MANTIS: A PREDICTIVE DRIVING DIRECTIONS RECOMMENDATION SYSTEM

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Christopher Hoover

June 2013

COMMITTEE MEMBERSHIP

TITLE:	Mantis: A Predictive Driving Directions Recommendation System
AUTHOR:	Christopher Hoover
DATE SUBMITTED:	June 2013
COMMITTEE CHAIR:	Franz Kurfess, Ph.D. Computer Science California Polytechnic State University San Luis Obispo, CA
COMMITTEE MEMBER:	Alexander Dekhtyar, Ph.D. Computer Science California Polytechnic State University San Luis Obispo, CA
COMMITTEE MEMBER:	Anurag Pande, Ph.D. Civil and Environmental Engineering California Polytechnic State University San Luis Obispo, CA

Abstract

Mantis: A Predictive Driving Directions Recommendation System

Christopher Hoover

This thesis presents Mantis, a system designed to evaluate possible driving routes and recommend the optimal route based on current and predicted travel conditions. The system uses the Bing Maps REST service to obtain a set of routes. Traffic data from the California Department of Transportation's Performance Measurement System (PeMS) is then used to estimate travel times for these routes. In addition to simple travel time estimation based on instantaneous traffic conditions, Mantis can use historic data to predict traffic speeds at future times. This allows Mantis to more effectively account for regularly repeating traffic patterns such as rush hour, increasing the accuracy of its travel time estimates. Mantis is also capable of monitoring traffic incidents reported by the California Highway Patrol and identifying incidents that will be encountered along a route's path.

Contents

List of Tables

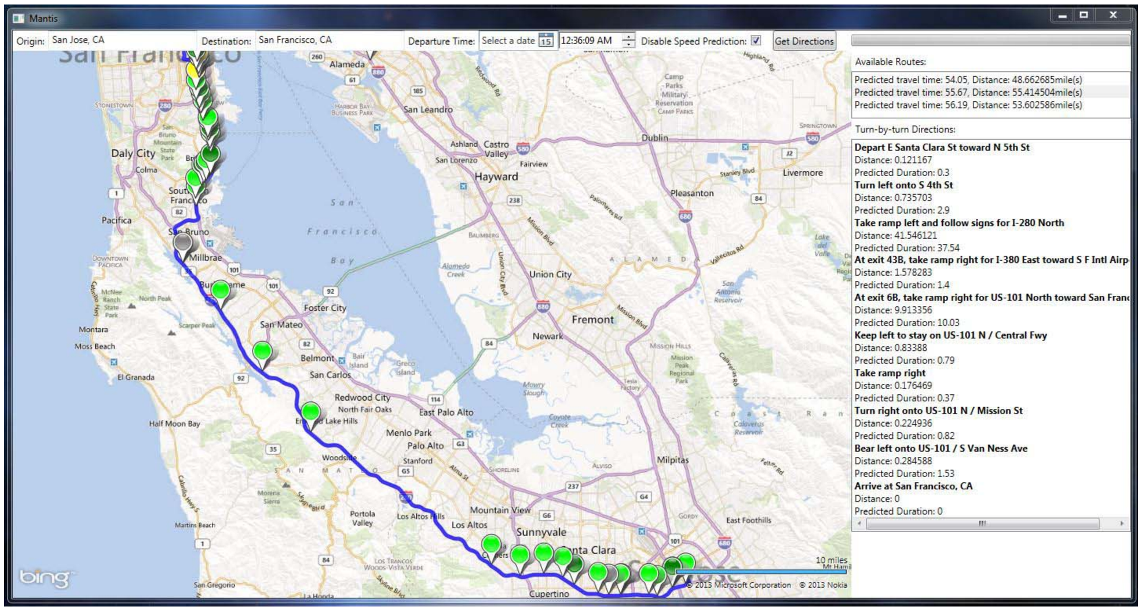
List of Figures

Chapter 1

Introduction

With the growth of technology like web applications and smart phones, it is now easier than ever to simply go online and get detailed driving directions to any address almost instantly. Many such services will also estimate travel time to the destination based on static factors such as speed limits along the route. However, many other dynamic conditions also influence the actual travel time for a given route. For instance, in urban areas, the day and time of travel have a substantial impact on overall traffic levels. A freeway may be the fastest route on weekends or in early afternoon, but morning or evening peak hour may reduce that same freeway's pace to a crawl, drastically increasing commute time. Accidents and other unpredictable hazards may cause delays at any time, and road closures may force drivers to adjust their planned routes with little to no warning. Thus, a truly accurate prediction of travel time for a route is often impossible when based only on static properties of the route itself.

In light of these challenges, I have developed Mantis, a driving directions recommender system that evaluates several possible routes to a destination using a variety of auxiliary data sources. Mantis is capable of monitoring freeway traffic speeds in real time using data from a detector network maintained by the California Department of Transportation, or CalTrans.



Origin: San Jose, CA
Destination: San Francisco, CA
Departure Time: Select a date 12:36:09 AM
Disable Speed Prediction: ✓

Available Routes:
Predicted travel time: 54.05, Distance: 48.662685mile(s)
Predicted travel time: 55.67, Distance: 55.414504mile(s)
Predicted travel time: 56.19, Distance: 53.602586mile(s)

Turn-by-turn Directions:
Depart E Santa Clara St toward N 5th St
Distance: 0.121167
Predicted Duration: 0.3
Turn left onto S 4th St
Distance: 0.735703
Predicted Duration: 2.9
Take ramp left and follow signs for I-280 North
Distance: 41.546121
Predicted Duration: 37.54
At exit 43B, take ramp right for I-380 East toward S F Intl Airp
Distance: 1.578283
Predicted Duration: 1.4
At exit 6B, take ramp right for US-101 North toward San Franc
Distance: 9.913356
Predicted Duration: 10.03
Keep left to stay on US-101 N / Central Fwy
Distance: 0.83388
Predicted Duration: 0.79
Take ramp right
Distance: 0.176469
Predicted Duration: 0.37
Turn right onto US-101 N / Mission St
Distance: 0.224936
Predicted Duration: 0.82
Bear left onto US-101 / S Van Ness Ave
Distance: 0.284588
Predicted Duration: 1.53
Arrive at San Francisco, CA
Distance: 0
Predicted Duration: 0

Figure 1.1: Mantis main screen.

Mantis also detects California Highway Patrol incidents as they are reported on the CHP website, and is capable of determining which of these incidents lie along a given route's path. In addition to real-time traffic monitoring, Mantis attempts to predict future traffic speeds at specific locations using past traffic data from CalTrans. The ultimate goal of Mantis is to suggest the best driving route based on both current and expected conditions.

Mantis is written as a desktop application in C#. The primary GUI component is a Bing Maps map view, which is used to display the selected route's path. This pane also displays other useful information as pushpin icons. Traffic measurement stations are indicated by green, yellow, red, or black pushpins, corresponding to low, moderate, high, and very high congestion, respectively. In addition, Mantis displays any highway patrol incidents as yellow-orange pushpins.

Above the map view are the primary user inputs. Aside from allowing the user to specify a departure and arrival location, Mantis also supports selection of a desired departure date and time, which permits users to evaluate potential routes in advance. Users can also choose to enable or disable speed prediction. If this feature is disabled, Mantis will use current real-time traffic speeds in its travel time calculations. Otherwise, Mantis will attempt to predict traffic speeds for the selected date and departure time.

The right-hand pane displays route information. At the top right of the main window, Mantis displays a list of possible routes. The user can select a different route by clicking on its entry in the list. Mantis automatically refreshes the map view to show the new route's information. Finally, detailed instructions for following the selected route are given in the pane below the route selection area. Distance and predicted travel time are given for each individual step in the route.

Chapter 2

Background

2.1 Traffic Detectors

The foundation of my work is the network of traffic sensor stations operated by the California Department of Transportation, or Caltrans. This sensor network is primarily composed of wire loop detectors, which use loops of wire as their sensor mechanism. Each such wire loop rests in a slot cut in the freeway's pavement. An electronic signal is constantly sent through these wire loops to create an electromagnetic field. When a vehicle passes over a loop, it acts like a conductor and disrupts the loop's field. This is detected by a roadside Vehicle Detector Station, or VDS, that can monitor and control several individual loops. Finally, the raw data collected by the VDSs is sent to Caltrans over a cellular network. [14]

2.2 Performance Measurement System (PeMS)

The PeMS (Performance Measurement System) project began in the late 1990s, with the goal of creating a tool to guide improvements to freeway efficiency. It was deployed for

California on a state-wide scale in 2002 [16]. Currently, PeMS collects and analyzes real-time traffic data for nine of the twelve California districts.

2.2.1 Usage Scenarios

A report by Pravin Varaiya [15] describes a variety of uses for PeMS. This discussion is summarized below.

First, transportation managers can use PeMS as a performance analysis tool. Its ability to compute quality and performance metrics based on incoming data allows managers to evaluate current freeway performance compared to the past, or to examine trends in performance over time. PeMS can also compare the performance of individual freeways, which can help to identify areas of poor performance and to guide prioritization of resources for improvements.

For traffic engineers, PeMS is again useful as an analysis tool. In particular, the system can help to identify areas and times of peak congestion. By examining historical data, an engineer can use PeMS to determine if new data is anomalous or not. This can alert engineers to problems with the detectors themselves.

Transportation researchers and planners can use PeMS to guide their decisions as well. Since building new freeways is an expensive proposition, [15] argues it is more cost-effective to analyze and improve existing freeways instead. According to [15], its computed quality measures can aid in determining theoretical potential for improving specific freeways. For instance, [15] discusses a scenario in which PeMS could be used to determine a freeway's maximum traffic capacity. This could, in turn, provide a target traffic flow for ramp metering systems.

PeMS is not only useful for Department of Transportation workers. The system also allows Value-Added Resellers, or VARs, to access the data it collects in real time via an FTP server. This data can then be used freely in third-party traffic applications. Finally, PeMS also offers limited route planning features for travelers. Using the traffic data it collects, PeMS can calculate travel times for freeway segments. [15] also indicates that PeMS can display historical travel time distributions for particular freeway stretches. These features are only applicable for portions of individual freeways. However, [15] indicates that at the time of its writing, no other system provided an equivalent service.

2.2.2 Estimating Traffic Speeds

The PeMS website [8] includes a description of the algorithm that PeMS uses to estimate traffic speeds. The data processing procedure used in PeMS is outlined in Figure 2.1. Because my work is primarily concerned with the traffic speeds PeMS reports, this discussion focuses on the speed computation portion of the process.

The detectors used in PeMS report two values every thirty seconds: flow (the number of vehicles detected over the detector during the sample period) and occupancy (the percentage of the sample period that the sensor detected a vehicle passing over it). Before estimating traffic speeds, PeMS first aggregates these 30-second values into 5-minute samples. For each lane at a VDS, flow values are summed, while occupancy values are averaged.

Once the raw data has been aggregated, PeMS estimates traffic speeds based on the collected flow and occupancy data. PeMS uses adaptive algorithms described in [12]. This paper first notes that speeds can be estimated using a simple formula:

$$v(t) = g(t) \times \frac{c(t)}{o(t) \times T} \tag{2.1}$$

For a sample period t (with duration T), $v(t)$ is the computed speed, $c(t)$ is flow, $o(t)$ is occupancy, and $g(t)$ is the "g-factor", or effective vehicle length. Accurately estimating g-

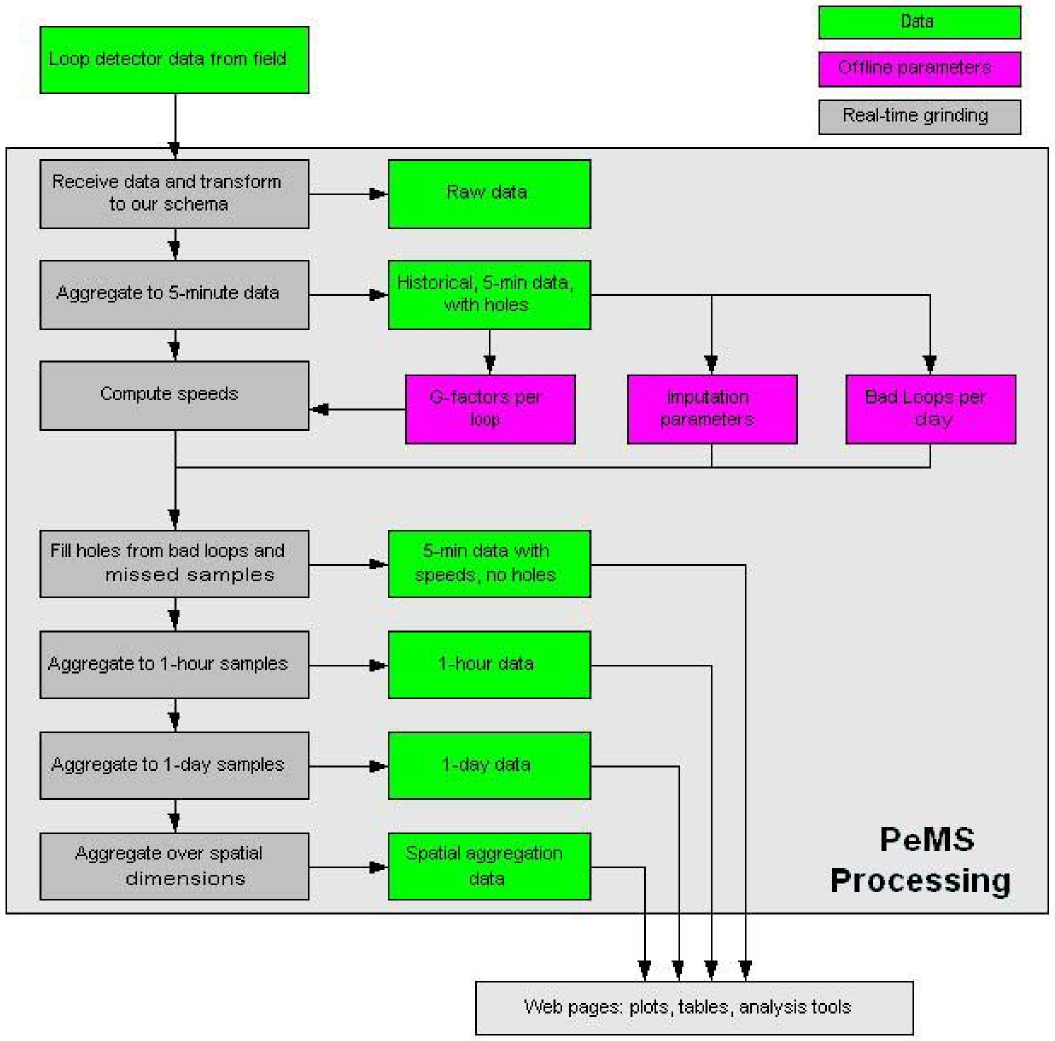

Data
Offline parameters
Real-time grinding
Loop detector data from field
Receive data and transform to our schema
Raw data
Aggregate to 5-minute data
Historical, 5-min data, with holes
Compute speeds
G-factors per loop
Imputation parameters
Bad Loops per day
Fill holes from bad loops and missed samples
5-min data with speeds, no holes
Aggregate to 1-hour samples
1-hour data
Aggregate to 1-day samples
1-day data
Aggregate over spatial dimensions
Spatial aggregation data
PeMS Processing
Web pages: plots, tables, analysis tools

Figure 2.1: Data Processing Flow in PeMS

factors is crucial for obtaining accurate speed estimates. However, many detectors monitored by PeMS are single-loop detectors, which cannot directly measure g-factors. In addition, as discussed in [12], g-factors are highly variable between individual detectors and over time for the same detector. Thus, PeMS must attempt to estimate g-factors as part of its calculations.

[12] states that equation 2.1 can be rewritten as follows, where v_{free} is the free-flow traffic speed (i.e., the speed when traffic is uncongested) and is assumed to be known:

$$g(t) = \frac{o(t) \times T}{c(t)} \times v_{free} \tag{2.2}$$

[12] notes that this equation is not applicable in congested conditions, and also fails if no vehicles are detected during the sample period. In order to compensate for these problems, [12] presents the following formulas:

$$g_{filt}(t) = \begin{cases} (1-p) \times g_{filt}(t-1) + p \times g_{inst}(t) & \text{for valid measurement and uncongested traffic} \\ g_{filt}(t-1) & \text{otherwise} \end{cases} \tag{2.3}$$

g_{inst} is calculated using the formula for $g(t)$ given above. p represents a "time constant" parameter, which is set to two hours in PeMS.

The one remaining problem is that incorporating p delays g-factor results by its value. [12] solves this problem by using historical g-factors and the g-factor curve's periodic nature that is, its tendency to regularly repeat over the course of weekdays. This allows PeMS to compensate for the delayed g-factor by shifting the g-factor curve forward by two hours. Thus, the final g-factor estimation formula is:

$$g(t) = g_{filt}(t) + [g_{hist}(t+\tau) - g_{hist}(t)] \tag{2.4}$$

g_{hist} is the historic g-factor and τ is the time constant. This g-factor formula can then be used in equation 2.1 to estimate traffic speeds.

After obtaining initial speed estimates, PeMS also attempts to fill in ”holes” in its data through imputation. The system uses a variety of imputation techniques, including linear regression based on neighboring detectors, temporal medians, and cluster medians. For further information on these techniques, refer to [8].

2.3 Bing Maps Routing Algorithm

Until recently, Bing Maps used a modified version of Dijkstra's algorithm for its routing calculations. However, in early 2012, Microsoft announced the implementation of a new routing engine with improved performance [9].

The current Bing Maps routing algorithm is designed to accommodate a variety of route cost metrics, such as travel time, total path distance, and so forth. It is composed of the following three stages:

1. **Metric-independent preprocessing**. In this phase, the road network graph (G) is partitioned into connected cells consisting of no more than U vertices each (where U is an input parameter for the algorithm), such that the number of arcs with endpoints in different cells (boundary arcs) is minimized.

2. **Metric customization**. This phase constructs a new graph H, consisting of all boundary vertices (vertices with at least one neighbor in a different cell) and boundary arcs from G. H also includes a clique for each cell. These cliques are created by adding an arc between each pair of boundary vertices within the same cell, with the arc's cost equal to the shortest path cost between the vertices (or infinity if no path exists).

These path costs are computed using Dijkstra's algorithm. The paper points out that graph H is an overlay graph, and does not alter paths or arc costs from G.

3. **Query**. When a route is requested between vertices s and t, the routing engine runs bidirectional Dijkstra's algorithm on a new graph composed of H, the cell containing s, and the cell containing t.

Stage 1 relies only on static graph topology, including speed limits, physical length of road segments, and so on. Since these features rarely change, stage 1 runs infrequently, and thus can be slow compared to the other stages. Stage 2, on the other hand, must be run once for each route cost metric. Since customization for different cost metrics is a primary design goal for this algorithm, stage 2 must run quickly if the engine is to perform well overall.

[11] discusses a variety of optimizations that were tested during the algorithm's development, including clique reduction (sparsification) and goal-direction, which attempts to avoid unfavorable search directions using additional precomputed information. However, although these optimizations do improve the algorithm's space efficiency and query runtime, the paper indicates that these improvements are only moderate. Furthermore, they sacrifice preprocessing efficiency, increase implementation complexity, and make the algorithm's performance more dependent on the cost metric. The authors therefore opted for a simpler implementation that uses simple cliques and a matrix-based representation. Each cell from the overlay graph is represented as a matrix that stores the distances between its entry and exit vertices. The paper states that although this approach does not yield the fastest query times in every case, it makes metric customization fast and space efficient, while still offering sufficient query performance for real-time use.

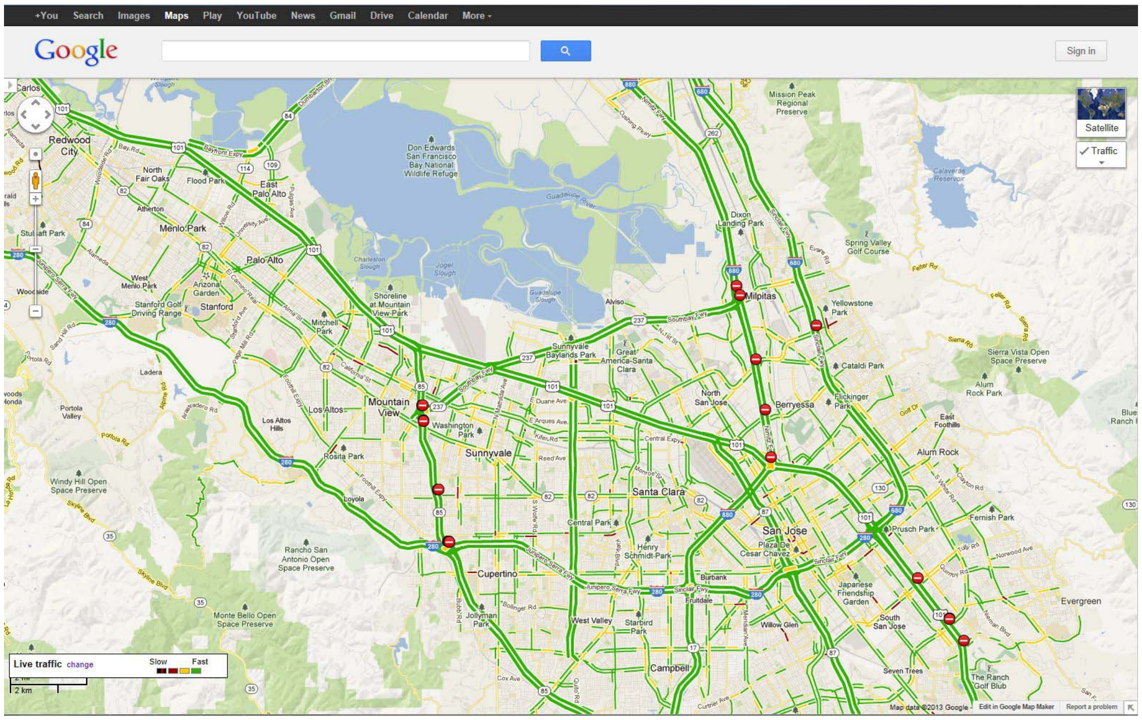


Figure 2.2: Main Google Maps window.

