrier between them. Other examples relate to the over/under relationship between a flyover road and the lower road.

Euclidean space

Coordinatized Euclidean space is a highly organized kind of space, with many well defined operations and relationships. It transforms spatial properties into the properties of tuples of real numbers. There are mainly three classes of objects: Points, Lines, polygons, and several kinds of transformations among the objects, which include Euclidean transformations, similarity transformations, affine transformations,projective transformations and topological transformations. The classification is based on the particular properties that it preserves of embedded objects.

Error possibilities in this space model include: Coordinates for nodes may be off, the length of a segment may be different from the distance between the endpoints along a segment, or there may be an error due to curvature approximation.

Network space

The basic concepts of this model include nodes, edges, and connectivity. Algorithms on a graph include finding the shortest path, DFS, topological sorting, etc.

Possible errors include: Missing edges between two nodes; Missing intersections, non-existent intersections(e.g. flyover shown as intersection), and non-existent nodes.

3.2 Non-Spatial errors

Attribute information

The attribute information describes the attributes, and there are several attributes associated with road map data. Each may introduce different types of errors.

The available attributes in popular maps include: Road type, speed information, major landmark point, zip code in address, and road names. The possible errors here are misclassifying road types, misspelling names, inconsistency errors, disagreement of a road.

Missing features

The other important errors may be missing necessary features. Since we describe features on a map, there are some features which are important for high level control missing: e.g. Length of road, traffic information of road that can be used to add in-time travel guide. And traffic time information is also helpful for ATIS.

4 Evaluation and Validation Methods

Validation procedures are needed to ensure the high quality of road-map data. Example methods include (i) range checks on sensor measurements, (ii) cross entry (iii) road network checking (e.g. checking that benchmark routes are attached to the network), (iv) path-consistency checking (e.g. checking that the road names match for different segments, except for the suffix), (v) peer review by engineers, and (vi) field testing.

4.1 Alternative methods

Range checking helps in finding the errors on map data. Traffic sensor data such as speed should be in a range of [0..speedlimit]. Data beyond this range indicates errors in map data. Other reasonable range criteria include: coordinates for road intersections within the Twin Cities should be within the coordinate of Minnesota; the length of a segment should be greater than the straight line distance between endpoints.

Cross-entry checking relies on the comparison of a map against a more accurate dataset. For example, road-names on a map may be checked against the post-office street address database, assuming the latter is more reliable.

Road-network checking is testing the map via checking that benchmark routes are attached to the network. The shortest-path algorithm should usually yield a well-known optimal path. The contrary case may imply some error in the map. For example, the shortest distance path from the EE/CSci Bld to the Peking Garden restaurant should be along Washington Ave., which is almost a straight line joining the two points of interest. If it is not, then possible map errors may include missing segments or an incorrect segment length.

Path-consistency checking is to check whether the road names match for different segments, except for their suffix(e.g. SE, NE, etc.). For example, each segment of Washington Ave. should be called by the same name. If one segment is called Washington Ave., and the other is called Washington Road, then inconsistency exists.

Peer review by engineers is a manual inspection. Engineers look through maps and find errors based on their experiences and knowledge on the real world. Field testing is also a manual checking process. It is done by bringing the map to the real world and investigating the data on the spot to see if they correctly represent the real roads.

4.2 How does routing relate to map accuracy?

Route-planning is described as finding a path from a source node to a destination node under some specific condition, such as the shortest travel distance or the shortest travel time. The problem starts from a source, and the goal is the destination. The routing problem computes an optimal path between the given node-pair(source-destination). The optimization is based on several factors: minimum total path cost, or minimum path travel time. Also, the traffic condition of each road could be used as a factor to compute the path.

A critical component of the routing service is a complete and accurate road-map suited for navigation. Through the WebRouting developed in this phase, we can identify and classify the errors in the road-map database. Through the WebRoute system, we can perform two kinds of checking: Road-network checking and path-consistency checking. The WebRoute allows commuters to select their source and destination, and it computes a route based on that information. Commuters will then provide comments about the quality of the computed route to help evaluate the underlying road-map. This routing application provides a framework so that we can not only test the performance of several kinds of algorithms but can also test map accuracy.

There are several web sites providing web routing service, such as MapsOnUs, MapInfo, ESRI, and SmallWorld(See table 1). Developed by Lucent Technologies, MapsOnUs is one of the most useful functional WWW routing sites. It collaborates simultaneously on travel-planning over the Web, and it integrates yellow pages, directions, and mapping features. The users can plan routes and search for locations within the entire United States. After specifying the source, intermediate, and destination addresses, the map server will respond with the shortest path, along with turn by turn directions. The registered users can also edit their own hot lists and set the markers, such as where they live, where they work, their current positions, etc. The route type choice provides the Fastest Route, the Shortest Route, Major Highway Prefered, or Major Highway Avoided. The other routing options are Plan Route or Plan Best Tour. Plan Best Tour minimizes the total distance by reordering the intermediate destinations as needed. Plan Route uses the intermediate destinations order chosen by users.

