High Performance Spatial Visualization of Traffic Data

Final Report

Prepared by

Shashi Shekhar, Chang-Tien Lu, Alan Liu

Computer Science Department
University of Minnesota
Minneapolis, MN. 55455

Feburary 2002

Submitted to

Minnesota Department of Transportation
Office of Research Services
395 John Ireland Boulevard, MS 330
St. Paul, Mn. 55155

The contents of this report reflect the views of the authors, who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the views or policies of Minnesota Department of Transportation at the time of publication. This report does not contitute a standard, specification, or regulation. The authors and the Minnesota Department of Transportation do not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because the are considered essential to this report.

Abstract

The objectives of this project are to develop high performance spatial tools and techniques to generate critical visualizations of loop-detector traffic data collected at the Traffic Management Center in Minneapolis, Minnesota, and to explore new spatial data structures and algorithms to address the performance bottlenecks in the process of producing novel interactive traffic visualization.

In this project, we constructed a web-based video-like visualization software package for observing rapid summarization of major trends. This package can be used to visualize the effects of a sudden increase in load on the traffic network after scheduled events for the planning of traffic management for future similar events. In the underlying database, we modeled the traffic data as a data warehouse to facilitate the query engine for on-line analytical processing used in the visualization software. We also extend our visualization package to support several data mining techniques, e.g., clustering, classification, and outlier detection. In addition, we identified the performance bottlenecks in the generations of various visualizations and developed efficient algorithms to address the bottlenecks.

This web-based software system is available at:
http://www.cs.umn.edu/research/shashi-group/vis/traffic.htm

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The visualization of loop-detector traffic data can help in the identification of potentially important patterns embedded in the data. Many of the current visualization techniques do not scale to large data sets and are not practical for interactive visualization and “what if” analysis. The goal of the project is two-fold. First, we develop spatial alogithms which can help speed-up the processing time of visualization algorithms. Second, we will provide a data warehousing framework for integrating different multi-dimensional views of traffic data. In this chapter, we introduce the background and objectives of this project, and the four tasks of the planned work, namely, requirement analysis, bottleneck identification, visualization algorithm design, and prototype software development. We also give a brief suvey of current traffic visualization system.

1.1 Background

High-performance visualization techniques are becoming crucial as the amounts of traffic data collected by an ever increasing network of sensors, is becoming too large to be analyzed manually than the ability to analyze the data manually. For example, producing a three dimensional visualization of speed as a function of time and highway network space for a single day's worth of traffic data can take up to a week using the current tools. The performance bottleneck forces visualizations to be limited to a very small sample of traffic data out of the data set collected by the sensors. This bottleneck makes it hard for traffic researchers to perform interactive visualizations for asking "what-if" questions. Interactively changing the variables selected for visualization or drilling down to smaller subnets (e.g. a traffic zone, a corridor, a lane, single hour) for further details is difficult due to the performance bottlenecks caused by the large volume of traffic data.

1.2 Objectives

The objective of this project is to develop high performance spatial tools and techniques to generate critical visualizations of loop-detector traffic data collected at the Traffic Management Center. This project aims at developing new spatial data-structures and algorithms to address the performance bottlenecks in the process of producing and revising current as well as novel interactive visualization including some of the following:

- Summary traffic maps of the Twin Cities metropolitan area for a variety of aggregate information including annual traffic volume and daily hours of congestion.
- Multi-dimensional surface visualization such as speed as a function of time and highway network location.
- Techniques to visualize relationships between traffic attributes(e.g. volume and occupancy) over different dimensions(e.g. space and time)
- A video-like visualization of traffic data for an approximate but rapid summarization of major trends. This can also be used to visualize the effects of a sudden increase in load on the traffic network after scheduled events so that traffic management can plan for future similar events.

The spatial techniques will enable visualization of the basic loop-detector data(i.e. volume, occupancy) as well as information derived from traditional and novel analysis methods. Derived information may include identification of cyclical and recurring patterns embedded in the traffic data.

It may also include classification of of major sources, sinks and bottlenecks in the road network.For example, it is well known that the downtown Minneapolis, depending upon the time of day, is the biggest sink or source of traffic in the Twin-Cities road network. What is not obvious is that the second biggest source-sink is the area surrounding I-494 between Bloomington and Edina and not the downtown St. Paul. Such analysis is an example of spatial clustering, a data mining technique which has the potential of discovering "nuggets" of information otherwise hidden in the data.

The specific objectives of the project are these: a) To review the relationship between visualization algorithms and scalability in order to identify the performance bottlenecks. b) To organizae the multidimensional attribute traffic space into a lattice framework for better understanding. c) To use the lattice framework to efficiently produce summary plots(including video) which may be beneficial to the traffic analyst.

First, we will examine the relationship between visualization and scalability in order to identify the performance bottlenecks. After we examine the relationship, we will present an algorithm which satisfies the standard data mining objectives [3]:

Avoid Repeated Scan: The visualization algorithm should require atmost one scan of the database.

Anytime Algorithm: The visualization is able to produce the "best" results at anytime during the computation.

Limited RAM: The algorithm works with a limited RAM and buffer allocated by the user.

Incremental: The algorithm proceeds in an incremental fashion: In the presence of new data the algorithm can use previous results without starting the computation afresh.

Forward-only cursor: The data being visualized may be a result of an expensive join algorithm over a potentially distributed data warehouse. Thus the algorithm must operate with a forward-only cursor over a view of the database.

We will organize the multi-dimensional space of traffic attributes into a lattice structure. This will help in the identification of important nodes in the lattices. Each node of the lattice can potentially generate a new visualization schema. We will identify the redundant nodes and optimize our algorithm generation by using summary information from parent/child nodes.

Through interviews with a traffic expert we will identify the important nodes of the lattice and use them to generate static and dynamic summary plots. A software implementation of results will be used to validate our approach.

1.3 Tasks

We divided our work into four tasks, namely, requirement analysis, bottleneck identification, visualization algorithm development, and prototype software construction.

1.3.1 Task 1: Requirement analysis

We met with officials and researchers at the TMC to identify the important visualization requirements of loop-detector data. We also studied the work flow of generating, revising, and repurposing of visualizations at TMC. We queried TMC researchers to understand which visual metaphors may be most suitable for the analysis of traffic data. For spatial data, the "map" has been the oldest, most prolific and successful handle for representing spatial data. Thus we will determine the ways that maps may be extended to integrate conceptual traffic dimensions.

1.3.2 Task 2: Identification of bottleneck in visualization

We started with a specific visualizations, e.g., maps, towards the identification of performance bottlenecks. This was possible due to the development of a map based user-friendly interface for the traffic database, which has been recently developed at CTS by PI for TMC researchers. The map based interface allows interactive generation of map based visualization, such as the summary maps for annual traffic volumes or daily hours of congestion. We used "code profiling" and other performance measurement techniques to identify the performance bottlenecks in the generations and revisions of the visualizations. For example, we determined if it may be profitable(performance-wise) to separate out the visual mining from the non-visual mining by duplicating the database even though this may potentially lead to consistency problems in the database. We also tried and leverage our work from previous projects on data-achiving to reduce performance bottlenecks.

1.3.3 Task 3: Develop visualization algorithms for spatially index data

We developed spatial data-structures and algorithms to address the performance bottlenecks in the generation of various visualizations. For example, the generation of a summary map visualization may take advantage of the aggregate descriptions (e.g., annual traffic volume). A possible way to speed up such queries is to maintain aggregate information such as daily traffic volumes. Similar performance improvement opportunities will be identified for generation of other visualizations, e.g. multi-dimensional visualization of speed as a function of time and highway network space. We developed a lattice framework where each node corresponds to a subspace of the multi-dimensional conceptual space. A lattice framework is conducive to "what-if" scenarios and allows integration into other high performance visual kernels such as Vega. We also tried to address the important question of the validity of results derived from aggregate data. As is well known, analysis of aggregate data can easily lead to conclusions which are contradictory to those arrived on the base data. We investigated whether the recent wavelet compression techniques proposed may, to a certain extent, help in ameliorating this problem. (For example, by mapping the problem of aggregation (compression) to that of scale.)

1.3.4 Task 4: Prototype software development

We validated our results via a software implementation. We determined if our software development can be seamlessly integrated with the proposed mini-CAVE display medium in the ITS lab. The database was installed in the ITS laboratory. Part of the research was on the Computer Science department machines. We will work with TMC staff as well as ITS staff to ensure compatibility. We explored web-based development to reduce protability issues.

1.4 Literature Survey

There have been many studies which focus on the visualization of traffic data, though none of them address the constraints of large data sets on visualization. The research can be divided into four categories: visualization for the specialist end user, visualization for the non-expert end user, the role of a GIS, and higher dimensional (three and above) visualization. Hill et al. [14] summarizes the Traffic Flow Visualization and Control(TFVC) system under development for the Long Island Expressway on Long Island. They conclude that accurate real-time visualization of traffic data can aid in incident detection and management, which is one of the critical elements of Advanced Traffic Management System(ATMS).

Prevedourous et al. [24] have developed a visualization system for the non-expert end user. They claim that their product does not require any training for users to understand the displayed results. The roadway layouts and vehicles are realistic and the adjacent land use are in full display.

The role of Geographic Information System(GIS) to aid traffic visualization has been the subject of many studies. GIS allow for effective visualization of spatially referenced data. In [12], a prototype software system is proposed for visualization of dynamic traffic data. The system provides controls which allow the user to filter, symbolize and replay the data to reveal patterns and trends over varying time spans. Baecher et al. [2] have addressed the issue of rapidly integrating data available in different formats into a common visualization tool. The easy availability of GPS systems was the motivation behind the study by Quiroga et. al [27] to integrate GPS and GIS data to study the problem of traffic congestion. They conclude that by limiting the aggregation of GPS data to 0.2 miles, considerale savings in data storage can be achieved while at the same time allowing for a consistent analysis of traffic scenarios.

One of the fundamental limitations of current GIS is its inability to handle multi-dimensional data. In [4] the authors first present an overview of the common problems associated with the rendering of three dimensional data. They note that visualization of data must honor the original data points, take account of heterogeneity of the medium where the data was collected, and make transparent the effects of various interpolating and extrapolating methods employed. They then suggest practical advice about overcoming these aforementioned limitations. The issue of complexity, hardware and software requirements, production time and cost associated with the visualization of transportation data is addressed by [22].

Combining data mining with a GIS was the focus of a study conducted by Chase et al. [5]. The National Bridge Inventory Database(NBI) contains 6.2 billion bytes of data on highway and bridges in the United States. The authors used data mining, datawarehousing and spatial visualization techniques(GIS) to discover implicit patterns and relationships embedded in the data. Using the newly discovered relationships, a model was proposed to link the relationship between bridge deterioration and climate conditions. Generalized linear and additive regression techniques were used to build the model.

Chapter 2

Task 1: Requirement Analysis

For spatial data, the “map” has been the oldest, most prolific and successful means of representing spatial data. The focus in this chapter is to identify important visualization requirements which will help transportation professionals to analyze traffic flow and patterns. In this chapter, we analyze the different requirements from different aspects and introduce the Twin-Cities traffic data archives, the concept of data warehouse and its support of traffic data visualization, and discuss several essential visualization utilities. In addition, we extend the visualization utilities to support data mining techniques, e.g., classification, clustering, and outlier detection.

2.1 Twin-Cities Traffic Data

In 1995, the University of Minnesota and the Traffic Management Center(TMC) Freeway Operations group started the development of a database to archive sensor network measurements from the freeway system in the Twin Cities. The sensor network includes about nine hundred stations, each of which contains one to four loop detectors, depending on the number of lanes, as shown in Figure 2.2(a) Sensors embedded in the freeways and interstate monitor the occupancy and volume of traffic on the road. At regular intervals, this information is sent to the Traffic Management Center for operational purposes, e.g., ramp meter control, as well as research on traffic modeling and experiments. Figure 2.1 shows a map of the stations on highways within the Twin-Cities metropolitan area, where each polygon represents one station. The interstate freeways include I-35W, I35E, I-94, I-394, I-494, and I-694. The state trunk highways include TH-100, TH-169, TH-212, TH-252, TH-5, TH-55, TH-62, TH-65, and TH-77. I-494 and I-694 together form a ring around the Twin-Cities. I-94 passes from East to North-West, I-35W and I-35E run in a South-North direction. Minneapolis downtown is located on the intersection of I-94, I-394, and I-35W, and downtow St. Paul is located at the intersection of I-35E and I-94.

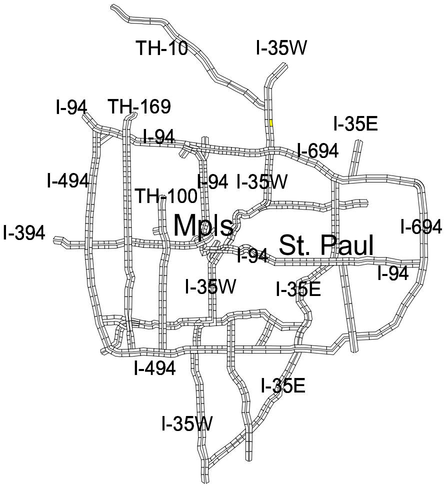
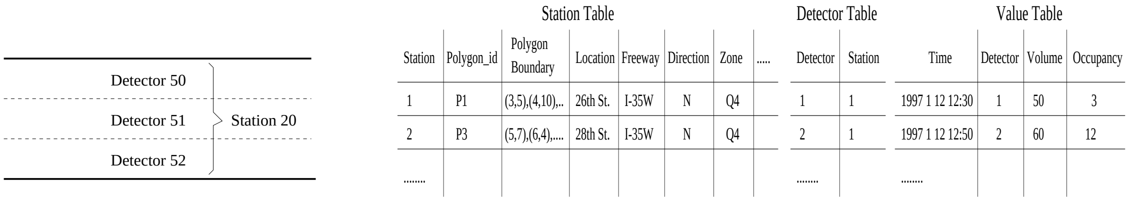


Figure 2.1. Detector map at the station level

Figure 2.2(b) shows the three basic data-tables for the traffic data. The *station* table stores the geographical location and some related attributes for each station. The relationship between each detector and its corresponding station is captured in the *detector* table. The *volume* table records all the volume and occupancy information within each 5-minute time slot at each particular station.



Station Table

Station	Polygon_id	Polygon Boundary	Location	Freeway	Direction	Zone	.....
1	P1	(3,5),(4,10),..	26th St.	I-35W	N	Q4	
2	P3	(5,7),(6,4),....	28th St.	I-35W	N	Q4	
....							

Detector Table

Detector	Station
1	1
2	1
....	

Value Table

Time	Detector	Volume	Occupancy
1997 1 12 12:30	1	50	3
1997 1 12 12:50	2	60	12
....			

(a) Relationship between detectors and stations

(b) Three basic tables

Figure 2.2. Detector-station Relationship and Basic Tables

We use Figure 2.3 to illustrate data flows and the required modules of our system. The basic map and raw data are cleaned, transformed, and loaded into the data warehouse module, which provides the multidimensional views and the OLAP operations for data visualization, as well as a variety of data mining analysis tools, e.g, classification, clustering, outlier detection. The discovered patterns or rules are then visually displayed as maps or charts for further interpretation. We describe the underlying data warehouse component, visualization components, and data mining techniques in forthcoming subsection.

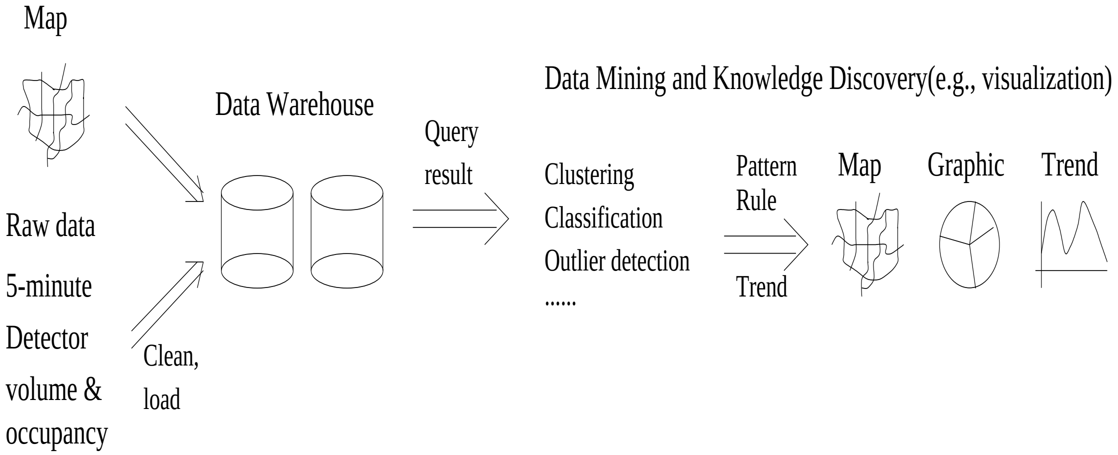


Figure 2.3. Data-flow and main modules in our system

2.2 Users of Transportation Traffic Visual Tools

Transportation Managers

Transportation managers generally want to know the current performance of the traffic and compare it to historical performance. He can make it by using this web-based traffic tools to display the traffic video in some day and the average traffic vedeo in the same weekday in history. If he finds something spectial, he might initiate a query about where location that the abnormal situation happens is. Which locations are worst performer. He might allocate resources to improve the worst performance or he can send this information to radio station for broadcasting.

Traffic Engineers

When traffic engineers get report that abnormal performance appear either from car drivers or from the intelligent traffic system, he might use outlier analysis tools to find which detectors did not work or he likes to know how congestion start or spread.

Travelers and Commuters

Travelers are most intereted to know where the congestion happen in real time and is it possible for them to make to the destination in time for a meeting or event such as basket ball game.

Researchers and Planners

It is impossible for state to spend millions dollars to build more and more freeways to relieve the traffic congestion. Planner might wonder how the information techniques can be used to reduce congestion. How the ramps and meters are located to help improve the traffic performance?

2.3 Data Warehouse Support

A data warehouse(DW) [6,7,15–17,19,18] is a repository of subject-oriented, integrated, and non-volatile information, aimed at supporting knowledge workers(executives, managers, analysts) to make better and faster decisions. Data warehouses contain large amounts of information which are collected from a variety of independent sources and is often maintained separately from the operational databases. Data warehouses maintain historical, summarized, and consolidated information, and are designed for on-line analytical processing (OLAP) [10,11]. The data in the warehouse are often modeled as a multidimensional space to facilitate the query engines for OLAP, where queries typically aggregate data across many dimensions in order to detect trends and anomalies [23].

There is a set of numeric measures that are the subjects of analysis in a multidimensional data model. Each of the numeric measures is determined by a set of dimensions. In a traffic data warehouse, for example, the measures are volume and occupancy, and the dimensions are *time* and *space*. Dimensions are hierarchical by nature. In Figure 2.4, for example, the *time* dimension can be grouped into "Hour","Date","Month","Week","Season", or "Year", which form a lattice structure indicating a partial order for the dimension. Similarly, the *Space* dimension can be grouped into "Station","County","Freeway", or "Region". Given the dimensions and hierarchy, the measures can be aggregated in different ways. The SQL aggregate functions and the group-by operator only produce one out of all possible aggregates at a time. A data cube [13] is an aggregate operator which computes all possible aggregates in one shot.

