UMD Computer Science
NATSRL Education and Outreach Program

Final Report

Carolyn Crouch
Computer Science Department
University of Minnesota Duluth

Donald Crouch
Computer Science Department
University of Minnesota Duluth

Richard Maclin
Computer Science Department
University of Minnesota Duluth

January, 2004

Published by
Minnesota Department of Transportation
Office of Research Services
Mail Stop 330
395 John Ireland Boulevard
St. Paul, Minnesota 55155-1899

Acknowledgements

The research team gratefully acknowledges the support of CTS and the Minnesota Department of Transportation. The contributions of both the undergraduate (Hemal Lal, Michael Perkins, and Jeff Sharkey) and graduate (Deodatta Bhoite, Anirruda Mahan) researchers have been instrumental to the success of this work.

Table of Contents

Executive Summary

A primary objective of the NATSRL Education and Outreach Program is to provide opportunities for undergraduate students to participate in transportation-related research activities. The research described in this report resulted in the design and implementation of a novel data warehousing method for maintaining and updating transportation data and the design and implementation of various compression methods which can be effectively used to minimize the size of the raw data files when the transportation data is distributed. This foundation provides the basis for our current research—a complex web interface to allow users with no knowledge of computer programming and little knowledge of the data to explore the transportation data quickly and easily. The project has also proved effective in achieving its educational objectives, both with respect to the undergraduate researchers (two of whom have entered graduate school, while a third is still participating in the 2003-2004 project) and graduate researchers, one of whose thesis will be based in large part on this research. This project demonstrates that even undergraduate computer science students, using practical yet innovative approaches, can learn and contribute to the solutions of important real world problems in the area of transportation research.

1. Introduction

A primary objective of the NATSRL Education and Outreach Program project undertaken by the Computer Science Department at the University of Minnesota Duluth is to provide opportunities for undergraduate students to participate in transportation-related research activities. Collaborative research between students and faculty is particularly rewarding if proper care is taken to select projects that are both challenging and feasible. Undergraduate students must be taught the art and practice of conducting research and provided experience in the process performing research. Students chosen to work on the NATSRL projects develop an increased awareness of both transportation issues and the application of computer science to the solution of these issues.

For this project we pursued two areas of research: (1) the design and implementation of a data warehouse for storing a set of traffic sensor data (described below), and (2) the design and implementation of knowledge-based compression methods for maintaining archives of the sensor data. These projects form the basis for a future project which will focus on making the sensor data available for exploration via a web interface.

To understand the research undertaken in this project, we first discuss the traffic data that is being manipulated.

1.1. TMC Traffic Sensor Data

The data used for this research is gathered by the Traffic Management Center (TMC) from over 4000 loop sensors on the freeway system in Minnesota. Most of the sensors are located in the Twin Cities area. Each of the sensors takes two readings of the traffic every 30 seconds. The collected data is then daily packaged into a single zip file and is archived and made publicly available by Dr. Taek Kwon [1].

The sensor provides two types of readings:

1. *Volume*: The number of vehicles passing the sensor every 30 seconds.
2. *Occupancy*: The time during which the sensor was occupied within the 30 second interval.

Valid data in the volume files ranges from 0 to 40 since not more than 40 vehicles can pass a sensor in 30 seconds. Valid data in occupancy files ranges either from 0 to 1800 or from 0 to 1000 depending on how it is formatted (c30 or o30). The c30 file format divides the 30 second interval into 1800 equal parts, whereas the older, o30 file format divides the 30 second interval into 1000 equal parts.

Volume data can be represented in one byte whereas occupancy requires one word to represent every 30 seconds. Thus a file containing 24 hour traffic data for a sensor contains 2880 bytes of volume data and 5760 bytes of occupancy data per sensor.

Traffic data is available from the year 1999 to the current date. But many new sensors were added during that period; not every sensor has data for all 5 years. And since sensors malfunction from time to time, data can be missing or invalid.

To illustrate the differences between volume and occupancy, consider the following four conditions:

- Low Volume, Low Occupancy – very few cars are passing a sensor and the sensor has cars above it for only a short period of time. This would be an ideal time to travel.
- Low Volume, High Occupancy – in this condition, very few cars are passing but a car is occupying the sensor fairly often. This situation occurs when traffic is moving very slow and backs up. This condition may well correspond to a traffic jam.
- High Volume, Low Occupancy – in this condition, many cars are passing but occupancy remains low. Thus traffic is likely to be moving very well with little congestion.
- High Volume, High Occupancy – many cars are passing a sensor and there are cars over the sensor fairly often. This would suggest that the road is congested but that traffic is still moving.

The sensors are located on various highways in the state of Minnesota, primarily in the Twin Cities area. They cover major highways and some of the major connecting roads (such as I-35, I-94, I-694, I-494, I-35E, I35W, etc.).

Sensor data is available from 1999 to the present. Thus we have up to 5 years of data for the 4500 sensors. The total number of data points available for each sensor is approximately:

5 years * 365 days * 24 hours * 120 (half minutes) * 2 (measures) = 10,512,000 data points.

For 4000 sensors, this means up to 4.2 X 10^{10} data points. And of course the data continues to grow daily. Although disk space is relatively inexpensive, the amount of storage needed to hold this data and the time it would take to process it makes direct storage infeasible at this time. Thus we performed research to look for methods to deal with this amount of data.