Company Name	Web site
MapsOnUs	http://www.MapsOnUs.com/
ESRI	http://www.esri.com/
SmallWorld	http://www.smallworld-us.com/
MapInfo	http://www.mapinfo.com/
MapQuest	http://www.mapquest.com/
InterGraph	http://www.intergraph.com/

Table 1: Web site for GIS company

The ESRI has two routing application demos in its Web site, Where To Go in San Francisco and the Bus Itinerary Query. In Where to Go in San Francisco, the users input their current address or select a landmark, and the server will show where features are in the neighborhood and how to get to them. However, its user interface is not as friendly as that of MapsOnUs, and the routing is limited within San Francisco. The Bus Itinerary Query is a prototype of a public access internet connecting to the Seattle area Metro Transit route, along with schedule data. It provides walking directions as well as bus itineraries.

MapInfo provides all the tools needed to develop and deploy powerful spatial analysis and mapping applications, and their products can connect directly to server-side databases such as Oracle, Infomix, Sybase and others.

SmallWorld products offer a unified platform for modeling complex physical networks and enabling applications that require spatially referenced data. Their core products include a rich object-oriented development environment, a highly effective CASE tool, and a unique approach to transaction management that allows multiple users to design additions, make changes and open new work orders using the same spatial data at the same time.

4.3 Our Java Routing

Our WebRoute software system is implemented using JAVA, and it consists of two component classes: Client and Server. Because of the problem of system resource management, we now put these two classes on the client side, and the server only provides data through the web. We separate the functions to different component classes as to achieve the modualization, which allows easy maintenance and algorithm changes.

Why JAVA

Java is an object-oriented programming language allowing the execution of interactive software across computer networks, thus allowing a single program to be run on different platforms. Using Java, computation may be distributed across many machines, lowering the load on a single machine.

Architecture

Applet: we use the JAVA language to implement a client interface. Java provides a good AWT library for the design. The GUI for our web site is very user-friendly, providing several ways for users to choose a routing source and a destination. If the users have knowledge of the area on the map, they can input the source and destination position by clicking on the intersection of the map. The Applet also provides a list of the intersection street names, and a hot list of frequently visited places for users to choose. After the users have chosen a source and destination, the Java Applet sends the requirement to the server class and receives the path from the server class later on. After it has received the path, the path is represented graphically in the map, colorfully connecting the source and destination. The message panel also shows the corresponding information from the server and gives the users turn-by-turn directions within this shortest route. The server processes all the map and road data and also computes the optimal path between the source and the destination. It accepts requests from the Client and also sends the resulting path to the client. When there are map information requests come from the interface, the server has to distinguish what kind of requests they are, and then determines if path computing is necessary.

Data Server: The map data contain the road type, start and end position. If the road conditions, such as road speed and congestion, are included in the map/road data and are updated hourly, we can optimize the path on several criteria.

4.4 System Demo

Interactive map interface

The map includes the major roads in Minneapolis, and the users select the source and destination by either clicking on the position of the map, or by choosing the road intersection from the Gazateer. The source and destination window displays the identification of these two points and sends this query information to the routing server. After computing the route path according to the chosen algorithm and criteria, the routing server sends back the final path. This calculated information is shown in the map with a colored line and in the message window with turn-by-turn directions.

Source and destination window

When an intersection is clicked on the map, the identification associated with this intersection is displayed in the source and destination window(Figure 1). The first selected intersection is the source, and the second is the destination. The directions calculated during the path finding will be based on this order. After setting the source and destination, the Java applet sends the pair to the routing server and then computes the path. For computing the next path, just click on another intersection, and the new source will appear while the destination field is cleared to accept the new destination.

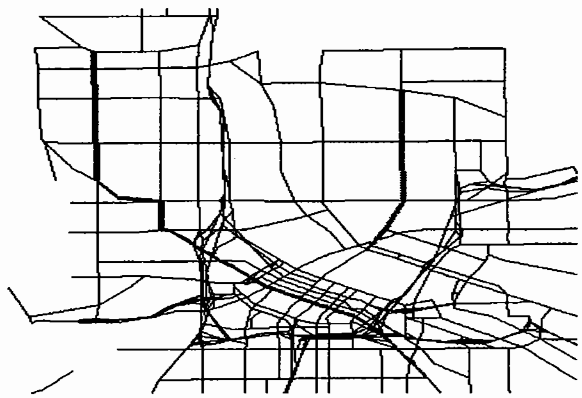


Figure 1: Source and Destination window

Message Window

The message window displays the turn-by-turn directions provided by the routing server(Figure 2). Following these directions, the users will travel from source to destination. It also shows the status of the network connection.