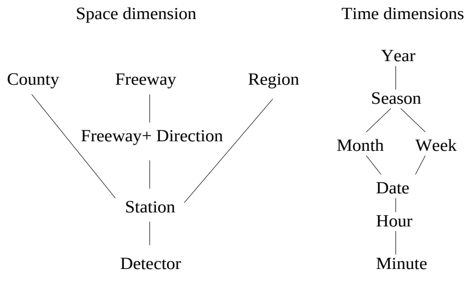


Figure 2.4. Concept Hierarchies for Dimensions

2.3.1 Basic Concepts

Aggregation hierarchy: The CUBE operator [13] generalizes the histogram, cross-tabulation, roll-up, drill-down, and sub-total constructs. It is the N-dimensional generalization of simple aggregate functions. Figure 2.5 shows the concept for aggregations up to 3-dimensions. The dimensions are Year, Highway, and Month. The measure is sales. The 0D data cube is a point which shows the total summary. There are three 1-D data cubes: Group-by Highway, Group-by Month, and Group-by Year. The three 2-D data cubes are cross tabs, which are a combination of these three dimensions. The 3D data cube is a cube with three intersecting 2D cross tabs. Figure 2.6 shows the tabular forms of the total elements in a 3D data cube after a CUBE operation. Creating a data cube requires generating a power set of the aggregation columns.

A tabular view of the individual sub-space data-cubes of Figure 2.5 is shown in Figure 2.7. The union of all the tables in Figure 2.7 yields the resulting table from the data cube operator. The 0-dimensional sub-space cube labeled “Aggregate” in Figure 2.5 is represented by Table “VOLUMES-L2” in Figure 2.7. The one-dimensional sub-space cube labeled “By Month” in Figure 2.5 is represented by Table “VOLUMES-L1-C” in Figure 2.7. The two-dimensional cube labeled “By Month & Year” is represented by Table “VOLUMES-L0-C” in Figure 2.7. Readers can establish the correspondence between the remaining sub-space cubes and tables.

The cube operator can be modeled by a family of SQL queries using *GROUP BY* operators and aggregation functions. Each arrow in Figure 2.7 is represented by a SQL query. In Table 2.1,

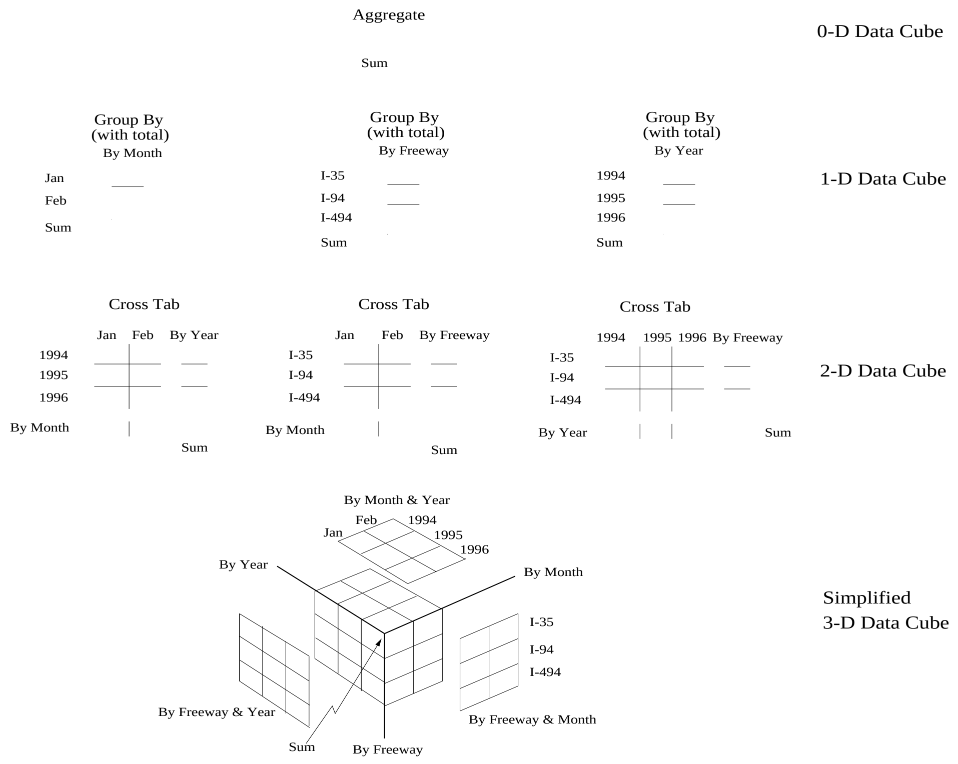


Figure 2.5. The 0-D, 1-D, 2-D, and 3-D data cubes [13]

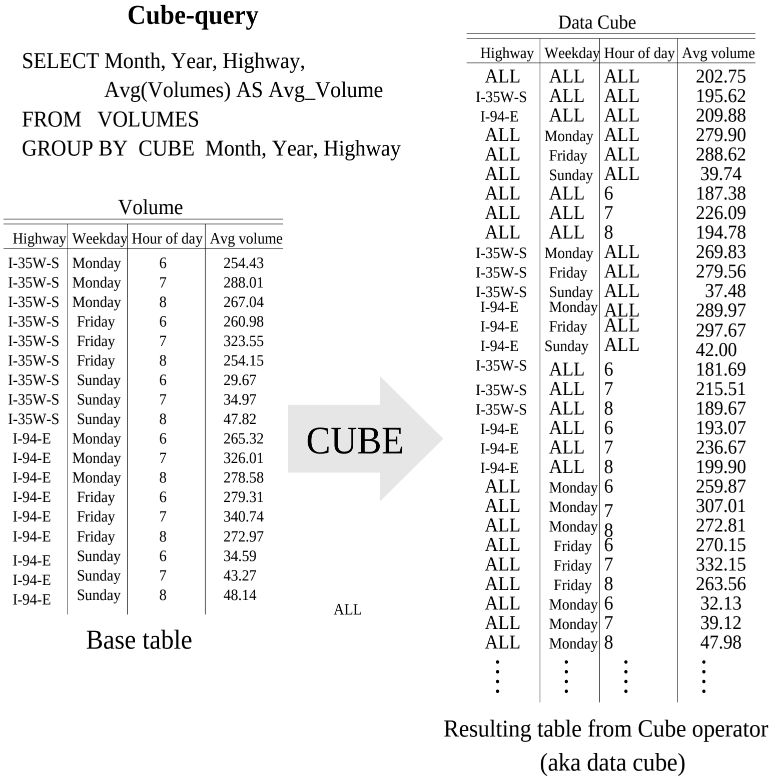
we provide the corresponding queries for the five arrows labeled $Q1, Q2, \ldots, Q5$ in Figure 2.7. For example, query $Q1$ in Table 2.1 aggregates "Volumes" by "Year" and "Freeway," and generates Table "VOLUMES-L0-A" in Figure 2.7. From this table, we can see that 7AM is the busiest time on weekdays. By contrast, 8AM is the busiest time of the day.

The *GROUP BY* clause specifies the grouping attributes which should also appear in the *SELECT* clause so that the value resulting from applying each function to a group of tuples appears along with the value of the grouping attribute(s).

Q1	**SELECT** 'ALL', Year, Freeway, **SUM**(Volumes) **FROM** VOLUMES-Base **GROUP BY** Year, Freeway
Q2	**SELECT** 'ALL', 'ALL', Freeway **SUM**(Volumes) **FROM** VOLUMES-L0-A **GROUP BY** Freeway
Q3	**SELECT** 'ALL', 'ALL', 'ALL' **SUM**(Volumes) **FROM** VOLUMES-L1-A
Q4	**SELECT** 'ALL', 'ALL', Freeway, **SUM**(Volumes) **FROM** VOLUMES-Base **GROUP BY** Freeway
Q5	**SELECT** 'ALL', 'ALL', 'ALL', **SUM**(Volumes) **FROM** VOLUMES-Base

Table 2.1. Table of **GROUP BY** queries

What is an Aggregation Hierarchy used for? : To support OLAP, the data cube provides



Cube-query

```
SELECT Month, Year, Highway,
          Avg(Volumes) AS Avg_Volume
FROM   VOLUMES
GROUP BY  CUBE  Month, Year, Highway
```

Volume

Highway	Weekday	Hour of day	Avg volume
I-35W-S	Monday	6	254.43
I-35W-S	Monday	7	288.01
I-35W-S	Monday	8	267.04
I-35W-S	Friday	6	260.98
I-35W-S	Friday	7	323.55
I-35W-S	Friday	8	254.15
I-35W-S	Sunday	6	29.67
I-35W-S	Sunday	7	34.97
I-35W-S	Sunday	8	47.82
I-94-E	Monday	6	265.32
I-94-E	Monday	7	326.01
I-94-E	Monday	8	278.58
I-94-E	Friday	6	279.31
I-94-E	Friday	7	340.74
I-94-E	Friday	8	272.97
I-94-E	Sunday	6	34.59
I-94-E	Sunday	7	43.27
I-94-E	Sunday	8	48.14

Base table

Data Cube

Highway	Weekday	Hour of day	Avg volume
ALL	ALL	ALL	202.75
I-35W-S	ALL	ALL	195.62
I-94-E	ALL	ALL	209.88
ALL	Monday	ALL	279.90
ALL	Friday	ALL	288.62
ALL	Sunday	ALL	39.74
ALL	ALL	6	187.38
ALL	ALL	7	226.09
ALL	ALL	8	194.78
I-35W-S	Monday	ALL	269.83
I-35W-S	Friday	ALL	279.56
I-35W-S	Sunday	ALL	37.48
I-94-E	Monday	ALL	289.97
I-94-E	Friday	ALL	297.67
I-94-E	Sunday	ALL	42.00
I-35W-S	ALL	6	181.69
I-35W-S	ALL	7	215.51
I-35W-S	ALL	8	189.67
I-94-E	ALL	6	193.07
I-94-E	ALL	7	236.67
I-94-E	ALL	8	199.90
ALL	Monday	6	259.87
ALL	Monday	7	307.01
ALL	Monday	8	272.81
ALL	Friday	6	270.15
ALL	Friday	7	332.15
ALL	Friday	8	263.56
ALL	Monday	6	32.13
ALL	Monday	7	39.12
ALL	Monday	8	47.98
⋮	⋮	⋮	⋮

Resulting table from Cube operator
(aka data cube)

Figure 2.6. An example of data cube

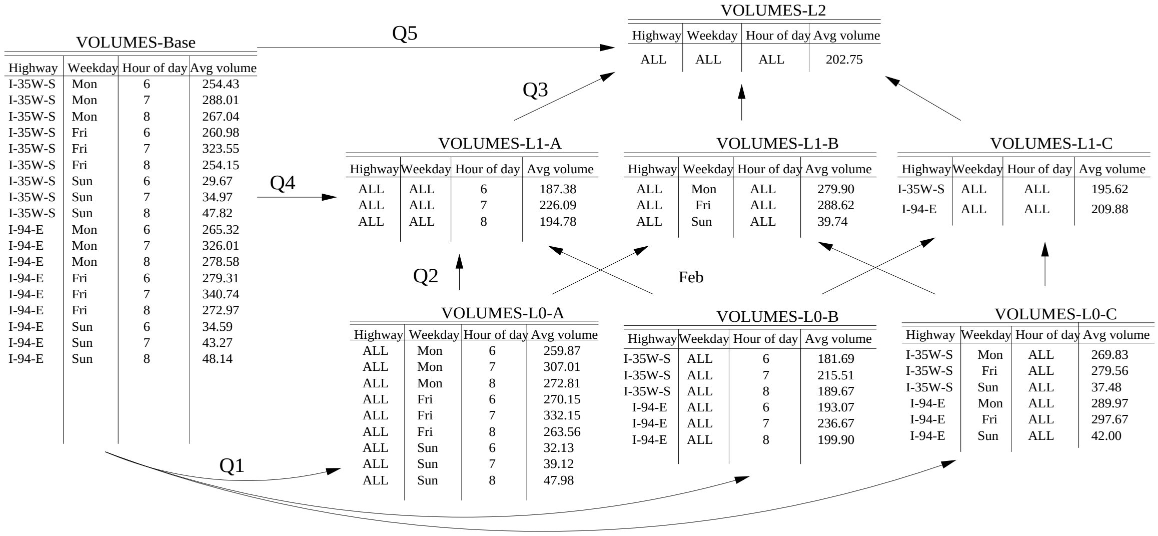
the following operators : roll-up, drill-down, slice and dice, and pivot. We now define these operators.

- Roll-up: increasing the level of abstraction. This operator generalizes one or more dimensions and aggregates the corresponding measures. For example, Table VOLUMES-L0-A in Figure 2.7 is the roll-up of Table VOLUMES-Base on the Highway dimension.

- Drill-down: decreasing the level of abstraction or increasing detail. It specializes in one or a few dimensions and presents low-level aggregations. For example, Table VOLUMES-L0-A in Figure 2.7 is the drill-down of Table VOLUMES-L1-A on the Year dimension. Similar to Table VOLUMES-L0-A, 7AM is the busiest time.

- Slice and Dice: selection and projection. Slicing into one dimension is very much like drilling one level down into that dimension, but the number of entries displayed is limited to that specified in the slice command. A dice operation is like a slice on more than one dimension. On a 2-dimensional display, for example, dicing means slicing on both the row and column dimensions.

  Table 2.2 shows the result of slicing for "I-35W-S" on the highway dimension from the Table VOLUMES-L2 in Figure 2.7.



VOLUMES-Base

Highway	Weekday	Hour of day	Avg volume
I-35W-S	Mon	6	254.43
I-35W-S	Mon	7	288.01
I-35W-S	Mon	8	267.04
I-35W-S	Fri	6	260.98
I-35W-S	Fri	7	323.55
I-35W-S	Fri	8	254.15
I-35W-S	Sun	6	29.67
I-35W-S	Sun	7	34.97
I-35W-S	Sun	8	47.82
I-94-E	Mon	6	265.32
I-94-E	Mon	7	326.01
I-94-E	Mon	8	278.58
I-94-E	Fri	6	279.31
I-94-E	Fri	7	340.74
I-94-E	Fri	8	272.97
I-94-E	Sun	6	34.59
I-94-E	Sun	7	43.27
I-94-E	Sun	8	48.14

VOLUMES-L2

Highway	Weekday	Hour of day	Avg volume
ALL	ALL	ALL	202.75

VOLUMES-L1-A

Highway	Weekday	Hour of day	Avg volume
ALL	ALL	6	187.38
ALL	ALL	7	226.09
ALL	ALL	8	194.78

VOLUMES-L1-B

Highway	Weekday	Hour of day	Avg volume
ALL	Mon	ALL	279.90
ALL	Fri	ALL	288.62
ALL	Sun	ALL	39.74

VOLUMES-L1-C

Highway	Weekday	Hour of day	Avg volume
I-35W-S	ALL	ALL	195.62
I-94-E	ALL	ALL	209.88

VOLUMES-L0-A

Highway	Weekday	Hour of day	Avg volume
ALL	Mon	6	259.87
ALL	Mon	7	307.01
ALL	Mon	8	272.81
ALL	Fri	6	270.15
ALL	Fri	7	332.15
ALL	Fri	8	263.56
ALL	Sun	6	32.13
ALL	Sun	7	39.12
ALL	Sun	8	47.98

VOLUMES-L0-B

Highway	Weekday	Hour of day	Avg volume
I-35W-S	ALL	6	181.69
I-35W-S	ALL	7	215.51
I-35W-S	ALL	8	189.67
I-94-E	ALL	6	193.07
I-94-E	ALL	7	236.67
I-94-E	ALL	8	199.90

VOLUMES-L0-C

Highway	Weekday	Hour of day	Avg volume
I-35W-S	Mon	ALL	269.83
I-35W-S	Fri	ALL	279.56
I-35W-S	Sun	ALL	37.48
I-94-E	Mon	ALL	289.97
I-94-E	Fri	ALL	297.67
I-94-E	Sun	ALL	42.00

Figure 2.7. An example of group-by

Highway	Weekday	Hour of day	Avg volumes
I-35W-S	ALL	ALL	195.62

Table 2.2. Slice on the value I-35W south of the Highway dimension

Table 2.3 shows the result of dicing into the value Monday on weekday dimension and I-35W-S on highway dimension from Table VOLUMES-L2 in Figure 2.7.

- Pivoting: re-orienting the multidimensional view of data. It presents the measures in different cross-tabular layouts

Table VOLUMES-L0-B in Figure 2.7 tells us I-94 is busier than I-35W south, which is consistent with Table VOLUMES-L1-C. From Table VOLUMES-L0-C and VOLUMES-L1-B, Friday is busier than Monday. Traffic is lightest on Sunday.

2.3.2 Traffic Data Warehouse

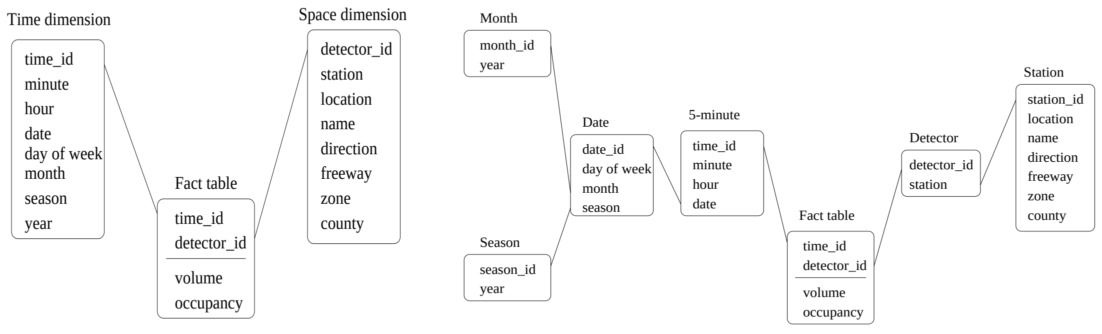
In the traffic data warehouse the measures are *volume* and *occupancy*, and the dimensions are *time* and *space*. Dimensions are hierarchical by nature. In Figure 2.4, for example, the *time* dimension can be grouped into "Hour","Date","Month","Week","Season", or "Year", which form a lattice structure indicating a partial order for the dimension. Similarly, the *Space* dimension can be grouped into "Station","County","Freeway", or "Region". Given the dimensions and hierarchy, the measures can be aggregated in different ways, The SQL aggregate functions and the group-by operator only

Highway	Weekday	Hour of day	Avg volumes
I-35W-S	Monday	All	269.83

Table 2.3. Dice on the value Monday and I-35W-S of highway dimension

produce one out of all possible aggregates at a time. A data cube [13] is an aggregate operator which computes all possible aggregates in one shot.

Most data warehouses use star or snowflake schemas to present a multidimensional data model [6, 15, 7]. Figure 2.8(a) shows a star schema representation for the traffic data. The table in the center is called a *fact* table, which connects to other dimension tables. In traffic data warehouse 2.2(b), the *volume* table can be thought of the *Fact* table. The *Detector* and *Station* tables together form the space dimension. The *time* dimension data can be derived from calendar models. A snowflake schema, as shown in Figure 2.8(b), provides a refinement of the star schema by decomposition of the dimension tables. Notice that the *Detector* and *Station* tables are separate in the snowflake schema of the traffic data warehouse. The aggregate structure of the dimensional tables in star schemas may be more appropriate for many analyses [7]. Accordingly, we use a star schema to construct the traffic data set. We describe the aggregate functions and the data cube operator in the rest of this section.



(a) A Star Schema

(b) A Snowflake Schema

Figure 2.8. Design Schema

2.3.3 Operations on The Traffic Data Cube

The cube operator [13, 30] generalizes the histogram, cross-tabulation, roll-up, drill-down, and subtotal constructs. It is the N-dimensional generalization of simple aggregate functions. Table 2.4

is the *base* table for the traffic data cube. This is produced by joining the dimension tables with the *Fact* table in a star schema. Recall the traffic data warehouse star schema(Figure 2.8(a)). The *base* table has columns for all attributes of the *fact* table and various dimension tables. The key column/attribute for the *base* table are the same as those for the *fact* table. Due to the lack of space, Table 2.4 omits a few columns, e.g., season, day of week.

There are two dimensions: *time* and *space*, and two measures: *volume* and *occupancy*. Each dimension has its corresponding attributes. The lower attributes can be aggregated to higher or derived attributes according to their concept hierarchies as shown in Figure 2.4.