2.4 Related Systems

2.4.1 Google Maps

Google offers a freely-accessible web service for route finding and travel time estimation. Its map view features a color-coded traffic overlay view. When in use, the traffic overlay draws color-coded segments on top of roadways to visually indicate traffic speeds. This traffic data is also used to estimate travel time for driving routes. Currently, the service also displays clickable alerts indicating lane closures and similar activity.

Google Maps obtains traffic speed data from GPS-capable devices while they travel. According to Google, historical traffic data is also involved in their travel time calculations [7]. However, no details are available on the algorithm used to calculate these travel times.

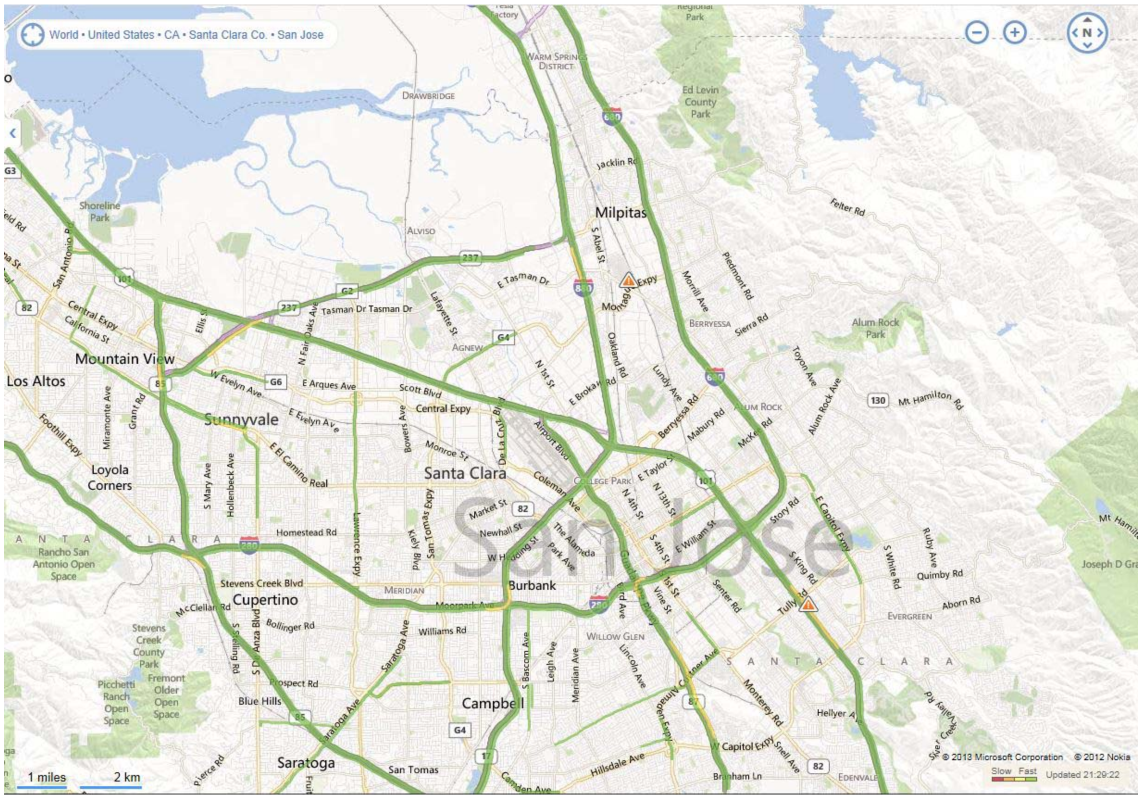


Figure 2.3: Main Bing Maps window.

2.4.2 Bing Maps

Similar to Google Maps, Bing Maps is another free web-based mapping and routing service. Its features appear to be comparable to those of Google Maps in most respects. In particular, it features a color-coded traffic overlay view similar to that of Google Maps, and can use traffic information to estimate travel time to the destination.

2.5 Cassandra

Cassandra is a data storage tool developed by Facebook. The system is designed to be both highly available and highly scalable, supporting continuous growth while dealing with component failures in a massively distributed environment. Cassandra was originally developed to support Facebook's Inbox Search feature, and therefore had to support high write throughput, scale with Facebook's user base, and replicate data across data centers to reduce access latency [13]. Now an Apache software project, Cassandra has been adopted both by other Facebook services and other companies, including Netflix, eBay, Reddit, and more [1].

Cassandra's smallest data unit is the column, which consists of a name, value, and timestamp. Rows are comosed of a row key and a set of columns. Unlike rows in a relational DBMS, rows in Cassandra are not required to conform to any specific structure, and may include varying sets of columns [5].

Chapter 3

Architecture and Design

The high-level architecture for Mantis is given in Figure 3.1.

In the remainder of this section, I will describe Mantis's major components and their interactions, along with the motivation for my design decisions.

3.1 Traffic Data Collector

This component is a student virtual machine (VM) running on a server owned and operated by the Cal Poly CSL staff. It acts primarily as a proxy between the Route Evaluation Engine and the CalTrans traffic data server. It runs a simple Java program that connects to the CalTrans server via FTP and periodically downloads its most recent data files. The files are stored on the VM's hard disk, where they are in turn downloaded by the evaluation engine's Traffic Downloader sub-component.

This indirect download process is necessary because of the CalTrans FTP server's access control policies. The server requires that all requests for data downloads must originate from a set of specific IP addresses associated with the account used to connect to the server. Thus,

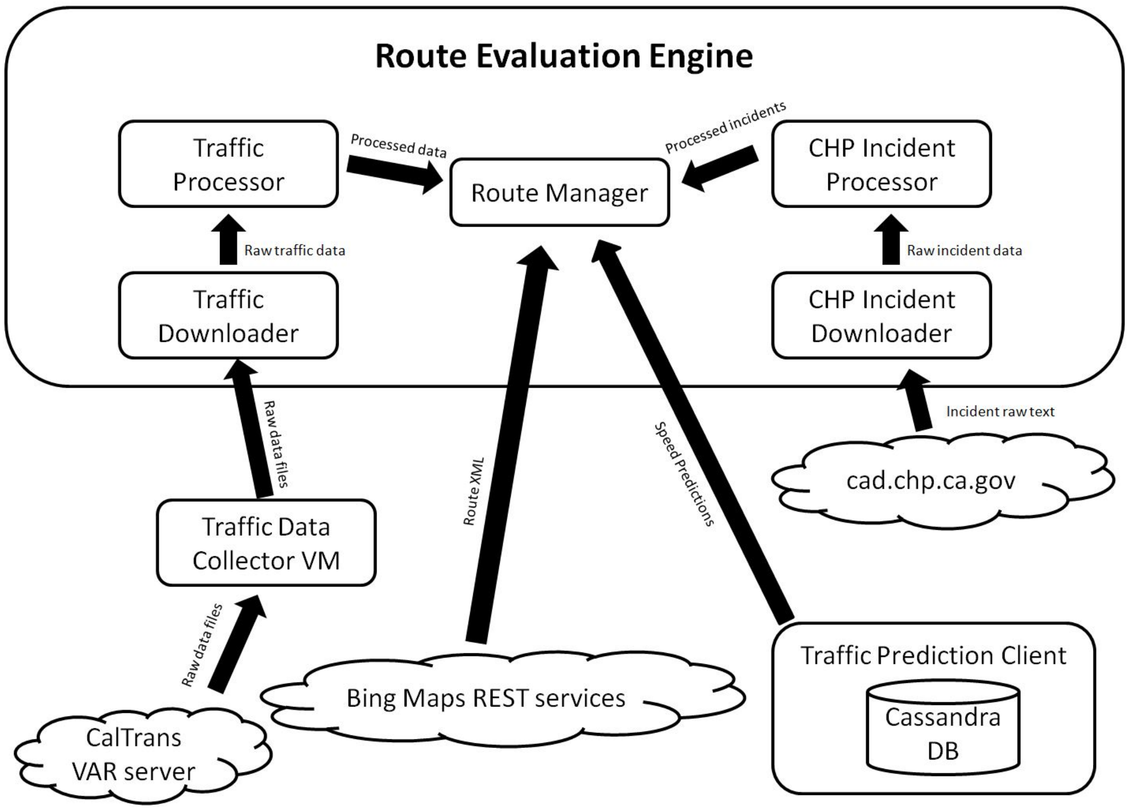


Figure 3.1: Overall System Architecture.

in order to access the CalTrans server, I needed a collection point with a static IP address. My personal computers do not offer such a guarantee, but CSL student VMs are configured with static IP addresses. Although this process has no efficiency or performance benefits, it does allow Mantis to reliably access CalTrans real-time traffic data without regard to the evaluation engine's host system configuration.

3.2 Traffic Prediction Client

This component is responsible for storing archived CalTrans traffic data and using it to predict speeds for particular locations and times. It is composed of a Java client program, which implements the prediction logic, and a single-node Cassandra database, which runs on the same host machine and acts as its data repository. It accepts queries for predictions over a standard TCP socket connection, and responds with its prediction over the same connection.

This component's existence as an independent entity is in part due to its origin as a class project. In order to make it easy for the rest of my project team to access and work on the prediction client, I decided to make the client platform-agnostic by using Java. This allowed the client to run on a Linux VM accessible to the whole team over the Internet. When I began to integrate the component into Mantis after the conclusion of that class project, I felt that it would be best to reuse and extend the existing Java client, rather than rewriting the prediction logic in C# and directly integrating it into the Route Evaluation Engine.

Using a network connection for communication between the prediction client and the evaluation engine also promotes overall system flexibility. In particular, this allows the prediction client to run either on the same host as the evaluation engine or on a remote system.

3.3 Route Evaluation Engine

This component is the foundation of Mantis. As its name suggests, it is responsible for obtaining and processing information from each data source in order to rank all candidate routes to the destination. Unlike the previous secondary components, the evaluation engine is written in C# as a Windows application.

Data collection and processing is handled by a set of downloader and processor component pairs. In each pair, the downloader is responsible for contacting a remote data source and obtaining data from the source in its raw form, while the processor consumes the raw data and converts it into a corresponding in-memory representation. For example, the traffic downloader handles communication with the traffic data collector VM and downloads raw traffic data files to the local hard disk. The traffic processor then reads these files and constructs an object-oriented representation of the data. Likewise, the CHP incident downloader connects to the CHP website and collects raw incident information, while the CHP incident processor creates an OO representation of the data that can be easily consumed by higher-level components. These processing procedures are described further in 4.

The Route Manager component has two primary functions. First, it contacts the Bing Maps REST service and downloads up to three possible routes to the destination in XML format. As part of this task, it also creates a list of Route objects to represent the downloaded routes. The details of this conversion process are described in 4. Second, the Route Manager contains scoring logic that determines how to order the set of Routes in order of desirability. This process is also covered in greater detail in section 4.

Chapter 4

Implementation

This section gives a detailed discussion of Mantis's implementation and low-level operations. The primary aim of this section is to comprehensively describe how Mantis produces its results. For each type of data used in Mantis, this section describes the data's raw format, shows how Mantis converts the data into a useful internal representation. Finally, this section discusses Mantis's route analysis algorithms at length, showing how the processed data is used to predict travel times and to ultimately assess each route's relative quality.

4.1 Traffic Data

4.1.1 VDS Metadata

In addition to real-time measurements from their VDSs, CalTrans also provides a VDS configuration file that provides definitions and metadata for each detector in their system. This data is available as an XML file, and can be downloaded from the PeMS website.

Raw Data

The VDS configuration file is organized as a list of district elements, each corresponding to a California administrative district. Each district element contains a list of all counties within the district, and each county in turn gives a list of all cities within the county. Following its county list, each district lists all VDSs that fall within it. Finally, a district may also contain a list of freeway crossings, each of which is associated with a VDS.

A sample of the VDS configuration file's contents is given below:

Listing 4.1: Sample of VDS configuration file

```
<counties>
  ...
  <county id="85" name="Santa Clara">
    <cities>
      ...
      <city id="68000" name="San Jose"/>
      <city id="69084" name="Santa Clara"/>
      <city id="70280" name="Saratoga"/>
      <city id="77000" name="Sunnyvale"/>
    </cities>
  </county>
  ...
</counties>
<detector_stations>
  <vds id="401180" name="500’ N of Lake St" type="ML" county_id="75"
     city_id="67000" freeway_id="1" freeway_dir="N" lanes="2" cal_pm="
     6.20" abs_pm="443.298" latitude="37.790676" longitude="
     -122.470123" last_modified="08/30/2012"/>
  <vds id="401372" name="NB 001 to SB101&NB101" type="ML" county_id="
     75" city_id="67000" freeway_id="1" freeway_dir="N" lanes="2"
     cal_pm="6.90" abs_pm="443.998" latitude="37.800668" longitude="
```

```
        -122.469422" last_modified="08/30/2012"/>
...
<crossings>
     <crossing county_id="81" description="ANO NUEVA PRESERVE-LT"
        freeway_id="1" freeway_dir="N" cal_pm="1.321" abs_pm="389.643"/>
     . . .
     <crossing county_id="81" description="GAZOS CREEK RD-RT" freeway_id=
        "1" freeway_dir="N" cal_pm="5.782" abs_pm="394.104"/>
. . .
</crossings>
</district>
```

Because Mantis only uses VDS definitions in its calculations, the county, city, and crossing elements are not discussed further, and are included here only for completeness.

Each “vds” element in the configuration file corresponds to a Vehicle Detector Station (VDS), and includes the following attributes:

1. **id**: A unique identifier for the station. Data files use this attribute to associate samples with the stations that reported them.
2. **name**: A description of the station, often referring to its location.
3. **type**: A two-letter code corresponding to a station type defined in the station configuration file. The possible values are as follows:
   - CD: Collector/Distributor
   - FF: Freeway-Freeway Connector
   - FR: Off Ramp
   - HV: HOV
   - ML: Mainline

- OR: On Ramp

4. **county_id**: ID of the county containing the detector.

5. **city_id**: ID of the city containing the detector. If the detector does not fall within a city's limits, this attribute is empty.

6. **freeway_id**: The freeway number the detector monitors - for example, 101, 880, 87.

7. **freeway_dir**: The freeway's direction of travel at the detector (N, S, E, or W). Note that this direction may not necessarily match the compass direction of travel at the detector.

8. **lanes**: The number of freeway lanes at this detector.

9. **cal_pm**: The detector's California postmile location. This quantity is not a distance measure, but is instead assigned to specific locations along a freeway's path as a reference point. California postmile values are never changed after being assigned (even if the freeway's path is modified), and they reset at county lines [4] [3].

10. **abs_pm**: The detector's absolute postmile location. Unlike the California postmile, the absolute postmile is a true distance measure. It is defined as the distance from a freeway's beginning to a location along its path. This is not a straight-line distance, but rather reflects the distance along the freeway's path [4] [3].

11. **latitude**: The detector's estimated latitude coordinate.

12. **longitude**: The detector's estimated longitude coordinate.

13. **last_modified**: Date of this entry's most recent modification.

Processing

Since each station's properties are given directly as attributes of vds elements, very little additional processing is needed to load station definitions from the configuration file. Using XPath expressions, Mantis iterates over the file's list of district elements, and for each one, retrieves its list of vds elements. Each vds element's attributes are then retrieved by name and used to initialize a VDS C# object, which is inserted into a hash table using the VDS's ID as the key.

4.1.2 Real-Time Data

CalTrans offers access to their real-time traffic data through an FTP server accessible to PeMS accounts with Value-Added Reseller (VAR) privileges. This FTP server is updated with the detector network's most recent raw measurements every 30 seconds, while 5-minute aggregate samples are updated every 5 minutes.

As discussed in 3, Mantis uses a virtual machine set up by Cal Poly's CSL staff as an intermediate collection point for this data. Because this VM is behind the CSL network's firewall, it is inaccessible by unsecured protocols such as FTP. Mantis therefore uses a .NET assembly for the WinSCP secure file transfer program to connect to the VM via SCP. The WinSCP assembly allows C# programs to automate WinSCP functions, such as establishing a session and downloading files from remote hosts.

Raw data samples are expressed primarily as Flow and Occupancy measurements. Flow is simply the number of vehicles that traveled past the detector during the sample period, and is typically measured in vehicles per hour. Occupancy indicates the percentage of the sample period that the detector was on, ranging from 0 to 100. According to PeMS documentation, occupancy can be used as an indication of traffic density at a detector [3].

Raw Data

Both 30-second and 5-minute data files are formatted as comma-delimited ASCII text files, with each line containing data for one sample from a single VDS. The 30-second files contain completely raw measurements, while the 5-minute files contain values either averaged over or calculated for the entire sample period.

Each line in the 30-second files contains the following fields, for lanes 1-N:

- **VDS_ID**: Identifier for the VDS that generated this sample.
- **LANE_COUNT**: Deprecated.
- **LOOP_COUNT**: Deprecated.
- **LOOP_N_FLOW**: Measured flow for lane N.
- **LOOP_N_OCCUPANCY**: Measured occupancy for lane N.
- **LOOP_N_STATUS**: Deprecated.

A sample of a 30-second file's contents is given in Listing 4.2.

Listing 4.2: Example of 30-second traffic data

```
402174,,,13,.2794,,12,.17,,7,.09,,,,,,,,,,,,,,,,,,,,,,
402175,,,17,.1,,10,.0694,,6,.09,,,,,,,,,,,,,,,,,,,,,,
402176,,,15,.1,,15,.12,,15,.13,,,,,,,,,,,,,,,,,,,,,,
402177,,,14,.12,,8,.09,,10,.09,,,,,,,,,,,,,,,,,,,,,,
402178,,,9,.32,,8,.22,,8,.1394,,,,,,,,,,,,,,,,,,,,,,
```

The detector configuration file described in the previous section includes lane counts for all VDSs, so for each 30-second sample, Mantis determines the number of lanes (loops) in the sample by looking up the generating VDS's lane count.

The format of each sample in the 5-minute files is as follows:

1. **VDS_ID**: Identifier for the VDS that generated this sample.
2. **Flow**: Total flow for the 5-minute sample period, calculated over all lanes.
3. **Occupancy**: Average occupancy for the 5-minute period over all lanes.
4. **Speed**: Average speed over all lanes.
5. **VMT**: Vehicle Miles Traveled. Represents the total number of miles traveled by all vehicles on a certain section of freeway for some period of time.
6. **VHT**: Vehicle Hours Traveled. Gives the total time spent traveling by all vehicles on a certain section of freeway for some period of time.
7. **Quality**: Calculated as VMT / VHT. Used as a measure of a freeway's performance, with high values indicating good performance and low values indicating poor performance.
8. **Travel_Time**: (not in use)
9. **Delay**: Delay incurred by congested freeway conditions, measured in vehicle-hours.
10. **Num_Samples**: Total number of samples from this detector in the 5-minute period.
11. **Pct_Observed**: Percentage of samples from individual lanes that was included in this sample's aggregate values.

A sample of a 5-minute data file's contents is given in Listing 4.3.

Listing 4.3: Example of 5-minute aggregated traffic data

```
401475,587,.1139,56.7,296.435,5.2,56.7,,.67,50,100
400231,396,.0977,65.3,929.808,14.2,65.3,,0,30,0
401901,748,.0899,55.9,265.54,4.8,55.9,,.67,60,100
401097,513,.1013,61.2,94.905,1.6,61.2,,.09,40,0
402621,207,.2138,17.9,196.65,11,17.9,,7.96,20,100
```

Processing

As with the station configuration data, real-time traffic data requires only simple processing. The first line in any real-time data file gives the timestamp for the file's contents, and each subsequent line is a data sample. Each data sample line is split into an array of strings using ',' as a delimiter. The first step in processing a sample is to extract the associated VDS ID, which is always the first field, and consequently always at position 0 in the array. If no matching VDS definition is found, the sample is discarded. Otherwise, the sample's other fields are also extracted based on their positions in the array.

Since the 30-second and 5-minute data files contain different fields, two different classes are used to represent real-time traffic data samples. 30-second samples are represented by the RawVDSData class. This type contains an internal array of RawLaneSample objects, one for each lane at the detector's location. Each RawLaneSample in turn contains the flow and occupancy values for its lane. Similarly, the AggregateVDSData class, which stores VDS ID, flow, occupancy, speed, VMT, VHT, individual sample count, and timestamp, is used to represent 5-minute data samples.

Each real-time sample's fields are used to initialize a new object of the appropriate data sample type. Finally, each such object is inserted into a hash table with the VDS ID used as the key. Two separate hash tables are maintained, one for each type of real-time data. These tables are stored as static fields of the TrafficDataProcessor class, and are accessible via read-only properties.

4.1.3 Archived Data

In addition to their real-time data FTP server, CalTrans also maintains an online archive of all traffic data collected by their system. This data can be downloaded from the Data Clearinghouse section of the PeMS website. Mantis uses this data to predict traffic speeds at specific locations and times.

Raw Data

Like the real-time data previously discussed, archived traffic data is offered as comma-delimited ASCII text files. Each archived data file gives all samples collected for a single day, with each individual sample given on a separate line.