1.2. Analyzing Traffic Sensor Data

Our research involved investigating methods for effectively building a data warehouse from the archival data, allowing large-scale analyses of the data along multiple axes. We also investigated different methods for compressing the data that can in many cases result in significant space savings (making it possible to store, access and distribute the data more quickly). Sections 1.2.1 and 1.2.2 briefly describe our approach. Chapters 2 and 3 discuss this research in greater depth.

1.2.1. Building a Sensor Data Warehouse

A primary use of sensor data involves performing analyses to answer questions about historical traffic flow. We constructed a data warehouse using a data cube model that would allow for multiple analyses of the data. The idea behind a data cube is that the data is summarized in a matrix defined by two or more axes that can be investigated and contrasted to discover interesting trends in the historical data (see Ch. 2).

We started by choosing a level of the data to use as a summary. For this work, we concentrated on a very short version of the data set (a 5 minute duration summary) as well as a longer term version of one hour's duration. The one hour summary is very compact yet yields many interesting data artifacts when queried. The minute duration summary makes it possible to explore local trends while still smoothing many of the data artifacts.

The data gathered is defined by two key independent attributes (time and location) and measures two dependent variables (volume and occupancy). We applied the somewhat novel approach of using more than one axis in the cube for each of the independent attributes. For example, with respect to time, we defined our data cube as having multiple time axes. The advantage of this approach is that time has multiple aspects that may be of interest. For example, we might be interested in looking at data for a particular region of sensors that covers all the morning rush hour statistics for the period from Jan. 1, 2000 to July 1, 2000. To select this data, we slice along the first time dimension to select only those time periods covering the morning rush hour and along the second time axis for those periods in the first half of 2000. In a similar fashion, we defined a model using more than one location axis in the cube. This allows the user to select sensors by region and by the interaction of the sensors (e.g., all sensors along a north-to-south route across the Twin Cities). Our interactions with Mn/DOT personnel suggest that such a warehouse could prove beneficial for researchers attempting to view historical trends in the traffic flow data and to construct reports on traffic flow.

1.2.2. Methods for Compressing the Data

Dr. Taek Kwon has examined a number of standard techniques for compressing the archived sensor data [1]. We examined several approaches to compression with the objective of achieving significant reductions in data size. Some approaches take advantage of background knowledge about the data.

Our initial efforts focused on determining and implementing an effective form of lossless compression, with the aim of including this approach as part of a larger system. For this work we analyzed well-regarded existing methods and implemented variations of these methods in an effort to produce an effective lossless compression technique for this data.

Our second approach focused on performing a lossy compression of the data. In lossy compression, some specific aspects of the data may be lost, but the compressed result should nevertheless be close to the original data (and hopefully much smaller in size). The objective was to use an analysis of the historical data to compress the data more effectively. The data could then be compressed in one of several ways. For example, the data could be segmented into those data points that appear *normal* (e.g., assuming a normal distribution for a sensor at a particular representative time period such as weekend rush hour, those points that fall within the 95% confidence interval for the distribution of points) and those that are not (i.e., those points showing significant deviation). The former points can be represented with a single bit encoding, and all other points would use a more complex encoding. Although such an encoding would lose the actual values associated with each sensor for that time period, it would retain all of the information about those points that are likely to be outliers for various reasons (e.g., mechanical failures, traffic blockage, etc.). The resulting version of the data would not replace the original archive but could be used as a supplement and made available to those interested in looking for

deviations in the data. This type of analysis can also be useful in setting up tools that look for deviations in the data in real time.

1.3. Organization of this Report

The remainder of this report is organized as follows. In Chapter 2 we present our design and implementation of the data warehouse. Chapter 3 describes our experiments with compression. Chapter 4 is a short discussion of related work. We conclude in Chapter 5 with an overview of our results and suggestions for future research.

2. Building a Traffic Data Warehouse

A main focus of this research was to design and implement a data warehouse using the traffic sensor data. The resulting data representation captures key aspects of the data while ensuring efficient storage and querying of the data. In this chapter, we describe the basic concept of a data warehouse and discuss our implementation of the traffic data warehouse.

2.1. Data Warehouses

With the ubiquitous growth of computers in many areas, such as commerce (supermarkets, chain stores, ..) and science (advances in astronomy, genetics, etc.), the amount of data available about such enterprises is growing at a phenomenal rate. A data warehouse is an important tool in organizing and understanding the enormous amounts of historical data generated by any organization and in making decisions based on the knowledge realized from that data. [2]

Han and Kamber [2] define a data warehouse as a "*a semantically consistent data store that serves as a physical implementation of a decision support data model and stores the information on which an enterprise needs to make strategic decisions.*" Inmon [3] defines a data warehouse as "*a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process.*"

Both definitions state that the data warehouse is a useful tool in support of decision support systems (DSS). We expand below on Inmon's definition, which **gives us** four important properties of the data in a data warehouse.

A data warehouse must be subject-oriented. In other words, before creating a data warehouse, the purpose for creating it should be clear. For example, if a store wants to determine customer buying patterns, then it should store its sales records as opposed to other records (such as payroll records) in the data warehouse.