Space Dimension							Time Dimension							Measures	
Det_id	Stat_id	Freeway	Direction	Zone	County	...	Time_id	Yr	Mn	Day	Hr	Min	...	Vol.	Occu.
1	1	I-35W	N	6M	Hennepin	..	1	1997	1	15	6	20	..	80	15
2	1	I-35W	N	6M	Hennepin	..	1	1997	1	15	6	20	..	75	12
3	10	I-35W	S	8P	Hennepin	..	1	1997	1	15	6	20	..	65	18
4	10	I-35W	S	8P	Hennepin	..	1	1997	1	15	6	20	..	45	17
5	100	I-94	E	7L	Scott	..	1	1997	1	15	6	20	..	120	30
6	100	I-94	E	7L	Scott	..	1	1997	1	15	6	20	..	115	35
7	120	I-94	W	8Q	Ramsey	..	1	1997	1	15	6	20	..	134	25
8	120	I-94	W	8Q	Ramsey	..	1	1997	1	15	6	20	..	125	15
....	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..

Table 2.4. The base table of traffic data cube

To support OLAP, the data cube provides the following operators : roll-up, drill-down, slice and dice, and pivot. We now define these operators.

- Roll-up: aggregate. This operator generalizes one or more dimensions and aggregates the corresponding measures. For example, Table 2.5 is the roll-up of the *base* Table 2.4 in both *space* and *time* dimension. The dimension hierarchy in Figure 2.4 shows that while the *space* dimension is aggregated to the *freeway* level, the *time* dimension is aggregated to the *year* level. The aggregate function used in this example is the average on a daily basis.

Space Dimension	Time Dimension	Measures	
Freeway	Year	Volume(Avg. per day)	Occupancy(Avg. per day)
I-35W	1997	60,345	20.3
I-35E	1997	69,730	14.5
I-94	1997	86,782	19.5
....	..	..	..

Table 2.5. Example of roll-up

- Drill-down: disaggregate. This operator specializes in one or a few dimensions and presents low-level aggregations. For example, we drill down Table 2.5 in *time* dimension, adding the *month* attribute.

- Slice and Dice: selection and projection. Slicing into one dimension is very much like drilling

Space Dimension	Time Dimension		Measures	
Freeway	Year	Month	Volume(Ave. per day)	Occupancy(Avg. per day)
I-35W	1997	1	55,340	23.3
I-35W	1997	2	65,645	10.1
I-35W	1997	3	68,395	24.3
...				
I-35E	1997	12	85,345	20.9
I-35E	1997	1	55,375	24.3
I-35E	1997	2	62,335	12.1
I-35E	1997	3	70,945	23.0
...	..	..	..	..

Table 2.6. Example of drill-down

one level down into that dimension, but the number of entries displayed is limited to that specified in the slice command. A dice operation is like a slice on more than one dimension.

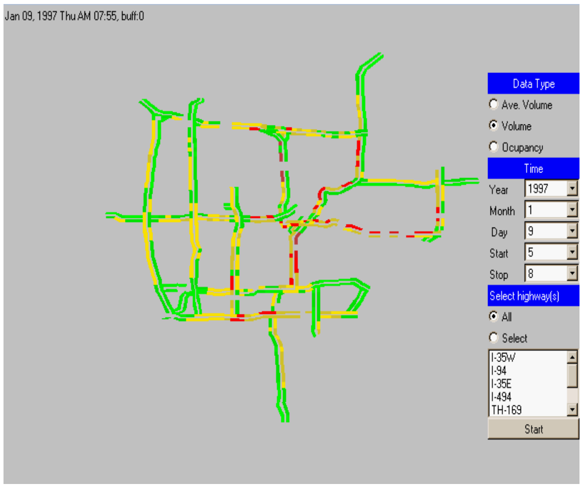
- Pivoting: re-orienting the multidimensional view of data. This operator presents the measures in different cross-tabular layouts. It is more typical of spreadsheets. For example, rows may represent months of a year. Column may represent freeways. Cells may show average volume per day. This spreadsheet would result from pivoting part of Table 2.6.

2.4 Visualization Utilities

Discussions with traffic officials and researchers at the Traffic Management Center helped us identify several essential visualization utilities to support traffic pattern analysis. These utilities include traffic video visualization, traffic volume map, attribute visualization, video map comparison, and traffic flow visualization.

2.4.1 Traffic Video Visualization

A video-like visualization of traffic data can provide an approximate but rapid summary of major trends. It can also be used to visualize the effects of a sudden increase in load on the traffic network after scheduled events so that traffic management can plan for future similar events. Figure 2.9 shows an example of the traffic video for traffic volume of all stations from 7:55 AM to 8:00AM on Jan 9th, 1997. The right portion of the figure is the interface designed for users to specify the display parameters, including 1) data type: average volume, total volume, or occupancy; 2) time: date of display, starting time, and stopping time; 3) highway: total highway, or a particular highway.


Jan 09, 1997 Thu AM 07:55, buff:0

Data Type
○ Ave. Volume
● Volume
○ Ocupancy

Time
Year 1997
Month 1
Day 9
Start 5
Stop 8

Select highway(s)
● All
○ Select

I-35W
I-94
I-35E
I-494
TH-169

Figure 2.9. Traffic Video

2.4.2 Traffic Volume Map

Summary traffic maps of the metropolitan area for a variety of aggregate information includes highway-level traffic volume and occupancy. Figure 2.10 shows the summary volume map for highway I-35W South Bound on Monday, Januarary 6, 1997. The X-axis is the 5-minute time slot for the whole day and the Y-axis is the label of the stations installed on the highway, starting from the top in the north end to the bottom in the south end. In this figure, we can easily observe three traffic patterns: 1) the morning rush hour (6-9AM) in the northern part of the highway; 2) the evening rush hour (3-6AM) in the southern part of the highway; and 3) morning to evening (6:00AM to 6PM) busy traffic volume for stations located within downtown to the junction with Highway 61, that is, the route from downtown Minneapolis to the Minneapolis/Saint Paul airport.

(Traffic volume legend ?)

2.4.3 Attribute Visualization

Visualization of traffic attributes (e.g., volume, occupancy) as a function of time allows identification of outliers or identification of groups of stations with similar behavior. Figure 2.11 shows the total traffic volume of station 15, located at the intersection of I-35W and Highway 61, on Monday, Januarary 6, 1997. The X-axis is the time interval; the Y-axis is the measure of volume. As can be seen, the average 5-minute traffic volume was higher than 230 from 6:00AM to 6:00PM. Notice the abrupt drop of the traffic volume from 462 to 0 at 2:09PM, which is caused by the reboot of traffic data recorder.

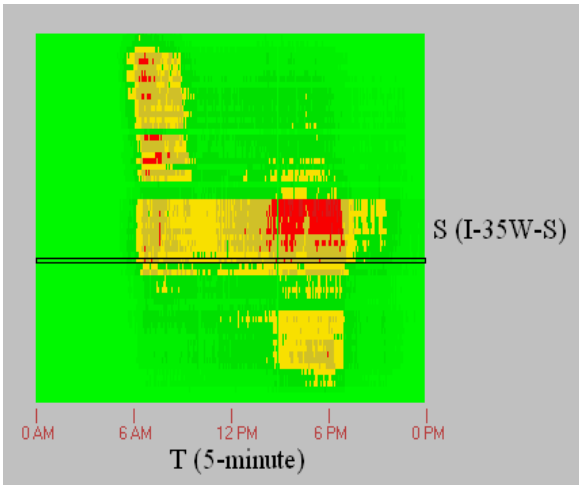


Figure 2.10. Summary traffic map

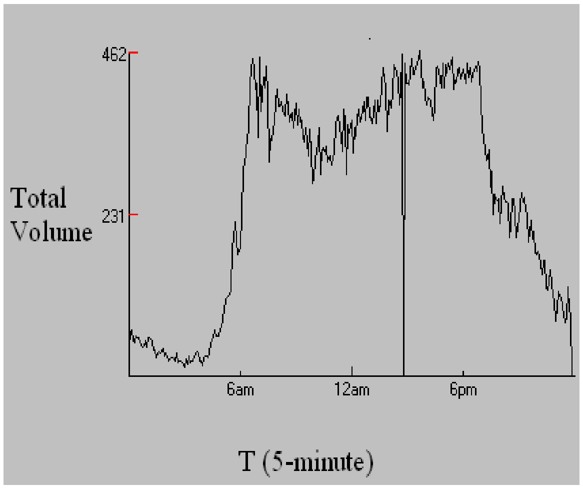
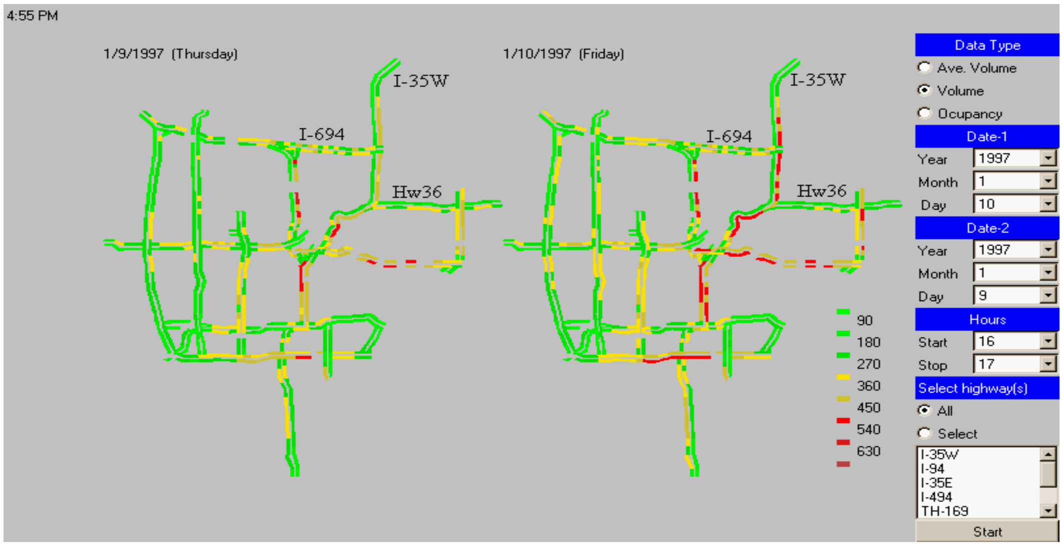


Figure 2.11. Attribute visualization

2.4.4 Video Map Comparison

Our system provides a comparison utility that allows users to simultaneously observe the traffic flow on two different dates. Figure 2.12 shows the traffic flow on Thursday, January 9, 1997 and Friday, Januarary 10, 1997 during 4:55PM to 5:00PM. The figure indicates that the busy traffic flow starts earlier on Friday for Highway I-35W North Bound between the downtown area and its intersection with Highway I-694. An important application of this component is that users can compare the effect of applying ramp meter control on the same weekday.



Data Type
○ Ave. Volume
● Volume
○ Ocupancy

Date-1
Year 1997
Month 1
Day 10

Date-2
Year 1997
Month 1
Day 9

Hours
Start 16
Stop 17

Select highway(s)
● All
○ Select

I-35W
I-94
I-35E
I-494
TH-169

Figure 2.12. Comparison of traffic video of two different days

2.4.5 Traffic Flow Visualization

This component combines the functionality of the summary traffic map, i.e., volume map, and the visualization of attributes. Figure 2.13 shows an example of the traffic flow visualization component. The left portion of the figure is the volume map for Highway I-94 on Monday, Januarary 6, 1997. In the volume map, the X-axis is the 5-minute time interval; the Y-axis is the label of the stations installed on the Highway I-94 East bound, starting from the top in the east end to the bottom in the west end. The chosen station station ID 135, as labeled by the vertical line, is located at the intersection of Highway I-94 and 49th Avenue. The right portion of the figure corresponds to the whole day of traffic of station 135 on Januarary 6,1997, where the X-axis represents the 5-minute time intervals and Y-axis is the total traffic volume for each station in a 5-minute interval.

2.4.6 The visualization display component for a specific highway

A video visualization for a chosen highway is important for users to analyze a particular traffic flow. Figure 2.14 shows a frame of the traffic video for Highway I-94 on Januaray 6 at 7:55AM. The right portion of the figure is the interface designed for users to specify display type, time interval, and highway. This figure shows a busy traffic flow between downtown Minneapolis and downtown St. Paul. In addition, the high traffic volume from Brooklyn Park to downtown Minneapolis on highway I-94 East bound can also be observed.

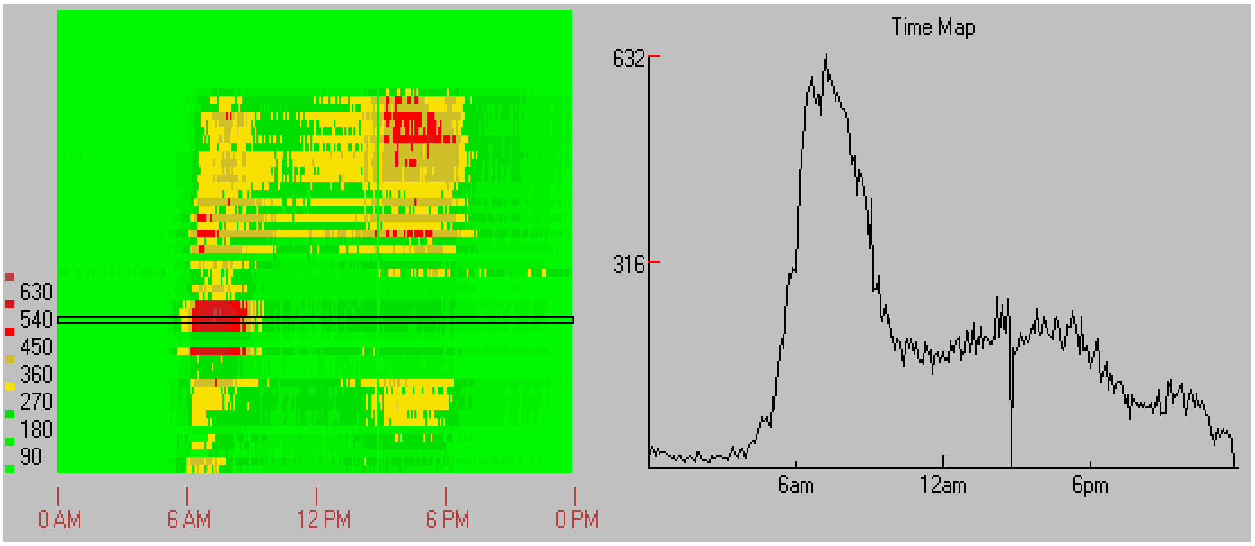


Figure 2.13.

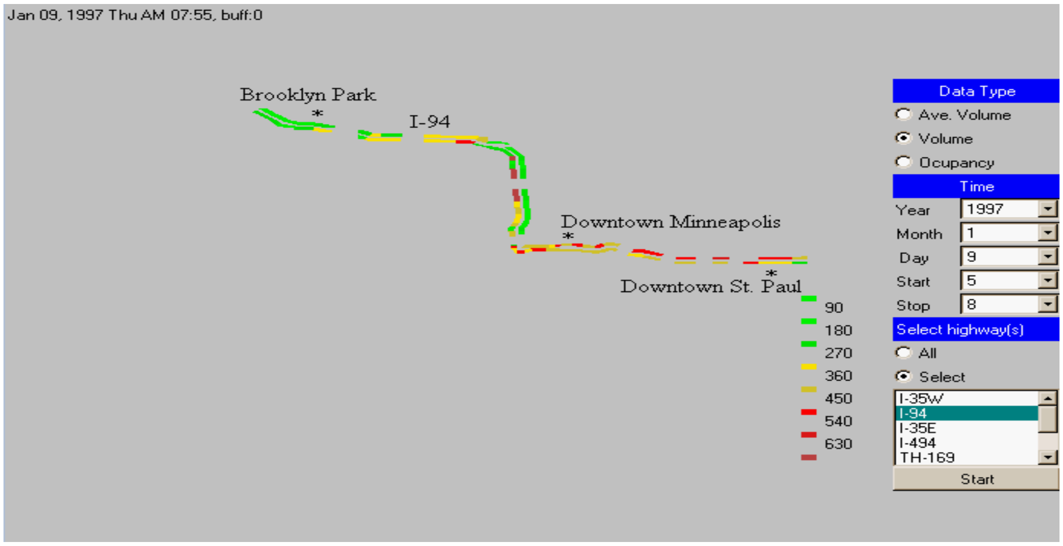
2.5 Traffic Data Mining and Visualization

Traffic data mining is concerned with the identification of useful and interesting patters in traffic data. Knowledge discovery tools, e.g., visualization, help traffic professionals study the patterns identified by data mining. In this section, we examine the application of visualization for different kinds of data mining process, e.g., classification, clustering, and outlier detection.

2.5.1 Classification

Given a set of example objects, called the training set, each of which contains n attributes(features) and one class label, the objective of classification is to analyze the training set and build a model for each class using the n attributes in the data set. The class models are then used to classify the test set, in which the class labels are not provided.

A decision-tree-based classification [25] is a supervised learning method that constructs decision trees from a set of training examples. In our traffic dataset, we apply the C.5 decision tree clustering algorithm [26] provided by a data mining software Clementine [9] to classify the bottleneck station, as shown in Figure 2.15(a). The bottleneck stations were pre-determined by MNDOT TMC. The training set is the average traffic volume and occupancy per-lane on 5-minute interval for January 15, 1997, while the testing set is the traffic flow on January 17, 1997. The training accuracy was 89% and testing accuracy is 87%. The decision tree for determining a bottleneck station is shown in Figure 2.15(b).



Jan 09, 1997 Thu AM 07:55, buff:0

Data Type

○ Ave. Volume

● Volume

○ Ocupancy

Time

Year 1997

Month 1

Day 9

Start 5

Stop 8

Select highway(s)

○ All

● Select

I-35W
I-94
I-35E
I-494
TH-169

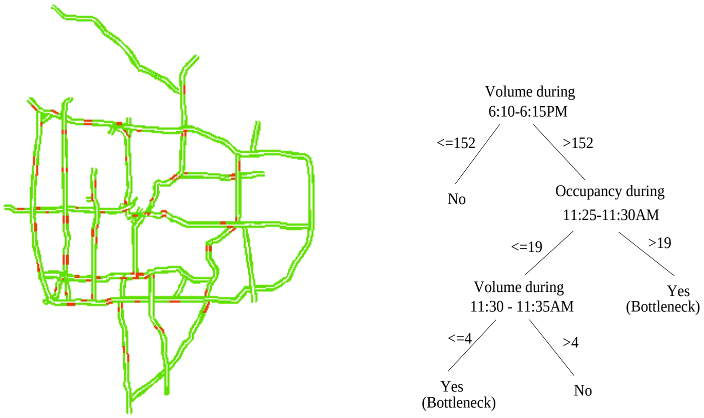
Figure 2.14. Traffic video for highway I-94 on 1/6 1997

2.5.2 Clustering

Clustering, or unsupervised classification, is the process of grouping a set of abstract objects into different clusters such that objects within a cluster are more similar to one another and objects in separate clusters are less similar to one another. Each object in a cluster contains a set of attributes. Euclidean distance is a commonly used similarity measure between data points if the attributes are continuous.

It is of interest to group stations which exhibit similar traffic flow patterns. Here, each station S_i is modeled as a point in a n-dimensional space with the form $S_i =< t_1, t_2, \ldots t_j, \ldots, t_n >$, where t_j denotes the traffic volume for station i at time j and n is the number of the time slots.