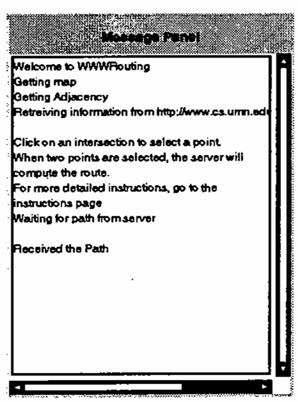


Figure 2: Message Panel

Gazeteer Window

If the user does not know an exact location on the map, the Gazeteer window provides an easy way to set a road intersection(Figure 3). To use this feature, double click on a street name in the left window, and the list of streets it intersects will show in the right window. Select the proper cross street by double clicking on the road name. The setting will be displayed in the source and destination window.

Figure 3: Gazateer Window

Routing algorithm option

This option allows the users to select different types of routing algorithms or criteria(Figure 4). The users can choose the Dijkstra shortest path or the A* algorithm; both will calculate the same routing path, but with a different computation time. The user can also select the shortest travel distance or the time between a source and a destination. However, these features have not been implemented in the current version, since it needs dynamic road conditions to calculate the shortest travel time.

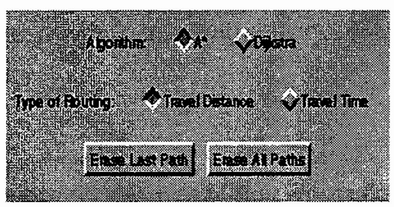


Figure 4: Routing algorithm option

5 Testing the Validation Procedure

Based the data from Etak, we illustrate various types of road-map errors via a sequence of examples.

- **Example 1: Non-existent road-segment**
  In the central part of figure 5 Church Street is close to the north side of Washington Avenue, and the south part of Union Street after Washington Avenue does not exist. Besides, the second block north of Washington Avenue, and between Union Street and Harvard Street is non-existent.

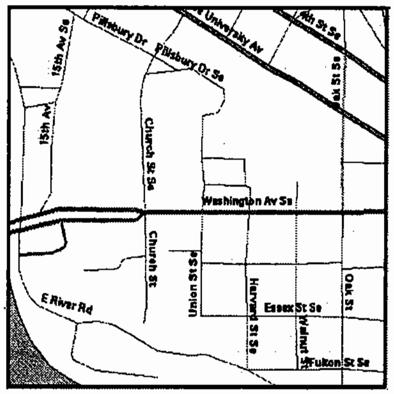


Figure 5: Non-existent road-segment

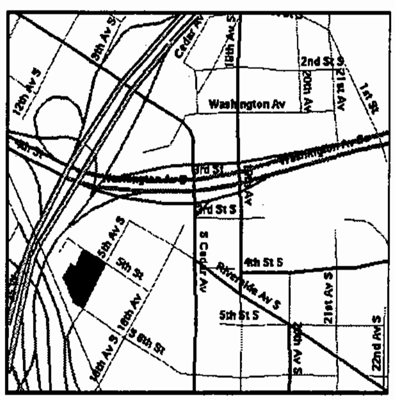


Figure 6: Non-existent road-intersection(19th Ave. S, Washington Ave. S)

- **Example 2: Non-existent road-intersection**
  In the central part of figure 6, the graph shows an incorrect intersection of Washington Ave. with 19th Ave. Actually, 19th Ave. passes over Washington Ave.

- **Example 3: Non-existent intersection with railroad**
  In figure 7, the graph shows an incorrect railroad intersection at 15th Ave. and 4th Street. In fact, the railroad intersects with 15th Ave. and University Avenue, and the railroad should be shifted south one block.

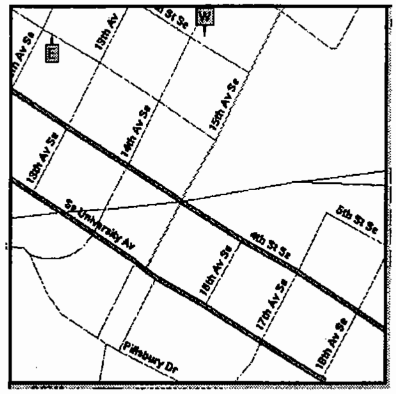


Figure 7: Non-existent road-intersection with railroad

- **Example 4: Topological error (Over/under)**
  In the central part of figure 8, from Huron Street SE to the east of Highway 94, the ramp should be above Highway 94, not under it, as shown in the graph.