Most of the data gathered for a data warehouse comes from heterogeneous sources, such as relational databases, palm-tops, flat files, etc. To guarantee consistency, this data must be cleaned and integrated before it is stored in the data warehouse. Decision support systems make decisions based on past experience and patterns. In order to make such decisions, the historical data must be stored. Data in a data warehouse is typically never removed, because the more historical data we have, the better we are able to support our decisions. Thus, the data in a data warehouse must be non-volatile.

In order to learn from past experience, an organization analyzes its historical data and searches to identify various patterns. The ability to recognize patterns in the data helps the organization to improve its performance by basing planning strategies are on them. For example, the department of transportation may realize after looking at the historical traffic flow data that the traffic is often slow on a particular road, so they may then provide an alternate route to improve the traffic conditions. A data warehouse facilitates such analyses of large amounts of data by summarizing

data drawn from heterogeneous sources and presenting it in a manner which helps the users find rules for the data.

2.2. Implementing a Data Warehouse: The Data Cube Model

The data in a data warehouse can be summarized using the data cube model. A data cube is an aggregation operator suggested by Gray et al. [4]. To model the given data using a data cube, one first determines the dimensions of the data, the measure or fact used for analysis, and then summarizes the data along those dimensions.

Dimensions are the perspectives or entities along which an organization wants to keep the records. For example, in our case we want to record traffic data with respect to year, time of the year and sensor location. These are the dimensions for our data warehouse. Measures are the dependent variables in our model. For example, we want to find patterns in the number of cars passing a sensor (volume), so that will become our measure.

Figure 2.1 shows how a data cube might look in 3 dimensions for hypothetical traffic data.

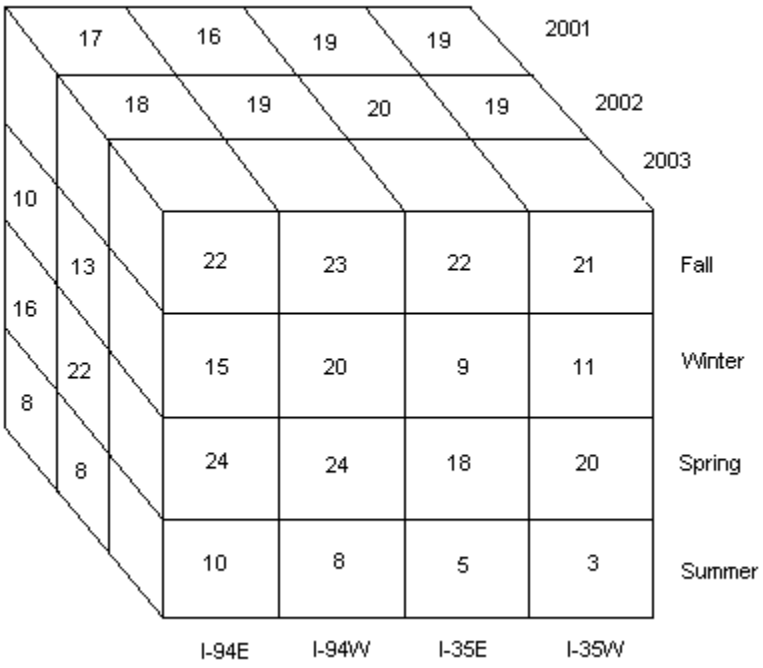


Figure 2.1. Sample data cube for traffic data. This diagram shows how a data cube might look like in three dimensions. The dimensions used in this data cube are highways, year, seasons in year. The measure used is volume (number of cars passing a sensor). A number in a cube represents average hourly traffic on a particular highway during a particular season in a particular year. For example, the leftmost bottom cube has volume 10, means that on average 10 cars passed on every sensor on I-94 E during summer of 2003.

The three dimensions used in the data cube in Figure 2.1 are highways, year and seasons of the year. A number of highways may be viewed, but we have selected only four of them. The data is recorded for three years (2001-2003), and all the four seasons are included. We measure the average hourly traffic on each of the sensors located on a particular highway during a particular season and year. By visualizing a cube in this manner, we can see the data as a whole instead of in tabular form. It is easy to find patterns in such data. We can easily determine that the average traffic on all the highways has increased compared to previous years. We also note that the volume of traffic is particularly low in summer.

The data summarized in the data cube can be analyzed for patterns using various operations on the data cube. Examples of such operations include roll-up, drill-down, slice, dice and pivot.

The drill-down operation adds an additional dimension or divides a dimension into more parts to drill down into the cube. For example, if we want to look at the average traffic during every month of the year (rather than every season), then we drill-down into the time of the year dimension. This drill-down operation gives us a low-level view of the data.

Roll-up reduces one or more dimensions and aggregates the values along remaining dimensions. For example, if we want to see the average I-94 and I-35 traffic, instead of breaking it by their bounds (East and West) we can perform a roll-up on the highway dimension to produce a more high-level view of the data.

The slice operation takes a slice of the data cube. For example, if we want to look at 2003 data for all highways and for all seasons, then we take a slice along the year dimension.

The dice operation is similar to the slice operation, except that it is taken across more than one dimension. Thus we can take a dice along the year and highway dimensions to view only the 2003 data for highway I-35.