Spatial zone formation consists of segmenting traffic stations into smaller pieces that are relatively homogeneous in some sense. While these zones can be specified directly by researchers, particularly when certain areas are of interest, hierarchical clustering provides a general data mining approach for automatically creating a "zone hierarchy." The leaf nodes are the individual station, while intermediate nodes represent larger groups of contiguous stations. The regions at the lowest level represent smaller, more homogeneous regions, while regions at higher levels represent larger, but less homogeneous regions. There has been some research into clustering contiguous spatial data [8, 32], but it is still a relatively new problem. Figure 2.16 shows an example of clustering I-35W north bound into three homogeneous zones, where stations within each zone exhibit similar traffic pattern. Figure 2.16(a) is the data(average traffic volume) map of each 5-minute time slot v.s. station located in the I-35W north bound. The X-axis is the 5-minute time slot for the whole day and



(a) Bottleneck Stations from MNDOT

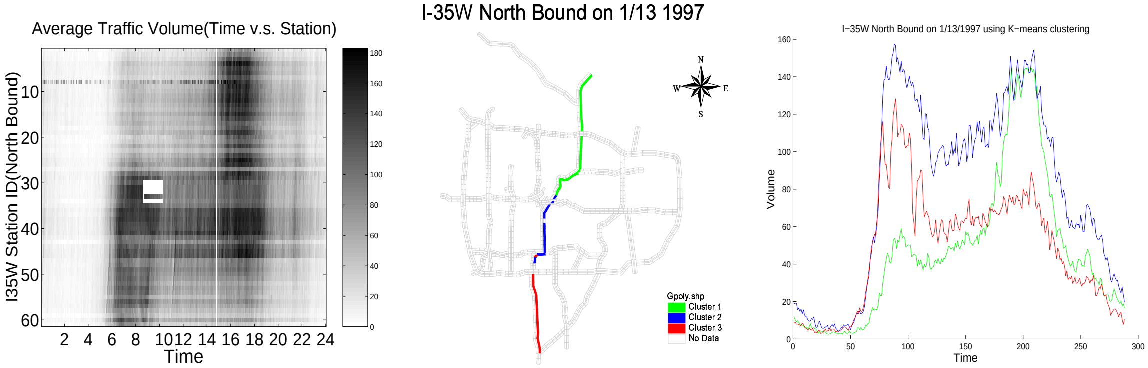
(b) Decision Tree

Figure 2.15. A Classification Example

the Y-axis is the label of the stations installed on the highway, starting from 1 in the north end to 61 in the sound end. We call this a *Space-Time* volume map. Each station is modeled in the form $S_i =< t_1, t_2, \ldots t_j, \ldots, t_n >$, where n is 288, denoting the 5-minute time slots within one day. The clusters after applying the K-means clustering algorithm are shown in Figure 2.16(b). The distance between two station vectors is the Euclidean distance, i.e., sum of the square of differences in volume at each time-point. We looked for three clusters of stations, primarily because the data map(Figure 2.16(a)) shows at least three distinct groups of stations. The results of the clustering algorithms can be visualized in attribute space (volume over time) for further knowledge discovery. Figure 2.16(c) shows the average volume within each cluster. The differences between each cluster can be easily observed. The stations in cluster 1 have high traffic volume during the afternoon rush hour; the stations in cluster 3 have peak traffic volume during the morning rush hour; the stations in cluster 2 exhibit high traffic volume during both morning and afternoon rush hour. Table 2.7 summarizes all the time periods with peak traffic volume (greater than a pre-defined threshold) within each cluster.

In Figure 2.16, we have shown three different ways of visualizing the station-time-volume relationship. This is analogous to pivoting, which is a standard means of data exploration in OLAP and data warehousing. The fact that traffic data has a spatial-temporal component allows for intuitive visualization of the different pivot operations.

Assuming that the three plots in Figure 2.16 are generated from traffic data of a "typical" day, an unusual but repeatable deviation from the normal pattern can help traffic managers plan for unexpected events which may disrupt the flow of traffic. For example, how will a major traffic accident, a sport game, or an entertainment event affect the composition of the three identified clusters? Such questions can be answered by plotting historical traffic data of days when such events were known to have occurred.



(a) Data(average volume) map

(b) K-means clustering using volume data

(c) Average volume within each cluster

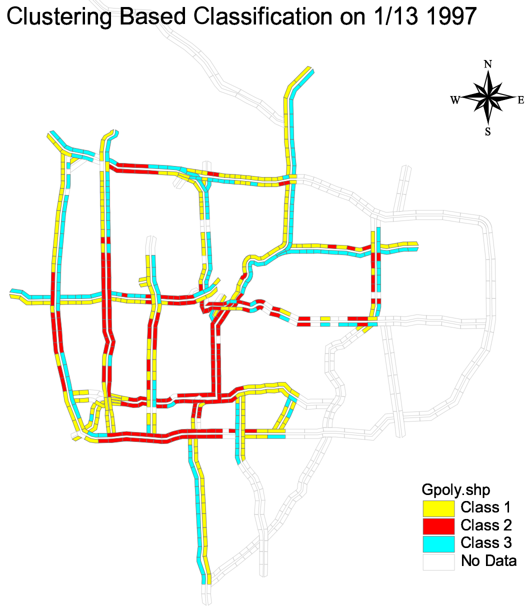
Figure 2.16. I35W North Bound

	I-35W North Bound		
	Cluster 1	Cluster 2	Cluster 3
No of Station	27	18	16
Duration (minutes) greater than threshold (90)	200	740	135
Time period greater than threshold (90)	2:45PM-2:50PM, 3:10PM-6:20PM	6:20AM-10:00AM, 10:20PM-6:55PM	6:15AM-6:45AM, 7:05AM-8:35AM, 9:15AM-9:20AM

Table 2.7. Description of each cluster

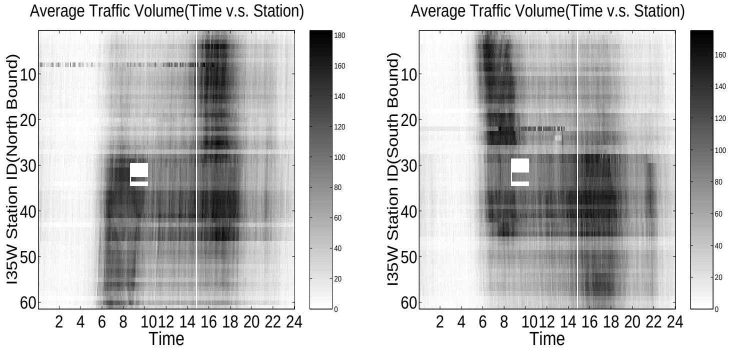
2.5.3 Outliers/Exceptions detection

Knowledge discovery tasks can be classified into four general categories: (a) dependency detection (e.g. association rules) (b) class identification (e.g. classification, clustering) (c) class description (e.g. concept generalization), and (d) exception /outlier detection [20]. Most research has concentrated on the first three categories, which correspond to patterns that apply to a large percentage of objects in the dataset. In contrast, outlier detection focuses on a very small percentage of data objects, which are often ignored as noise. An outlier in a set of data is an observation or a point that is considerably dissimilar to or inconsistent with the remainder of the data [28]. Figure 2.17



shows an example of traffic flow outliers. Figures 2.17(a) and (b) are the volume maps for I-35W North Bound and South Bound, respectively, on Tuesday, Januarary 21, 1997. The X-axis is the 5-minute time slot for the whole day and the Y-axis is the label of the stations installed on the highway, starting from 1 in the north end to 61 in the south end. The abnormal dark blue line at time slot 177 and the dark blue rectangle during time slot 100 to 120 on the X-axis and between station 29 to 34 on the Y-axis can be easily observed from both (a) and (b). Moreover, station 9 in Figure 2.17(a) exhibits inconsistent traffic flow compared with its neighbor stations.

The identification of spatial oultiers, e.g., station 9 in the previous example, can help traffic managers to quickly respond to faulty detectors or



(a) I-35W North Bound

(b) I-35W South Bound

Figure 2.17. An example of outlier

2.5.4 Association Rules Discovery

Given a set of records, each record containing some number of items, it is desirable to discover the dependency rules such that the occurrence of an item can be predicted based on occurrences of other items. Agrawal et al. [1] proposed a formal model to define the association rules. Let $\mathcal{I} = I_1, I_2, \ldots, I_m$ be a set items and T be a database of transactions. An association rule is an implication of the form $X \Longrightarrow I_j$, where X is a set of some items in $\mathcal{I}$, and I_j is a single item in $\mathcal{I}$ that is not present in X. The rule $X \Longrightarrow I_j$ holds in the set of transactions T with confidence c if at least $c\%$ of transactions in T that contain X also contain I_j. The support for the rule is defined as a fraction of a transaction in T that contains the union of items in X and I_j. Confidence is an indication of the rule's strength and support is a measure of statistical significance. Association rules are one possible approach for capturing long range dependencies (in space and time) hidden in traffic data. For example the following rule may be captured by the judicious use of association rule discovery: *A major traffic accident on Station A of Highway X during time $T1$ and $T2$ results in unusual high traffic volume on Station B of Highway Y during $T2 + 2$ and $T2 + 3$.* Unlike correlation analysis, the promise of association rule analysis is that such dependencies do not have to be hypothesized but are discovered automatically.

2.5.5 Sequential Pattern Discovery

Given a set of objects, with each object associated with its own timeline of events, the problem of mining sequential patterns is to find the rules that predict strong sequential dependencies among different events.

Discovering interesting patterns, e.g., correlations, from general traffic datasets is challenging due to their spatial-temporal, multi-scale nature and their large size (10's of gigabytes). Sequential pattern discovery consists of two major components, namely, modeling of events and algorithms for finding spatial-temporal patterns formed by these events. Forming events is challenging due to the influence of neighboring areas on the properties of a spatial zone and the multi-scale nature of the data. Algorithms for finding patterns need to address the impact of redundancies due to overlap across neighborhoods, (which require redefinition of traditional measures of correlation and association), incorporate spatial-temporal properties and traffic domain knowledge to prune and filter uninteresting patterns, and scale to large data sets.

2.6 Conclusion

We illustrate the requirements of a software system for traffic data visualization. This software system contains three components, namely, data warehouse support, a core visualization component, and a data mining module. The data warehouse stores historical traffic data to support on-line analytical processing(OLAP) and data aggregation; the core visualization component provides several visualization utilities which allow users to efficiently compare and contrast traffic flow; the data mining component is essential for semi-automatic trend analysis and pattern discovery.

Chapter 3

Task 2: Identification of bottleneck in visualization

In this chapter, we explain the process where we followed to identify the bottlenecks are using our visualization system. First we did code profiling on traffic video to find out the amount of time spent by each subsystem. The subsystem which costs most of the time is the bottleneck.

3.1 Performance

This section describes the factors that affect the performance of displaying traffic data. In terms of performance, we have two concerns. One is the time used by the database. The other is the response time, especially for traffic video.

We use code profiling on traffic video and try to improve the performance roughly to get rid of bad design decisions. After all this effort, what remains will be the true bottlenecks of our system.

3.1.1 Performance Profiling

In our visualization software system, there are two possible bottlenecks. One involves the network and the other is the database. Code profiling allows us to find out which one is our bottleneck. Of course, both can be bottlenecks.

In our experiments, we used three timers to measure the times used by each subsystem. The first timer, T_{all}, represents the time from the click of the 'Start' button until the applet finished receiving all the traffic data. This can be seen as total time. The second one, T_{db}, represent the time spent by the database and network tranportation between the database and CGI. Because of the relatively small amouts of data, network transportation contributes little time to T_{db}. We can take T_{db} as the database time. $T_{network}$ is the time spent on network transportation from CGI to the applet. Because the synchronization of the timers on both the Applet and CGI is hard to achieve and the CGI server is quick enough to ignore the time spent by excution of the program on the server, we calcuated $T_{network}$ by substracting T_{db} from T_{all}.

In the following code profiling experiments, we requested 3-hour traffic data, from 5am to 8am for all stations on a specific date. The refreshing rate was 2 snapshots/second. The bandwidth of the network was 100 MBps.

In our first experment, Experiment 1, the CGI program requested traffic data of a snapshot from the database, sent the data to CGI, and repeated this process unitl all the snapshots were displayed. Figure 3.1 shows the flow of data in Experiment 1. After receiving all the data, GUI started to display. The reason for displaying after receiving all the data was to avoid video jitter. Table 3.1 shows the results using different tables.

Table used	Total time(sec)	database time(sec)	network time(sec)
value_per_day	95.628	94.829	0.799
five_min	296.487	295.296	1.191

Table 3.1. Experiment 1. Fetch the data for a snapshot.

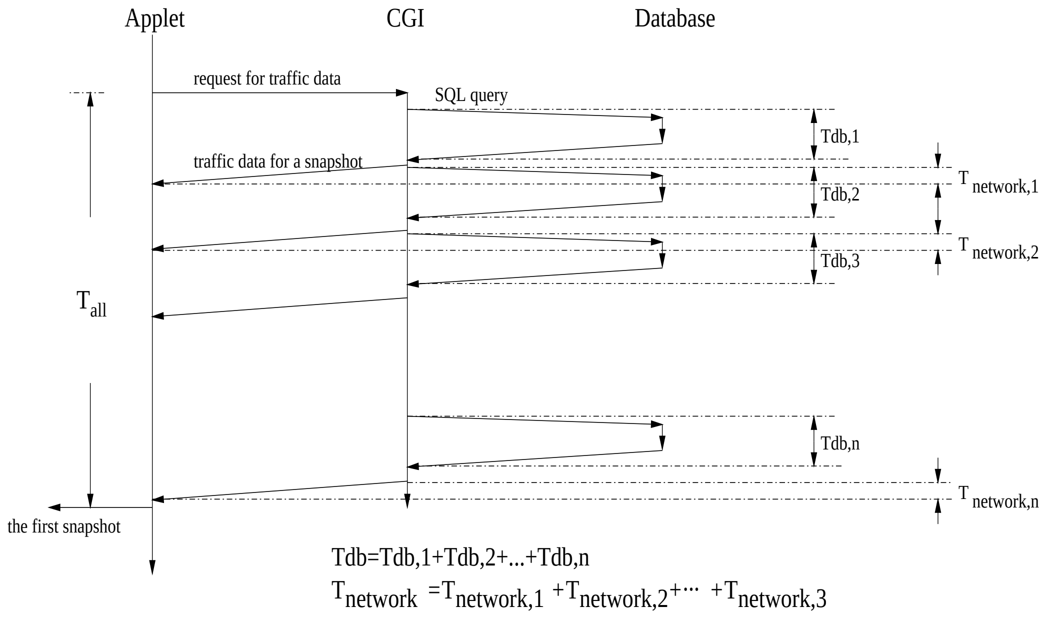


Figure 3.1. Performance Profiling for experiment 1

The results show that 99 percent of the time is used by the database. The time used on network transportation accounts for less than 1 percent. It is clear that the database is our bottleneck.

Performance vs. database table

Different tables also gives different performance. The value_per_day table is almost 3 times faster than the five_min table. The reason is that the five_min table has overhead of storing date, time and station id for each measure, which resluts in more disk space needed to store the data and increases the disk I/Os.

Performance decreases a little when table grows bigger

Because of the huge amounts of traffic data, one question arises: does the database time increase as the size of the data grows? Table 3.2 shows the times under different data set sizes. As can be seen, the time increase slightly when the data grows.

3.2 Database

In the previous section, we discovered that the database is the bottleneck of our application. In this section, we try a different approach to reduce the database time.

Rows of data	Total time(sec)	database time(sec)	network time(sec)
58,964	84.962	84.065	0.894
112,191	85.593	84.832	0.761
171,122	86.775	85.996	0.779
229,387	88.036	87.222	0.814
274,444	89.418	88.513	0.905
333,214	95.628	94.829	0.799

Table 3.2. performance in different sizes of data set

3.2.1 Different query approach

In Experiment 1, the program fetched data for a snapshot per SQL query. Every fetching incurred disk I/O, which is extremely expensive and should be avoided. In the case of using the value_per_day table, the size of a row was small, and each row had a good chance of being located in a single disk block. Reading traffic data for the first snapshot brings the traffic data of the whole day into memory. We should reuse these data to reduce number of disk I/O.

The idea here to improve performance is to fetch a whole day's worth of data, instead of just a snapshot, even though some of the data are not needed. This is the idea behind Experiment 2. Figure 3.2 represents the flow of traffic data in Experiment 2. Table 3.3 shows the results of such improvement. We see a speed-up of 12.

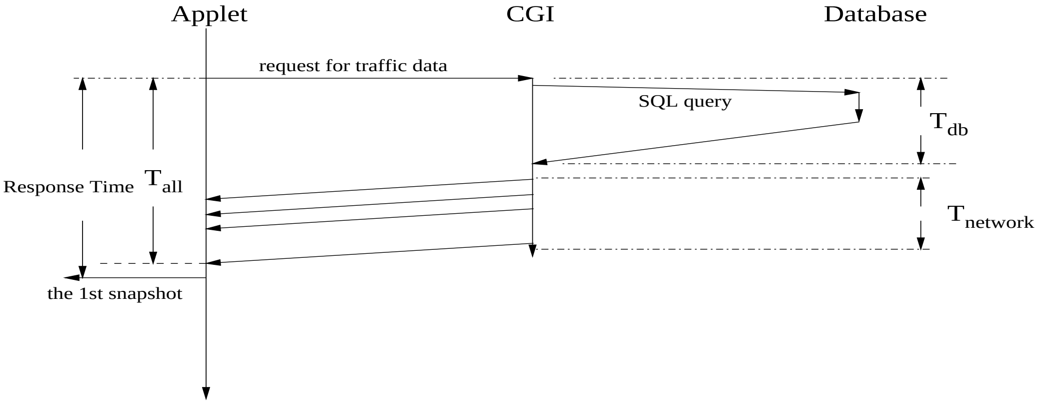


Figure 3.2. Retrieving the whole day instead of a snapshot

T_{all}	T_{db}	$T_{network}$
7.030	3.76	3.27

Table 3.3. Experiment 2. Fetch data of the whole day.

3.3 Network

Simple calculation would not show that the network is our bottleneck. There are 774 stations in our system. Assume CGI program returns (station number,value) pairs to applets. Each station number and value in a string is 3-bytes long at most. For each snapshot of a highway map, we need to transfer (3+3)*774 = 4644 bytes. Assume the refreshing rate is 2 snapshots per second. The bandwidth we need is 4644*2 = 9288 bytes ≈ 9KB. In our experiment environment, we use fast Ethernet, which is 100MBps and can handle the work easily.

3.3.1 Pipelining to reduce response time

Another issue is the response time and it is more for traffic video. Response time is defined as the time between the click of the 'Start' button and the appearance of the first snapshot. An acceptable reponse time is 8 seconds.

In Experiment 1, in order to avoid video jitter, the safer approach was to display the video after receiving all the traffic data. In the world of streaming video, , the displaying client saves some frames to be displayed in a buffer. Then the displaying and receiving data run concurrently to get both performance and quality. We can use this idea in our traffic video.

The database will affect response time. In Experiment 2, the database time was 3.76 seconds. Not taking network transportation into account, the response time was at least 3.76 seconds. When users request more traffic data, this number will increase. The drawback of this approach is that the CGI is waiting for snapshots that are not needed immediately. A reasonable approach is to transfer one snapshot per transaction.

The key concept of pipelining is to display video and receive data simutaneously. The reasons this can save time are twofold. First, after GUI displays a snapshot, it goes to sleep untill it needs to display the next snapshot. Second, network I/O is blocking I/O. The system sleeps between the two network I/O's. These two processes are like producer-consumer and should be able to run concurrently. Figure 3.3 shows the new workflow of pipelining. After caching some snapshots, GUI starts to display images as it keeps receiving traffic data. The results are shown in Table 3.4. We see about 4.5 seconds response time for any duration.