Archived data is available for download as 30-second, 5-minute, 60-minute, and 1 day sample periods. Mantis uses the 5-minute period format, since this gives good data granularity but also reduces the number of individual samples compared to the 30-second format. Thus, only the 5-minute format is discussed here.

The 5-minute archived data files contain the following fields. Note that fields 13-17 occur once for each individual lane (numbered 1..N) at the VDS. Mantis only uses sample timestamps, VDS IDs, and average speeds in its calculations, but the other fields are also listed for completeness.

1. **Timestamp**: Timestamp for the beginning of the sample's 5-minute period, given in MM/DD/YYYY HH24:MI:SS format.
2. **Station**: ID for the VDS that produced this sample.

3. **District**: District number for the VDS that produced this sample.
4. **Freeway #**: Freeway number for the VDS.
5. **Direction of Travel**: The freeway travel direction for the VDS. May be N, S, E, or W.
6. **Lane Type**: Type code corresponding to a type given in the station configuration file.
7. **Station Length**: Distance in miles or kilometers covered by the VDS.
8. **Samples**: Total number of individual samples received during the 5-minute period.
9. **Percent Observed**: Percent of individual lane samples that were directly observed during the sample period.
10. **Total Flow**: Flow for the 5-minute sample period, summed over all lanes.
11. **Average Occupancy**: Occupancy for the 5-minute period, calculated over all lanes and expressed as a number between 0 and 1.
12. **Average Speed**: Average speed in miles/hr for the sample period.
13. **Lane N Samples**: Good sample count for lane N during the sample period.
14. **Lane N Flow**: Flow for lane N during the sample period.
15. **Lane N Average Occupancy**: Occupancy for lane N during the sample period.
16. **Lane N Average Speed**: Average speed in miles/hr for lane N during the sample period.
17. **Lane N Observed**: Binary field. 1 indicates observed data, while 0 indicates imputed data.

A sample of a 5-minute archived data file is given in Listing 4.4.

Listing 4.4: Example of 5-minute archived traffic data

```
00:00:00,400000,4,101,S,ML,.275,40,0,33,.009,67.6,10,7,.0064,73.2,0,...
09/11/2012
00:00:00,400001,4,101,N,ML,.445,50,100,38,.0069,71,10,3,.0022,76.5,1,...
09/11/2012
00:00:00,400002,4,101,S,ML,.31,50,0,263,.0384,68.9,10,50,.0314,76.2,0,...
09/11/2012
00:00:00,400006,4,880,S,ML,.34,40,100,131,.0239,68.6,10,19,.01,74.8,1,...
09/11/2012
00:00:00,400007,4,101,N,ML,.365,50,100,118,.0166,72.4,10,20,.0116,76.5,...
09/11/2012
00:00:00,400009,4,80,W,ML,.12,0,0,34,.009,68.3,0,8,.0072,72.5,...
09/11/2012
00:00:00,400010,4,101,N,ML,.45,40,100,71,.0128,68,10,6,.0041,74.8,...
09/11/2012
00:00:00,400011,4,101,S,ML,.225,39,40,71,.0148,67.8,0,12,.013,74.1,...
09/11/2012
00:00:00,400014,4,101,N,ML,.29,40,75,52,.0106,63.4,10,3,.0018,74.8,...
```

Processing

Unlike the real-time traffic data, which is updated and parsed dynamically on demand, the archived data is loaded into a Cassandra database, and thus is only processed in its raw form once.

The Cassandra database is structured as follows. Each row key is a VDS ID concatenated with a date (in MM/DD/YYYY format), while column names are full sample timestamps formatted as long integers (representing the number of milliseconds since Jan 1, 1970). Finally, each cell contains the average speed for the VDS ID in its row key at the timestamp

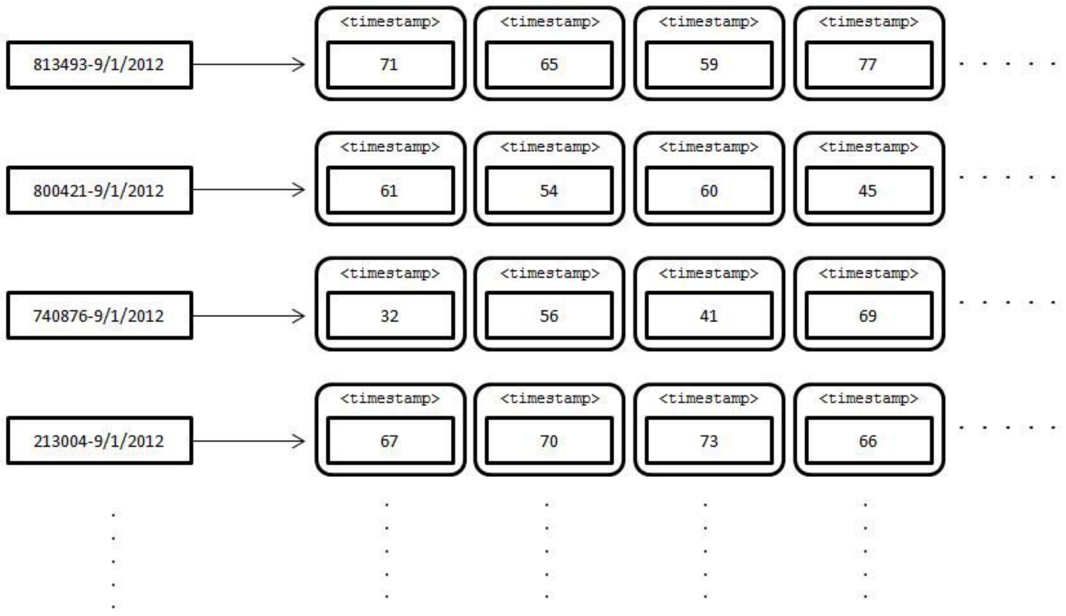


Figure 4.1: Diagram of Cassandra database format.

given by its column. See Figure 4.1 for a depiction of the database's structure.

Each archived data file is read one line at a time, with each line being split into a string array using ',' as a delimiter. The sample's timestamp, VDS ID, and average speed (at indices 0, 1, and 11, respectively) are then extracted for insertion into Cassandra. The VDS ID and timestamp are used to construct the sample's row key as described above, and then the average speed value is inserted using its converted full timestamp as the column name.

4.2 Highway Patrol Incidents

The California Highway Patrol (CHP) maintains a webpage that is automatically updated with all reported incidents every 60 seconds. This page, accessible at http://cad.chp.ca.gov, allows Mantis to actively monitor highway patrol incidents and detect incidents that fall along particular routes.

4.2.1 Raw Data

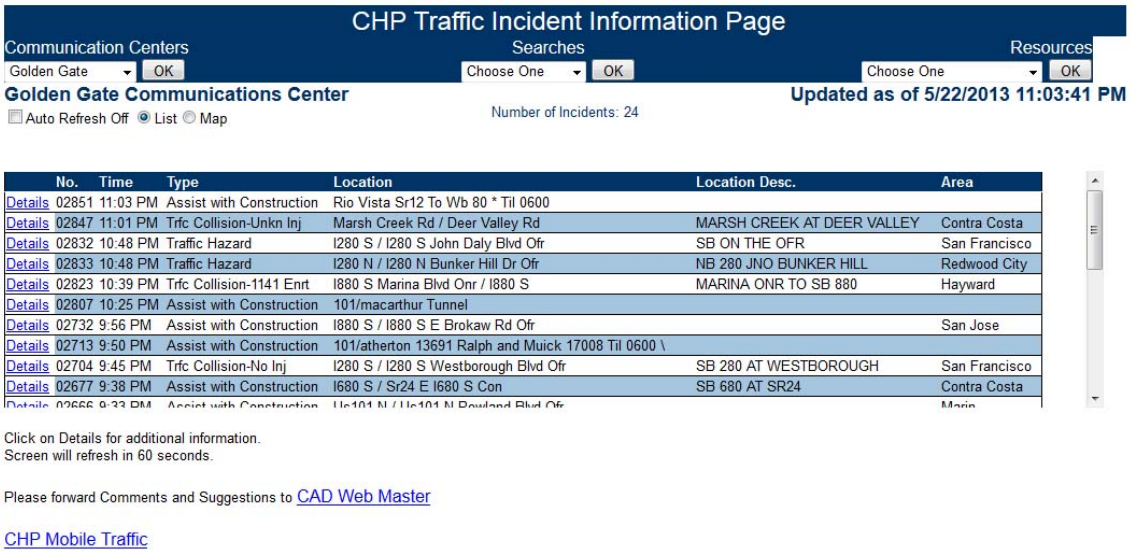
For each California region, the CHP incident information page lists incidents in an HTML table, with the following columns:

1. **No.**: Integer identifier for the incident.
2. **Time**: Timestamp for the incident.
3. **Type**: Shorthand classification of the incident based on its nature. For instance, collisions are assigned a type of "Trfc Collision" (along with a brief description of injury severity).
4. **Location**: Human-readable description of the incident's location.
5. **Location Desc.**: Written description of the incident's location, similar to the previous column.
6. **Area**: General area in which the incident occurred - for example, "San Francisco," "Napa," "San Jose".

Each row in the table also includes a link to a more detailed view of the corresponding incident.

4.2.2 Downloading and Processing

Although the CHP offers incident information via their website, they do not offer this information for direct download. Obtaining the data is further complicated by the fact that the incident information page only displays incidents for one region at a time. In order to view all current incidents, a user must select each region in turn. However, the incident page uses the same URL for all regions, so simply using standard HTTP requests to access the


CHP Traffic Incident Information Page

Communication Centers: Golden Gate — OK | Searches: Choose One — OK | Resources: Choose One — OK

Golden Gate Communications Center

Auto Refresh Off [✓ List] Map

Number of Incidents: 24

Updated as of 5/22/2013 11:03:41 PM

	No.	Time	Type	Location	Location Desc.	Area
Details	02851	11:03 PM	Assist with Construction	Rio Vista Sr12 To Wb 80 * Til 0600		
Details	02847	11:01 PM	Trfc Collision-Unkn Inj	Marsh Creek Rd / Deer Valley Rd	MARSH CREEK AT DEER VALLEY	Contra Costa
Details	02832	10:48 PM	Traffic Hazard	I280 S / I280 S John Daly Blvd Ofr	SB ON THE OFR	San Francisco
Details	02833	10:48 PM	Traffic Hazard	I280 N / I280 N Bunker Hill Dr Ofr	NB 280 JNO BUNKER HILL	Redwood City
Details	02823	10:39 PM	Trfc Collision-1141 Enrt	I880 S Marina Blvd Onr / I880 S	MARINA ONR TO SB 880	Hayward
Details	02807	10:25 PM	Assist with Construction	101/macarthur Tunnel		
Details	02732	9:56 PM	Assist with Construction	I880 S / I880 S E Brokaw Rd Ofr		San Jose
Details	02713	9:50 PM	Assist with Construction	101/atherton 13691 Ralph and Muick 17008 Til 0600 \		
Details	02704	9:45 PM	Trfc Collision-No Inj	I280 S / I280 S Westborough Blvd Ofr	SB 280 AT WESTBOROUGH	San Francisco
Details	02677	9:38 PM	Assist with Construction	I680 S / Sr24 E I680 S Con	SB 680 AT SR24	Contra Costa
Details	02666	9:33 PM	Assist with Construction	Us101 N / Us101 N Rowland Blvd Ofr		Marin

Click on Details for additional information.
Screen will refresh in 60 seconds.

Please forward Comments and Suggestions to CAD Web Master

CHP Mobile Traffic

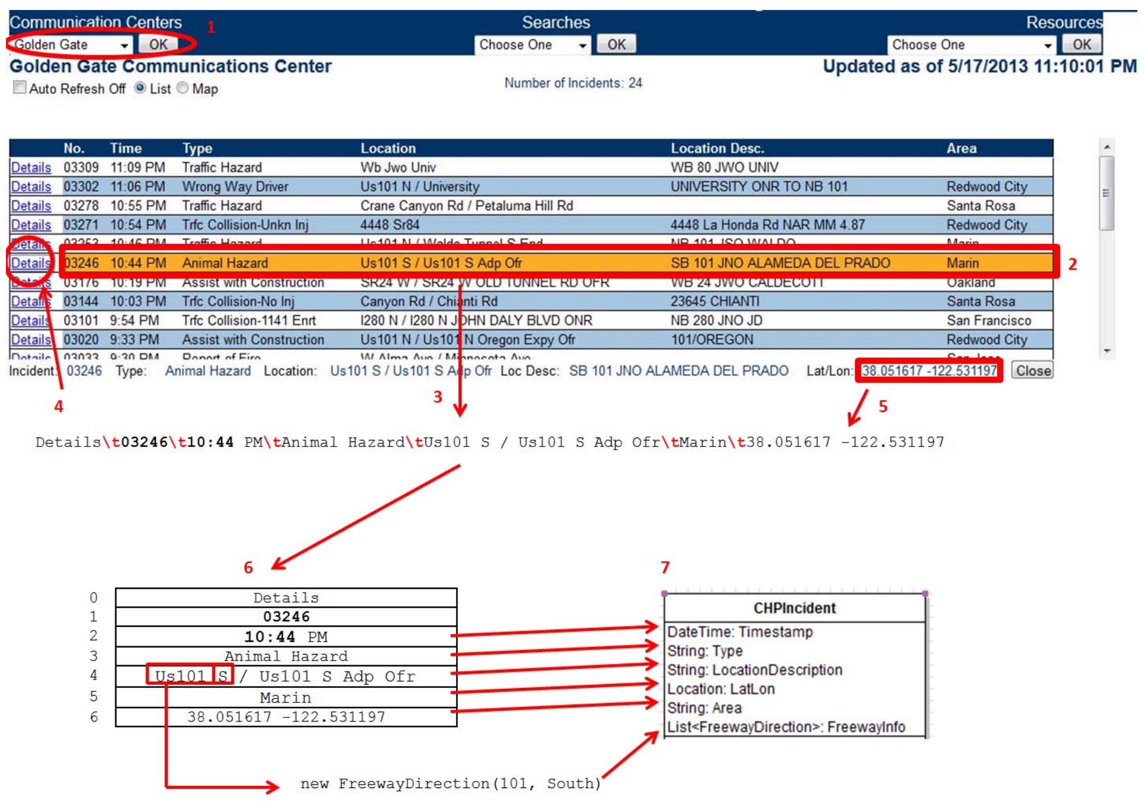
Figure 4.2: The CHP Incident Information webpage (http://cad.chp.ca.gov).

incident page is not sufficient to collect all current incidents. Thus, Mantis uses an automated web browser instance to access the incident information page. In particular, Mantis relies on the Selenium browser automation library to launch an automated instance of Firefox.

The procedure for downloading and parsing incident data is visualized in Figure 4.3.

After directing the automated Firefox instance to the incident information page, Mantis extracts incident data for the first region directly from the HTML table. After this data has been collected, Mantis uses Selenium to access the webpage's interactive controls and select the next region. This process is repeated for each remaining region. The incident downloader component ultimately produces a list of incidents represented as strings, with individual fields separated by tabs.

As part of this process, Mantis must simulate a click on the link in each row's first cell. This displays a details pane below the main incident table, which gives latitude and longitude coordinates for the incident. This information is extracted separately, and added to the string representation of its corresponding row.



Golden Gate Communications Center

Updated as of 5/17/2013 11:10:01 PM

Auto Refresh Off [✓ List] Map

Number of Incidents: 24

	No.	Time	Type	Location	Location Desc.	Area
Details	03309	11:09 PM	Traffic Hazard	Wb Jwo Univ	WB 80 JWO UNIV	
Details	03302	11:06 PM	Wrong Way Driver	Us101 N / University	UNIVERSITY ONR TO NB 101	Redwood City
Details	03278	10:55 PM	Traffic Hazard	Crane Canyon Rd / Petaluma Hill Rd		Santa Rosa
Details	03271	10:54 PM	Trfc Collision-Unkn Inj	4448 Sr84	4448 La Honda Rd NAR MM 4.87	Redwood City
Details	03253	10:46 PM	Traffic Hazard	Us101 N / Waldo Tunnel S End	NB 101 JSO WALDO	Marin
Details	03246	10:44 PM	Animal Hazard	Us101 S / Us101 S Adp Ofr	SB 101 JNO ALAMEDA DEL PRADO	Marin
Details	03176	10:19 PM	Assist with Construction	SR24 W / SR24 W OLD TUNNEL RD OFR	WB 24 JWO CALDECOTT	Oakland
Details	03144	10:03 PM	Trfc Collision-No Inj	Canyon Rd / Chianti Rd	23645 CHIANTI	Santa Rosa
Details	03101	9:54 PM	Trfc Collision-1141 Enrt	I280 N / I280 N JOHN DALY BLVD ONR	NB 280 JNO JD	San Francisco
Details	03020	9:33 PM	Assist with Construction	Us101 N / Us101 N Oregon Expy Ofr	101/OREGON	Redwood City
Details	03033	9:30 PM	Report of Fire	W Alma Ave / Minnesota Ave		San Jose

Incident: 03246 Type: Animal Hazard Location: Us101 S / Us101 S Adp Ofr Loc Desc: SB 101 JNO ALAMEDA DEL PRADO Lat/Lon: 38.051617 -122.531197 Close

```
Details\t03246\t10:44 PM\tAnimal Hazard\tUs101 S / Us101 S Adp Ofr\tMarin\t38.051617 -122.531197
```

0	Details
1	03246
2	10:44 PM
3	Animal Hazard
4	Us101 S / Us101 S Adp Ofr
5	Marin
6	38.051617 -122.531197

```
CHPIncident
DateTime: Timestamp
String: Type
String: LocationDescription
Location: LatLon
String: Area
List<FreewayDirection>: FreewayInfo
```

```
new FreewayDirection(101, South)
```

Figure 4.3: Getting CHP incident data.

Once the raw incident data is obtained, the incident processor component splits each raw incident into a string array using the tab character as a delimiter. The incident's timestamp, type, location description, latitude/longitude, and area are then extracted based on their positions in the array and used to initialize a new CHPIncident object's corresponding properties. In addition, the incident processor searches the location description text for freeway numbers and directions using regular expressions. Any sequence of digits following text such as "I", "US", and so forth is treated as a freeway number. Similarly, any compass direction, either in full ("East") or abbreviated ("E") form, is treated as the corresponding freeway direction. Each such pairing is encapsulated in a FreewayDirection object and added to a list stored in the CHPIncident object. Finally, all such CHPIncident objects are added to a list maintained as a static field of the incident processor.

4.3 Routes

4.3.1 Raw Data

Mantis obtains raw route responses from the Bing Maps REST service in XML format. Each possible route is given in a separate XML file. The raw route XML contains a wealth of information, but Mantis primarily relies on the following elements:

- **Bounding Box**: Gives latitude and longitude coordinates that define a bounding box containing the route's entire path.
- **Itinerary Item**: Given in a list. Itinerary items represent a single step in following the route. Each itinerary item element contains the following important sub-elements:
  - **Travel distance**: Distance covered by this step.
  - **Travel duration**: Bing Maps' estimated time to complete this step.

  - **Maneuver point**: Latitude/longitude coordinates marking the beginning of this step.
  - **Instruction**: Human-readable instructions for following this step.
  - **Detail**: Gives further information about a route step. An itinerary may contain zero to multiple Detail elements. Each one can in turn contain a road or freeway name and type, along with start and end path point indices (see below).

- **Route Path**: Gives information about the route's actual path. This element contains a Line sub-element, which in turn contains a list of Point sub-elements. Each Point gives latitude and longitude coordinates for a single point along the route's path. When connected by line segments, these points trace out the route's path as a sequence of latitude/longitude coordinates. Path point indices in other elements refer to positions in this list of Point elements.

An example itinerary item is given in Listing 4.5 [6].

Listing 4.5: Example itinerary item XML

```
<ItineraryItem>
  <TravelMode>Driving</TravelMode>
  <TravelDistance>1.843</TravelDistance>
  <TravelDuration>251</TravelDuration>
  <ManeuverPoint>
        <Latitude>44.979942</Latitude>
        <Longitude>-93.264141</Longitude>
  </ManeuverPoint>
  <Instruction maneuverType="TurnRight">Turn right onto MN-65 South / S
     Washington Ave</Instruction>
  <CompassDirection>southeast</CompassDirection>
  <Detail>
        <ManeuverType>TurnRight</ManeuverType>
        <StartPathIndex>2</StartPathIndex>
        <EndPathIndex>11</EndPathIndex>
        <Name>S Washington Ave</Name>
        <CompassDegrees>113</CompassDegrees>
        <Mode>Driving</Mode>
        <PreviousEntityId>0</PreviousEntityId>
        <NextEntityId>0</NextEntityId>
        <RoadType>Arterial</RoadType>
        <RoadShieldRequestParameters>
          <Bucket>50428</Bucket>
          <Shield>
                <RoadShieldType>3</RoadShieldType>
                <Label>65</Label>
          </Shield>
        </RoadShieldRequestParameters>
        <LocationCode>118-06272</LocationCode>
        <LocationCode>118N06272</LocationCode>
        <LocationCode>118-05155</LocationCode>
        <LocationCode>118-05154</LocationCode>
        <LocationCode>118N05154</LocationCode>
        <LocationCode>118-05153</LocationCode>
  </Detail>
  <Exit/>
  <TollZone/>
  <TransitTerminus/>
  <IconType>Auto</IconType>
  <Time>0001-01-01T00:00:00</Time>
  <TransitStopId>0</TransitStopId>
  <SideOfStreet>Unknown</SideOfStreet>
</ItineraryItem>
```

4.3.2 Processing

Mantis uses C#'s built-in XPath support to query route XML documents and access elements by name. For elements that directly contain values of interest, the Route Manager simply queries those elements and accesses their InnerText property to obtain the needed information. Otherwise, the Route Manager retrieves a list of all child elements with a certain name and iteratively processes them.