The pivot is a visualization operation. It will rotate the data cube with respect to a corner, thereby changing the view of the data cube to help us find patterns in different sets of dimensions.

2.3. The Traffic Data Cube Schema

In most data warehouses, temporal features such as time of the day, day of the week, etc., are considered to be on the same elementary axis (time). This makes it difficult to create summaries that slice time in a number of ways. For example, if we want to examine rush hour traffic during a business day, we are required to break the time dimension in two ways, hours and days of the week. This requires a complex time filter if we consider the days of the week and time of the day along the same dimension as a concept hierarchy. Therefore we chose to split the time dimension into multiple natural time dimensions like time of the day, day of the week, time of the year and year itself.

We also need the spatial aspect of the traffic data. The sensors provide information about the highways in which they are embedded. But in order to perform a query that summarizes the data along highways as well as spatial location on the highways, we have to split the spatial dimension. For example, suppose we want to compare traffic on the north section of I-35 to that on the south section. It is be impossible to do such analysis if we have kept the highway

information and the location (latitude, longitude) information along one dimension. Thus, we split the spatial dimension into the sensor and location dimensions.

We keep the volume (number of cars passing a sensor) and occupancy (proportion of time the sensor was occupied) as the two measures for the data warehouse.

The dimensions of a data cube specify the independent features used to define the cells. We have six data features defining our cube:

1. Time of the day: data summarized with respect to time of the day by each hour.
2. Day of the week: data summarized with respect to day of the week.
3. Time of the year: data summarized with respect to time of the year by each month.
4. Year: data summarized by each year.
5. Sensor: data summarized by sensor numbers with highway information.
6. Location: data summarized by location information about each sensor (i.e., latitude and longitude).

The measures in a data cube are the dependent values stored in that cube entry. For the traffic sensor data we include:

1. Volume: number of cars passing each sensor in 30 seconds averaged over the duration chosen for our data cube (see below).
2. Occupancy: proportion of time the sensor was occupied during those 30 seconds averaged over the duration chosen for our data cube (see below).

A key factor affecting the size and efficiency of a data warehouse is the granularity of each of its dimensions. The granularity of the dimension tables must be carefully chosen so that it does not become too coarse or too fine. If the granularity is very coarse, then the data will lose its interest and become too general. If the granularity is very fine, then the data warehouse will become very large. Querying in this case is inefficient, making analysis very difficult.

In one version of the dataset we chose to summarize the time of the day by every hour. Thus the cardinality for this dimension is 24. We added a Boolean flag associated with the hour to indicate rush hour. In a second version, we summarize over every 5 minutes; the cardinality of this dimension is 288.

The day of the week dimension is summarized by every day of the week; the cardinality for this dimension is 7. We added a Boolean flag to differentiate between weekday and weekend.

The time of the year dimension is summarized by every month in the year. Its cardinality is 12. We added additional fields to the month, including season and quarter of the year. This will make analysis of seasonal trends in traffic possible.

The year dimension simply represents the data summarized by each year. The cardinality of this dimension depends on the number of years for which this data is available; currently its cardinality is 5.

The sensor dimension is summarized by each sensor. We keep highway and bound information (west-bound, east-bound) for each sensor in the table. The cardinality of this dimension depends

on the number of sensors that are recording data. Currently, there are about 4800 sensors recording data.

The location dimension is also summarized by each sensor. We also store latitude and longitude of each sensor. The cardinality of this dimension is identical to that of the sensor table and depends on the number of sensors recording data.

The dimension tables contain only the static information. The schema is based on the star schema used for data warehouses, in which there is one central fact table and multiple supporting dimension tables. The fact table we designed uses surrogate keys to bind the dimensions with the fact tables. We store the volume and occupancy for each combination of the dimensions.

The surrogate keys help to accommodate future changes to the schema. If the number of sensors changes, we do not want to modify the fact table (a costly operation). Instead we use surrogate keys (rather than actual sensor numbers) for the primary and foreign keys in the database schema.

2.4. Implementation Details

The data warehouse is built using MySQL™ database. We use the 4.0.15a version of MySQL for Sun Solaris. The database server is a quad processor machine with each processor having 450 MHz speed. It has 1GB RAM and a 70GB disk space.

The traffic data was downloaded from the tdrl.d.umn.edu server. A Perl script was used to unzip the data and separate the volume and occupancy files. The operating system parameters were modified to store this large number of files.

We created a Java program to summarize the data by hour and then upload the data into the data warehouse. It takes 12-15 days to upload the current data into the database after summarization. But once uploaded, the warehouse can easily be updated with each day's data in just a few minutes.

Following is the list of tools that were built to process this data:

1. *unpack.pl* : Perl script to uncompress the data
2. *MyDataLoader1.java*: Java program to summarize and load the data into the data warehouse. This summarizes the data by 1 hour intervals.
3. *MyDataLoader2.java*: Java program to summarize and load the data into the data warehouse. This summarizes the data by 5 minute intervals.
4. *Wrapper.java*: This provides a wrapper for MyDataLoader. It calls the class iteratively for all sensors and all time periods of the year.
5. *upload.sh*: This script calls Wrapper.java for a complete year (or more depending upon how it is configured).

We have also built a data visualization tool in Java in order to understand the data. This tool summarizes the data by the hour and displays it using bar graph visualization.