Another interesting task is to to determine the proper size of the initial buffer. The larger the buffer is, the longer is the response time. The size of the buffer is determined by the incoming rate and consuming rate. In our setup, GUI always consumes 2 snapshots per second. The incoming rate varies depending on the database and network. 100MBps is fast enough for our purpose. It can transfer a snapshot in a very short time. The database plays an important role and will be the

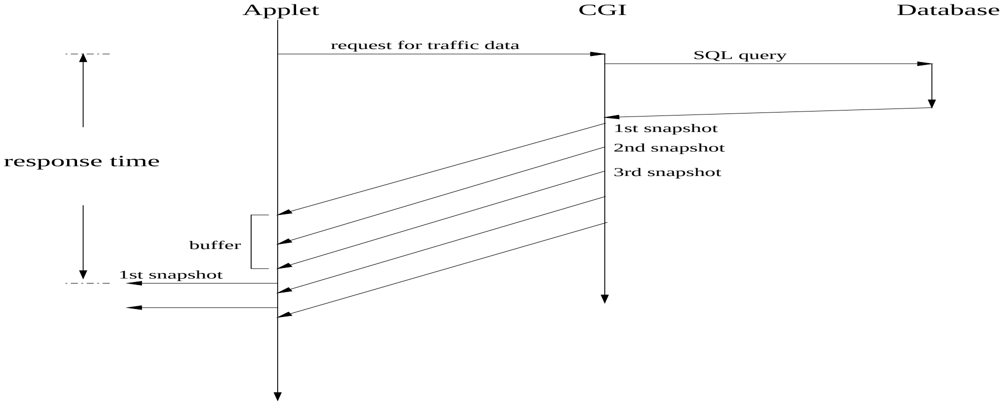


Figure 3.3. Pipelining traffic data.

No Pipelining(sec)	Pipelining(sec)	Duration
7.23	4.687	5AM ∼ 8AM
8.82	4.66	5AM ∼ 10AM
11.216	4.616	5AM ∼ 12AM
12.619	4.526	5AM ∼ 2PM
14.821	4.536	5AM ∼ 4PM
17.295	4.517	5AM ∼ 6PM

Table 3.4. Experiment 3. Response time. The buffer size is three snapshots. value_per_day is used.

topic of next section.

In Experiment 3, the buffer size was 3 snapshots. In the experiment, GUI consumed 2 snapshots per second. As shown in Table 3.3, GUI received 3*12/3.27∼11 snapshots per second. Since the incoming rate was faster than the consuming rate, the initial buffer size could be as small as possible. We see that the response time was kept around 4.5 seconds for all cases.

3.4 Conclusion

The code profiling experiments revealed that the database is the bottleneck. Due to the small amounts of the traffic data for traffic video, network bandwidth is not a problem. With regard to response time, we should manage the transactions between the database and CGI and Applet appropriately to get the most out of our system. Buffering can be used to improve the response time.

The only remaining issue is the database performance. Even though some improvement has been made to the database, it is not enough for a high-performance system. In our next section, we ddress the database performance.

Chapter 4

Task 3: Develop Visualization Algorithm

In this chapter, we propose solutions and algrithms to improve database performance. First, we discuss some database issues which affect performance. Based on the discussion, we try to fine-tune the performance. Second, we develop efficient algorithms which can be used by visualization module. We introduce an algorithm for spatial outlier detection which can be applied to automatically detect faulty traffic stations and quickly identify abnormal traffic patterns.

4.1 Introduction

In the previous section, we determined that the database is the bottleneck. In this section we discuss the use of indexes to improve database performance. We will also develop an efficient algorithm which finishes in one scan.

4.1.1 Experiment design

Parameters

Two factors affect our database performace. They are disk I/Os and indexes. Given a table, the database time is proportional to the number of rows accessed. Indexes influence how the database searches for records. Without indexes, database will do table scan, which takes much time to accomplish.

Table design

In Section 2.3.1, we mentioned the data cube as a tool to support visualization. The dimensions and measures of the cube should be fields of our table. Table five_min(Table 5.3) shows the idea of a data cube. Field sid is the space dimension. Field record_dttm is the time dimension. Volume, avg_volume and occupancy are the measures.

An alternative design to reduce the overhead of sid and record_dttm when storing each measure is shown in Table 5.1. Each row in the value_per_day table stores traffic data from a station for the whole day. Field sid is the space dimension, record_date, the time dimension, data0∼data287, the measures at different time of day.

Query design

Our first step was to find out the queries used in our system. General speaking, there are two applications. One is traffic video, which needs the data of one day. The other is data cube visualization, which always requires data of several days. Regardless of the different methods of visualization, the number of rows of data which are accessed are the determining factor for performance. Table 4.1 are the benchmark queries to evaluate the perfomance of a solution.

Q1 and Q2 are for the traffic video. Q1 requires data from all stations on Jan 1st, 1997 while Q2 requires data from stations on I-35W and Jan 1st, 1997. Q3∼Q8 are for the data cube visualization. The design of these queries is to access different number of rows.Q3 requests data of one day, which contains 1897 rows of data. Q4 reads 13331 rows.

Num	Query
Q1	Get volume for all stations from 5AM to 8AM on Jan 1st, 1997.
Q2	Get average volume for all stations on I-35W south from 5AM to 8PM on Jan 1st, 1997.
Q3	Get average volume vs time of day on Jan 1st, 1997.
Q4	Get average volume for each time of day and each weekday from Jan 1st, 1997 to Jan 7th, 1997.
Q5	Get average volume for each time of day and each weekday from Jan 1st, 1997 to Jan 14th, 1997.
Q6	Get average volume for each time of day and each weekday from Jan 1st, 1997 to Jan 21th, 1997.
Q7	Get average volume for each time of day and each weekday for the month of Jan, 1997.
Q8	Get average volume for each time of day and each weekday for the month of Jan and Feb, 1997.

Table 4.1. Queries

In this experiment, we measured the database time. The database time was calculated from the time the SQL query was submitted to the time the program received all the data from database. During the experiment, five_min was faster than value_per_day for the traffic video. But value_per_day worked faster for the data cube visualization. To see how our methods improved the performance, we chose the best table for each application. Thus Q1∼Q2 was tested using five_min while Q3∼Q8 used value_per_day.

4.1.2 Relevant Characters of MySQL Database Server

There are some constraints imposed by the database server. The database we used in our implementation was MySQL Distrib 3.23.38. The constraints we are interested are:

- All indexes are B-trees.
- The size of a database is limited. There is about 500MB free space on the MySQL server.
- Indexes are created automatically for primary key fields.

4.2 Use Index to Improve Performance

As is known, indexes can speed up the search for records. Without indexes, a database may have to do table scans to find out all candidates. MySQL has three kinds of indexes, namely primary, unique, and normal indexes.

- Primary index: index built on primary key fields.
- Unique index: index built on fields which have unique values.
- Normal index: index built on any fields. It is also known as a secondary index.

Note that the primary index described here is a little different. Usually, if a table uses a primary index, the data records are physically ordered on disks. But here, the primary index just means an index built on primary key fields.

In the experiment, we configured the database to use no index, primary index and secondary index in order to see how these affect the performnace. Table 4.2 shows results of the benchmark queries. Note that Q1∼Q2 used the five_min table and Q3_simQ8 used the value_per_day table.

Query	no index	primary index	seconday index	rows accessed	total number of rows
Q1	283.776	2.000	2.090	22860	5649984
Q2	313.359	0.638	1.057	4428	5649984
Q3	1.517	0.278	0.320	1897	171122
Q4	2.67	1.78	1.841	13331	171122
Q5	3.708	3.592	3.615	26637	171122
Q6	4.944	5.403	5.354	39898	171122
Q7	6.627	6.776	6.819	58964	171122
Q8	11.772	11.966	11.849	112191	171122

Table 4.2. Impact of indexing on performance.

4.2.1 Improvement for traffic video

In traffic video, use of index significantly improves the performance. In Q1, we see a speed-up of 141. Use of the primary index and seconday index show no difference. The speed-up for Q2 is even higher, about 491. This is because the fields in the search conditions of Q2 are exactly the index fields. In Q1, only the record_dttm is used.

Use index to reduce response time

With indexing, the five_min table outperforms the value_per_day table for traffic video because it reads the traffic data for the snapshot that is needed immediately. Table 4.3 shows that the response time can be further reduced to 1.5 seconds. Buffering is used.

Duration	Response time
5AM ~ 8AM	1.512
5AM ~ 10AM	1.523
5AM ~ 12AM	1.512
5AM ~ 2PM	1.562
5AM ~ 4PM	1.512
5AM ~ 6PM	1.512

Table 4.3. Response time using five_min

4.2.2 Improvement for data cube visualization

In data cube visualization, we see that the performance enhancement using indexes is limited. This is best seen in Figure 4.1. When accessing less than 15 percent of rows(point A), use of index helps the database find the records. Fifteen percent is the break-even point, where using indexes spend the same amount of time as using no index. When accessing more than 15 percent to 34 percent, using no index is faster than using index. After 34 percent, the database always does table scans, even with indexing.

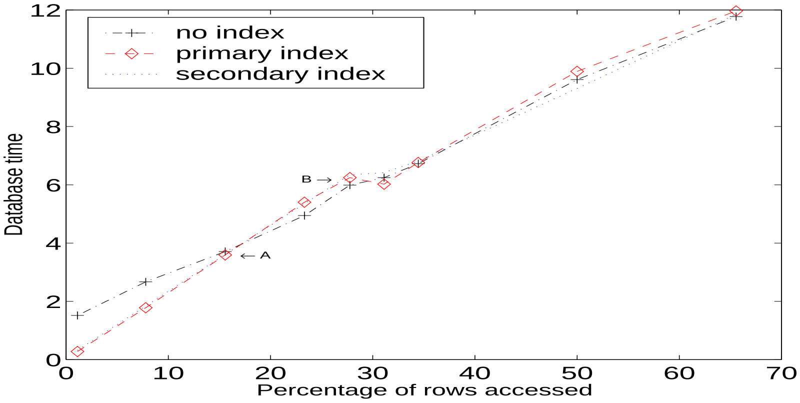


Figure 4.1. Data Cube Visualization and Index.

From the previous section, we can see that indexing does not help the database to improve performance when our programs access too many rows. Even when an index does help to improve performance, the speed-up is 5.46, which is poor.

Computing from raw data is not a good idea because of the huge amounts of data. In fact, the data cube mentioned in Section 2.3.1 can be used to improve performance. We can see from Figure 2.6 that the aggregation weekday has been done and the results are saved in the table. In our case, instead of reading raw data, we can read the aggregated results from this table. The rows accessed will be far fewer than those accessed raw data.

Data cube visualization using different tables

Different table design can result in different performance. Table 4.4 shows the results of using different tables. Primary key indexes are built on both the five_min and value_per_day tables. dt_cube table has secondary index (s, record_dt). five_min and dt_cube only have data of one month while value_per_day has 3 months. Figure~fig:performance of five min and value per day clearly shows the huge improvement of value_per_day, even though value_per_day contains more data the five_min. value_per_day is 17∼28 times faster than five_min.

One table to notice is the dt_cube table, which definition can be found in Tabel 5.2. It is the VOLUMES-L0-A table in Figure 2.7. When using this table, the speed-up is 346, compared with five_min table.

Query	value_per_day	five_min	dt_cube
Q3	0.278	7.89	0.080
Q4	1.78	34.52	0.219
Q5	3.592	58.45	0.249
Q6	5.403	81.87	0.263
Q7	6.776	116.01	0.335

Table 4.4. Impact of indexing on performance.

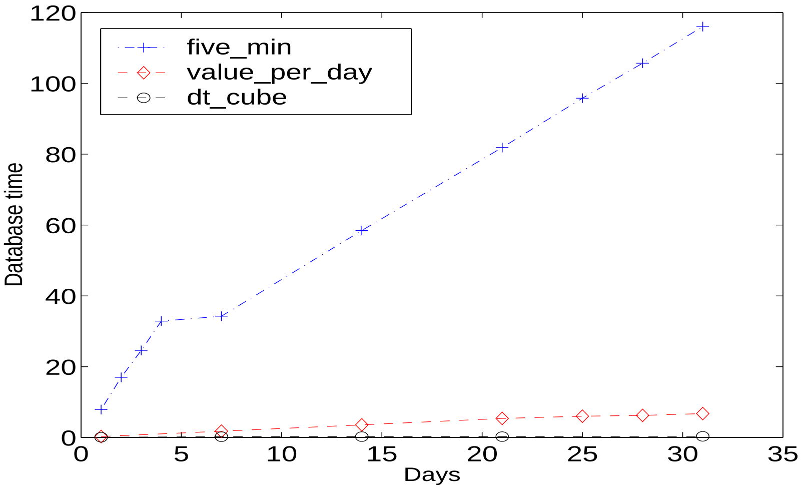


Figure 4.2. Performance of value_per_day and five_min

4.3 Develop High Performance Algorithm

Among the great number of stations on highways in Twin Cities area, one of the interesting applications is to find out the outliers, the faulty stations whose behavior is abnormal. By 'abnormal', we mean the station behaves strangely compared to its neighbors.

With the help of data cube visualization, we can easily find out the outliers. Take Figure 2.17 as an example. There is a horizonal 'line' near station 10. The behavior of this station is very different compared to its neighbors.

Even though visualization provides an easy way to discover outliers, inspecting thousands of pictures is difficult. For example, there are 18 highways in our system. Each highway runs in 2 directions. If one picture is generated per day/highway/direction. This means 2*18=36 pictures are generated per day. To inspect one month's data, the personnel need to look at about 1000 pictures, which is tedious and error-prone. Thus we need methods to discover suspected outliers so that they can be inspected manually, more easily.

4.4 Spatial Outlier: Introduction

A spatial outlier is a spatially referenced object whose non-spatial attribute values are significantly different from those of other spatially referenced objects in its spatial neighborhood. Informally, a spatial outlier is a local instability (in values of non-spatial attributes) or a spatially referenced object whose non-spatial attributes are extreme relative to its neighbors, even though they may not be significantly different from the entire population. For example, a new house in an old neighborhood of a growing metropolitan area is a spatial outlier based on the non-spatial attribute house age. We provide a general definition of spatial outliers and propose efficient spatial outlier detection algorithms to detect abnormal stations in the traffic data set.

In the traffic data set, each station is a spatially referenced object with spatial attributes(e.g., location) and non-spatial attributes(e.g., measurements). Spatial arrangement of stations can be modeled as a spatial graph [29]. A directed edge from station s_1 to station s_2 indicates the existence of a road segment allowing traffic to move from s_1 to s_2. This graph is called a spatial graph because nodes, i.e., stations, are located in a Euclidean space [33] where each node has a location specified by coordinates, e.g., <highway, mile point>. The non-spatial attributes include sensor-id and traffic measurements (e.g., volume, occupancy). We are interested in discovering the location of stations whose measurements are inconsistent with those of their neighbors. This spatial outlier detection task is formalized as follows.

Let the traffic sensors constitute a collection of spatially referenced objects. The location of a sensor represents a spatial attribute and is represented by the symbol x. A traffic measurement (e.g., volume) constitutes a non-spatial attribute space and is represented as $f(x)$. The neighborhood of x, $N(x)$, is the set of traffic sensors adjacent to the sensor located at x. We note that the neighborhood relationship is based on directed edges in the underlying spatial graph. Thus sensors on opposite sides (e.g., I-35W north bound and I-35W south bound) are not neighbors even if the pairwise Euclidean distance is small. A sensor is compared to its neighborhood using the function $S(x) = [f(x) - E_{y \in N(x)}(f(y))]$, where $f(x)$ is the attribute value for a data record x, $N(x)$ is the set of neighbors of x, and $E_{y \in N(x)}(f(y))$ is the average attribute value for the neighbors of x. The statistic function $S(x)$ denotes the difference of the attribute value of a sensor located at x and the average attribute value of $x's$ neighbors.

[1] Examination of other categories of dimensions, e.g., temporal, is beyond the scope of this paper and may be explored in future work.

4.4.1 Definition of S-Outliers

Consider a spatial framework $SF =< S, NB >$, where S is a set of locations $\{s_1, s_2, \ldots, s_n\}$ and $NB : S \times S \to \{True, False\}$ is a neighbor relation over S. We define a neighborhood $N(x)$ of a location x in S using NB, specifically $N(x) = \{y \mid y \in S\,, NB(x, y) = True\}$.

Definition: An object O is an S-outlier(f, f^N_{aggr}, F_{diff}, ST) if $ST\{F_{diff}[f(x), f^N_{aggr}(f(x),N(x))]\}$ is true, where $f : S \to R$ is an attribute function, $f^N_{aggr} : R^N \to R$ is an aggregation function for the values of f over neighborhood, R is a set of real numbers, $F_{diff} : R \times R \to R$ is a difference function, and $ST : R \to \{True, False\}$ is a statistic test procedure for determining statistical significance.

Example 1. The spatial outliers defined in Section 1.1 are examples of S-outliers. We can define respective components in the traffic application domain as follows. The f is the non-spatial attribute, namely, traffic volume. The neighborhood aggregate function $f^N_{aggr}(x) = E_{y \in N(x)}(f(y))$ is the average attribute value function over neighborhood $N(x)$. The difference function $F_{diff}(x)$ is $S(x) = [f(x) - E_{y \in N(x)}(f(y))]$, i.e., the arithmetic difference between attribute function $f(x)$ and neighborhood aggregate function $f^N_{aggr}(x)$. Let $\mu_{s(x)}$ and $\sigma_{s(x)}$ be the mean and standard deviation of the difference function F_{diff}; then the significance test function ST can be defined as $Z_{s(x)} = |\frac{S(x) - \mu_{s(x)}}{\sigma_{s(x)}}| > \theta$.

Example 2. A $DB(p, D)$-outlier [21] is also an example of an S-outlier. For a k dimensional data set T with N objects, an object O in T is a $DB(p, D)$-outlier if at least a fraction p of the objects in T lies greater than distance D from O [21]. Assuming f^N_{aggr} is the number of objects within distance D from object O, the statistical test function ST can be defined as $\frac{(Total\ number\ of\ objects) - f^N_{aggr}(x)}{(Total\ number\ of\ objects)} > p$. The DB-outlier subsumes many other definitions of global outliers [21].

4.5 Spatial Outlier Detection: Problem Definition and Proposed Algorithms

In this section, we provide a formal definition of the problem of designing computationally efficient techniques for detecting spatial outliers. Earlier sections presented a definition of spatial outliers and showed that the definition subsumes other quantitative spatial outlier definitions. Table 4.5 shows examples of difference function F_{diff} and statistic test function ST for different quantitative spatial outlier detection methods. Difference function F_{diff} computes parameters that are used by statistical test function ST to verify the outlierness of a node. We show F_{diff} and ST functions for Spatial statistic $Z_{s(x)}$, Scatterplot, and Moran scatterplot approaches to summarize the lemmas presented in the earlier section. For example, in the scatterplot approach, the difference function computes the error term ϵ, which is the value of the vertical distance between a node and the regression line in the $X - Y$ plane and is defined as $F_{diff} : \epsilon = E(x) - (m * f(x) + b)$, where $E(x)$, the average attribute value of neighbor nodes of x, is the Y-axis value; $f(x)$, the attribute value of node x, is the X-axis value; the m and b are the slope and intercept of the scatterplot line in the $X - Y$ plane.

The computation needs of spatial outlier detection are divided into two parts, model building and test result computation. Model building computes aggregate functions used by the difference function F_{diff} and statistic test function ST, as shown in the last row of Table 4.5. We discuss the computation of the aggregate functions and propose algorithms for model building and test result computation.