Itinerary Items

A route is primarily represented in Mantis by a list of ItineraryItem objects. The ItineraryItem class stores the corresponding route step's maneuver location, instruction text, road name and type, start and end path point indices, distance and duration, and freeway number. Most of this information is directly available from the corresponding ItineraryItem element in the raw route XML. However, Bing Maps does not provide freeway information directly for itinerary items, so additional processing is needed to determine which itinerary items fall on freeways.

Mantis uses regular expressions to extract freeway numbers from itinerary items in XML format. For each itinerary item, Mantis first retrieves the item's Detail sub-elements and applies this regular expression to their Name child elements:

```
U[sS].?([0-9]+)|I.?([0-9]+)|S[rR].?([0-9]+)|CA.?([0-9]+)
```

The expression will match any sequence of numbers preceded by "US", "I", "SR", or "CA", which are all freeway prefixes. If the expression does not find a match in this step, Mantis applies the same regular expression to the itinerary item's instruction text. When the expression finds a match, the numeric portion of the matching string is extracted and used as the itinerary item's freeway number. If both attempts to find a match fail, the itinerary item is assigned a freeway number of -1, indicating no freeway information could be found. In order to be identified as involving a freeway, an itinerary item must have a valid freeway number.

Path Points

In addition to a list of ItineraryItem objects, a route in Mantis also includes an array of PathPoint objects, each of which corresponds to a Point element from the raw route XML. A PathPoint's primary function is to store a latitude/longitude coordinate pair. Each PathPoint also stores the freeway number and travel direction (if any) for that point on the route, along with a flag indicating whether or not the path point is on a freeway. While the latitude and longitude coordinates can be directly extracted from the route XML, determining a PathPoint's freeway number and direction requires additional processing.

Before parsing the list of path points itself, Mantis identifies all path point indices at which the freeway direction of travel may change, along with the associated travel direction at that point. This information is represented as a hash table that maps a path point index to the freeway direction at that point. To construct this hash table, Mantis examines each itinerary item in the order they must be followed. If the itinerary item's OnFreeway property returns true, its start path index (that is, the index of the first path point considered part of the itinerary item) and its freeway travel direction are inserted into the hash table. The freeway direction is determined by searching for compass directions (e.g. "South", "N") in the itinerary item's instruction text.

After the table has been constructed, Mantis moves on to processing the path points themselves. For each path point, if its index is a valid key in the hash table, Mantis retrieves the direction associated with the index and assigns it to the PathPoint being created. This direction is used for all subsequent path points as well, until the next direction change is detected. The freeway number for the path point is determined by searching for the last itinerary item with a start path index less than or equal to the point's index, and retrieving the item's freeway number. Finally, the PathPoint's own OnFreeway property is set to match the itinerary item's OnFreeway property.

Identifying Stations Along the Route

The final step in constructing a route representation from raw data is finding all VDSs along the route. Mantis accomplishes this by iterating through the route's list of itinerary items and, for each consecutive pair of path points in the current itinerary item, searching for VDSs that fall approximately between the two points based on their latitude and longitude coordinates. To do this, the distance from the VDS being considered to the line between the path points is calculated. If the distance is below a certain threshold, the VDS is treated as approximately on the line between the points. In addition, Mantis checks to make sure a candidate VDS's freeway number and direction match the itinerary item's freeway number and direction. For the complete algorithm, refer to Listing 4.6.

Listing 4.6: Finding VDSs for a route.

```
GetStationsForItineraryItems()
{
  foundFirst = false

  for i = 0..itineraryItems.length
  item = itineraryItems[i]
  if item.OnFreeway
    endIndex = (i < itineraryItems.length - 1) ?
    itineraryItems[i+1].StartPathIndex :
    item.EndPathIndex
    currentDir = item.Direction
    for pathIndex = item.StartPathIndex..endIndex
    P = pathPoints[pathIndex]
    Next = pathPoints[pathIndex+1]
    inManeuver = IsPathPointWithinItineraryItemManeuver(pathIndex+1)
    stations = GetStationsBetweenPathPoints(P, Next, currentDir, item.
        Freeway,
    inManeuver)
    for each s in stations
      neighborPoints.add(s.Id -> { pathIndex, pathIndex+1 })
      if !foundFirst
      foundFirst = true
      pointBeforeFirst = pathIndex
      pointAfterLast = pathIndex+1
      foundStations.add(s.Id -> s)
    item.StationInfo = { pointBeforeFirst, pointAfterLast, foundStations.
        Values }
    foundStations.clear()
}

GetStationsBetweenPathPoints(P1, P2, dir, fwy, inManeuver)
```

```
{
for each s in allStations
  if (dir == s.Direction and fwy == s.Freeway)
    if (IsStationBetweenPoints(P1, P2, s, inManeuver))
      stations.add(s)

return stations
}

IsPathPointWithinItineraryItemManeuver(i)
{
  for each item in itineraryItems
    if item.StartPathIndex <= i <= item.EndPathIndex
      return true
  return false
}

IsStationBetweenPoints(P1, P2, s, inManeuver)
{
  valid = false
  U = ((s.latitude - P1.latitude) * (P2.latitude - P1.latitude))
     + ((s.longitude - p1.longitude) * (P2.longitude - P1.longitude))
  Udenom = (P2.latitude - P1.latitude)^2 + (P2.longitude - P1.longitude)^2
  U = U / Udenom
  projectedPoint.latitude = P1.latitude + (U * (P2.latitude - P1.latitude)
     )
  projectedPoint.longitude = P1.longitude + (U * (P2.longitude - P1.
     longitude))
  minX = min(P1.latitude, P2.latitude)
  maxX = max(P1.latitude, P2.latitude)
  minY = min(P1.longitude, P2.longitude)
  maxY = max(P1.longitude, P2.longitude)
  valid = projectedPoint.latitude >= minX
    and projectedPoint.latitude <= maxX
    and projectedPoint.longitude >= minY
    and projectedPoint.longitude <= maxY
  if valid == true
    maxDistance = inManeuver ? 0.015 : 0.5
    if Distance(projectedPoint, s) <= maxDistance
      return true
    else
      return false
  else
    return false
}
```

For each VDS discovered during this process, Mantis stores the indices of the points between which it was discovered in a hash table, neighborPoints, keyed by VDS ID. This hash table is stored as a property of the Route object being constructed. In addition, for each itinerary item in the route, Mantis creates an instance of the ItineraryItemStationInfo class and stores it in the StationInfo property of the associated ItineraryItem object. The ItineraryItemStationInfo's primary purpose is to maintain a list of all VDSs discovered for its ItineraryItem. It also stores the index of the path point immediately before the ItineraryItem's first VDS, along with the index of the path point immediately after the ItineraryItem's last VDS. This additional information is used later during travel time estimation.

As can be seen in Listing 4.6, the procedure GetStationsBetweenPathPoints is used to find all VDSs that fall between two consecutive path points. The first test verifies that a candidate VDS's freeway number and direction match those of the itinerary item being processed. Any VDSs that pass this test must also lie approximately on the line between

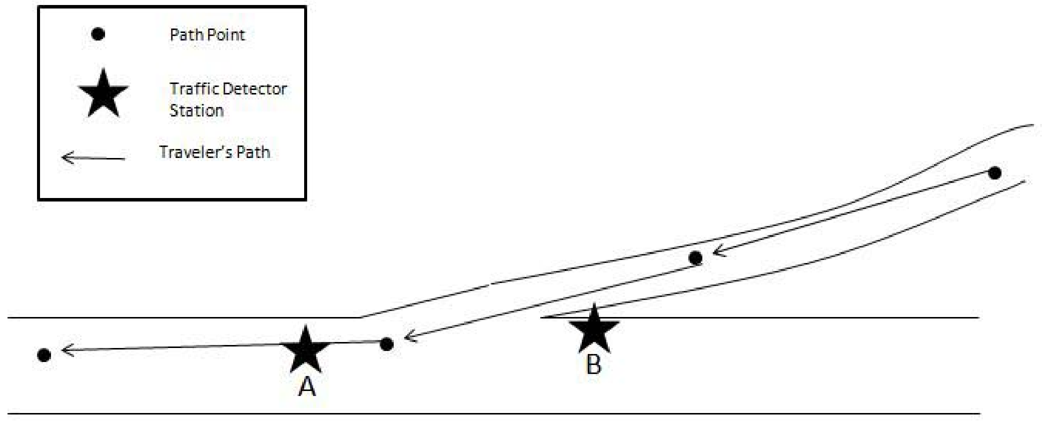


Figure 4.4: VDSs near on-ramps.

the two path points. Any VDS that satisfies all these conditions is added to the list of VDSs to be returned.

Note that the list of VDSs returned from the function GetStationsBetweenPathPoints is sorted by distance from P1 (the first path point in the pair) in ascending order. Thus, the VDS closest to P1 will be first in the returned list, and the VDS most distant from P1 will be last. This step ensures that each itinerary item's VDS list will be sorted in the order that a traveler would encounter the VDSs.

It is important to note that IsStationBetweenPoints uses a lower distance threshold for path points that fall within an itinerary item's initial maneuver. For example, for an itinerary item that calls for traveling on a freeway, the item's maneuver corresponds to merging onto the freeway using an on-ramp. On-ramps almost always are very close to the freeway itself. Thus, if the test used to identify VDSs approximately along the line between two points is not strict enough, VDSs on the freeway before the merge point may be incorrectly included.

Figure 4.4 gives an example of this situation. VDS A is after the on-ramp's merge point, and should therefore be included in the itinerary item's list of VDSs, since a traveler merging onto the freeway will pass it. However, VDS B is before the merge point, so the traveler will not directly pass it. Thus, VDS B should be excluded.

4.4 Estimating Travel Time

This section describes the algorithm Mantis uses to estimate the time required to complete a route. At a high level, the algorithm iterates through the route's list of itinerary items in order from the route's beginning to its end, computing the travel time for each one and accumulating a total travel time.

4.4.1 Overview

Every itinerary item can be decomposed into three segments:

1. From the step's beginning to the first VDS. This segment typically corresponds to freeway entry, such as an on-ramp.
2. From the first VDS to the last VDS.
3. From the last VDS to the beginning of the next step. Similar to 1, this segment typically corresponds to exiting the freeway via an off-ramp.

In addition, itinerary items can be partitioned into two general classes: those containing at least one VDS, and those containing no VDSs. The former category is processed in terms of the above three segments. Travel time for such itinerary items is calculated as the sum of the segments' individual travel times. Since itinerary items with no VDSs have no real-time traffic information available, Mantis simply uses Bing Maps' estimated travel time for these itinerary items.

Listing 4.7: **Estimating a route's travel time.**

```
EstimateTravelTime(useRealTime, departTime)
{
  currentTime = departTime
  for item in itineraryItems
    stationCount = item.StationInfo.StationCount
    if !item.OnFreeway OR stationCount == 0
      travelTime += item.TravelDuration
      currentTime = currentTime.AddSeconds(item.TravelDuration)
    else
      t1 = TimeToPointBeforeFirstStation(item, item.StationInfo)
      travelTime += t1
      currentTime = currentTime.AddSeconds(t1)
      t2 = TimeFromPointToNextStation(item)
      travelTime += t2
      currentTime = currentTime.AddSeconds(t2)
      for i = 0..stationCount-2
        s1 = item.StationInfo[i]
        s2 = item.StationInfo[i+1]
        t3 = TimeBetweenStations(s1, s2, useRealTime, currentTime)
        travelTime += t3
        currentTime = currentTime.AddSeconds(t3)
      t4 = TimeFromStationToNextPoint(item.StationInfo[stationCount-1],
         item.StationInfo)
      travelTime += t4
      currentTime = currentTime.AddSeconds(t4)
      t5 = TimeToNextItineraryItem(item)
      travelTime += t5
      currentTime = currentTime.AddSeconds(t5)
  return travelTime
}
```

4.4.2 Step 1: Travel Time to the First VDS

To estimate travel time from an itinerary item's beginning to its first VDS, Mantis iteratively processes consecutive pair of path points. Beginning from the item's first path point, the algorithm computes the travel time to the next point by calculating the distance between them using the Haversine formula [2] and assuming a speed of 60 mph. This process is repeated for each consecutive point pair until the point immediately before the itinerary item's first VDS. Mantis then uses the same procedure to find the travel time from the last point to the first VDS. The sum of all such travel times gives the total travel time for this portion of the itinerary item.

Listing 4.8: Calculating travel time to the first VDS.

```
TimeToPointBeforeFirstStation(item, itemStationInfo)
{
  dist = 0
  for i = item.StartPathIndex..itemStationInfo.
      PointIndexBeforeFirstStation-2
    P1 = pathPoints[i]
    P2 = pathPoints[i+1]
    dist += Distance(P1.Location, P2.Location)
  return (dist / 60) * 3600) // convert to seconds
}

TimeFromPointToNextStation(item)
{
  dist = Distance(pathPoints[item.StationInfo.PointIndexBeforeFirstStation
      ],
  item.StationInfo[0])
  return (dist / 60) * 3600 // convert to seconds
}
```

4.4.3 Step 2: Travel Time Between the First and Last VDS

The next step is to estimate travel time for the itinerary item's freeway component, where VDSs provide speed information. As with the previous step, Mantis computes the travel time for this segment as the sum of travel times between consecutive pairs of VDSs. Functions DistanceBetweenStations and GetSpeedForStations in Listing 4.9 are discussed in the following subsections.

Listing 4.9: Travel time between two VDSs.

```
TimeBetweenStations(s1, s2, useRealTime, currentTime)
{
  dist = DistanceBetweenStations(s1, s2)
  speed = GetSpeedForStation(s1, useRealTime, currentTime)
  return (dist / stationSpeed) * 3600 // convert to seconds
}
```

Calculating Distance Between VDSs

Computing the distance between consecutive VDSs is complicated by the fact that VDSs are not guaranteed to be separated only by straight sections of freeway. Indeed, it is common for a freeway to curve somewhat between VDSs. The straight-line distance between consecutive VDSs is thus likely to be shorter than the actual distance between them along the freeway's path. In order to accurately compute distances between VDSs, Mantis uses the difference between their Absolute Postmile locations as the distance between them. Since absolute postmile is defined as the distance between a point and the freeway's start point following the freeway's path [4], this gives the actual freeway distance between two consecutive VDSs.

However, although this technique works for most situations, I have observed some cases in which the difference between two VDSs' absolute postmiles is less than the straight-line distance between them. This anomaly is most likely due to errors in the VDS configuration file. Thus, when computing distance between two VDSs, Mantis first compares these two distances. If the straight-line distance is greater, Mantis estimates the freeway distance to the second VDS by using the path points between them, following the procedure described in Step 1.

Listing 4.10: Finding distance between VDSs.

```
DistanceBetweenStations(s1, s2)
{
  absPmDist = |s2.AbsPm - s1.AbsPm|
  latLonDist = Distance(s1, s2)
  if (latLonDist - absPmDist) > 0.1
    s1Neighbors = neighbors.get(s1.Id)
    s2Neighbors = neighbors.get(s2.Id)
    dist = Distance(s1, pathPoints[s1Neighbors.NextPoint])
    for i = s1Neighbors.NextPoint..s2Neighbors.PrevPoint
      dist += Distance(pathPoints[i], pathPoints[i+1]
    dist += Distance(pathPoints[s2Neighbors.PrevPoint], s2)
  else
    dist = absPmDist
  return dist
}
```

Getting Speeds for VDSs

When computing travel time between consecutive VDS pairs, Mantis uses the speed at the first VDS for the section of freeway between the VDSs. If speed prediction is disabled, this speed is obtained from the latest 5-minute real-time data sample for the VDS. Otherwise, the algorithm contacts the speed prediction client program to obtain a speed prediction for the VDS.

Speed predictions in Mantis are based on both location (in the form of a VDS ID) and expected arrival time to the location. To that end, Mantis maintains a predicted arrival time for the current point on the route. As the algorithm processes a route, it accumulates travel time for the portion already processed. This allows Mantis to calculate an expected arrival time to intermediate locations based on the user's predicted departure time. In particular, since the algorithm separately calculates travel time between individual VDSs, Mantis can calculate an arrival time for each VDS as it is visited by simply adding the total accumulated travel time to the departure time. This allows the route evaluation engine to obtain a speed prediction for any VDS. The prediction program that calculates these predictions is described in the following section.

Listing 4.11: Getting speed for a VDS.

```
GetSpeedForStation(s1, useRealTime, currentTime)
{
  sData = GetAggregateStationData(s1)
  if !useRealTime
    speed = GetPredictedSpeedForStationAndTime(s1, currentTime)
  else if  sData is null OR sData.Speed == VALUE_MISSING
    speed = 60
  else
    speed = sData.Speed

  return speed
}
```

The function GetPredictedSpeedForStationAndTime in Listing 4.11 simply sends a message to the prediction program requesting a prediction for the indicated VDS and time.

Predicting Traffic Speeds

As discussed in 3, Mantis uses an independent speed prediction program. The route evaluation engine and the predictor interact using a client-server pattern. The predictor listens for network connections on a standard TCP socket. To obtain a speed prediction, the route evaluation engine connects to the predictor and sends a simple message in this format:

```
<VDS-ID> <MM/DD/YYYY> <H24:MM:SS>
```

Each such message is a request for a speed at the indicated VDS and time. The predictor replies with a single floating-point number, which is the predicted speed in miles per hour.

The prediction program bases its predictions on historical data for the same day of the week as the provided arrival time. For instance, to predict speed for a Friday, the prediction program will only consider speeds from the database that were measured on Fridays. This decision was made to better account for peak hour traffic patterns. Since freeway traffic is typically much lighter on weekends than on weekdays, it is important to only use data from weekends to predict speeds for weekends, and likewise for weekdays.

In addition, the prediction program only considers past speeds from approximately the same time of day as the expected arrival time. This design decision was also motivated largely by peak hour traffic patterns. On weekdays, freeway traffic typically moves much slower in the morning and evening, and significantly faster in early afternoon, early morning, and late evening. Thus, it is important to use speeds from the same time of day as the arrival time, rather than indiscriminately using speeds from any time of day. Currently, the prediction program considers all speed measurements from within 15 minutes of the expected arrival time.

The pseudocode for the speed prediction algorithm is given in Listing 4.12.

Listing 4.12: Speed prediction algorithm.

```
PredictSpeed(sId, arrivalTime)
{
  dates = GetAllPreviousDates(arrivalTime)
  allSamples = new Map<Long, Double>
  for each d in dates
    key = ConstructKey(sId, d)
    rangeStart = GetRangeStartForDate(d)
    rangeEnd = GetRangeEndForDate(d)
    allSamples.put(sliceQuery(rangeStart, rangeEnd, key)

  return ExponentialSmoothing(allSamples, 0.5)
}

GetAllPreviousDates(time)
{
  current = time
  while current > FIRST_DATE_IN_DB
    dates.add(current)
    current.day -= 7

  return dates
}

ConstructKey(sid, d)
{
  return sid + ":" + d.month + "/" + d.day + "/" + d.year
}

GetRangeStartForDate(d)
{
  d2 = d
  d2.minute -= 15
  return d2
}

GetRangeEndForDate(d)
{
  d2 = d
  d2.minute += 15
  return d2
}

SliceQuery(rangeBegin, rangeEnd, key)
{
  q.key = key
  q.range = rangeBegin..rangeEnd
  resultSet = q.execute()
  for each r in resultSet
    resultsMap.put(r.column -> r.value)

  return resultsMap
}
```

Before calculating a prediction, the predictor must first determine which dates will be used in the prediction. As discussed above, the predictor only uses data from the same day of the week as the requested timestamp. Specifically, the predictor constructs a list of all dates preceding the provided timestamp in 1-week increments, halting when it reaches the earliest date for which data is available.

For each date in this list, the predictor queries the Cassandra database for speeds from the specified VDS. In order to retrieve all speeds from within 15 minutes of the specified arrival timestamp, the predictor uses slice queries, which fetch all values with a specified row key and a column name falling in a specified range. The combined results of all individual queries are then used to calculate a prediction for the indicated VDS and time with exponential smoothing. The formula for this calculation is given below [10]:

$$S(t) = \alpha \times A_{t-1} + (1 - \alpha) \times S_{t-1} \tag{4.1}$$

This formula operates on a sequence of values A_t, A_{t-1}, $\ldots$, A_0, where A_t is the most recent speed measurement retrieved from the previous query and A_0 is the oldest. α is a smoothing factor, which can be any number between 0 and 1. In Mantis, it is set to 0.5. Beginning by setting S_0 equal to A_0, the predictor uses the formula to iteratively compute a sequence of exponentially smoothed values. The last of these values is uses as the predicted speed.

4.4.4 Step 3: Travel Time From Last VDS to Last Path Point

After calculating the travel time to the itinerary item's last VDS, Mantis uses a procedure similar to that described in Step 1 to estimate the remaining travel time for the item. The function TimeFromStationToNextPoint is first used to find the travel time from the item's last VDS to the next path point. TimeToNextItineraryItem then accumulates travel time between each following consecutive pair of path points, ending with the itinerary item's last path point. If the current item is the last item in the route, this end point is the path point at the position indicated by the item's EndPathIndex property. Otherwise, the end point is the first path point in the next itinerary item.