We also constructed some SQL scripts to build the data warehouse schema and to optimize the data warehouse by building indexes. A summary of these SQL scripts follows.

1. *mysqlschema.sql*: SQL script to create the data warehouse schema for 1 hour summarizations.
2. *mysqlschema2.sql*: SQL script to create the data warehouse schema for 5 minute summarizations.
3. *insertGen.pl*: perl script to create a SQL script for inserting sensor data and location data.
4. *insertLocation.sql*: SQL script for inserting location data into the data warehouse.
5. *insertSensor.sql*: SQL script for inserting sensor data into the data warehouse.
6. *optimize.sql*: SQL script to create indexes and thus optimize the query timings.

2.5. Some Characteristics of our Data Warehouse

The size of the database is a critical factor. It is determined by the number of dimensions and the granularity of those dimensions. We have six dimensions. However, location and sensor dimensions have a one-to-one mapping so we can safely consider them as one dimension in our calculations for size of the database.

The size of the database is calculated as follows:

Number of rows = time of day X day of week X time of year X years X number of sensors

= 24 X 7 X 12 X 5 X 4500

= 45.36 million

If each row is 12 bytes then the database size is about 544 MB.

We also built another data warehouse based on 5 minute summarizations. The size of this database is:

= 288 X 7 X 12 X 5 X 4500

= 544 million

If each row is 12 bytes then the database size is about 6.5 GB

In order to assess how efficiently data can be accessed, we performed several sample queries measuring query time both before and after indexing (to check the amount of optimization resulting from indexing).

Figure 2.2 shows a sample query performed on the data warehouse. As we see, a very significant time reduction results from the use of the indices. The data warehouse was optimized to make the queries, like this example, more efficient.

The query times usually vary from fractions of seconds to a few seconds, depending on the complexity of the query involved. Usually, the larger the number of dimensions we have to group, the larger the query time. For example, the query in Figure 2.3 takes longer to execute because it involves 4 dimensions and we are considering the whole dataset, not a part of it.

```
mysql> select SR_ID, DOW_ID, AVG(VOLUME), AVG(OCCUPANCY)
    -> from TRAFFIC_DATA
    -> WHERE YR_ID=4
    -> AND (SR_ID=305 OR SR_ID=306)
    -> GROUP BY SR_ID, DOW_ID;

Timings before indexing
Time 1: 53.86 sec
Time 2: 51.77 sec
Time 3: 52.06 sec

Query 1 timings after indexing
Time 1: 0.52 sec
Time 2: 0.28 sec
Time 3: 0.28 sec
```

Figure 2.2. Sample query 1 and corresponding timing information for data warehouse.

```
Query 2
SELECT YR_ID, TOY_ID, DOW_ID, TOD_ID, AVG(VOLUME), AVG(OCCUPANCY)
FROM TRAFFIC_DATA
GROUP BY YR_ID, TOY_ID, DOW_ID, TOD_ID;

Query 2 timimg: 4 min 25.36 sec
```

Figure 2.3. Sample query 2 and corresponding timing information for data warehouse.

We also found out that the Group By clause is more efficient than combining the results of two separate queries. For example, Figure 2.4 shows the results of one complex query that is more efficient than two simpler queries.

```
Query 3
mysql> SELECT T.SR_ID, D.DOW_WEEKDAY_FLAG, AVG(VOLUME), AVG(OCCUPANCY)
    -> FROM TRAFFIC_DATA T, DAY_OF_WEEK D
    -> WHERE T.DOW_ID = D.DOW_ID
    -> AND T.YR_ID=4
    -> AND (T.SR_ID=305 OR T.SR_ID=306)
    -> GROUP BY T.SR_ID, D.DOW_WEEKDAY_FLAG;
Query 3 timing: 0.31 sec


Query 4
mysql> select SR_ID, AVG(VOLUME), AVG(OCCUPANCY)
    -> from TRAFFIC_DATA
    -> where (DOW_ID=1 OR DOW_ID=7)
    -> AND YR_ID=4
    -> AND (SR_ID=305 OR SR_ID=306)
    -> GROUP BY SR_ID;

Query 5
mysql> select SR_ID, AVG(VOLUME), AVG(OCCUPANCY)
    -> from TRAFFIC_DATA
    -> where (DOW_ID BETWEEN 2 and 6)
    -> AND YR_ID=4
    -> AND (SR_ID=305 OR SR_ID=306)
    -> GROUP BY SR_ID;
Query 4 + 5 timing: 0.26sec + 0.28sec = 0.54sec
```

Figure 2.4. Sample of one complex query (3) that is more efficient than two simpler component queries.

3. Knowledge-Based Compression Methods

In this chapter, we summarize our experiments in the compression of NATSRL traffic data. The data exists in binary form. Data is collected by each sensor on a daily basis. Each sensor produces two files:

1. Sensor_no.v30 : Volume file, stores the volume of traffic passing that sensor.
2. Sensor_no.c30 : Occupancy file, stores the amount of time the sensor was occupied.

Each sensor writes data every 30 seconds (one byte for a .v30 file and 2 bytes for a .c30 file). Thus a .v30 file is 2880 bytes and a .c30 file is 5760 bytes in length. There are about 4500 sensors. As not every sensor produces data every day, the average total data per day is about 32 Megabytes.