Test Computation			
Spatial Outlier Definition	Spatial statistic $Z_{s(x)}$	Scatterplot	Moran scatterplot
Difference function F_{diff}	$S(x) = [f(x) - E(x)]$	$\epsilon = E(x) - (m * f(x) + b)$	$Z_i = \frac{f(x)-\mu_f}{\sigma_f}$, $I_i = \sum_j W_{ij} Z_j$
Statistic test function ST	$\mid \frac{S(x)-\mu_s}{\sigma_s} \mid > \theta$	$\mid \frac{\epsilon-\mu_\epsilon}{\sigma_\epsilon} \mid > \theta$	$(Z[f(i)]) \times (\sum_j (W_{ij} Z[f(j)])) < 0$
Aggregate function used in F_{diff} and ST	μ_s, σ_s	m, b, μ_ϵ, σ_ϵ	μ_f, σ_f

Table 4.5. Examples of F_{diff} and ST functions for different approaches

4.5.1 Problem Definition

Given the components of the S-outlier definition, the objective is to design a computationally efficient algorithm to detect the S-outliers. The components of the S-outlier definition are restricted via constraints to allow computational efficiency while preserving the correctness of Lemmas showing that various existing spatial outlier detection tests (e.g., Scatterplot, Moran scatterplot, Spatial statistic $Z_{s(x)}$) are special cases of S-outliers. Thus the algorithms proposed in this section are useful in building models to detect spatial outliers via a variety of existing techniques. The following optimization problem characterizes the problem of designing efficient algorithms for detecting spatial outliers:

Spatial Outlier Detection Problem

Given:

- A spatial framework S consisting of locations $s_1, s_2, \ldots, s_n$
- A neighborhood relationship $N \subseteq S \times S$
- An attribute function $f : s_i \to R$
- A neighborhood aggregate function $f_{aggr}^N : R^N \to R$, where N is the maximum neighbor number for a location

- A comparison function $F_{Diff}(f, f^N_{aggr})$
- Statistic test function $ST : R \rightarrow \{True, False\}$

Design: An efficient algorithm to detect S-outliers,

i.e., $\{s_i \mid s_i \in S, s_i\, is\, an\, S - outlier\}$

Objective:

- Efficiency: to minimize the computation time

Constraints:

- F_{diff} and ST can be computed using algebraic aggregate functions of values of $f(x)$ and f^N_{aggr}
- The size of the data set is much greater than the main memory size
- Computation time is determined by I/O time

Aggregate functions can be grouped into three categories, namely, distributive, algebraic, and holistic [13,31]. An aggregate function F is called distributive if there exists a function G such that the value of F for a data set can be computed by applying a G function to the value of F in each partition of the whole data set. In most cases, $F = G$. Examples of distributive aggregate functions include *count*, *max*, and *sum*, as shown in Appendix B. An aggregate function F is algebraic if F of a data set can be computed using a fixed number of aggregates from each partition of the data set. *Average*, *variance*, *standard deviation*, *maxN*, *minN* are all algebraic aggregate functions. Illustrations are available in Appendix B. An aggregate function F is called holistic if the value of F for a data set cannot be computed using a constant number of aggregates from each partition of the data set. We note that algebraic and distributive aggregate functions can be computed by a single scan of a data set even when the data set is too large to fit in the main memory. In comparison, for a holistic aggregate function, e.g., *median*, the entire data set has to be stored in memory for processing. In processing a data set with a size greater than the size of memory, extra disk scans are required to calculate the holistic aggregate function.

For each node, say x, the attribute function $f(x)$ contains the attribute value of x. The neighborhood aggregate function f^N_{aggr} computes a value using the attribute value of x and the attribute value of $x's$ neighboring nodes. The distributive aggregate function computes the aggregate value (e.g., *sum*, *count*) of the attribute value and neighborhood aggregate value for all nodes. The algebraic aggregate function computes the statistic values for all nodes, e.g., *mean* and *standard deviation*, and can be derived using the values computed in the distributive aggregate functions. The comparison function F_{diff} and statistic test function ST for the quantitative spatial outlier definition can be computed using algebraic aggregate functions of values from $f(x)$ and f^N_{aggr}. Table 4.6 shows the algebraic aggregate functions for different quantitative definitions of spatial outliers. Each column shows the computation structure of the attribute function, neighborhood aggregate function, distributive aggregate functions, and algebraic aggregate functions for each spatial outlier detection approach. For example, in the scatterplot approach, the attribute function is $f(x)$; the neighborhood aggregate function f^N_{aggr} is $E(x) = \frac{1}{k}\sum_{y \in N(x)} f(y)$; the distributive aggregate functions $D^{G_i}_{aggr}$ are $\sum f(x)$, $\sum E(x)$, $\sum f(x)E(x)$, $\sum f^2(x)$, $\sum E^2(x)$; and the algebraic aggregate functions $A^{G_i}_{aggr}$ are the slope m and the intercept b of the regression line, and the standard deviation σ_ϵ of the error term ϵ, all of which can be derived using the distributive aggregate functions.

By utilizing Table 4.6, we can compute the algebraic aggregate functions in one single scan of the spatial self-join of the station data set using the neighbor relationship. For example, the standard deviation of the error term ϵ in the scatterplot approach can be computed using the values computed in the distributive aggregate functions. In a naive approach, however, two data scans of the spatial self-join may be used, where the first scan computes the slope and intercept of the regression line, and the second scan calculates the statistic values (e.g., *mean* and *standard deviation*) of the error term.

Model Building			
Outlier Definition	Spatial statistic $Z_{s(x)}$	Scatterplot	Moran scatterplot
Attribute function f	f(x)	f(x)	f(x)
Neighborhood aggregate function f^N_{aggr}	$S(x) = f(x) - E(x)$	$E(x) = \frac{1}{k}\sum_{y \in N(x)} f(y)$	
Distributive aggregate functions: $D^{G1}_{aggr}, D^{G2}_{aggr}, .., D^{Gk}_{aggr}$	$\sum S(x)$, $\sum S^2(x)$, n(count)	$\sum f(x)$, $\sum E(x)$, $\sum f(x)E(x)$, $\sum f^2(x)$, $\sum E^2(x)$, n(count)	$\sum f(x)$, $\sum f^2(x)$, n(count)
Algebraic aggregate functions: $A^{G1}_{aggr}, A^{G2}_{aggr}, .., A^{Gk}_{aggr}$	$\mu_s = \frac{\sum S(x)}{n}$, $\sigma_s = \sqrt{\frac{1}{n}[\sum S^2(x) - \frac{(\sum S(x))^2}{n}]}$	$m = \frac{N\sum f(x)E(x) - \sum f(x)\sum E(x)}{N\sum f^2(x) - (\sum f(x)^2}$, $b = \frac{\sum f(x)\sum E^2(x) - \sum f(x)\sum f(x)E(x)}{N\sum f^2(x) - (\sum f(x))^2}$, $\sigma_\epsilon = \sqrt{\frac{S_{yy} - (m^2 S_{xx})}{(n-2)}}$, where $S_{xx} = \sum f^2(x) - [\frac{(\sum f(x))^2}{n}]$, $S_{yy} = \sum E^2(x) - [\frac{(\sum E(x))^2}{n}]$,	$\mu_f = \frac{\sum f(x)}{n}$, $\sigma_f = \sqrt{\frac{1}{n}[\sum f^2(x) - \frac{(\sum f(x))^2}{n}]}$

Table 4.6. Model building to compute the aggregate functions

4.5.2 Our Approach

The computational task in the spatial outlier detection problem can be divided into two subtasks: a) design an efficient computation method to compute the global statistical parameters using a spatial join and b) test whether spatial locations on a given path are outliers. We call the first task model building, the second task test result computation.

Model Building

An I/O efficient model building algorithm computes the algebraic aggregate functions, e.g., the *mean* and *standard deviation*, in a single scan of a spatial self-join from a spatial data set using a neighbor relationship. The computed values from the algebraic aggregate functions can be used by the difference function F_{diff} and statistic test function ST to validate the outlierness of an incoming data set. Algorithm 1 shows the steps of the Model Building algorithm. In the first step, the algorithm retrieves the neighbor nodes for each data object, say x; then it computes the neighborhood aggregate function f^N_{aggr}. The distributive aggregate functions are then aggregated using the attribute function $f(x)$ and the neighborhood aggregate function f^N_{aggr}. Finally, the algebraic aggregate functions are computed using the values from the distributive aggregate functions. Note that the data objects are processed on a page basis to reduce redundant I/O. In other words, all the nodes within the same disk page are processed before retrieving the nodes of the next disk page.

Model Building Algorithm

```
Input: $S$ is a spatial framework;
    $f$ is an attribute function;
    $N$ is the neighborhood relationship;
    $f^N_{aggr}$ is the neighborhood aggregate function;
    $D^{G1}_{aggr}, D^{G2}_{aggr}, \ldots, f^{Gk}_{aggr}$ are the distributive aggregate functions;

Output: Algebraic aggregate functions $A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr}$
for(i=1;i $\leq$ $|S|$ ;i++){
    $O_i$=Get_One_Object(i,S); /* Select each object from S */
    NNS=Find_Neighbor_Nodes_Set($O_i$,$N$,$S$);
   /* Find neighbor nodes of $O_i$ from S */
    for(j=1;j$\leq$ $|$NSS$|$;j++){
      $O_j$=Get_One_Object(j,NNS); /* Select each object from $NNS$ */
      $f^N_{aggr}$ = Compute_and_Aggregate($f(O_i), f(O_j)$);
    }
    Aggregate_Element($D^{G1}_{aggr}, D^{G2}_{aggr}, \ldots, D^{Gk}_{aggr}, f^N_{aggr}$,i);
    /* Add the element to global aggregate functions */
}
$< A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr} >$ = Compute_Algebraic_Aggregate($D^{G1}_{aggr}, D^{G2}_{aggr}, \ldots, D^{Gk}_{aggr}$);
/* Compute algebraic aggregate functions*/
return ($A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr}$).
```

Test Result Computation

The algebraic aggregate functions, e.g., *mean* and *standard deviation*, computed in the Model Building algorithm can be used to verify the spatial outlier of incoming data sets. The two verification algorithms are Route Outlier Detection (ROD) and Random Node Verification (RNV). The ROD algorithm detects the spatial outliers from a user specified route, as shown in Algorithm 2. The RNV procedure checks the outlierness from a set of randomly generated nodes. The step to detect outliers in both ROD and RNV are similar, except that the RNV has no shared data access needs across tests for different nodes. The I/Os for Find_Neighbor_Nodes_Set() in different iterations are independent of each other in RNV. We note that the operation Find_Neighbor_Nodes_Set() is executed once in each iteration and dominates the I/O cost of the entire algorithm. The storage of the data set should support efficient I/O computation of this operation. We discuss the choice for storage structure and provide experimental comparison in Sections 5 and 6.

Given a route RN within the data set S, the ROD algorithm first retrieves the neighboring nodes from S for each data object, say x, in the route RN; then it computes the neighborhood aggregate function f^N_{aggr} using the attribute value of x and the attribute values of $x's$ neighbors. The difference function F_{diff} is computed using the attribute function $f(x)$, neighborhood aggregate function f^N_{aggr}, and the algebraic aggregate functions computed in the Model Building algorithm. Node x can then be tested for outlierness using the statistical test function ST.

Route Outlier Detection(ROD) Algorithm

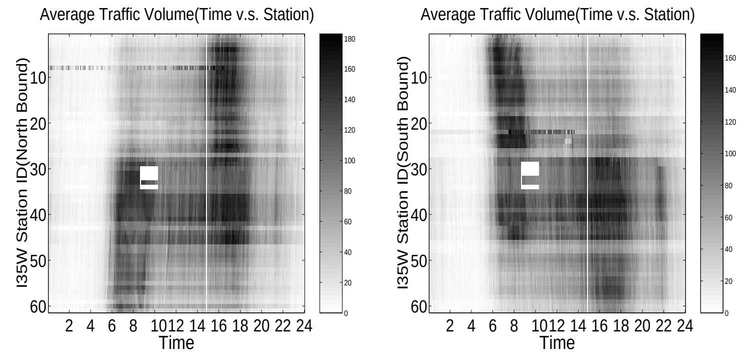
```
Input: $S$ is a spatial framework;
    $f$ is an attribute function;
    $N$ is the neighborhood relationship;
    $f^N_{aggr}$ is a neighborhood aggregate function;
    $F_{diff}$ is a difference function;
    $A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr}$ are algebraic aggregate functions;
    $ST$ is the spatial outlier test function;
    $RN$ is the set of node in a route;

Output: Outlier_Set.
for(i=1;i $\leq$ $|RN|$ ;i++){
    $O_i$=Get_One_Object(i,RN); /* Select each object from RN */
    NNS=Find_Neighbor_Nodes_Set($O_i$,$N$,$S$);
   /* Find neighbor nodes of $O_i$ from $S$ */
    for(j=1;j$\leq$ $|$NSS$|$;j++){
       $O_j$=Get_One_Object(j,NNS); /* Select each object from $NNS$ */
       $f^N_{aggr}$ = Compute_and_Aggregate($f(O_i), f(O_j)$);
    };
    $F_{diff}$ = Compute_Difference($f$,$f^N_{aggr}$, $A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr}$);
    if($ST(F_{diff}, A^{G1}_{aggr}, A^{G2}_{aggr}, \ldots, A^{Gk}_{aggr}$)== True){
        Add_Element(Outlier_Set,i); /* Add the element to Outlier_Set */
      }
  }
  return Outlier_Set.
```

4.5.3 Outliers Detected

We tested the effectiveness of our algorithm on the Twin-Cities traffic data set and detected numerous outliers, as described in the following examples.

Figure 4.3 shows one example of traffic flow outliers. Figures 4.3(a) and (b) are the traffic volume maps for I-35W north bound and south bound, respectively, on Januaray 21, 1997. The X-axis is a 5-minute time slot for the whole day and the Y-axis is the label of the stations installed on the highway, starting from 1 on the north end to 61 on the south end. The abnormal white line at 2:45PM and the white rectangle from 8:20AM to 10:00AM on the X-axis and between stations 29 to 34 on the Y-axis can be easily observed from both (a) and (b). The white line at 2:45PM is an instance of temporal outliers, where the white rectangle is a spatial-temporal outlier. Moreover, station 9 in Figure 4.3(a) exhibits inconsistent traffic flow compared with its neighboring stations, and was detected as a spatial outlier.



(a) I-35W North Bound

(b) I-35W South Bound

Figure 4.3. An example of an outlier

4.5.4 Conclusion

In this chapter, we proposed solutions to improve database performance. Using indexes speed up the database system by almost 300 times. Precomputation will enhance the data cube visualization. We also developed an efficient outlier detection algorithm which used an algebraic function and allowed one scan to carry out the outlier detection.

Chapter 5

Task 4: Prototype Software Implementation

In this chapter, we introduce the prototype software system in the traffic visualization system. First, we explain the design of the software architecture, including the Graphic User Interface(GUI), the Common Gateway Interface (CGI) web server, and the database server. We then illustrate the software utilities and supporting utilities in our system, such as video visualization and visualization of attributes. Finally, we show the implementation of map cube operations, such as roll up, drill down, and slide and dice, in the system. We also discuss some interesting patterns discovered in the visualization of the traffic data.

5.1 Prototype Software Design

5.1.1 Design Requirements

User Friendliness

Since users are all familiar with the world wide web, we decided that a web interface would be easier to use to interact with our system. Instead of FORM tag in HTML to interactive with users, we chose Java applets. Java provides abundant AWT components and useful design patterns, like Model View Control, to build complex interfaces.

Portability

Our system should be able to be accessed regardless of machines used. Since Java is touted as “Write once, run anywhere”, we chose it as our programming language.

Performance

Instead of using plain text files to store traffic data, we save the traffic data in a database. The disadvantage of using text files is that we need to do a full scan to find out the data we need, which is de facto table scan in database terms. In database, we can use indexes to help search for the needed data and thus avoid table scan.

5.1.2 Software Design

Based on the requirement analysis, we constructed an interactive web-based visualization system. This system contains three modules, namely, the Graphic User Interface (GUI), the Common Gateway Interface (CGI) web server, and the MySQL database server. Figure 5.1 shows the relation between these modules. The GUI accepts inputs from users and sends requests to the CGI. The CGI recevies the requests from the GUI and generates appropriate SQL queries to the MySQL database server, After receiving data from the database server, the CGI sends the data back to the GUI. We will introduce these modules in detail as follows.

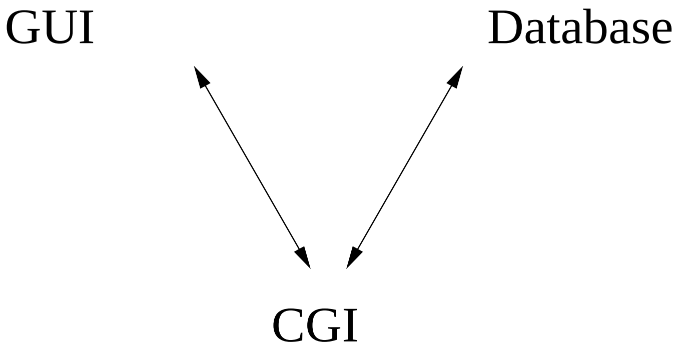


Figure 5.1. Software architecture

Graphic User Interface (GUI)

This module is responsible for getting inputs from users and displaying the results in a graphic manner. Based on the users’ inputs, it sends requests to the web server to request traffic data. After receiving traffic data, it transforms the data into different colors according to a color scheme and displays traffic maps on the screen. When displaying a highway map, each station is represented by a polygon. The geographical coordinates of the vertices of each station's polygon are stored in the database. This module transfers geographical coordinates to screen coordinates for proper displaying. The current module is implemented using Java applets. It can be plugged into web browsers, thus allowing it to accessed world-wide. This module was built using 19 Java programs with 4440 lines of code.

CGI web server

Implemented in Perl and installed in a web server, the CGI programs work as middle-tier between the database and GUI. It accepts requests from GUI and sends appropriate SQL queries to the database to request the needed data. The results are sent back to GUI as strings. For GUI to decipher the incoming strings, simple protocols between GUI and CGI were designed. The module contains 350 lines of Perl code.

Database server

We used My-SQL as the database server. There are 5 tables in the database, namely value_per_day, five_min, station, polygon and color. Tables 5.1 ∼ 5.6 show the definition and detailed structure of these tables.

Table 5.1 Value_per_day shows the stored traffic data of a station for a whole day. Each type of traffic data occupies a row in this table. Column sid is the station Id. record_date is the date of the record. record_type indicates what kind of data it is. It can be total volume, average volumn or occupancy. Column data0 is the measure for 00:00AM, data1 the measure for 00:05AM, and so on. Table 5.3 Five_min table represents a five-min traffic flow measure. sid is the station id. record_dttm indicates the date and time of this record. volume, avg_volume and occupancy are the measures for that station at that point of time. Table 5.4 stores the coordinates of the vertices of a polygon, which represents a station on the map. sid is the station id. sequence_number indicates an identity of the vertex. x and y are the geography coordinates. Table 5.5 stores information about a station. station_number is the station id. highway indicates the highway this station is on. junction reveals where the station is on the highway. direction indicates which bound of the highway this station is on. sequence_number is the sequence number of the station on the highway. It's numbered from north to south or west to east. For example, the northest station on highway I-35W South bound is given sequence number 0. key_station shows that this station is a station of interest. Table 5.6

Color table stores the color schemes. Column id is the id of the color. hex_code stores 24-bit RGB value of the color. scheme is the name of the color scheme.

column name	data type	primary key	comment
sid	smallint	Yes	station number
record_date	date	Yes	the date of the record
record_type	char	Yes	the type of the record
data0	unsigned smallint	No	data for 00:00 AM
data1	unsigned smallint	No	data for 00:05 AM
...	...	...	...
data287	unsigned smallint	No	data for 23:55 PM

Table 5.1. Value_per_day

column name	data type	primary key	comment
space	char(20)	Yes	s-dimension
record_dt	date	Yes	the date of the record
record_tm	time	Yes	time of day
avg_volume	float	No	

Table 5.2. Dt_cube

column name	data type	primary key	comment
sid	smallint	Yes	station number
record_dttm	datetime	Yes	
volume	smallint	No	total volume
avg_volume	smallint	No	average volume
occupancy	smallint	No	occupancy

Table 5.3. Five_min

5.2 Software Utilities

The software utilities we developed so far include three components: traffic video, visualization of attribute and data cube visualization.

column name	data type	primary key	comment
sid	smallint unsigned	Yes	
sequennce_number	smallint	Yes	
x	double	No	x coordinate
y	double	No	y coordinate

Table 5.4. Polygon

column name	data type	primary key	comment
station_number	int unsigned	Yes	
highway	char	No	the highway this station is in
junction	char	No	the junction this station is at
direction	char	No	
sequence_number	smallint	No	
key_station	bool	No	if it's an key station

Table 5.5. Station

5.2.1 Video Visualization

A video-like visualization of traffic data can be used for an approximate but rapid summary of major trends. This can also be used to visualize the effects of a sudden increase in load on the traffic network after scheduled events for the planning of traffic management for future similar events.