Listing 4.13: Travel time from the last VDS to the last point.

```
TimeFromStationToNextPoint(s, item)
{
  lastStationSpeed = GetSpeedForStation(s, useRealTime, currentTime) /
     3600 // convert to miles/second
  Q = pathPoints[item.StationInfo.PointIndexAfterLastStation]
  return Distance(s, Q) / lastStationSpeed
}

TimeToNextItineraryItem(item)
{
  dist = 0
  lastIndex = (item.Index < itineraryItems.Count-1) ?
  itineraryItems[item.Index+1].StartPathIndex : item.EndPathIndex
  for i = item.StationInfo.PointIndexAfterLastStation..lastIndex-1
  p1 = pathPoints[i]
  p2 = pathPoints[i+1]
  dist += Distance(p1, p2)
  return (dist / 60) * 3600 // convert to seconds
}
```

4.5 Detecting Incidents on a Route

For each route it evaluates, Mantis identifies all CHP incidents that lie along the route's path, using a procedure similar to that used to find traffic stations along the route. The pseudocode for this algorithm is given in Listing 4.14.

Listing 4.14: Finding CHP incidents on a route.

```
GetIncidentsOnRoute(route)
{
  for incident in incidents
    if incident.LatLon is not null
      for i = 0..route.PathPoints.Count-2
        P = route.PathPoints[i]
        Q = route.PathPoints[i+1]
        if IsIncidentBetweenPathPoints(P, Q, incident)
            if P.OnFreeway AND !incident.OnFreewayDir(P.Freeway, P.
                Direction)
            break
          incidentsOnRoute.Add(incident)
          break
  route.Incidents = incidentsOnRoute
}

IsIncidentBetweenPathPoints(P, Q, incident)
{
  valid = false
  U = ((incident.latitude - P1.latitude) * (P2.latitude - P1.latitude)) +
  ((incident.longitude - p1.longitude) * (P2.longitude - P1.longitude))
  Udenom = (P2.latitude - P1.latitude)^2 + (P2.longitude - P1.longitude)^2
  U = U / Udenom
  projectedPoint.latitude = P1.latitude + (U * (P2.latitude - P1.latitude)
     )
  projectedPoint.longitude = P1.longitude + (U * (P2.longitude - P1.
     longitude))
  minX = min(P1.latitude, P2.latitude)
  maxX = max(P1.latitude, P2.latitude)
  minY = min(P1.longitude, P2.longitude)
  maxY = max(P1.longitude, P2.longitude)
  valid = projectedPoint.latitude >= minX
    and projectedPoint.latitude <= maxX
    and projectedPoint.longitude >= minY
    and projectedPoint.longitude <= maxY
  if valid == true and Distance(projectedPoint, incident) <= 0.05
    return true
  else
    return false
}

OnFreewayDir(fwy, dir)
{
  for fwyDir in FreewayInfo
    if fwyDir.Number == fwy AND fwyDir.Direction == dir
      return true
  return false
}
```

As with VDSs, it is not enough to simply test for proximity to the route's path. It is also necessary to match the route's current freeway number and direction. OnFreewayDir performs this additional test. For each freeway previously detected in the incident's description text, the function compares the route path's freeway information to the incident's, and only returns true if a match is found.

4.6 Route Quality Assessment

After calculating each route's travel time and finding all relevant incidents for each route, Mantis uses this information to calculate holistic quality rankings for the routes. These rankings are expressed as numeric rankings, where high scores are superior to low scores.

Routes are presented to the user in the order determined by their scores, with the highest-ranked route displayed at the top of the route list.

Travel time is easy to incorporate into this calculation, since it is already a numeric quantity. However, determining the impact of a route's incidents on its score is less straightforward. To solve this problem, I developed a simple procedure for classifying incidents into severity categories, and assigned a numeric penalty to each category. Refer to Listing 4.15 for the scoring algorithm's complete pseudocode.

Listing 4.15: Route scoring.

```
RankRoutes(useRealTime, departTime)
{
  for r in routes
  r.EstimateTravelTime(useRealTime, departTime)
  GetIncidentsOnRoute(r)
  if r.EstimatedTravelTime > slowest
    slowest = r.EstimatedTravelTime

  for r in routes
  r.TravelTimeScore = slowestTravelTime - r.EstimatedTravelTime
  r.IncidentScore = CalculateIncidentScore(r)

  routes.SortByOverallScore()
}

CalculateIncidentScore(r)
{
  for i in r.Incidents
  score += i.Severity
  return score;
}
```

```

GetSeverityForIncidentType(type)
{
  if type in { "CLOSURE of a Road" }
  return Integer.MinValue
  else if type in { "Trfc Collision", "Traffic Hazard", "Traffic Advisory"
      , "Hit
  and Run", "Road/Weather Conditions" }
    return -10
  else
  return -3
}

CalculateOverallScore(r)
{
  return (travelTimeWeight * r.TravelTimeScore) + (incidentWeight *
r.IncidentScore) }
```

Note that in CalculateOverallScore, travelTimeWeight and incidentWeight are parameters of the scoring function. In Mantis, they are currently set to 0.5, giving equal weight to the travel time and incident components of the score.

CHP uses its own classification system for the incidents reported on its website. For some of these categories, it is relatively easy to intuitively assess their impact. One example is "CLOSURE of a Road" - this class of incident indicates the highest level of traffic disruption possible. As such, Mantis assigns an extremely steep penalty to incidents in this category, with the intent of invalidating any routes with road closures entirely. Since Mantis does not support recalculating route adjustments to navigate around road closures, its only response to routes that use a currently closed road is to not consider the route further.

Other incident classes, while not requiring such a drastic response, are still likely to significantly impact traffic flow. These categories include collisions/accidents, traffic advisories and hazards, and adverse road or weather conditions. However, it is difficult to quantitatively assess their impact based solely on the data provided on the CHP incident webpage. Some may drastically worsen traffic flow, while others may have a comparatively minor effect. Thus, Mantis uses a balanced approach with these incidents, assigning them a moderate penalty. This is intended to significantly reduce the route's overall quality, but not enough to remove it from consideration.

Finally, there are many other incident categories whose potential impact is even less clear. Mantis assigns any incidents not in the above two general categories a low penalty. Thus, while these incidents will still reduce a route's score, estimated travel time is likely to be the dominant factor.

Chapter 5

Evaluation

5.1 Speed Prediction

The objective of these experiments was to compare Mantis's speed predictions to actual observed traffic speeds.

I used the traffic data downloader program to periodically download real-time traffic speed data over several afternoons. For each of these samples, I used a small utility program to read the downloaded traffic data files and extract the speeds reported for a previously chosen set of VDSs. I then used Mantis's predictor program to generate predictions for these VDSs at the dates and times for which I obtained measurements. For these trials, the predictor's dataset included speed samples from September 1 November 14, 2012, and April 1 30, 2013. Note that, in order to limit the dataset's disk space requirements, only data for District 4 (the Bay Area) was used.

For the first set of such experiments, I used a group of VDSs on I-880 and US-101, near the 880/101 interchange in San Jose. Figure 5.1 shows the locations and IDS of these VDSs, along with an annotation indicating whether each VDS is on a northbound (N) or southbound (S) freeway.

I collected speed data for these VDSs on May 1-2 and May 8-9, 2013. Samples were taken at 3 PM, 4 PM, and 5 PM for all days, and also at 2 PM on May 8 and May 9. As described above, I used Mantis's predictor to generate predictions for this set of VDSs at the given times.

I also ran a second set of experiments using the same procedure. For these experiments, I used a set of VDSs on US-101 and I-280 between San Jose and San Francisco. The locations and IDs of these VDSs are given in 5.2.

Measurements for the VDSs in Figure 5.2 were recorded on May 14-17, 2013, at 3 PM, 4 PM, and 5 PM. As in the previous experiments, I used the predictor component to obtain predictions for these VDSs at the indicated dates and times. Complete results for these experiments are shown as column charts in the appendices.

5.1.1 Individual Cases

Figures 5.3 and 5.4 show examples of the results obtained from these experiments. Each colored series in these charts corresponds to speeds for one VDS. The predictor's accuracy can be judged by comparing the real-time measurements to the adjacent predicted speeds for the same time. Complete results for these experiments are given in the appendices.

In Figures 5.3 and 5.4, the predictor is generally quite accurate, especially for 4 PM and 5 PM. Similarly, Figure 5.4 shows good performance at 2 PM. It is also important to note that Figure 5.3 shows that the actual traffic speeds at 3 PM for VDS 400922 were somewhat lower than predicted. Conversely, in Figure 5.4, the predictor significantly underestimated the speed at 3 PM for VDS 400922. This VDS is located almost exactly at the intersection of freeways 880 and 101, which is often highly congested during evening peak hour. However, as these figures show, traffic speed can vary substantially between days, even at the same

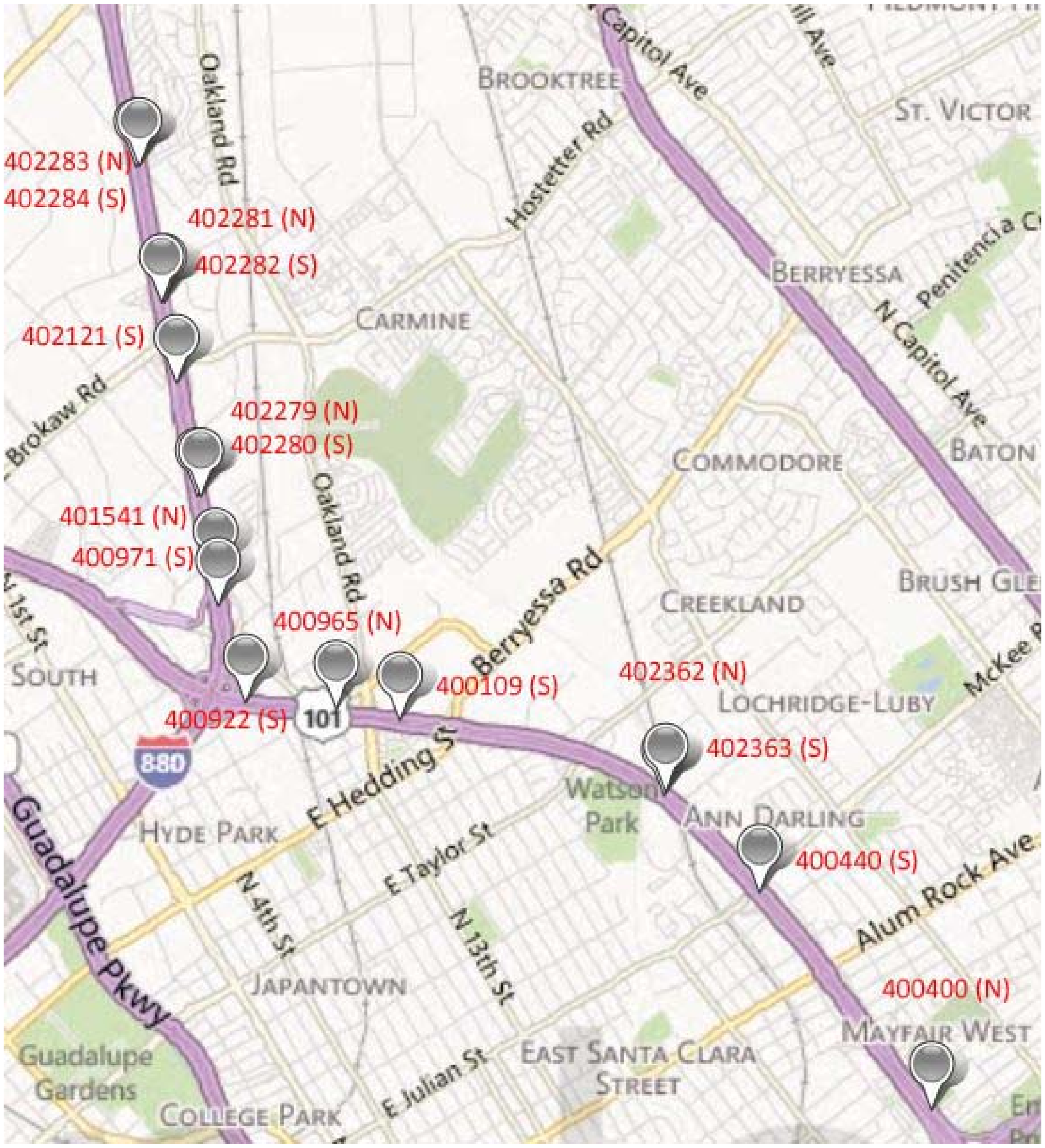


Figure 5.1: VDSs on I-880 and US-101.

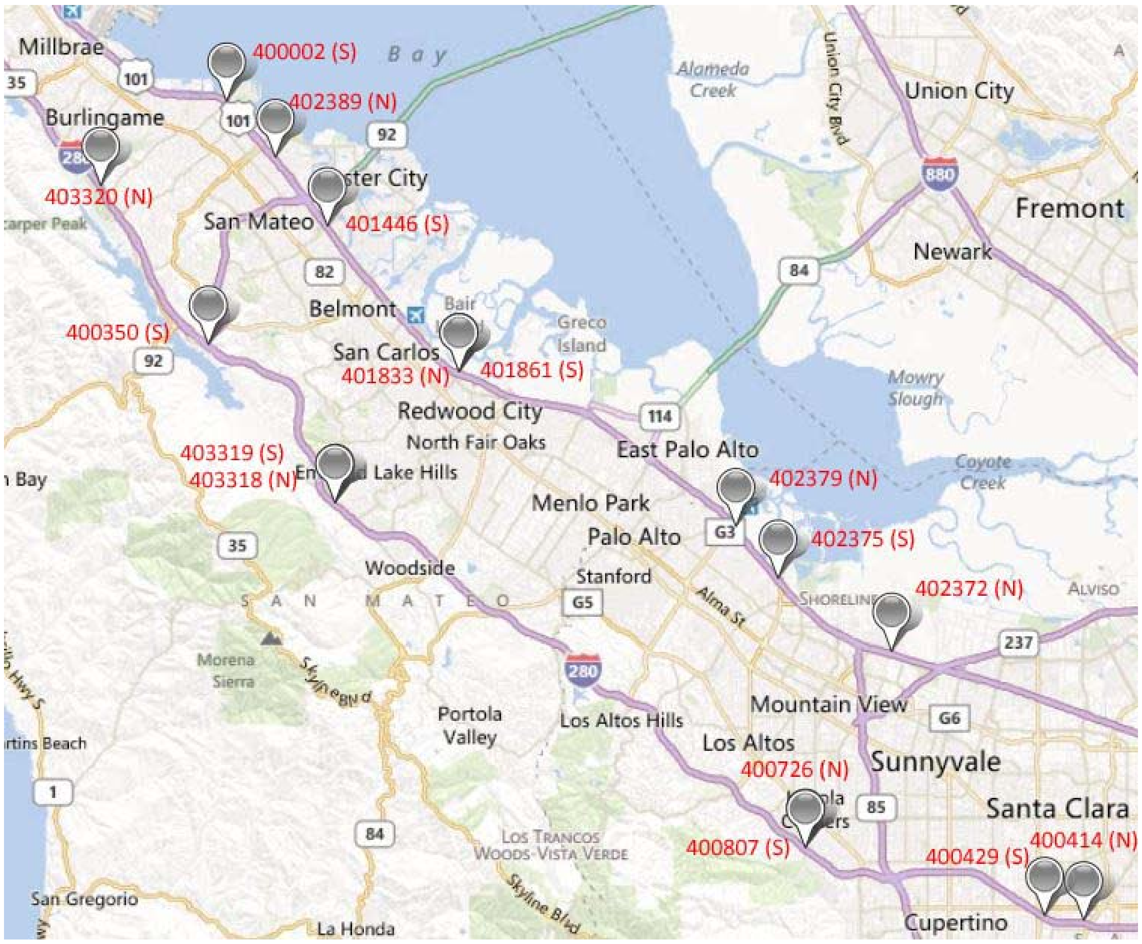


Figure 5.2: VDSs on I-280 and US-101.

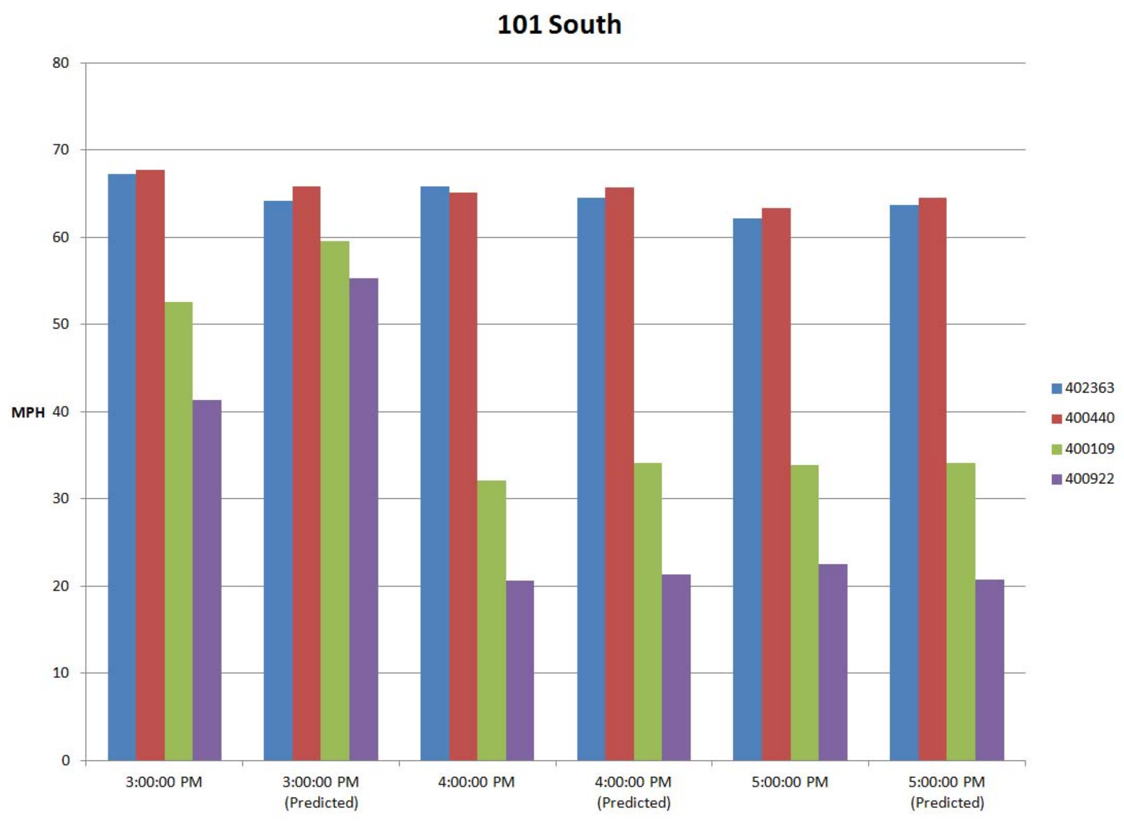


Figure 5.3: Sample of results for May 2.

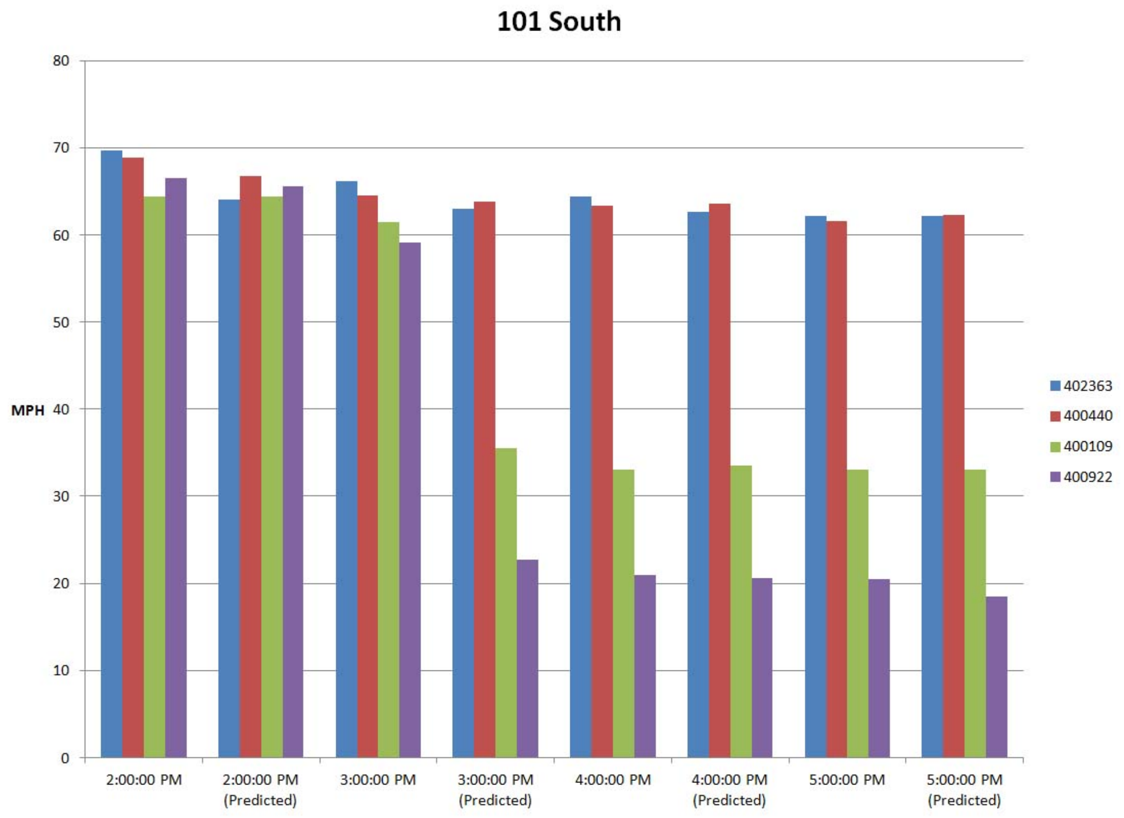


Figure 5.4: Sample of results for May 8.