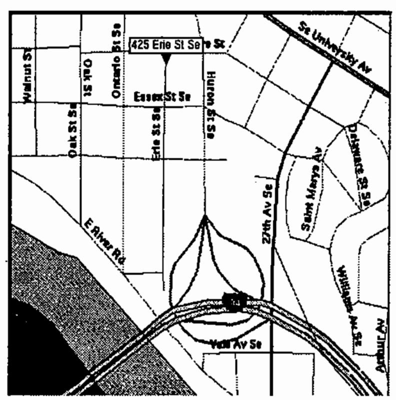


Figure 8: Topological error (overpass/underpass)

- **Example 5: Non-Spatial error: missing road name**
  In the bottom part of figure 9, there should have been a road name for the road adjacent to the north side of Mississippi river.
- **Example 6: Label placement error**
  In the upper left corner of figure 10, the road parallel to and between 2nd Street and 3rd Street should be Washington Avenue, not Cedar Ave. as we may infer from the road name placement on the map.
- **Example 7: Icon Placement error**
  In the central part of figure 11, Highway 35W does not show as a freeway icon, and icon 118 and 88 for country roads are associated with Highway 35W by mistake.
- **Example 8: Road-name inconsistency across segments**

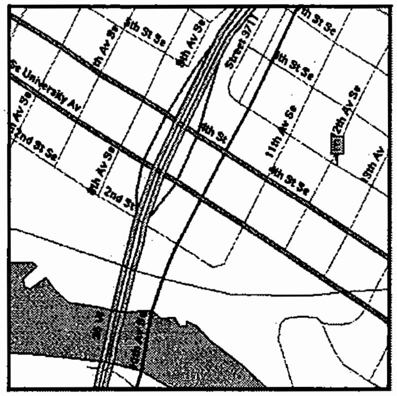


Figure 9: Non-Spatial error - Missing Feature(road name)

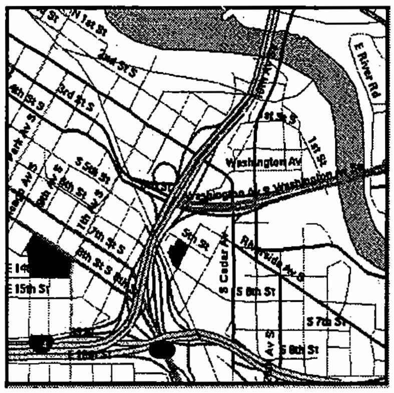


Figure 10: Label placement error

In Figure 12, we plan a route from 200 Union Street to 5001 Silver Lake Road. There is an inconsistency across 10, 11, and 12. The route goes from Silver Lake Road to Silver Lake Road NE, and back to Silver Lake Road, though the road is straight.

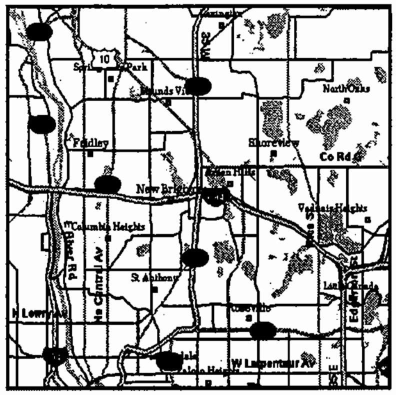


Figure 11: Icon-placement error

3	Less than .1 mi	TURN RIGHT onto CHURCH ST SE	0.3
4	0.1 mi	CONTINUE onto 17TH AV SE	0.4
5	0.1 mi	TURN LEFT onto US 52 (WEST)	0.5
6	0.6 mi	TURN RIGHT onto I 35 W (NORTH)	1.1
7	0.9 mi	BEAR RIGHT onto RAMP to NEW BRIGHTON BLVD	2.0
8	0.8 mi	TURN LEFT onto NEW BRIGHTON BLVD	2.7
9	1.3 mi	TURN LEFT onto SAINT ANTHONY BLVD	4.0
10	0.1 mi	TURN RIGHT onto SILVER LAKE RD	4.1
11	0.2 mi	CONTINUE onto SILVER LAKE RD NE	4.3
12	0.5 mi	CONTINUE onto SILVER LAKE RD	4.8

Figure 12: Road-name inconsistency across segments

6 Conclusion and future work

6.1 Accomplishment

In this WebRoute project, we designed a prototype user interface for displaying the shortest paths from source to destination. We use the JAVA language, which is platform-independent and allows the execution of interactive software across computer networks. Using Java, computation may be distributed across many machines, lowering the load on a single machine. We also evaluated other user interfaces available on the web sites to identify desirable functionalities. We found that MapOnUs is a good internet routing site, since it provides many convenient means for users to interactively choose a path. Our system goes a step forward in interactivity by providing an interactive map, on which users can click and choose their sources and destinations.

6.2 Future work

- Scale up to metro area map
- Develop new user interface for large map
- Change system architecture to client/server model

Since the large metro-area map contains large amounts of data, it is hard to display all of the desirable details in one interactive map. We plan to implement several display windows and to use zoom in and zoom out to control the map's hierarchical level. Client/Server architecture has better SC-ability.