Video visualization for one day

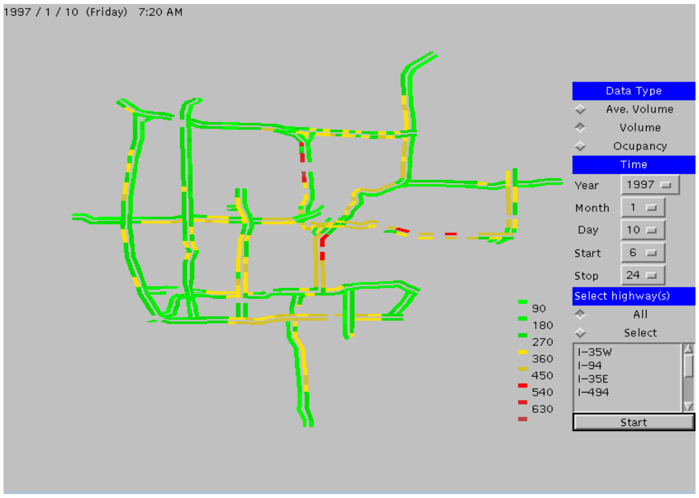
Figure 5.2 is a snapshot of traffic video. On the right of this figure are the selections users can select. Users can select which data type and date to display. Users can also select which hours of the day to display. All highways or some specific highways of interest can be selected. Figure 5.2 shows the traffic video for all highways while figure 5.3 shows only I-94, east and west bounds.

Video map comparison component

Our system provides a comparison utility that allows users to simultaneously observe the traffic flow on two different dates. For example, in 2001, Minnesota conducted an experiment of turning off ramp meters. Traffic domain masters may be interested in comparing the traffic before and after

column name	data type	primary key	comment
id	tinyint	Yes	
hex_code	int	No	24-bit RGB
scheme	char(80)	No	name of the scheme

Table 5.6. Color


1997 / 1 / 10 (Friday) 7:20 AM

Year 1997
Month 1
Day 10
Start 6
Stop 24

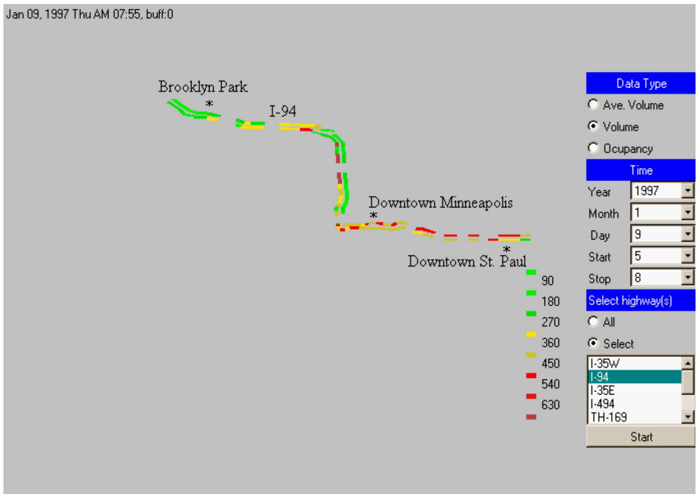
Figure 5.2. A Snapshot from Traffic Video

turning off the ramp meters. Figure 5.4 shows this utility. The interface of this module is almost the same as that of the previous module except that users can select one more date and it displays two highway maps at the same time of day.

5.2.2 Visualization of Attributes

Visualization of traffic attributes (e.g., volume, occupancy) as a function of time and selected highway locations allows identification of outliers (station or time-slots) as well groups of stations with similar behavior.

Figure 5.5 is a snapshot for this utility. The map to the right of the highway map is a so called volume map. The Y-axis of the map is stations on the selected highway. The X-axis is time of day. Users select a highway, date and data type of interest and the system will render the map on the right. If a user is interested in the behavior of a station, he/she can click on the map. The program will mark the station on the highway map. On the bottom are shown the measures for that station for the whole day.



Jan 09, 1997 Thu AM 07:55, buff:0

Data Type

○ Ave. Volume
● Volume
○ Ocupancy

Time

Year 1997
Month 1
Day 9
Start 5
Stop 8

Select highway(s)

○ All
● Select

Figure 5.3. Traffic Video for I-94

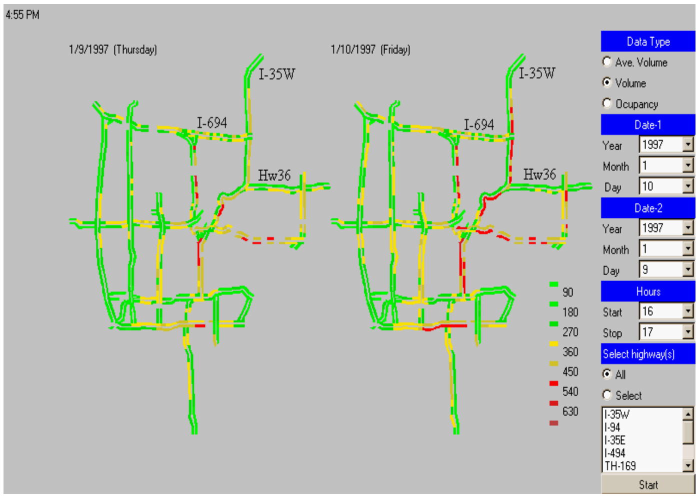
Also incorporated in this system is the outlier detection ability using the algorithm we developed in Chapter 4. This module calculates the average of the nearest 4 neighbor stations and the difference between this average and its measure. The differences from all stations should form a normal distribution. If the difference of a station falls out of the range of 3σ , this station is determined to be an outlier.

To discover outliers in this module, users can click on the 'Outlier' button and red bars will appear to mark the outlier stations on the map. The map on the right of figure 5.6 shows the result of outlier detection. These outliers may be caused by transission disruption, sensor malfunction, or traffic accident.

5.3 Visualization of Dimension Hierarchy

We designed a visualization system that can analyze traffic behavior based on spatial and temporal dimensional hierarchies. The spatial dimensions of interest are "station", "freeway", to "region"; the temporal dimensions of interest are "hour", "week", "month", "season", and "year". We use an example, Day of Year, to illustrate the interesting patterns discovered:

Day of Year over all stations demonstrates average traffic flow during the whole year for all stations. The X-axis is the time, starting from 0:00AM to 24:00PM; the Y-axis is the day of week, from Sunday to Saturday for the whole year. The low traffic flow on Saturday and Sunday can be


4:55 PM

1/9/1997 (Thursday)

1/10/1997 (Friday)

Data Type

○ Ave. Volume

● Volume

○ Ocupancy

Date-1

Year 1997

Month 1

Day 10

Date-2

Year 1997

Month 1

Day 9

Hours

Start 16

Stop 17

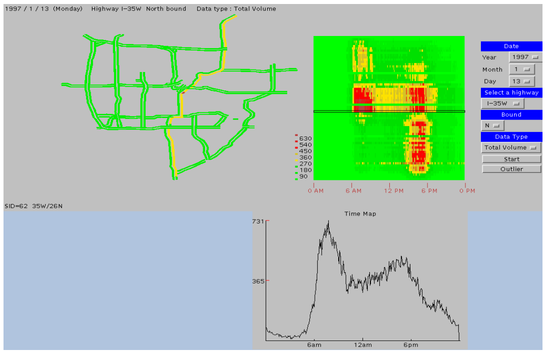
Select highway(s)

● All

○ Select

I-35W
I-94
I-35E
I-494
TH-169

Figure 5.4. Comparison of traffic video of two different dates.


1997 / 1 / 13 (Monday) Highway I-35W North bound Data type : Total Volume

Date

Year 1997

Month 1

Day 13

Select a highway

I-35W

Bound

N

Data Type

Total Volume

SID=62 35W/26N

Figure 5.5. Volume map

easily observed. The traffic patterns, e.g., rush hour periods, from Monday to Thursday are quite similar, but they differ from that of Friday, whose rush hour starts earlier. In addition, the periods of average high traffic volume on Saturday starting from noon to 6PM, while the high traffic volume periods on Sunday starting from 10:30PM to 7PM. This Day of Year traffic flow can also be used to analyze one particular station of interest.

5.3.1 Data Cube Visualization

We implemented a prototype system to illustrate the idea of displaying dimension hierarchy. Figure 5.7 shows the interface of this utility. This system adopts a DTS datacube. The map type allows users to pivot the cube to a facet, including 1-D data cube. Currently, Aggregation only supports aggregation on the Date dimension. Only visualiztion of DT and SD faces are implemented in the system.

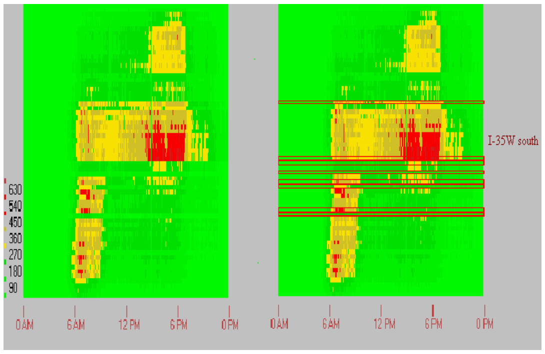
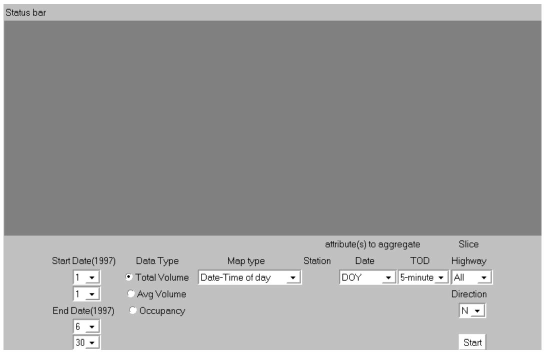


Figure 5.6. Outlier detection



Start Date(1997): 1, 1
End Date(1997): 6, 30
Data Type: ● Total Volume ○ Avg Volume ○ Occupancy
Map type: Date-Time of day
attribute(s) to aggregate: Station; Date: DOY; TOD: 5-minute
Slice: Highway: All; Direction: N

Figure 5.7. Data cube visualization

5.3.2 Traffic Dimension Hierarchy

We adopted a simple dimension hierarchy for our data cube. In such a system, there are 3 dimensions, D, T and S. D is the date, including year. T is time of day, from 00:00:00AM ~ 23:59:59. The granularity of T is 5 minutes in our system. S is the station. The dimensions of a data cube must be partially ordered. Thus in our system, we order the stations by highway ID and direction. Stations on a direction of a highway are ordered from north to south or west to east. Fig 5.8 shows the dimension hierarchy currently implemented in the prototype software.

Before describing Figure 5.8, we define data cube operators first. Pivoting is to rotate the cube to show a particular face. Slicing-dicing is to select some subset of the cube. Aggregating along a dimension is called a roll-up. The reverse of roll-up is called a drill-down.

As shown in Figure 5.8, pivoting the data cube to reduce the number of presentation attributes moves the hierarchy from top to bottom. For example, if we pivot the DTS data cube to show only D and T attributes, we get a DT matrix. We can slice and aggregate on any dimension before pivoting.

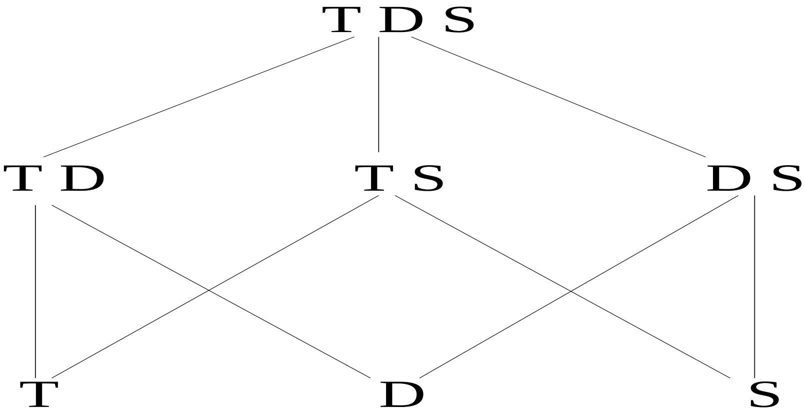


Figure 5.8. Traffic Dimension Hierarchy. In this hierarchy, time dimension is broke into Date(D) and TOD(T).

Every pivoting operation reduces the dimension by one. Pivoting the TDS cube to the TD face gives us the TD matrix (figure 5.10). The resulting TD cube can be pivoted again to the T face and we get the T matrix(figure 5.17).

Aggregation on dimensions can reduce the number of lattices. This corresponds to the roll-up operator of data cube queries. The aggregation on the T-dimension can be (morning rush hours, evening rush hours), (morning, evening, night, midnight). Current implementation only allows aggregation over the whole day. The attributes(stations) of the S-dimension are organized in a partial order. For example, they are organized in highway ID and direction. For each (highway, direction), the stations are ordered from north to south or west to east. Based on this order, the S-dimension can be aggregated according to highway Id's or the combination of highway Id's and direction of the highway. Another aggregation on S-dimension can be a zone, like downtown Minneapolis. Current implementation does not allow aggregation on the S-Dimension. D-dimension can be aggregated according to day of year, weekday or month. Current implementation allows all three aggregations on the D-dimension.

All the visualizations so far can be fit into this data cube framework. The Traffic video is somewhat special. Because we are displaying a map, the S dimension should be broken into 2 dimensions, x and y dimensions.The Traffic video then can be viewed as a slide show of the S^2T matrix. Note that D is sliced for a specific date. We can also aggregate D by weekday, then see the average traffic video for weekdays.

5.3.3 D-T matrix

Pivoting the cube to the D-T face gives us D-T matrix. The y-axis of this matrix is date. This matrix can answer the following query:

Find the average volume for each weekday and for each time of day.

To get Figure 5.9, we aggreage all stations and pivot to the D-T face. Based on figure 5.9 and aggregating along the D-dimension and grouping by weekdays, we can get weekday-TOD (figure 5.10). Figure 5.11 is similar to Figure 5.10 except that the aggregation is grouped by months.

Figure 5.12 is a little different from the previous figures. It aggregates all stations only on I-35W south and dates over a month and then slices the cube on the S-dimension for I-35W south. Finally, pivoting to the D-T face we see figure 5.12.

Figure 5.10 has an interesting observation. Rush hours on Friday begin earlier than on the other four weekdays.

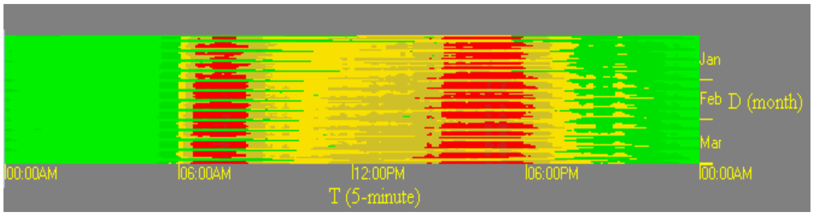


Figure 5.9. Day of year vs. Time of day.

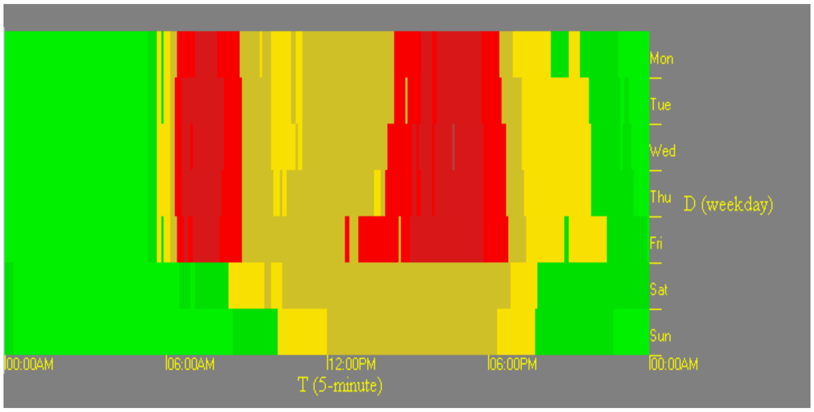


Figure 5.10. Weekday vs. Time of day.

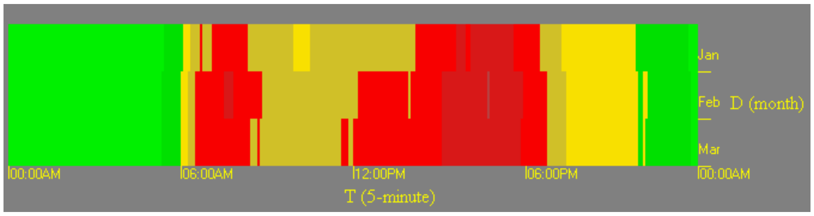


Figure 5.11. Month vs. Time of day.

5.3.4 D-S matrix

This matrix can answer the following query:

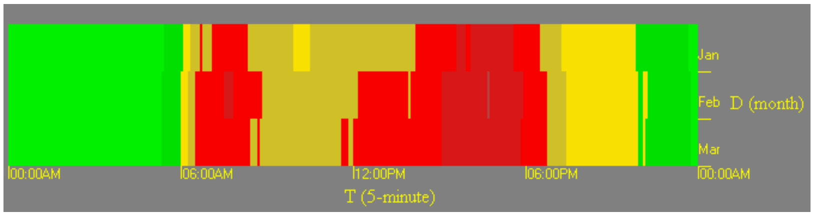


Figure 5.12. Month vs. Time of day on I-35W south

Find the average volume on a highway for each weekday.

The D-S matrix shows the measures for each station on a specific highway and on each day. Current implementation always slices for a specific highway. Its y-axis is the D-dimension and the x-axis is the stations of a highway and direction. Note that the partial order of the S-dimension applys here. the T-dimension is always aggregated over the whole day in the current implementation.

To obtain Figure 5.13 , we aggregate all times of the day, slice the S-dimension to leave only stations on I-35W south and pivot to the D-S face. This matrix shows that the pattern repeats itself every 7 days.

Based on Figure 5.13, we can aggregate the D-dimension and group by weekday and get Figure 5.14. If we aggregate the D-dimension and group by month on Figure 5.13, we have Figure 5.15

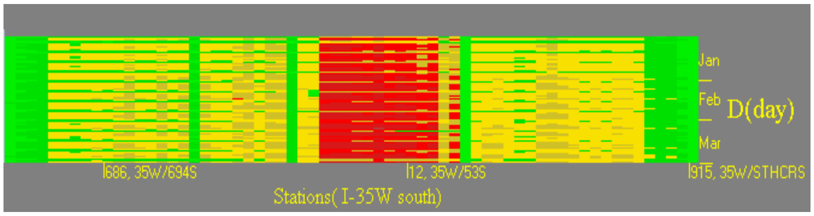


Figure 5.13. Day of year vs. stations on I-35W south.

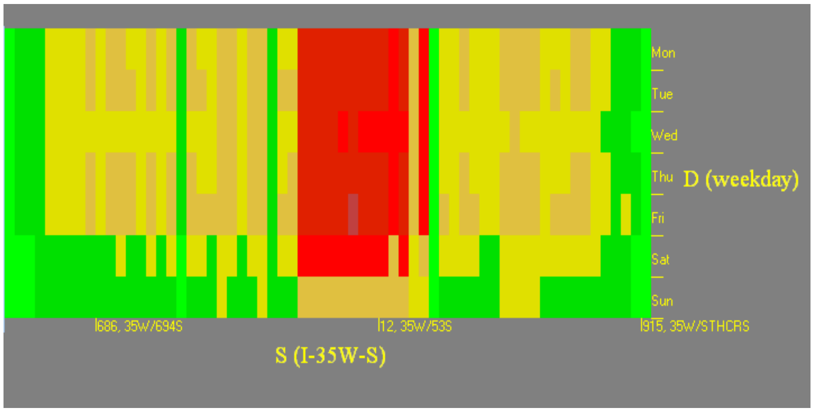


Figure 5.14. Weekday vs. stations on I-35W south.