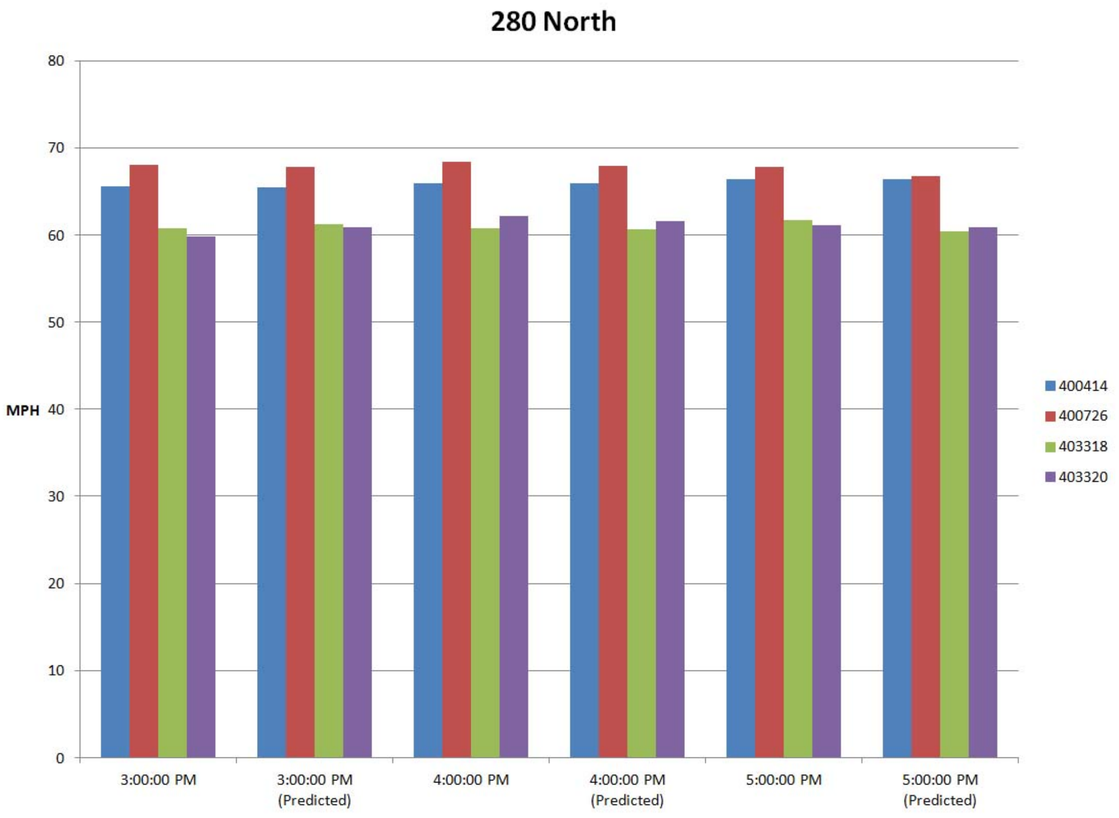


Figure 5.5: Sample of results for May 16.

location and time. Such fluctuations are very difficult to predict, as they can be due to numerous factors, such as accidents.

Figure 5.5 shows very good prediction accuracy for the VDSs on I-280 Northbound. Speeds at these locations are high and nearly constant over the measurement period, indicating that this section of freeway is not affected significantly by peak hour traffic. This pattern is repeated in the data for May 14, 15, and 17 as well. See the appendices for additional graphs.

In Figure 5.6, the predictor's results are not as consistent. In particular, it significantly underestimated the speed for VDS 402375 (on US-101 South between Palo Alto and Sunnyvale) at 3 PM and 4 PM. Its prediction for VDS 402375 at 5 PM is much closer to the

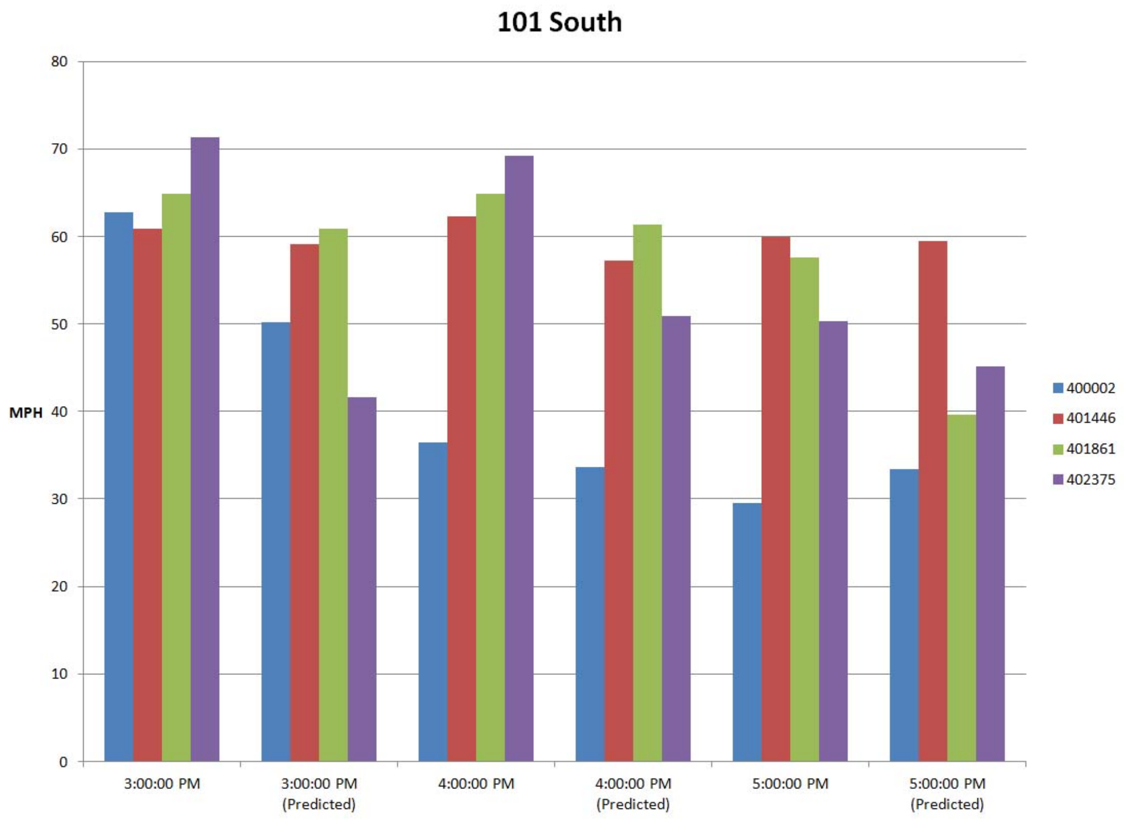


Figure 5.6: Sample of results for May 16.

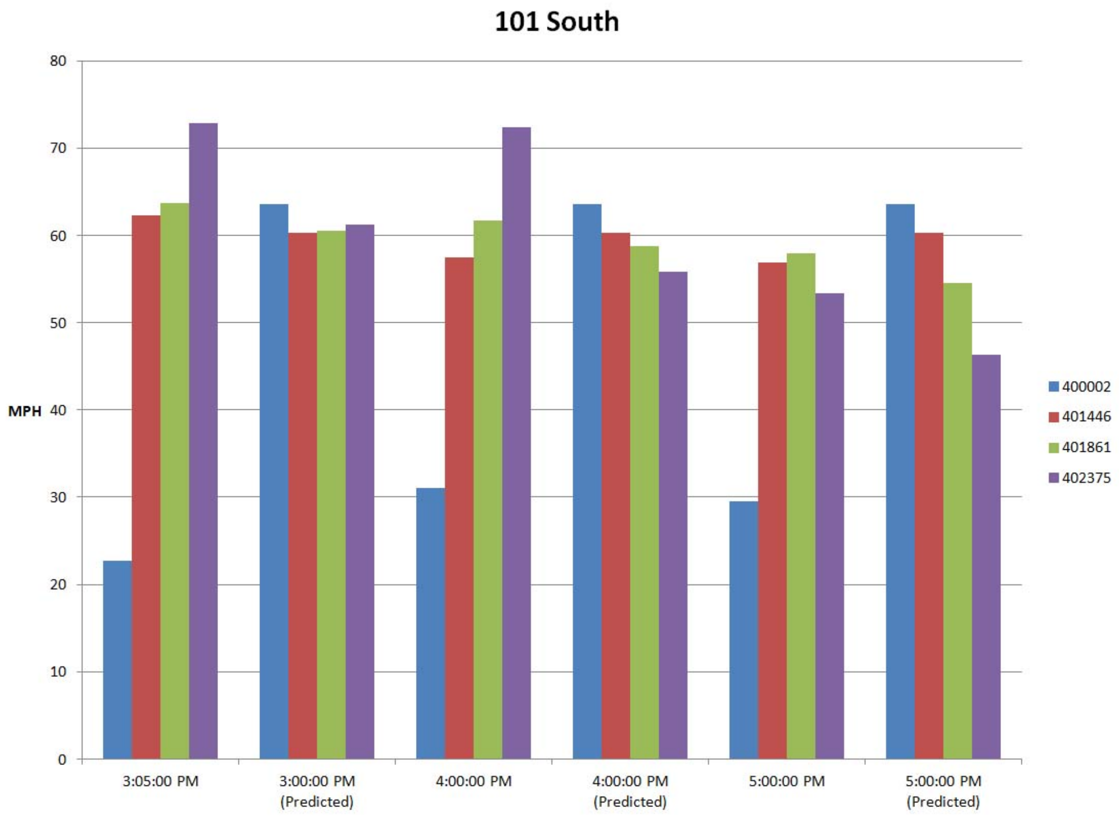


Figure 5.7: Sample of results for May 14.

observed speed, but its prediction at the same time for VDS 401861, which had been accurate at earlier times, is noticeably lower than the observed speed.

This unpredictability can be observed on other days as well. For instance, Figure 5.7 shows measurements and predictions for the same locations on May 14. The predictor again underestimates the speed for VDS 402375, though its prediction is more accurate than for May 16. Interestingly, it performs much better for VDS 401861, which reported fairly constant speeds on both days. However, VDS 400002's reported speeds for May 14 are much lower than the predictor's estimates. On May 16, the predictor generally estimated speeds for this VDS accurately.

These results suggest that certain portions of freeways are more difficult for the predictor than others. As previously noted, traffic conditions are always highly variable, even despite the generally predictable pattern of peak hour. The predictor cannot anticipate these anomalies using only historical speed data, so it is always possible that it will incorrectly estimate traffic speeds on days with unusual conditions, even if its predictions would be accurate under normal circumstances. Conversely, since the predictor's estimates are based on past traffic conditions, anomalous conditions in historical data may also impact the predictor's accuracy.

5.1.2 Overall Analysis

Figure 5.8 summarizes the results of my experiments. Each column shows the average prediction error for its corresponding VDS, where error is calculated as a percentage, using the following formula:

$$E(s) = \frac{\sum_{x=1}^{n} \frac{|P_x - A_x|}{A_x} \times 100}{n} \tag{5.1}$$

Here, P_x is the prediction for time x, A_x is the actual value for time x, and n is the number of samples for VDS s.

Figure 5.8 clearly shows that, as suggested by the previous discussion, the predictor performs much better for some VDSs than for others. This variability is visualized in Figures 5.9 and 5.10. VDSs with error less than 10% are shown in green, VDSs with error between 10% and 15% are shown in yellow, and VDSs with error greater than 15% are shown in red.

Tables 5.1 and 5.2 give further details on the VDSs with high error. This data seems to suggest some patterns. Seven of the ten VDSs with error above 10%, and four of the five VDSs with error above 15%, are on US-101. In addition, seven out of these ten VDSs are near on-ramps or off-ramps. Thus, it appears that US-101's traffic patterns may be generally more

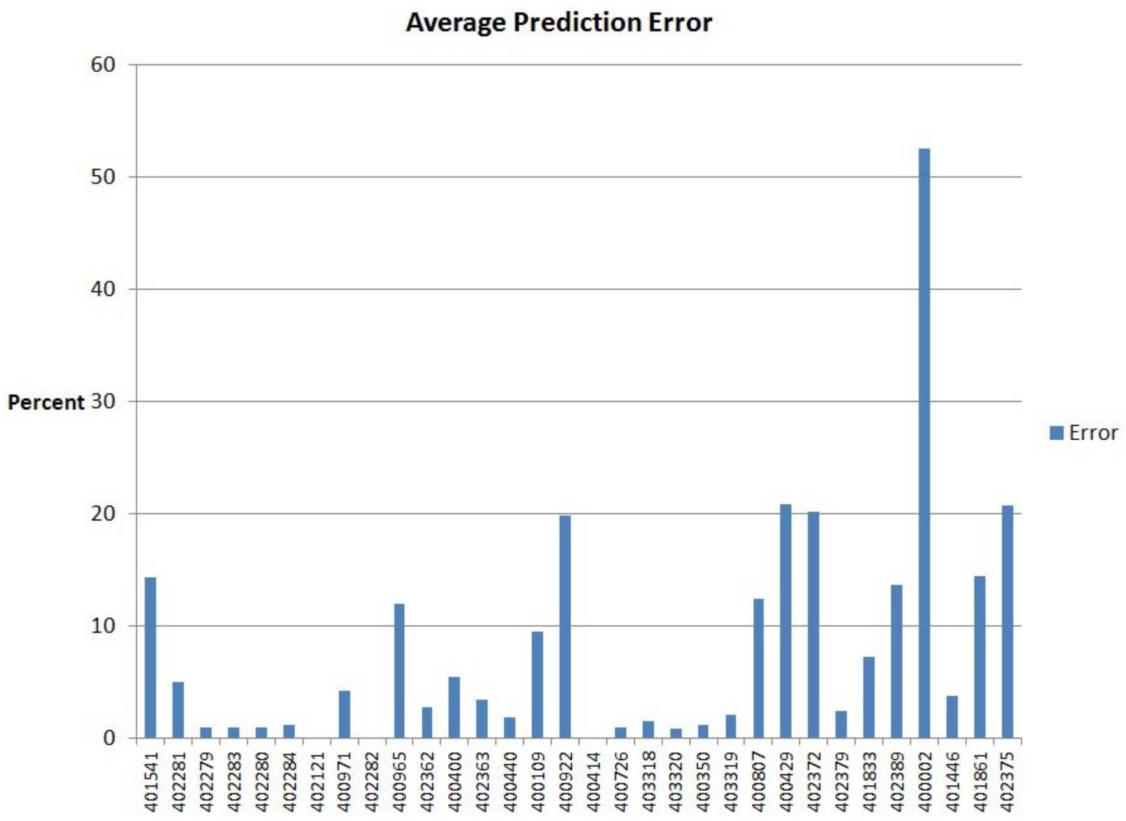


Figure 5.8: Average prediction error by VDS ID.

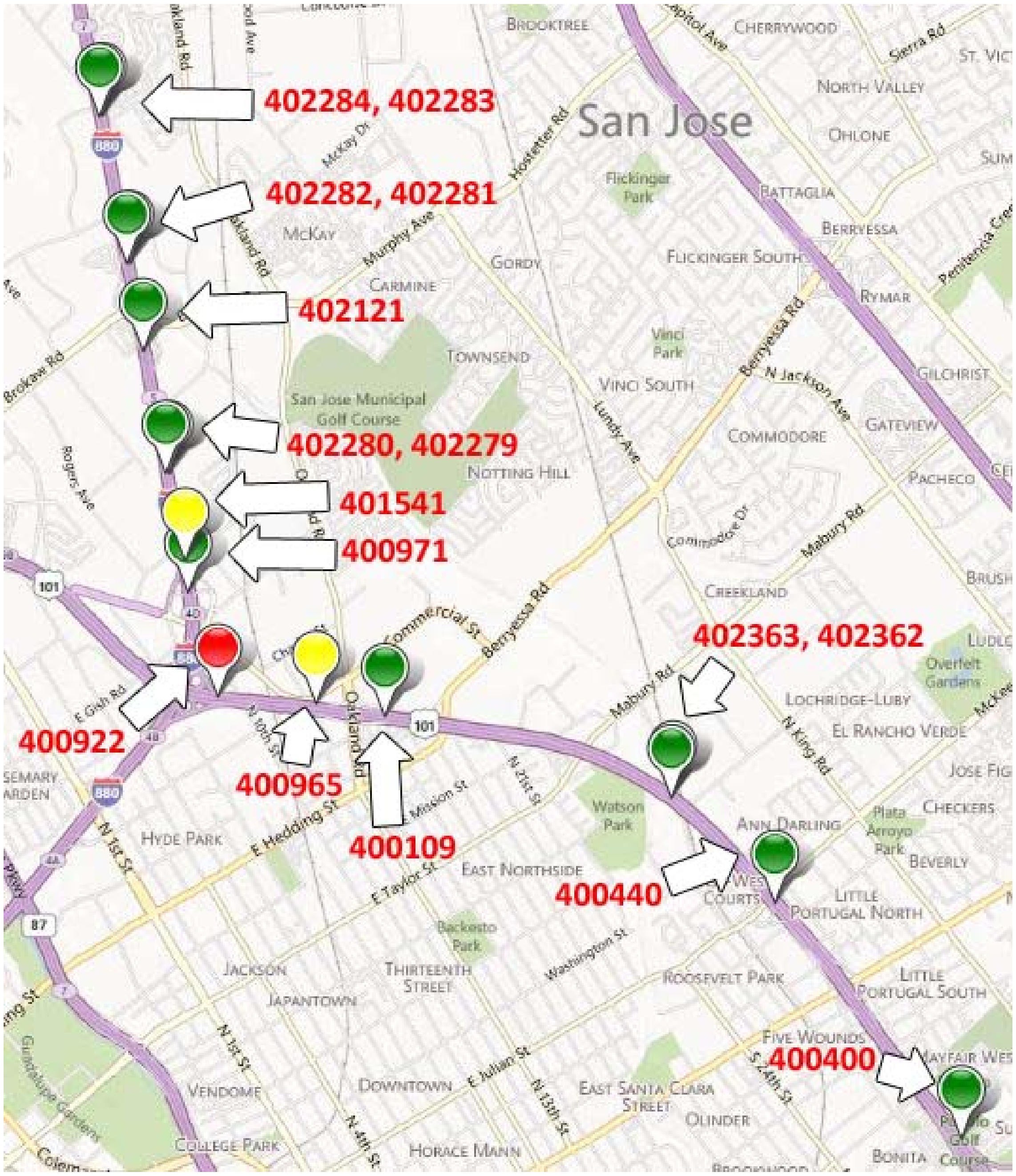


Figure 5.9: VDSs on US-101 and I-880 colored by average prediction error.

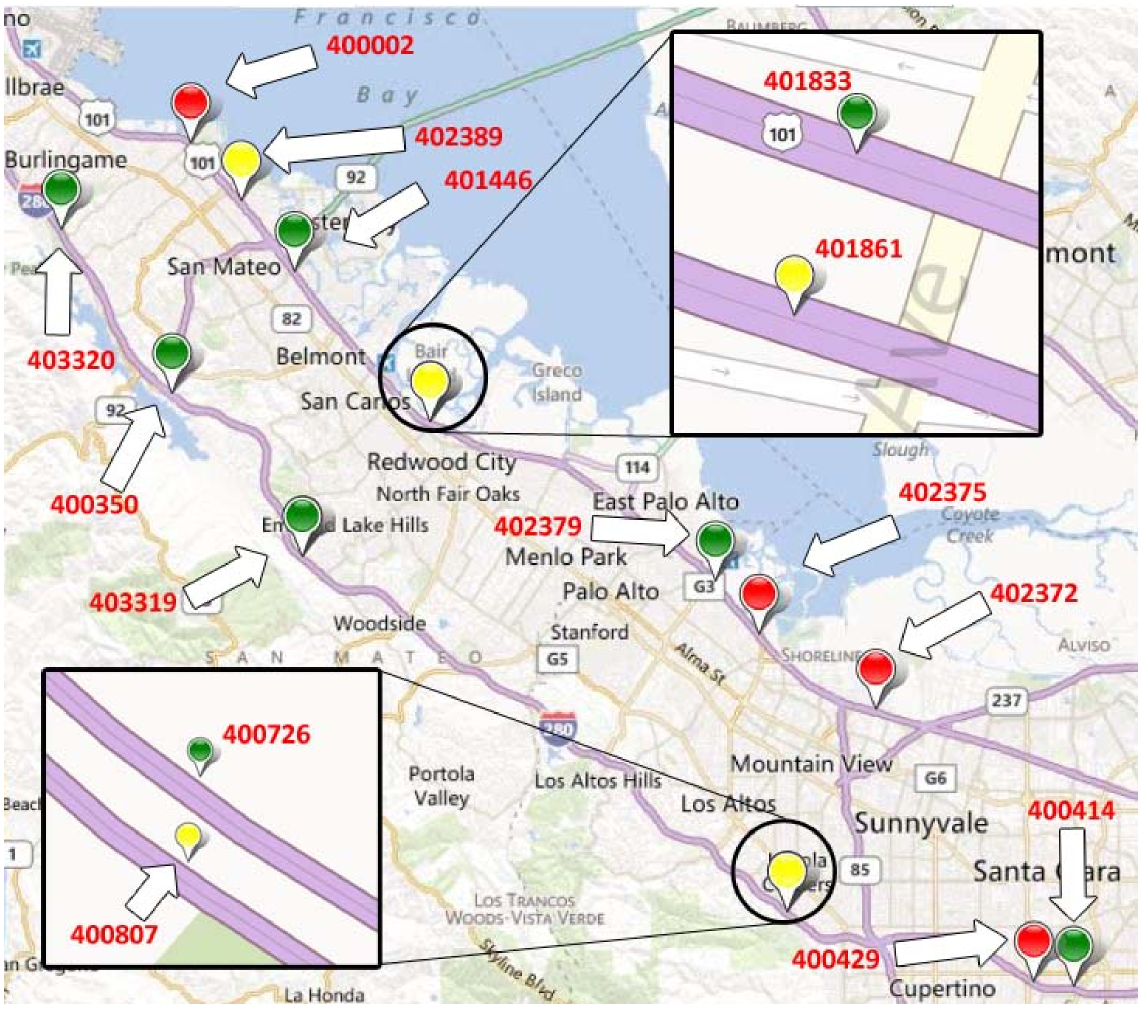


Figure 5.10: VDSs on US-101 and I-280 colored by average prediction error.