3.1. Lossless Compression

We started our investigations by looking at standard data compression applications. Some eight applications were studied, for Windows as well as UNIX platforms. We examined the following standard compression methods: Pkzip [5], Winzip[6], Tar-Gzip [7], Szip [8], Winrar [9], RK [10], PPMZ2 [11]. To test these tools we ran them on 40 days worth of data. The days were arbitrarily selected from the years 1999 to 2002. The average compression as obtained from these utilities ranged from 52% (pkzip) to 60%(szip). Table 3.1 shows the results for all of these methods.

Table 3.1. Compression results for various standard tools on a representative 30 days of data.

Tool Used	***Average Compression observed***
SZIP	60.39%
RK	57.22%
Stuffit	56.28%
Compressia	55.47%
WINRAR	52.89%
PKZIP	52.81%
Tar Gzip	52.22%
PowerArchiver	52.00%

The utility Szip was found to have (1) the highest compression ratio, (2) the fastest compression speed, and (3) the most consistent compression ratio.

These tests gave us an idea of how much compression a professional utility would provide. We also looked at various compression algorithms and strategies independently (that is, not as incorporated within one of the aforementioned techniques). We considered specifically run-length encoding, Huffman coding and a 6-bit representation. However, none of these algorithms gave satisfactory compression independently. So we then examined the feasibility of using these methods in sequence to take advantage of the characteristics of the data and the strengths of the methods themselves. The most promising algorithms in this regard appeared to be Run-Length Encoding (RLE) and the Burrows Wheeler Transform (BWT).

Run-Length Encoding is a very simple form of data compression in which *runs* of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count rather than as the original run. This is most useful for data that contains many such runs; for example, simple graphic images such as icons and line drawings.

The Burrows Wheeler Transform [12] is a transformation which uses a number of other steps to produce string sequences which consist of runs so that RLE can effectively compress them. Thus logically RLE should follow BWT in any algorithm sequence. BWT may be preceded by any compression algorithm if the need is felt. BWT took some time to code. The main concern was that it was very expensive, time-wise. We refined the code to make it more efficient.

The final sequence of algorithms was decided through many trial runs and experimentation:

1. Run Length Encoding
2. Burrows Wheeler Transform
3. Move to Front Transform
4. Run Length Encoding
5. Arithmetic coding

The average loss-less compression obtained through the use of this algorithm sequence was 60%. More detailed information is found in the next section.

3.1.1. An Example of the Burrows-Wheel Transform

The Burrows-Wheeler Transform (BWT) is an algorithm used in data compression. When a string is transformed by the BWT, none of its characters change, but rather the order of the characters changes. If the original string has several substrings that re-occur often, then the transformed string will have several places where a single character is repeated multiple times in a row. This is useful for compression, since compressing a string that has runs of repeated characters is easy (e.g., run-length encoding). Consider a simple example – transformation of the string 'banana' into 'bnnaaa'.

The transformation is done by sorting all rotations of the text and then taking the last column. The BWT includes a transformation called the Move-to-Front Transformation. BWT can be implemented with or without the MTF transform. However, if MTF is implemented, then BWT followed by run-length encoding will produce improved results. We implemented the Burrows-Wheeler Transform with the Move-to-Front transform. Below we present a step-by-step example.

BWT Transform and Move To Front - An Example

Original String = DRDOBBS

1) Permute by left shifting and wrapping.

String 0	D	R	D	O	B	B	S
S 1	R	D	O	B	B	S	D
S 2	D	O	B	B	S	D	R
S 3	O	B	B	S	D	R	D
S 4	B	B	S	D	R	D	O
S 5	B	S	D	R	D	O	B
S 6	S	D	R	D	O	B	B

	F						L
S 4	B	B	S	D	R	D	O
S 5	B	S	D	R	D	O	B
S 2	D	O	B	B	S	D	R
S 0	D	R	D	O	B	B	S
S 3	O	B	B	S	D	R	D
S 1	R	D	O	B	B	S	D
S 6	S	D	R	D	O	B	B

2) Sort Lexicographically.

3) Finish sort by formatting a new string. C(s) = Letters in last column "OBRSDDB" plus integer index 5 where first letter of the original string is now located.

4) Perform Move-To-Front (MTF) Transformation.

a) Create a list of all possible values numerically numbered

0 1 2 3 4

a b c d e

b) Get a character from input. Assume it is 'c'

c) Write out the number for 'c' so 2 then move c to the front of the list

0 1 2 3 4

c a b d e

d) Repeat steps b through c until the end of ... let's say next is 'd' write

out 3 and table is now dcabe.

Another example "tttWtwttt" = { 116, 0, 0, 88, 1, 119, 1, 0, 0 }.

e) Store the table somewhere in our compressed file.

5. Entropy encode everything with RLE (this seems to work very well in practice) or Huffman encoding.

Decoding: To recover the original string the following steps are required:

1) Undo the RLE

2) Undo MTF

3) Sort C(s) to determine the first letters of the previous matrix.

col1 col2

B.....O
B.....B
D.....R

D.....S
O.....D
R.....D
S.....B

4) Transform vector routes (e.g. match letter in row of first column with the first appearance of letter in second column)

B = 1
B = 6
D = 5
D = 4
O = 0
R = 2
S = 3

5) V = {1,6,4,5,0,2,3}Take the vector indexes and use them to rearrange string OBRSDDB. Basically, move between indexes. Start at primary index of 5. You get 'D'. Use that index into our vector table. Index 5 of the vector tells us to select index 2 of our string. That's 'R'. Index 2 in our vector table tells us to go to index 4 or our string which is 'D'. Continue process and string becomes DRDOBBS again.