References

[1] Bhasin, R (1994) "Rock Mass Characterization for Large Caverns in India and Norway Using a New Method of Recording and Presenting Geological Data", *International Journal of Rock Mechanics and Mining Sciences and Geomechanics Abstracts, Vol 31, Issue 1, pp87-91*

[2] Bretherton, WM, Jr (1994) "Accident Record Systems in Georgia", *ITE Journal, Vol 64, Issue 1, pp16-21*

[3] Deretsky, Z; Rodney, U; White, M (1993) "Conflation of Digital Maps", *IVHS America, pp105-108*

[4] Galer, M; Thomas, P; Bradford, M (1993) "Conducting In-Depth Accident Studies", *IVHS America, pp465-472*

[5] General Accounting Office (1995) "Aviation Safety: Data Problems Threaten FAA Strides on Safety Analysis System", *GAO-AIMD-95-27, 30p*

[6] Hentschel, W (1987), "Digital Maps for an Electronic Traffic Guide ", in German. *National Technical Information Service, Order No.: N88-25033/3/WTS.*

[7] Killick, J (1993) "Critical Data Standards Issues for IVHS", *IVHS America, pp465-472*

[8] Lerner-Lam, E; Francica, JR; Smith, WT; Rathi, A (1993) "Digital Map and Spatial Database Requirements for ATMS", *IVHS America, pp560-564*

[9] McElhaney, DR (1994) "An Update on Road Traffic Monitoring in the United States", *Seventh International Conference on Road Traffic Monitoring and Control, 26-28 April 1994. Proceedings, pp111-114*

[10] Mertikas, S; Wells, DE (1994) "The Estimation of Accuracy in Navigation", *Maritime Technical Information Facility Conference Paper, 19p*

[11] Queree, C (1994) "Travel Information Systems. Public Responsibility or Private Opportunity", *Towards an Intelligent Transport System. Proceedings of the First World Congress on Applications of Transport Telemetics and Intelligent Vehicle-Highway Systems, November 30-3rd December 1994, Paris, pp3097-3100*

[12] Spring, GS; Hummer, J (1995) "Identifying Hazardous Locations Using Geographic Information Systems. Final Report", *Southeastern Transportation Center - Region IV, 101p*

[13] Zavoli, WB (1993) "A System View of Digital Map Databases in IVHS and Its Implication for Standards", IVHS America, pp115-118

[14] Ning Jing, Yun-Wu Huang, and Elke Rudensteiner. "Hierarchical Optimization of Optimal Path Finding for Transportation Applications". In Proc. of ACM Conference on Information and Knowledge Management, 1996.

[15] S. Shekhar, A. Kohli, and M. Coyle. Path Computation Algorithms for Advanced Traveler Information Systems. In *Proc. of the 9th Intl. Conference on Data Engineering*, 1993.

[16] Shashi Shekhar, Andrew Fetterer, and Brajesh Goyal. A Comparison of Hierarchical Algorithms for Shortest Path Computation in Advanced Travel Information Systems. Technical Report 96-046, University of Minnesota, 1996.

Appendix A: Basic Concepts in Hierarchical Routing

A Basic Concepts in Hierarchical Routing

Traditional research examined the use of iterative breadth-first and best-first search algorithm. Another study examined A* and other estimator-based algorithms along with the effect of path lengths and edge costs on the relative performance of search algorithms. In these methods, time and space efficiency is a big problem, since they are all exponentially increasing in space. More research has been done on how to improve the efficiency of this computation, such as hierarchical graph routing and materialization. A fully materialized hierarchical algorithm for routing pre-computes and stores the shortest-path view and the shortest-path-cost-view for the graph fragments as well as for the boundary graph. The storage cost of the fully materialized approach can be reduced by a virtual or hybrid materialization approach, where none or few of the relevant views are pre-computed.

A.1 Flat-Graph Definitions

Graph $G = (N, E, C)$ is a flat graph consisting of a node set N, a cost set C, and an edge set E. The edge set E is a subset of the cross product $N \times N$. Each element (i, j) in E is an edge that joins node i to node j. Each edge (i, j) is associated with a cost $c_{i,j}$. Cost $c_{i,j}$ takes values from the set of positive real numbers. A node i is a neighbor of node j if $E_{i,j} \in E$. The degree of a node is the number of neighboring nodes. A path $P_{s,d}$, in a graph from a source node s to a destination node d, is a sequence of nodes (N_0, N_1, N_2, ..., N_k), where s = N_0, d = N_k, and the edges $E_{0,1}, E_{1,2}, ..., E_{k-1,k} \in E$. The cost of the path is the sum of the cost of the edges, i.e. $\sum_{i=1}^{k} c_{i-1,i}$. An optimal path from node i to node j is the path with the smallest cost.

A.2 Hierarchical Graph Definitions

The hierarchical graph has a 2-level representation of the original graph G. The lower level is composed of a set of fragments $F_1 \dots F_k$ of G. The higher-level graph is composed of the boundary nodes (BN) and is called the boundary graph (BG). Boundary nodes are defined as the set of nodes that have a neighbor in more than one fragment, i.e. $N_i \in BN \iff \exists E_{i,j}, E_{i,k} | FRAG(k) \neq FRAG(j)$. Edges in the boundary graph are called boundary edges, and the boundary nodes of a fragment form a clique, i.e. are completely connected. The cost associated with the boundary edge is the shortest-path cost through the fragment between the boundary nodes. A boundary edge is associated with a fragment identifier. A ***boundary path*** is the shortest path through the boundary graph.