VDS ID	Location Description	Average Error
401541	880 N, near 880/101 interchange, Gish Rd on-ramp	14.366%
400965	101 N, near 880/101 interchange, Oakland Rd on-ramp	11.992%
400807	280 S, near Loyola - no ramps	12.439%
402389	101 N, north of San Mateo - no ramps	13.708%
401861	101 S, near Redwood City, Whipple Ave on-ramp	14.431%

Table 5.1: VDSs with average error between 10 and 15 percent.

VDS ID	Location Description	Average Error
400922	101 S, near 880/101 interchange, on-ramps from 880	19.828%
400429	280 S, E of Cupertino, near on-ramp from Lawrence Expy	20.841%
402372	101 N, near Mountain View / Sunnyvale, Ellis St on-ramp	20.182%
400002	101 S, between San Mateo and San Francisco airpoirt - no ramps	52.48%
402375	101 S, between Palo Alto and Mountain View, San Antonio Rd off-ramp	20.764%

Table 5.2: VDSs with average error over 15 percent.

difficult to predict than those of I-280 or I-880. There also may be a correlation between proximity to on-ramps or off-ramps, as many of the more problematic VDSs are close to ramps. However, a more exhaustive study would be required to conclusively establish these relationships.

I also analyzed the predictor's overall performance by calculating the average error for each measurement time across all VDSs. As Figure 5.11 indicates, the predictor's performance does not seem to be influenced significantly by time of day. Rather, prediction accuracy appears to depend much more on location, with some locations proving much more difficult to accurately predict than others.

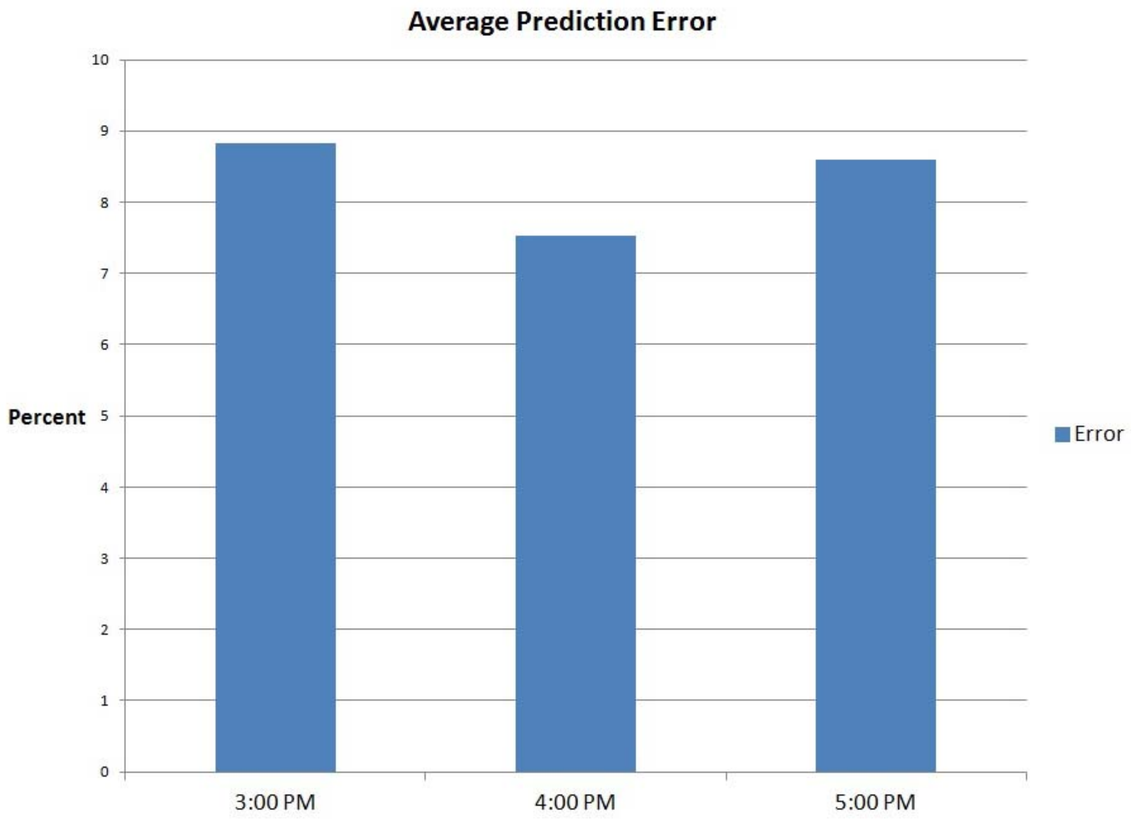


Figure 5.11: Average prediction error by measurement time.

Variable	Value
t Stat	2.909737524
Degrees of Freedom (df)	382
t Critical (two-tail)	1.966193433
P($T \leq t$) (two-tail)	0.003829006

Table 5.3: Results of paired t-test on prediction data.

Finally, to assess the statistical significance of the predictor's error, I conducted a paired t-test on my complete prediction experiment data set. Each pair consisted of an observed speed for a single VDS at a particular time and the predicted speed for the same VDS and time.

For this t-test, I used the following hypotheses:

- **Null hypothesis:** The average difference between the actual speeds and the predicted speeds is 0.
- **Alternate hypothesis:** The average difference between the actual speeds and the predicted speeds is not 0.

The test's results are given in Table 5.3. Because the computed t-value is greater than the critical t value, and the p-value is quite small (less than 5%), these test results support the alternate hypothesis stated above. It therefore appears that the predictor's error is statistically significant, which indicates that the predictor is in need of further refinement. Enlarging its historical data set is one possible way to improve its accuracy.

5.2 Comparison to Bing Maps and Google Maps

In order to assess Mantis's overall performance, I conducted a series of experiments comparing its recommendations with those from Bing Maps and Google Maps.

I used Bing Maps and Google Maps to get directions from San Jose, CA to San Francisco, CA and from San Francisco, CA to San Jose, CA in both morning and evening over several days. Bing Maps only displays one route, but it also includes an option to recalculate the route based on current traffic conditions. I used this feature to get traffic-optimized routes from Bing Maps. Google Maps gives several routes to the destination and displays an estimated travel time based on traffic for each one, so I chose the routes with the lowest predicted travel time as its recommended routes. Finally, I used Mantis to evaluate routes based on predicted conditions at the same days and times, and chose the routes with the lowest predicted travel times as its recommendations.

5.2.1 Sample Results

Before presenting overall results, I will consider two individual experiments. The first of these was performed on May 16, 2013 at approximately 5 PM. Figure 5.12 shows the route recommended by Bing Maps, while Figure 5.13 shows the route recommended by Mantis. In addition, Figure 5.14 shows the route that Mantis ranked second for this trial.

In this case, Mantis and Bing Maps both recommend traveling to San Francisco using I-280 North. In particular, the I-280 route certainly appears to have less traffic congestion than the US-101 North route shown in Figure 5.14. Thus, Mantis's recommendation seems sound in this case.

The second sample experiment was performed on May 17, 2013 at approximately 5 PM. Unlike in the previous case, Mantis and Bing Maps recommend different routes. Bing Maps suggests using I-280 South, while Mantis recommends US-101 South. Figures 5.15, 5.16, and 5.17 show these routes.

While Mantis's recommended route does not appear to be severely congested, it does show some areas with slower traffic. Mantis's second-ranked route, which primarily follows

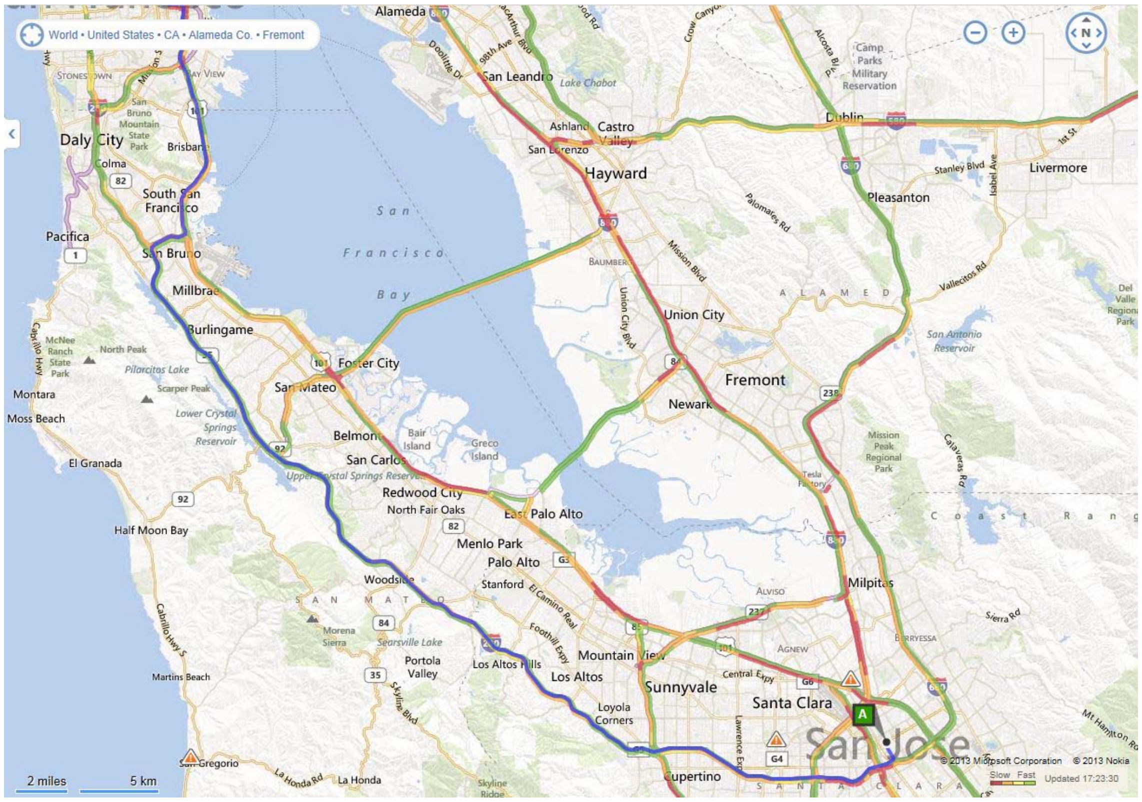


Figure 5.12: Bing Maps traffic-optimized route from San Jose to San Francisco, May 16, 5 PM.

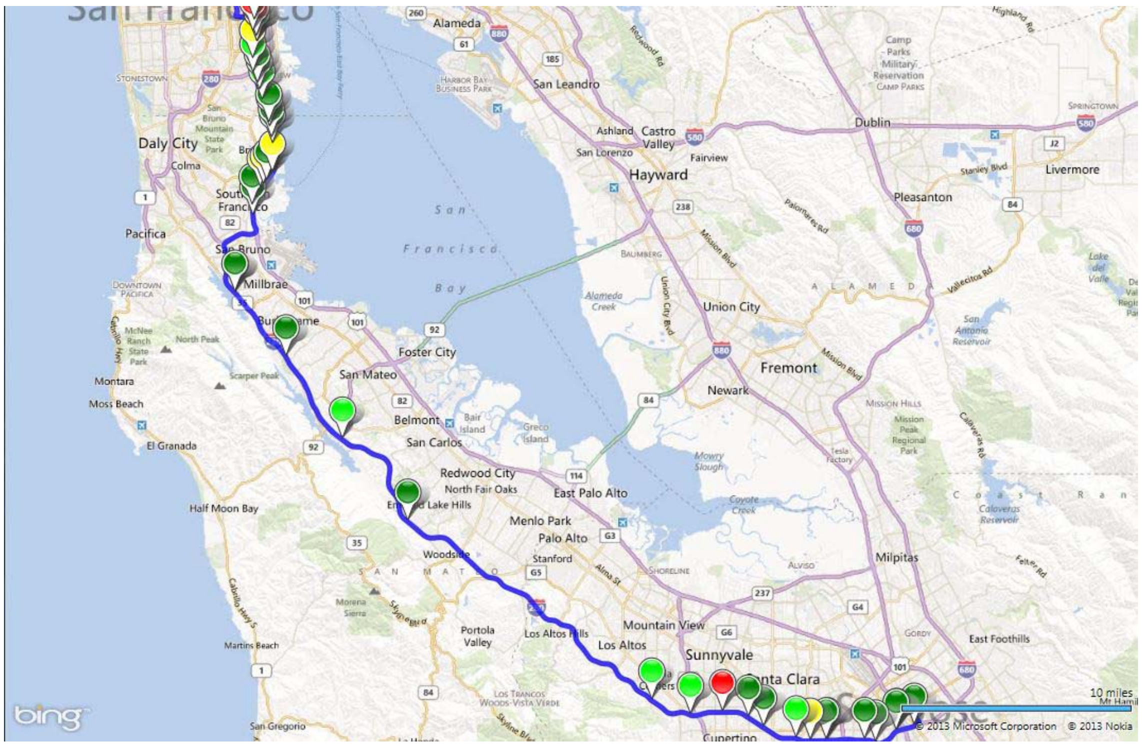


Figure 5.13: Mantis recommended route from San Jose to San Francisco, May 16, 5 PM.

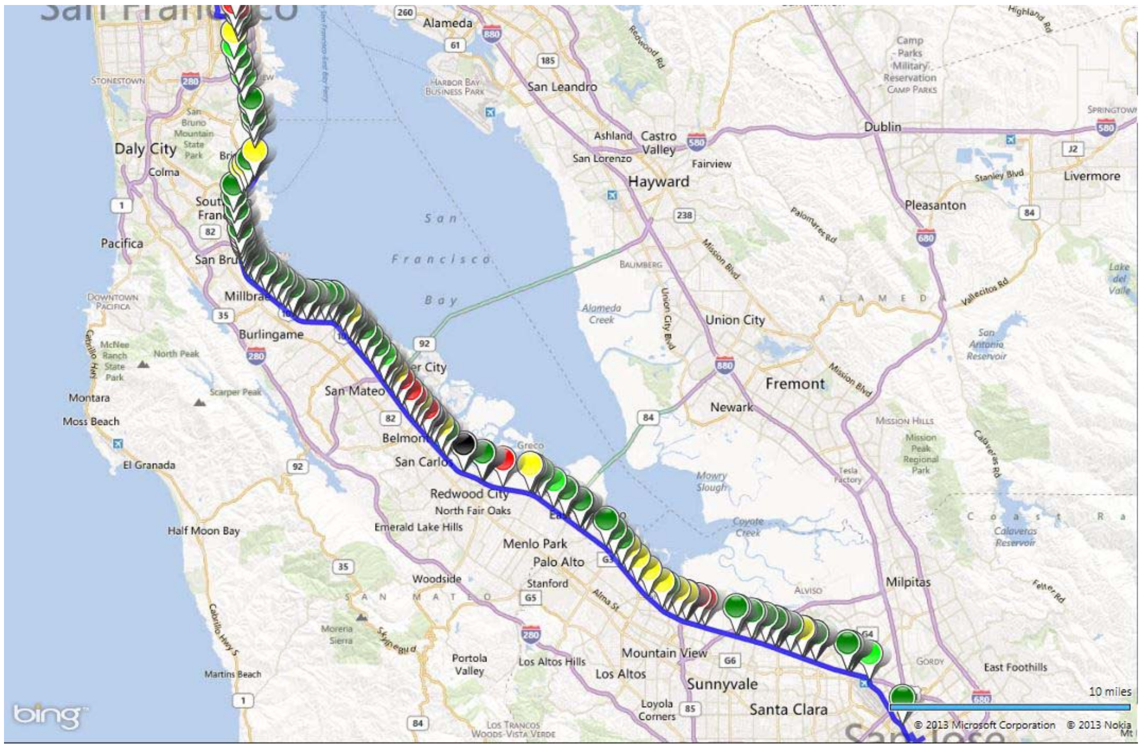


Figure 5.14: Mantis second-fastest route from San Jose to San Francisco, May 16, 5 PM.

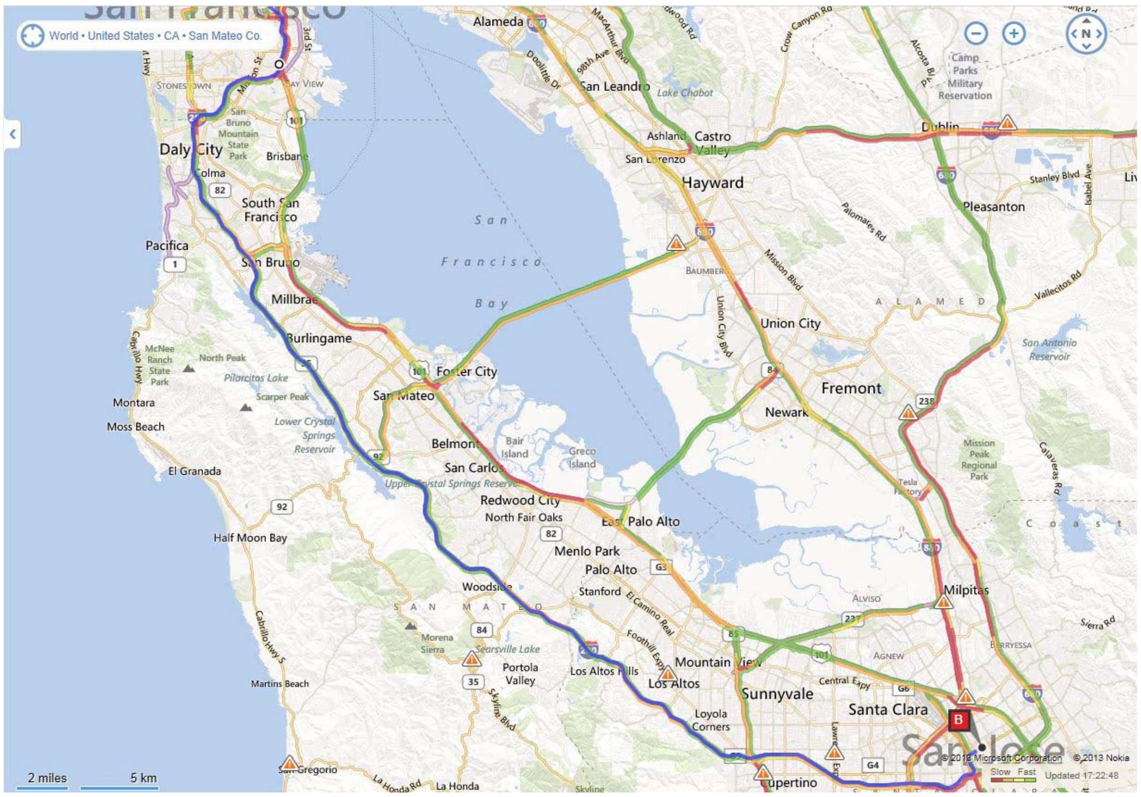


Figure 5.15: Bing Maps traffic-optimized route from San Francisco to San Jose, May 17, 5 PM.

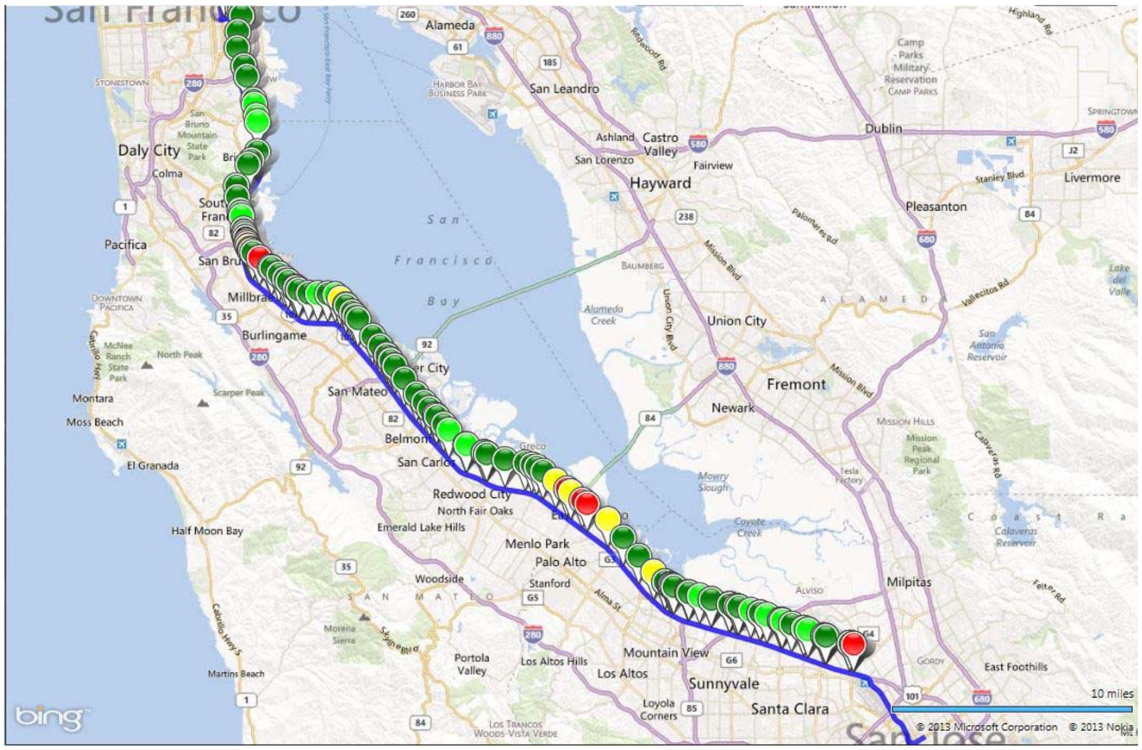


Figure 5.16: Mantis recommended route from San Francisco to San Jose, May 17, 5 PM.