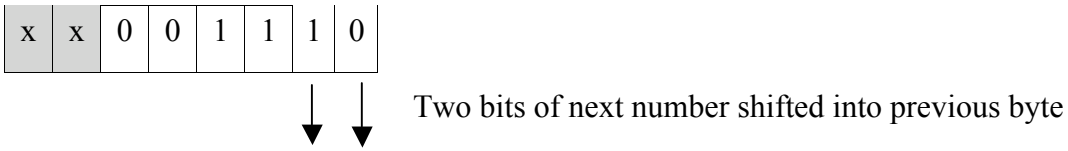
3.1.2 Other Methods

Other compression methods explored were:

- Huffman Coding
- Run-Length Encoding
- 6-bit representation

Huffman coding is an entropy-encoding algorithm used for data compression. The data to be compressed is considered as a sequence of strings. The strings likely to appear frequently are denoted by a shorter sequence of bits (code) and symbols which appear rarely are represented by a longer sequence.

While Huffman coding and run-length encoding are standard compression algorithms, *the 6-bit representation* is an innovative approach towards reducing file size. It utilizes the fact that the data inside the olume files has a fixed range and any number outside that range is considered erroneous. The given range of data for volume files is 0-40. These values need at most a 6-bit representation (out of the 8 in a byte). Thus 2 bits are actually wasted per byte. This approach uses these 2 bits to store a part of the next piece of information. Thus the numbers were actually shifted to the previous byte:



x	x	0	0	1	1	0	1

None of these three methods gave sufficient compression on its own. A combination of these methods was also tried, but it proved to yield no significantly improved compression and was more expensive, time-wise.

The results of these simple compression methods (independent) are shown in Table 3.2.

Table 3.2 Compression results for several algorithms on 30 days of test data.

Algorithm	***Average Compression obtained***
Run Length Encoding (lossy)	25%
Huffman Coding (lossy)	32%
BWT sequence	60%

3.2. Lossy Compression

The compression methods investigated to date were of the lossless type. Lossless data compression algorithms allow the original data to be reconstructed *exactly* from the compressed data. Lossless data compression is used when it is important that the compressed data can be uncompressed and returned to its original form.

Lossy data compression refers to the type of compression wherein some data may be lost when compression occurs—i.e., the file when decompressed may differ from the original but often is close enough to it to be useful. Lossy data compression is used exhaustively these days in mp3 files, storing video, telephony applications and image processing. As we can afford to 'lose' some data, lossy compression algorithms give us much more compression ratios than lossless ones.

Thus the advantage of a lossy method is that is many cases it produces a much smaller compressed file than any known lossless method. On the other hand, the advantage of a lossless method is that it gives 'true' data even after decompression.

We decided to try lossy compression on the traffic data in a search for higher compression ratios. In our first approach, data falling within certain standard deviations of the mean were replaced by their respective means. This is a preset method and to make the encoding dynamic, the data is partitioned into buckets, each containing approximately 15% of the data. This procedure was followed by the BWT sequence detailed above. Normal traffic data ranges as well as anomalies can be filtered out for observation. The compression obtained by the above mentioned method along with the original algorithm sequence was 77%.

With lossy compression, we can decide upon the amount of compression we want. There is a trade-off between the compression ratio and the fidelity of the data. As the compression ratio increases, so does the mean square error, which means the decompressed data will deviate more from the original. We looked at the variance and standard deviation of the lossy approach to determine the golden mean which would give us sufficient compression without sacrificing significantly with respect to the fidelity of the data.

Table 3.3. Compression percentage for various bucket sizes.

Bucket Size	***5.00%***	***10.00%***	***15.00%***	***20.00%***
Original Data Size (Mb)	30.7	30.7	30.7	30.7
BWT compressed data size (Mb)	9.3	8	7.2	5.9
Compression Obtained	70%	74%	77%	81%
Number of Buckets	26	14	10	7

A bucket size of 15% was selected as a balance between compression and mean square error. The average compression obtained from this configuration was 77%. Thus either lossless or lossy compression can be used as required.

4. Other Related Work

A description of the archived sensor data in unified traffic data format, its representation in common data format as done by Dr. Kwon and his group, and the evaluation of the various compression algorithms supported by CDF may be found in [13]. A good overview of the Caltrans freeway performance measurement system that extracts data from both real time and historical data is given in [14]. This system is based on Oracle and utilizes the underlying structure and software to considerable advantage.

Data mining and data warehousing are increasingly popular areas of research [2]. For this work, in the area of data warehousing, we are using the data cube model [15,16] of a warehouse implemented in a standard DBMS (MySQL). Although data warehousing has not been widely investigated for transportation data, some relevant research exists. Shekhar, *et. al*., [17] discuss a traffic data warehousing model to facilitate data mining and visualization. This paper gives a good review of data warehousing technology and its benefits (including a possible implementation of a data warehouse for spatial data based on a data cube), with an emphasis on the mining of the data and knowledge discovery. The research proposed in [17] focuses on addressing issues in finding special correlations in the data and is further developed in [18] with respect to spatial data mining. The mining of association rules [19] is an extremely active area of research in data mining and warehousing.