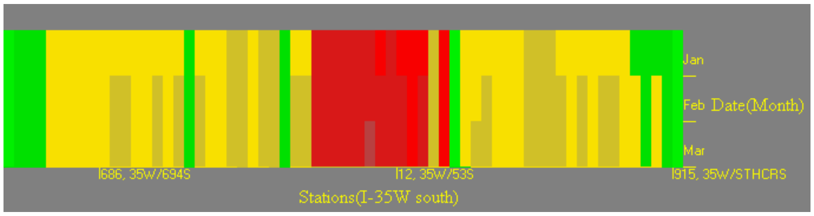


Figure 5.15. Month vs. stations on I-35W south.

5.3.5 S-T matrix

This matrix can answer the following query:

Find the volume at any time of day for all station on a highway.

In this matrix, the y-axis is the stations on a highway and the x-axis is time of day. Pivoting the cube to the S-T face gives the S-T matrix. This matrix is not implemented in this module but an alternative version has been implemented as the volume matrix.

Figure 5.16 is from section 5.2.2. It is an S-T matrix on I-35W south bound on Jan 6th, 1997. In terms of data cube, we slice for I-35W south and Jan 6th, 1997. Then we pivot the sub-cube to S-T face and get the figure. Note that there is no aggregation here.

Most of the time, we are not interested in both directions of a highway. Even though they are geographically near, stations on different directions of a highway are irrelevant.

Interestingly, we see morning rush hours on the section to the north of downtown Minneapolis. This means that most of the population working in downtown Minneapolis is from the northern suburbs. The section from downtown Minneapolis to I-62 is always busy during day time. We can also discover that there are three clusters of stations. Within each cluster, the stations have similar behavior.

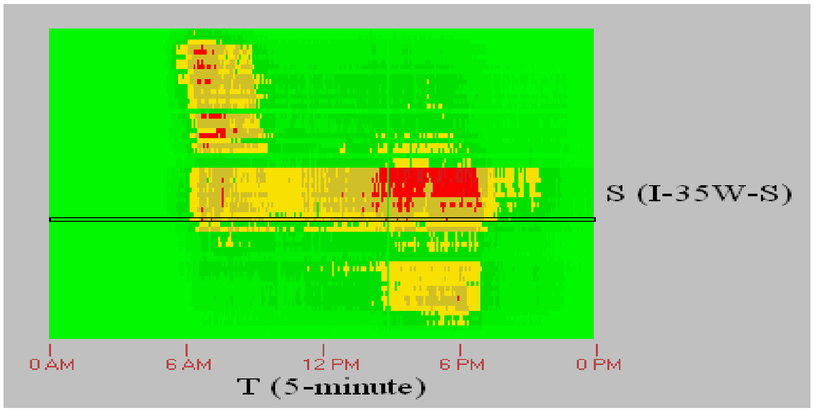


Figure 5.16. Stations on highway I-35W vs Time of day.

5.3.6 T matrix

The T matrix can answer the following query:

For a station on Jan 10, 1997, find the volume at each time of day.

Slicing on S-T/D-T matrix for the T-dimension gives us the T matrix. Current implementation is in the volume matrix module. Figure 5.17 is a 1-D T matrix from the S-T matrix module. The D-dimension and S-dimension are not aggregated in this matrix. To get this matrix, we can slice the S-T matrix for one station. Figure 5.17 shows the x-y plot of a T matrix. Another presentation of a 1-D matrix can be a color bar.

5.3.7 D matrix

The D matrix can answer the following query:

Display the total traffic volume for all stations for every month or every year.

The x-axis of The D matrix is the D-dimension, which can be aggregated into weekday, month or year. The y-axis is the traffic volume.

Although we have not yet implemented this component, the D matrix can be easily constructed in the similar method as the T matrix.

5.3.8 S matrix

The S matrix can answer the following query:

For Jan, 1997 show the average volume for every station on I-35W south.

The x-axis of the S matrix is the S-dimension, which can be aggregated into highway or region. The y-axis is the traffic volume.

Although we have not yet implemented this component, the D matrix can be easily constructed in the similar method as the T matrix.

5.3.9 Album

Answer the following query: For each highway and for each month in the year 1997, display the weekday vs time of day traffic pattern.

In addition to displaying a single matrix, we can also display series of matrixs which are related in some ways. Take the DT matrix for example. We may be interested in the DT matrix of different highways and in different months. In terms of cube operations, we can slice the S-dimension based on highway Id and direction. We also perform slicing on the D-dimension based on months. Such slicings results in many small data cubes. For every small cube, we aggregate the D-dimension by

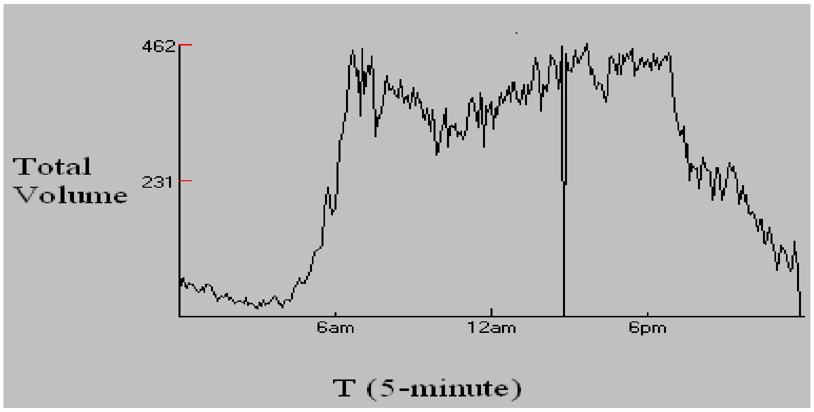


Figure 5.17. Time curve.

group by weekday. Pivoting these small cubes to the D-T face gives us an album shown in figure 5.19. Figure 5.18 illustrates this idea of data cube operations. The shaded faces in the figure are the 2-D matrix in our album. If we pivot the small cubes to the D-S face, we have a D-S album.

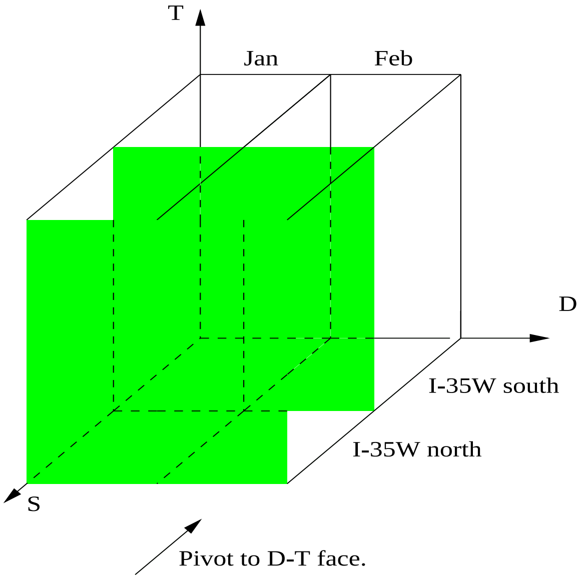


Figure 5.18. Slice data cube and pivot small data cubes to get album.

5.4 Supporting Utilities

5.4.1 Change color scheme function

We developed a convenient interface that enables users to change the color display for total traffic volume, average traffic volume, and average occupancy. This color change is a global modification that will affect all the visualization displays in the system.

5.4.2 Change color threshold function

The thresholds for different color displays can also be modified to better meet different users' requirements for various circumstances. For example, since the peaks of traffic volume on weekday and weekend are different, the intervals of traffic volume display for weekday and weekend should not be set the same.

5.4.3 Conclusion

In this chapter, we discussed three software components to visualize traffic data. Traffic video allows us to get an approximate but rapid summary of major trends. Visualization of attributes provides outlier detection and 2-D data cube visualization. It also has the useful function of highlighting the stations of interest. The data cube visualization component implemented a part of the data cube operations. Users can use it to visualize 2 2-D datacube.

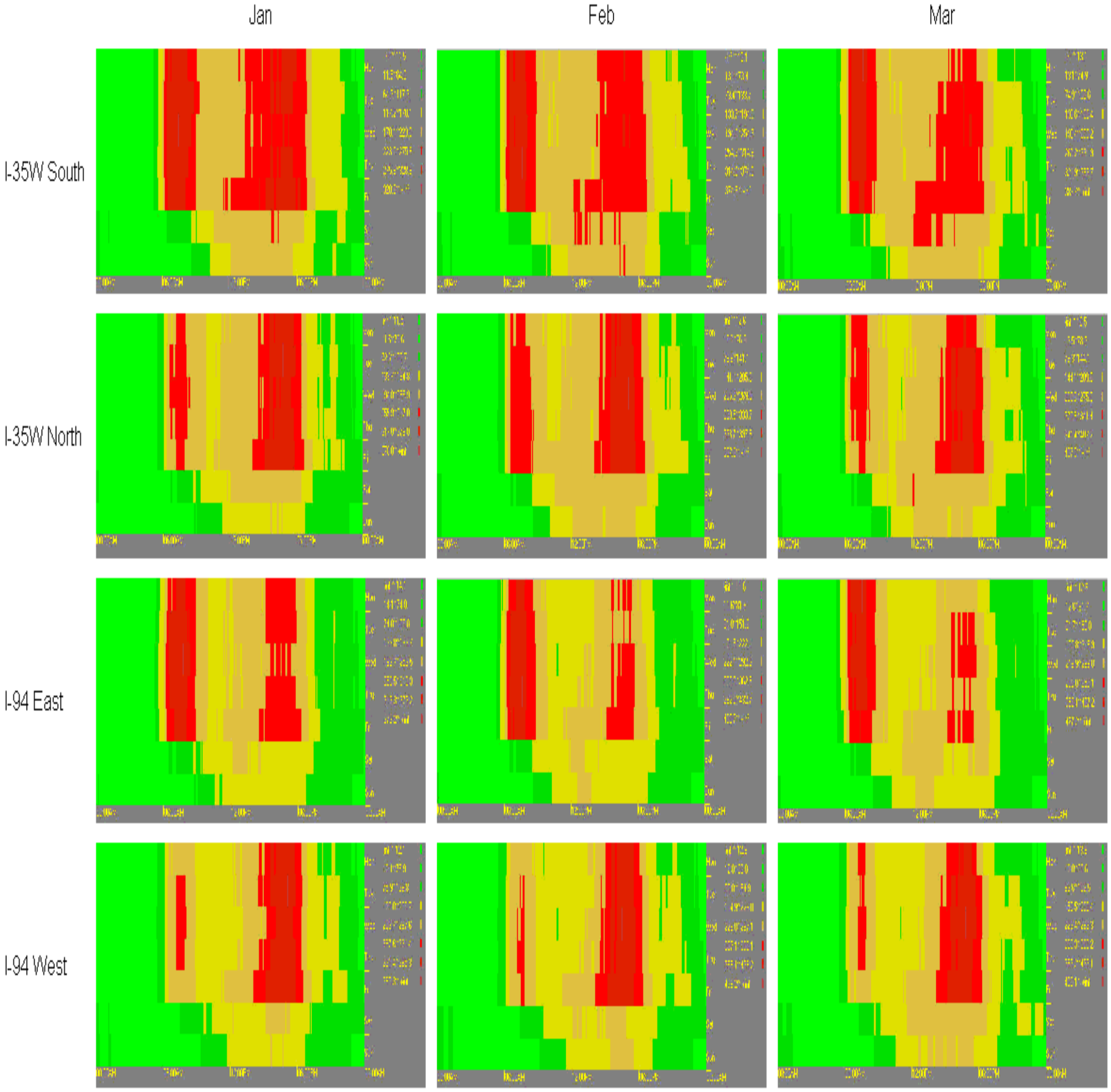


Figure 5.19. 2-D album

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Associations between Sets of Items in Massive Databases. In *Proc. of the ACM-SIGMOD Int'l Conference on Management of Data*, pages 207–216, May 1993.

[2] G. Baecher, J. Zarge, and J. Shapiro. Siteview: Practical geoenvironmental visualization. In *Transportation Research Record, (1556):170-176*, 1996.

[3] P. Bradley, U. Fayyad, and C. Reina. Scaline em(expectation-maximization) clustering to large database. In *Technical Reprot MSR-TR-98-35*, 1998.

[4] J. Burnetti and P. Massimini. Methodology for creating defensible three-dimensional visualizations. In *Transportation Research Record, (1526):166-169*, 1996.

[5] S. Chase, C. Nutakor, and E. Small. An in-depth analysis of the national bridge inventory database utilizing data mining, gis and advanced statistical methods. In Eight Transportation Board Conference on Bridge Management, number C-6; IBMC-047. In *Transportation Research Board*, 1999.

[6] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. In *Proc. VLDB Conference*, page 205, 1996.

[7] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. (1):65–74, March 1997.

[8] S. Chawla, S. Shekhar, W.-L. Wu, and U. Ozesmi. Modeling spatial dependencies for mining geospatial data: An introduction. In *Harvey Miller and Jiawei Han, editors, Geographic data mining and Knowledge Discovery (GKD)*, 1999.

[9] Clementine. Spss inc. http://www.spss.com/clementine/.

[10] E. Codd. Twelve rules for on-line analytic processing. In *Computerworld, April 13 1995.*

[11] E. Codd, S. Codd, and C. Salley. Providing olap(on-line analytical processing) to user-analysts: An it mandate. In *Arbor Software Corporation. Avaliable at http://www.arborsoft.com/essbase/wht_ppr/coddToc.html*, 1993.

[12] J. Ganter and J. Cashwell. Display techniques for dynamic network data in transportation gis. In *In Proceedings of the 1994 Geographic Information Systems for Transportation, pages 42-53*, 1994.

[13] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the Twelfth IEEE International Conference on Data Engineering*, pages 152–159, 1995.

[14] T. Hill. Traffic flow visualization and control(tfvc) improves traffic data acquisition and incident detection. In *Technical Report 0309058740, Transportation Research Board*, 1996.

[15] W. Inmon. *Building the Data Warehouse.* New York, NY: John Wiley & Sons, 1993.

[16] W. Inmon and R. Hackathorn. *Using the Data Warehouse.* New York, NY: John Wiley & Sons, 1994.

[17] W. Inmon, J. Welch, and K. Glassey. *Managing the Data Warehouse.* New York, NY: John Wiley & Sons, 1997.

[18] J. Widom. Research Problems in Data Warehousing. In *Proc. of 4th Int'l Conf. on Information and Knowledge Management(CIKM), Nov. 1995.*

[19] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data warehouse Lifecycle Toolkit.* Wiley, 1998.

[20] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 24th VLDB Conference*, 1998.

[21] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.

[22] H. Landphair and T. Larsen. Applications of 3-d and 4-d visualization technology in transportation. In *Technical Report 0309058740, Transportation Research Board*, 1996.

[23] I. Mumick, D. Quass, and B. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD*, pages 100–111, 1997.

[24] P. Prevedouros, D. Brauer, and R. Sykes. Development of interactive visualization tool for effective presentation of traffic impacts to nonexperts. In *Transportation Research Record,(1463):35-44*, 1994.

[25] J. Quinlan. Induction of decision trees. In *Machine Learning*, number 1, pages 81–106, 1986.

[26] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[27] C. Quiroga and D. Bullock. Architecture of a congestion management system for controlled-access facilities. In *Transportation Research Record, (1551):105-113*, 1996.

[28] S. Ramaswamy, R. Rastongi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Bell Laboratories, Murray Hill, NJ.*

[29] S. Shekhar and D.-R. Liu. Ccam: A connectivity-clustered access method for aggregate queries on transportation networks-a summary of results. *IEEE Trans. on Knowledge and Data Eng.*, 9(1), January 1997.

[30] S. Shekhar, C. Lu, X. Tan, S. Chawla, and R. R. Vatsavai. Mapcubes: A visualization tool for spatial data warehouses. In *Harvey Miller and Jiawei Han, editors, Geographic data mining and Knowledge Discovery (GKD)*, 1999.

[31] S. Shekhar, C. Lu, X. Tan, S. Chawla, and R. R. Vatsavai. Map cubes: A visualization tool for spatial data warehouses. In *Geographic Data Mining and Knowledge Discovery, Harvey Miller and Jiawei Han (eds.).* Taylor and Francis, 2001.

[32] P. Stolortz, H. Nakamura, E. Mesrobian, and et al. Fast Spatio-Temporal Data Mining of Large Geophysical Datasets. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining, AAAI Press*, pages 300–305, 1995.

[33] M. Worboys. *GIS: A Computing Perspective.* Taylor and Francis, 1995.

Appendix: Installation Procedures

.1 Overview

This document describes the steps needs to port the prototype visualization system(CubeView) to another web servers.

The key concept to port CubeView is:

- Put the CGI programs in some directory which can be accessed through web.
- Modify the Java source to reflect the changes of URL of CGIs
- Recompile Java code
- Put the resulted Java class files in a directory which can be accessed by browsers.

Note that some additional work can be done to make the porting without modifying Java source code. The CGI URL should be put in database or a flat text file. Every time the directory is changed, we just need to modify the database or text file.

The database could be any database using SQL as query language. Since the applets are executed by borwser, mostly IE, version 1.1 of Java compiler is enough.

The following sections describe the steps in detail.

.2 Steps

.2.1 Un-tar the CubeView.jar

Copy the CubeView.tar to a directory where to put all the files. Use tar utility to un-tar the whole package:

```
tar xvf CubeView.tar
```

In the CubeView directory, you will find some needed directories:

- cgi-bin: The CGI programs
- Applets: The Java Appelts
- java: The Java programs for data cube visualization
- doc: Documents, including API to Java programs.

.2.2 Modify Java Source Code

The CGI URLs are hard-coded in the source code without porting to other systems in mind when designing CubeView. Go to the directory of CubeView. Use *find* and *grep* commands to find out the places to modify:

```
find . -name “*.java”|xargs grep “http:”
```

Replace the URL before the file name with the new URL. For example, if the new URL for traffic_daily2.cgi is http://new/url/to/cube_view/cgi-bin/traffic_daily2.cgi, the replace 'http://www.cs.umn.edu/research group/vis' with 'http://new/url/to/cube_view'.

.2.3 Recompile Java Source Code

Due to different development stages, there are two directories to compile.

./Applets directory

In this directory, just type

```
javac *.java
```

to do the job.

./java directory

In this directory, use Ant build tool to build the class file:

```
ant jar-2dmap
```

It compiles all the Java source files in ./java directory and jar the class files and create a .jar file. This jar file is located in ./Applets directory

Note that you should require and install Ant on your system for building this directory.

.2.4 Change Permission

CGIs

Change mode to 755 for all CGIs in cgi-bin directory.

```
chmod 755 *.cgi
```

Java Class Files

Change mode to 744 for all Java class/jar files in Applets directory.

```
chmod 744 *.class chmod 744 *.jar
```

After all this, you are all set to use the CubeView to analyze traffic data.

.3 Loading New Traffic Data

The 5-min for all stations for a day is compiled into a text file for loading in the database. We use Perl programs to convert text file to database. Each line in the text file should be in the following format:

station_id, data_for_00:00, data_for_00:05,..., data_for_23:55

The file name of the text file should be in the following format:

TYYYYMMDD.txt

where

- T: Data type. A=Volume; B=Avg Volume; O=Occupancy
- YYYY: Year.
- MM: Month
- DD: Day of month

Use the ins_data2.pl in the ./cgi-bin directory to insert the traffic data to database. The table to insert the data is value_per_day as described in the final report. The command would look like:

```
perl ins_data2.pl A19970101.txt
```