Theorem 1 $BG.SP(s,d)rewrite[Edge \rightarrow] = G.SP(s,d)rewrite$ $[BoundaryNode{\rightarrow}BoundaryNode, InteriorNode{\rightarrow}, Edge \rightarrow]$ *if* s, d $inBN$.

Theorem 2 $p_g = (G.SP(s,d)rewrite[BoundaryNode{\rightarrow}\ BoundaryNode,\ InteriorNode{\rightarrow},\ Edge\ \rightarrow]) \in P_{set}$. *where* $P_{set} = \{p_{ij}rewrite[Edge \rightarrow] \mid p_{ij} = BG.SP(b_i, b_j),\ b_i \in BoundaryNodes(Fragment(s)),$
$b_j \in BoundaryNodes(Fragment(d))\}$

The shortest path $G.SP(s,d)$ between nodes s and d in G corresponds directly to the shortest path $BG.SP(s,d)$ between s and d in BG, where s and d are boundary nodes, as formalized in Theorem 1. The sequence of boundary nodes in $G.SP(s,d)$ is identical to that in $BG.SP(s,d)$, due to the

```
//BG = boundary graph
//CASE 1: Both Source s and Destination d are boundary nodes

boundaryPath = BG.GetPath(s,d);
path = ExpandBoundaryPath(boundaryPath);

//CASE 2: Source s is a local node and destination d is a
//         boundary node

c = INFINITY;
for each bN in BoundaryNodes(Fragment(s)) do
        if ((cpc = Fragment(s).SPC(s, bN) + BG.(bN,d)) < c) {
                c = cpc;
                minB = i;
        }
path = Fragment(s).SP(s, minB)
       + ExpandBoundaryPath(BG.SP(minB, d);

//CASE 3: Destination is a local node and source is a boundary
//         node
//Similar to CASE 2, but with source and destination reversed

//CASE 4: Both source and destination are local nodes
c = INFINITY;
for each sBN in BoundaryNodes(Fragment(s)) do
        for each dBN in BoundaryNodes(Fragment(d)) do
                if ((cpc=Fragment(s).SPC(s,sBN)
                     + BG.SPC(sBN,dBN)
                     + Fragment(d).SPC(dBN,d))< c) {
                                c = cpc;
                                minSB = sBN;
                                minDB = dBN;
                }
if (Fragment(s) == Fragment(d) && c > Fragment(s).SPC(s,d))
        path = Fragment(s).SP(s,d);
else
        path = Fragment(s).SP(s, minSB)
               + ExpandBoundaryPath(BG.SP(minSB, minDB))
        + Fragment(d).SP(minDB, d);
```

Figure 13: Hierarchical Routing Algorithm Template

construction of the boundary edges in BG. This observation can be generalized to the other node pairs in G. The sequence of boundary nodes in the shortest path $G.SP(s,d)$ between nodes s and d in G is the shortest path in BG between a boundary node in $fragment(s)$ and a boundary node in $fragment(d)$. This is stated in Theorem 2. The shortest path $G.SP(s,d)$ can be constructed by choosing a pair $<$ *a boundary node in* $fragment(s)$, *a boundary node in* $fragment(d)$ $>$, which minimizes the cost of $G.SP(s,sBN) + BG.SP(sBN,dBN) + G.SP(dBN,d)$.

Figure 13 gives a template algorithm for finding a path in a hierarchical graph. The following queries are sent to the database to compute the optimal path:

- **SP(s,d)** : This query returns the shortest path between nodes and d.
- **SPC(s,d)** : Query to find the cost of the shortest path from a source s to a destination d.
- **BoundaryNodes(f)** : Query that returns the set of boundary nodes for fragment f.
- **Fragment(n)** : This query returns the fragment identifier for interior node n.
- **ExpandBoundaryPath(boundaryPath)** : Query that expands the path through the boundary graph and returns the corresponding path in G by using the **ExpandBoundaryEdge(boundaryEdge)** query for each edge in the boundary path.
- **ExpandBoundaryEdge(boundaryEdge)** : This query expands a boundary edge and returns a corresponding path in G by computing the shortest path between the endpoints of the edge in the fragment.