For compression, we have examined a number of standard compression techniques, including SZIP [8], in building our tool. Our compression mechanism is based on many of the same components used in SZIP [8]. Our main advantage is that we are tailoring our compression technique to the data being compressed, using both historical information and knowledge about the domain.

Only recently, data warehouses have begun to be applied to transportation data. O'Packi et al. [20] implement a GIS linked spatial database to keep track of components such as road, bridge, accidents, etc. Papiernik et al. [21] implement a data warehouse to maintain and analyze performance measurements for the construction programs undertaken by VDoT. None of these analyze the traffic flow data. In this project, we have built a data warehouse to analyze the traffic flow on various freeways across Minnesota area, which will help MNDoT analyze patterns in the traffic flow on various freeways and plan for additional freeways if needed based on the empirical evidence provided by the data warehouse.

5. Conclusions and Future Directions

A primary objective of the NATSRL Education and Outreach Program is to provide opportunities for undergraduate students to participate in transportation-related research activities. The research described in this report resulted in the design and implementation of a novel data warehousing method for maintaining and updating transportation data and the design and implementation of various compression methods which can be effectively used to minimize the size of the raw data files when the transportation data is distributed. This foundation provides the basis for our current research—a complex web interface to allow users with no knowledge of computer programming and little knowledge of the data to explore the transportation data quickly and easily. The project has also proved effective in achieving its educational objectives, both with respect to the undergraduate researchers (two of whom have entered graduate school, while a third is still participating in the 2003-2004 project) and graduate researchers, one of whose thesis will be based in large part on this research. This project demonstrates that even undergraduate computer science students, using practical yet innovative approaches, can learn and contribute to the solutions of important real world problems in the area of transportation research.

References

1. T. Kwon, N. Dhruv, S. Patwardhan, E. Kwon. "Common Data Format Archiving of Large-Scale Intelligent Transportation Systems Data for Efficient Storage, Retrieval, and Portability," *Journal of the Transportation Research Board*: *Transportation Research Record 1836*:111-117, National Academy of Science, 2003.

2. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

3. W. Inmon. *Building the Data Warehouse*. John Wiley and Sons, Inc., 1996.

4. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. "Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals." *J. Data Mining and Knowledge Discover, 1(1)*:29-53, 1997.

5. S. Apiki. "Lossless Data compression," Byte 16(3): 309-312, 314, 386-387, March 1991.

6. http://www.winzip.com

7. P. Deutsch. GZIP file format specification version 4.3. RFC1952, Network Working Group, May, 1996.

8. M. Schindler. A Fast Block-sorting Algorithm for lossless Data Compression, 1996. The Szip homepage, http://www.compressconsult.com/szip/, 1997.

9. The WinRar website, http://www.win-rar.com

10. The RK website, http://rksoft.virtualave.net/rk.html

11. J. Cleary and I. Witten, "Data Compression Using Adaptive Coding and Partial String Matching", *IEEE Transactions on Communications*, 396-402, 1984.

12. Burrows, Michael, David Wheeler. "A Block-sorting Lossless Data Compression Algorithm." Digital Systems Research Center, Palo Alto, California, 1994.

13. N. Dhruv, Design of a Large Scale Archival Retrieval System for Transportation Sensor Data, M.S. Thesis, Department of Computer Science, University of Minnesota Duluth, May, 2002.

14. C. Chen, K. Petty, A. Skarbardonis, P. Varaiya, and Z. Jia. "Freeway Performance Measurement System-Mining Loop Detector Data," *Transportation Research Record 1748*, No. 01-2354, 96-102.

15. V. Harinarayan, A. Rajaraman and J. Ullman. “Implementing data cubes efficiently,” *ACM SIGMOD '96,* 205-216, 1996.

16. J. Widom. “Research problems in data warehousing,” *4th International Conference on Information and Knowledge Management*, Baltimore, MD, 25-30, 1995.

17. S. Shekhar, C. Lu, S. Chawla, P. Zhang. “Data Mining and Visualization of Twin-Cities Traffic Data,” technical report, Computer Science Department, University of Minnesota Twin Cities, 2000.

18. S. Chawla, S. Shekhar, W. Wu. “Modeling spatial dependencies for mining geospatial data: a Statistical approach,” technical report, Department of Computer Science, University of Minnesota Twin Cities, 2000.

19. R. Agrawal and R. Srikant. “Fast algorithms for mining association rules,” *Proc. 20th Int. Conf. Very Large Data Bases, (VLDB),* Morgan Kaufman, 487-499, 1994.

20. P. O'Packi, R. Dubois, N. Arementrout, and S. Bower. “Maine's approach to data warehousing for state departments of transportation,” *Transportation Research Record 1719, 1(1037):*227-232, 2000.

21. D. Papiernik, D. Nanda, R. Cassada, and W. Morris. “Data warehouse strategy to enable performance analysis,” *Transportation Research Record 1719, 1(0603):*175-183, 2000.