The hierarchical algorithm is composed of three steps: finding the relevant boundary-node pair in the boundary graph, computing the boundary path, and expanding the boundary path. The first step in determining the shortest path is to compute the boundary node through which the shortest path leaves the source's fragment and enters the destination's fragment. If both the source and destination are boundary nodes, then it is trivial. If the source is an internal node and the destination is a boundary node, the boundary node through which the shortest path leaves the source's fragment is found by querying the fragment graph for the cost of the path from the source to all of the boundary nodes of that fragment, and by querying the boundary graph for the cost of the shortest path from all of the boundary nodes of the source's fragment to the destination. The source-boundary-destination triple with the lowest aggregate cost determines the appropriate boundary node. The case where the source is a boundary node and the destination is an internal node is similar, but the roles of the source and destination are reversed. When both the source and the destination are internal nodes, the appropriate boundary node pair is found by querying the fragment graphs to determine the cost of the shortest path from the internal nodes to all of the boundary nodes of the fragment. Next, the boundary graph is queried to compute the shortest path cost between all of the pairs of boundary nodes. The path with the lowest aggregate cost determines the boundary-node pair. Once the appropriate boundary-node pair has been determined, the boundary graph is queried to determine the shortest path between those boundary nodes. The final step is to expand the boundary path by querying the fragments for the shortest path through them. Adjacent nodes in the boundary path form source/destination pairs on which the shortest-path query can be run in a fragment.

Lemma 1 *The hierarchical routing algorithm finds an optimal path from s to d.*

Appendix B: Degree of Materialization in Hierarchical Routing Algorithms

B Degree of Materialization in Hierarchical Routing Algorithms

B.1 View Materialization Terminology

A cost view (*CV*) on a graph materializes the cost of the shortest path between all of the node pairs in the graph. It does not store any path information. A compressed path view *CPV* stores the set of optimal paths between all points on the graph as a series of "hops". For a fragment graph, a partial materialization of the *CV*, the *C2B* or cost-to-boundary-nodes view stores the cost of the shortest path from the interior nodes of the fragment to the boundary nodes of the fragment.

Figure 14 shows *CPV* and *CV* for an example graph with 5 nodes and 8 edges. Each node is associated with a table, with one entry for all other nodes. The node table in *CV* contains the costs of the shortest paths between two nodes. For example, the cost from node C to node A is given as 5. To find the shortest path between those two nodes, the *CPV* table is used. Starting at the table for node C, the destination is looked up to get the next node on the path. In this case, the next destination is node E, and the *CPV* table for node E is used. From that table, node A is found to be the next hop, and the destination is reached.

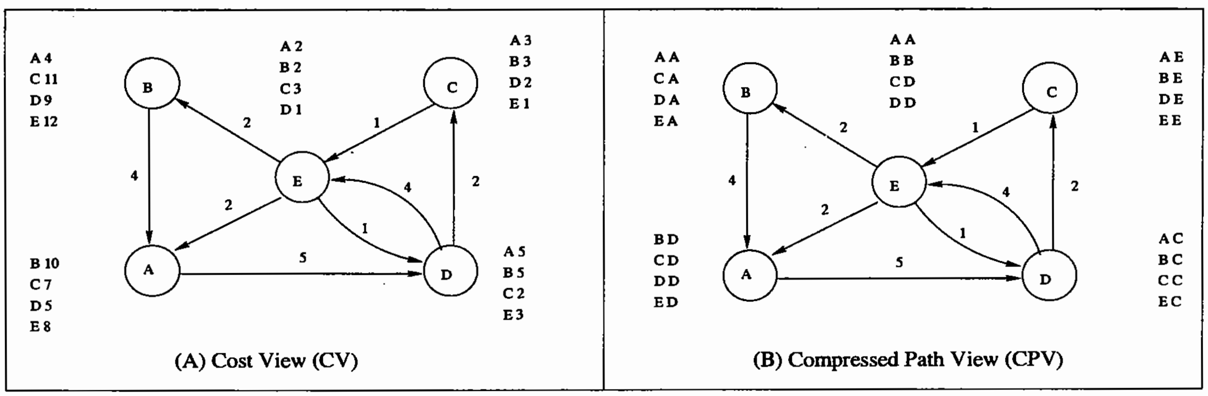


Figure 14: Path Closure and Cost Closure Examples

B.2 Asymptotic Analysis of Storage Cost Using a Grid Graph

	Adjacency list information		Compressed Path View		Cost View		Cost2-Boundary-Nodes
	per fragment graph	boundary graph	per fragment graph	boundary graph	per fragment graph	boundary graph	
Raw Storage	$F{*}G_f$	G_b	$N(\frac{N}{F}-1){*}2S_{id}$	$B(B-1){*}3S_{id}$	$N(\frac{N}{F}-1){*}(S_c+S_{id})$	$B(B-1){*}(S_c+S_{id})$	$(N-B)K{*}(S_c+S_{id})$
Asymptotic Storage	$O(N)$	$O(N)$	$O(N^{5/3})$	$O(N^{4/3})$	$O(N^{5/3})$	$O(N^{4/3})$	$O(N^{4/3})$
Real Size F=100	4M	3M	2.3G	32M	2.3G	32M	40M

Table 2: Asymptotic Size and Complexity of the Materialized Views

Symbol	Explanation	Symbol	Explanation
N	Number of nodes	S_c	size of storing a cost
S_{id}	Size of storing a node identifier	B	Number of boundary nodes
F	Number of fragments	G_b	Size of boundary graph in bytes
K	Average number of border nodes per fragment	L	Length of path through base graph
M	Length of path through boundary graph		

Table 3: Table of Notations

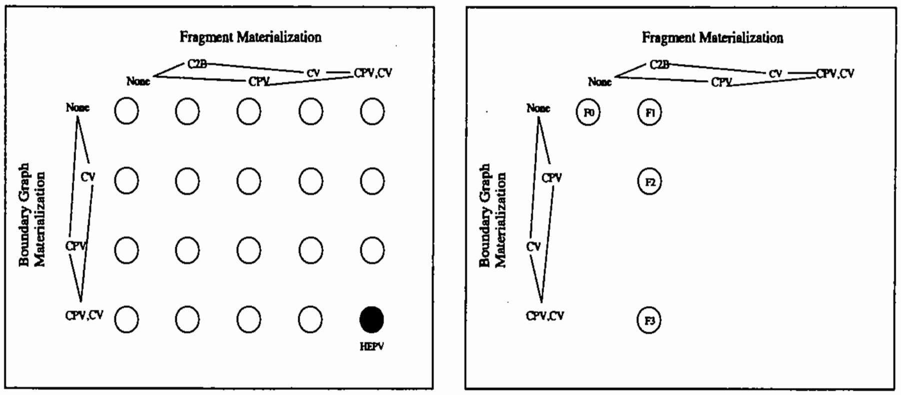
A summary of the parameters used in the remainder of the paper is given in Table 3. A square grid-graph of N nodes is an undirected planar graph. If $\sqrt{N}$ is an integer, the square grid-graph can be embedded in a Cartesian space with nodes at coordinates $(i * \sqrt{N}, j * \sqrt{N})$ with $i, j \in 0, 1, 2, ..., \sqrt{N} - 1$. Edges connect nodes $(i\sqrt{N}, j\sqrt{N})$ to nodes $((i+1)\sqrt{N}, j\sqrt{N})$ and $(i\sqrt{N}, (j+1)\sqrt{N})$. A grid-graph is often used in complexity analysis for transportation algorithms because of its regular structure and its similarity to urban road-maps [15].

In a graph decomposed into F fragments, each fragment has approximately N/F nodes. The average number of boundary nodes for each fragment is $K = 4\sqrt{N/F}$. The total number of boundary nodes is approximately $B = \sqrt{NF}$. Table 2 shows the view storage requirements. The storage formulas for the materialized views are shown first and are given in terms of N, B, K, and constants. F can be chosen to minimize the storage of the materialized views, as is studied in [16]. Setting F in terms of N, the asymptotic storage sizes for the views are computed. When $F = N^{1/3}$, the storage that minimizes the size of *CV* and *CPV* for the boundary graph, the values $K = N^{1/3}$ and $B = N^{2/3}$ are used to compute the asymptotic storage sizes for the views. The asymptotic analysis shows that the sizes of the *CPV* and *CV* views for the fragments are $O(N^{1.66})$, which is larger than the *CPV* and *CV* for the boundary graph and *C2B* for the fragments which grow at $O(N^{1.33})$. This is verified in the real sizes of the views given for $F = 100$ ($F \approx N^{2/5}$). The *CPV* and *CV* views for the fragments take two gigabytes of storage, while the other views are two orders of magnitude smaller, as shown in Table 2. We will not consider the materialization of the *CPV* on the fragments further in this paper. The size of the *CPV* for the base graph is $O(N^2)$, which would take about 100G to store.

B.3 Levels of Hybrid Materialization

A fully materialized view has all of the relevant information pre-computed and stored, while a virtual view has no information pre-computed. Hybrid materialization forms the middle ground between those two approaches: some of the relevant data is materialized, while some is left for computation. In terms of the hierarchical routing algorithm, three different levels of hybrid materialization could be used: table level, graph level, and algorithm level. Hybrid materialization with respect to a single table means that only part of the table is kept as a materialized view. An example of this is the *C2B* view which materializes only a part of the *CV* table for the fragment graphs. In this case, the *CV* table is fully materialized because it stores cost information between all pairs, while the *C2B* table only stores cost information between internal nodes and boundary nodes. The graph-level materialization chooses to materialize all of the views related to either the boundary graph or the fragment graphs, but not to both. Means that only materialization at the graph level are considered. The third definition of hybrid materialization is when we only materialize the tables necessary for the version of the hierarchical algorithm under consideration. Full materialization of all views for the hierarchical algorithm was researched in [14].

The hybrid materialization of the hierarchical algorithm (the focus of the remainder of the paper) will materialize a subset of the CV and CPV for the boundary graph and fragment graphs. Figure 15(a) shows the possible hybrid materializations for the hierarchical algorithm.



(a) Materialization Matrix

(b) Candidate Hybrid Materializations for Experiments

Figure 15: Degree of Materialization in Hierarchical Routing Algorithms