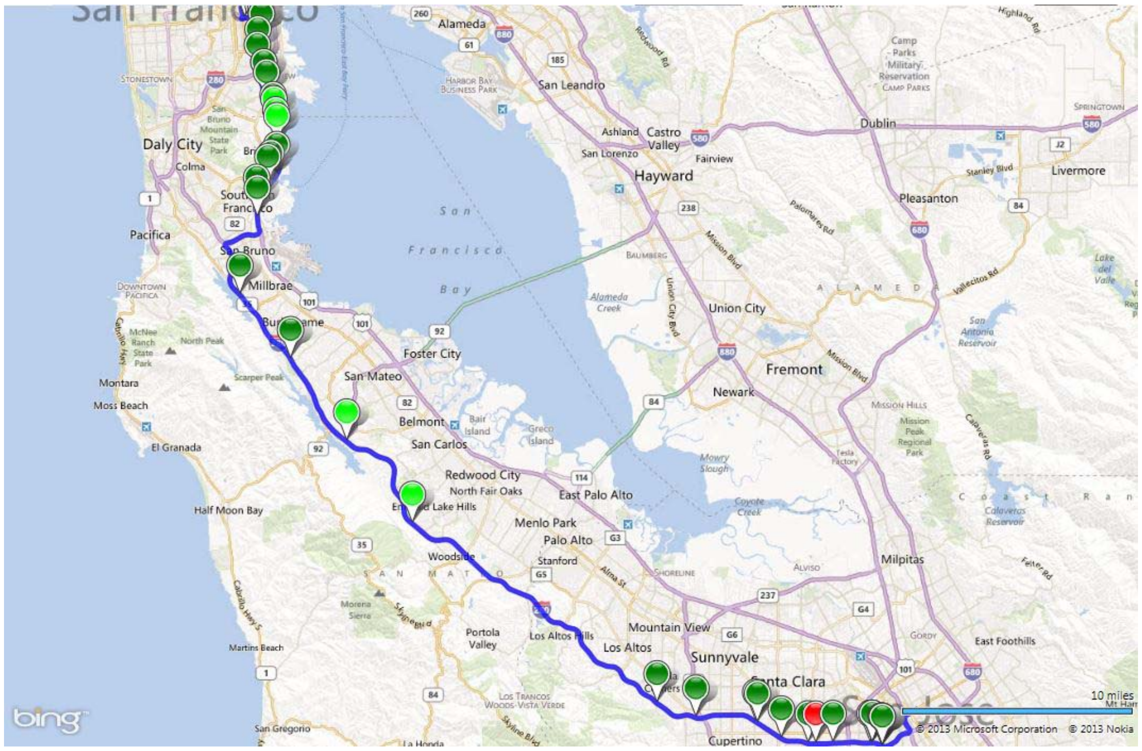


Figure 5.17: Mantis second-fastest route from San Francisco to San Jose, May 17, 5 PM.

I-280 South, appears to have minimal congestion. Even though the route following I-280 is several miles longer than the route following US-101, it seems reasonable that the higher congestion on US-101 could result in a higher travel time.

This behavior is at least partially due to a limitation that sometimes prevents Mantis from finding VDSs for freeway segments. The itinerary item containing instructions for merging onto CA-87 for the US-101 route does not specify a freeway direction in its instruction text. Since Mantis depends on the instruction text to determine the freeway direction for each itinerary item, it is unable to do so for this particular itinerary item. This in turn prevents it from finding VDSs for the itinerary item, since without a freeway direction it cannot positively identify the VDSs for that freeway that should be included. In such cases, Mantis is forced to fall back on the travel time given in the itinerary item's raw XML, which does not take traffic conditions into account. In this particular case, Mantis will estimate an unrealistically low travel time for CA-87 South, leading to a lower travel time estimate for the route as a whole. Since these directions were generated for a Friday evening, CA-87 South was almost certainly congested at the time. If Mantis was able to account for these conditions, its recommendation may have been different, and probably more accurate as well.

5.2.2 Overall Results

Tables 5.4 and 5.5 show the complete results of my experiments comparing Mantis to Bing Maps and Google Maps. All travel times in these tables are in minutes.

Overall, Mantis recommends a route that either exactly or nearly matches the route recommended by Bing Maps in 9 out of 14 cases. Mantis's recommendation matches the fastest route from Google Maps in 5 out of 8 cases.

Table 5.5 shows two cases in particular where Mantis's recommended route is quite different from the routes recommended by Google Maps and Bing Maps. On the evenings of

Time	Bing Route	Bing Time	Google Route	Google Time	Mantis Route	Mantis Time	Mantis 2nd Route	Mantis Time
5-14 9:15AM	280N, 101N	69	—	—	280N, 380E, 101 N	58.1	101N	58.7
5-14 5:00PM	280N, 380E, 101N	66	—	—	280N, 380E, 101N	60.95	101N	61.86
5-15 9:15AM	280N, 380E, 101N	70	—	—	101N	56.75	280N, 380E, 101N	58.15
5-15 5:15PM	280N, 380E, 101N	66	—	—	280N, 380E, 101N	60.79	101N	66.77
5-16 9:15AM	280N, 101N	70	280N, 380E, 101N	65	280N, 380E, 101N	57.86	101N	59.2
5-16 5:15PM	280N, 380E, 101N	66	280N, 380E, 101N	67	280N, 380E, 101N	62.22	101N	69.56
5-17 9:15AM	101N	65	280N, 380E, 101N	58	101N	55.42	280N, 380E, 101N	57.06
5-17 5:15PM	280N, 380E, 101N	67	280N, 380E, 101N	69	280N, 380E, 101N	59.53	101N	66.35

Table 5.4: Recommendations for routes from San Jose, CA to San Francisco, CA.

Time	Bing Route	Bing Time	Google Route	Google Time	Mantis Route	Mantis Time	Mantis 2nd Route	Mantis Time
5-15 9:15AM	101S, 380W, 280S	67	—	—	101S	58.72	101S, 380W, 280S	59.31
5-15 5:15PM	101S, 380W, 280S	67	—	—	80E, 880S, 101S	68.52	101S, 380W, 280S	69.69
5-16 9:15AM	101S, 380W, 280S	69	101S, 380W, 280S	59	101S, 380W, 280S	57.69	101S	58.82
5-16 5:15PM	101S, 280S	71	101S, 380W, 280S	73	80E, 880S, 101S	70.69	101S, 380W, 280S	72.22
5-17 9:15AM	101S	62	101S	50	101S	53.34	101S, 380W, 280S	57.1
5-17 5:15PM	101S, 280S	69	101S, 380W, 280S	68	101S	58.77	101S, 380W, 280S	59.42

Table 5.5: Recommendations for routes from San Francisco, CA to San Jose, CA.

May 15 and May 16, Mantis recommended a route to San Jose that uses I-80 Eastbound on the Bay Bridge and proceeds onto I-880 Southbound. This route is never recommended by the other two services, so it seems peculiar that Mantis would favor it. This is again at least partially due to Mantis's inability to determine the freeway direction for I-80, since the instructions for merging onto I-80 did not specify a direction explicitly. As in the previous case involving CA-87, Mantis's recommendation of this route was influenced by an unrealistically low travel time estimate for the I-80 portion of the route.

In terms of estimated travel time, Mantis is rarely consistent with the other two services, nearly always predicting a lower travel time than Bing Maps or Google Maps. It is interesting to note that Mantis's travel time estimates are generally closer to Google Maps' estimates than to Bing Maps' estimates. The average difference between Mantis's time predictions and Bing Maps' time predictions is 7.69 minutes, while the average difference between Mantis's time predictions and Google Maps' time predictions is 5.02 minutes.

5.3 CHP Incident Detection

Because Bing Maps and Google Maps both display visual alerts for traffic incidents and alerts, I conducted an experiment to evaluate Mantis's incident detection abilities compared to the other two services. I visited the Bing Maps and Google Maps websites, enabled their traffic overlay views, and moved their map views to the area surrounding the southern portion of the San Francisco Bay. I also used Mantis to display CHP incidents for the same region. Finally, for each of the three systems, I noted the number and type of incidents reported. Screenshots of each system's results are shown in Figures 5.18, 5.19, and 5.20.

Bing Maps seems to have the weakest incident detection, only displaying a handful of incidents. It is important to note, however, that it does detect scheduled construction locations that are not shown in Mantis or Google Maps.

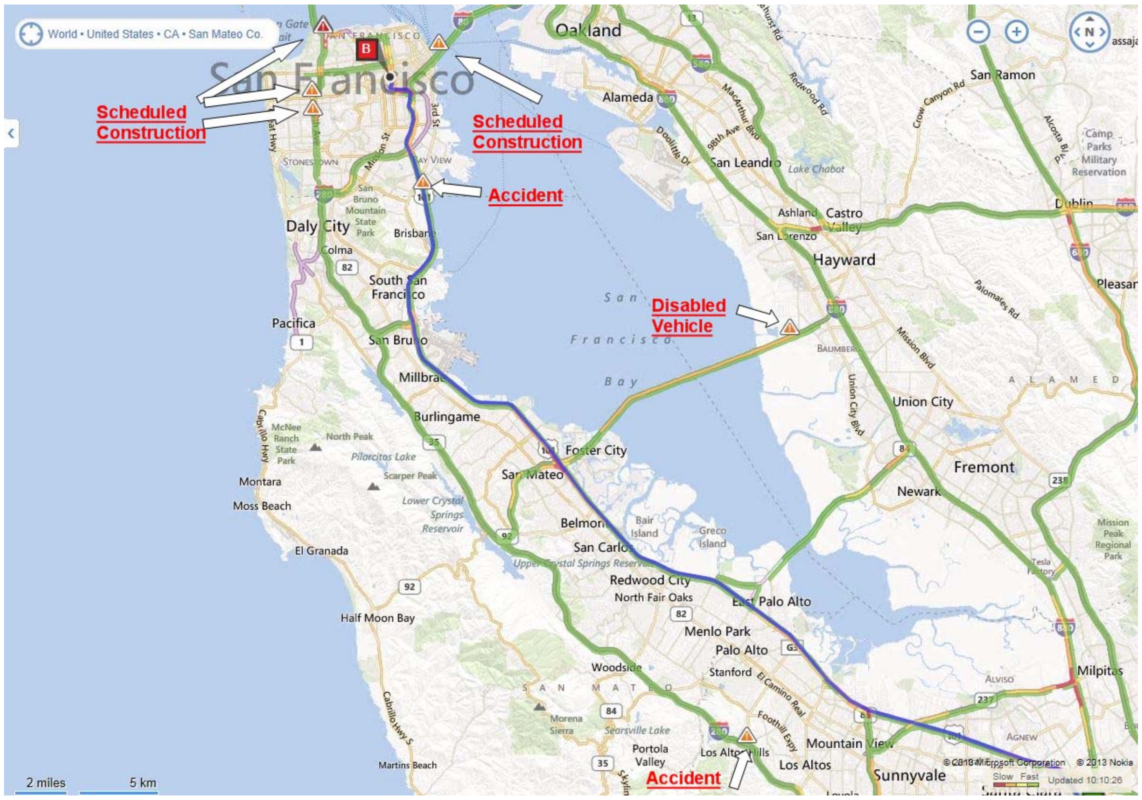


Figure 5.18: Bing Maps incidents for the Bay Area May 21, 2013, 10 AM.

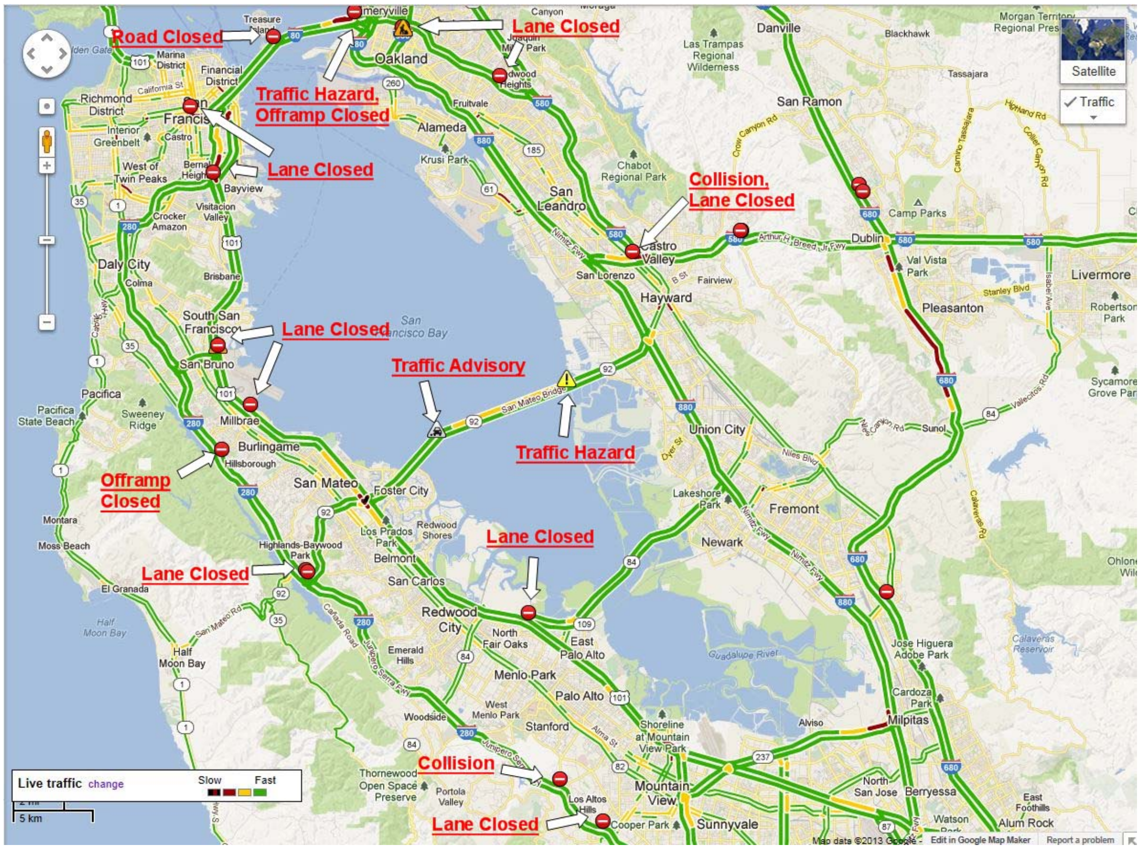


Figure 5.19: Google Maps incidents for the Bay Area May 21, 2013, 10 AM.

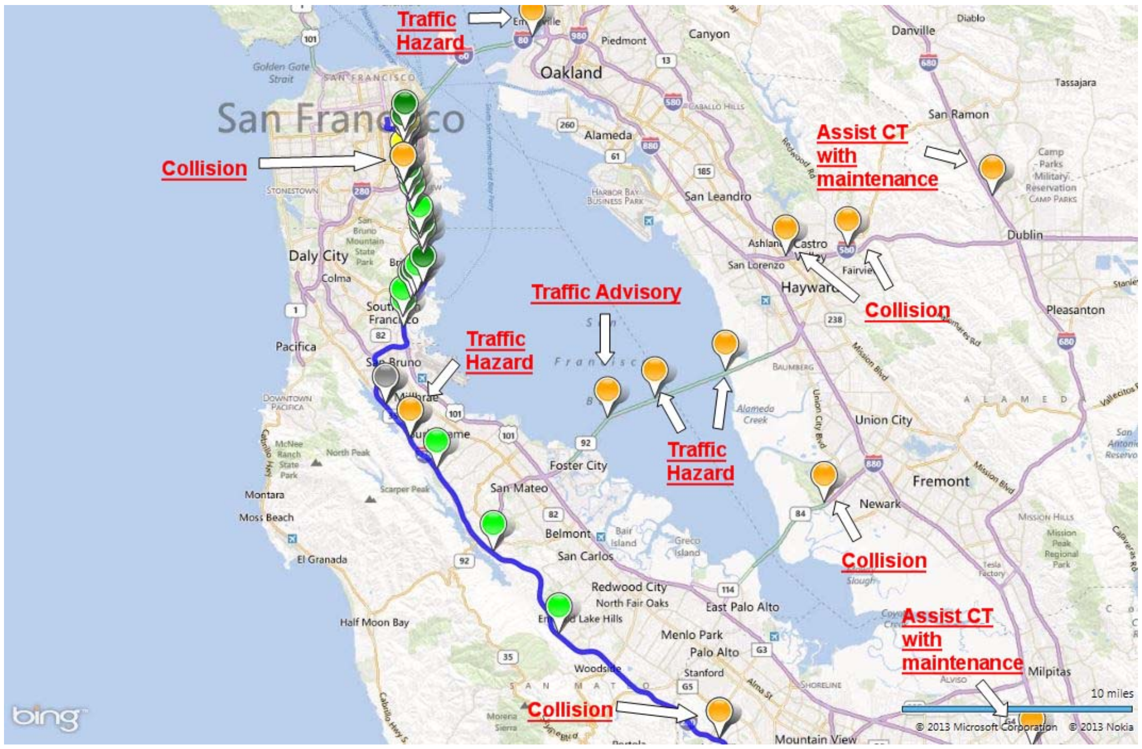


Figure 5.20: Mantis incidents for the Bay Area May 21, 2013, 10 AM.

Mantis shows significantly more incidents than Bing Maps. In particular, it detects a significant number of traffic hazard incidents that are not shown in Bing Maps.

Overall, Google Maps appears to have the most extensive incident detection capability. In addition to detecting collisions and generic traffic hazard incidents, it also detects a significant number of lane closures, which are not displayed in either Bing Maps or Mantis.

Based on these results, Mantis's greatest shortcoming in terms of incident detection is its inability to detect lane closures. The CHP incident website does not seem to be an adequate data source for these incidents. Remedying this deficiency would therefore require finding a suitable complementary data source for Mantis.

Chapter 6

Conclusions and Future Work

I believe my experiments have demonstrated that Mantis is a capable directions recommender system. However, there are also many opportunities for improving and expanding Mantis's capabilities.

First, as noted in the previous section, Mantis's prediction component is currently using a comparatively small dataset for only a portion of the California freeway network. Adding more data to the dataset would most likely improve the predictor's accuracy, and also expand its applicability to more locations.

Another potential extension is the inclusion of additional data sources and types. For instance, Mantis currently cannot effectively obtain data on lane closures or scheduled construction. Both can significantly affect traffic flow and travel times, so locating suitable data sources for these activities and incorporating them into Mantis would improve its ability to assess current travel conditions.

Finally, Mantis has only a limited ability to compensate for missing or incomplete data. This is particularly evident for routes with itinerary items that do not specify freeway directions in their instructions. As discussed in the previous section, when Mantis cannot determine the current direction of travel on a freeway, it cannot detect VDSs along that portion of the route. This significantly impairs its ability to accurately predict travel times for such routes. If Mantis was instead able to intelligently determine the current direction of travel even if it is not explicitly specified, the reliability of its estimates would be significantly improved.

Bibliography

[1] The apache cassandra project. http://cassandra.apache.org.

[2] Calculate distance, bearing and more between latitude/longitude points. http://www.movable-type.co.uk/scripts/latlong.html.

[3] Caltrans pems - glossary. http://pems.dot.ca.gov/?dnode=Help&content=help_glossary.

[4] Caltrans pems - value added resellers - file formats. http://pems.dot.ca.gov/?directory=Help&dnode=Help&content=var_fmt.

[5] Data modeling - datastax apache cassandra 1.1 documentation. http://www.datastax.com/docs/1.1/ddl/column_family.

[6] Driving route with route path example. http://msdn.microsoft.com/en-us/library/gg636956.aspx.

[7] Time in current traffic - google maps help. http://support.google.com/maps/bin/answer.py-?hl=en&answer=2549020&topic=1687356&ctx=topic.

[8] Caltrans pems - system calculations. http://pems.dot.ca.gov/?dnode=Help&content=help_calc, March 2007.

[9] Bing maps new routing engine - bing maps blog. http://www.bing.com/blogs/site_blogs/b/maps/archive/2012/01/05/bing-maps-new-routing-engine.aspx, January 2012.

[10] C. Cushing. Detecting netflix service outages through analysis of twitter posts. http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1765&context=theses.

[11] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA '11)*, May 2011.

[12] Z. Jia, C. Chen, B. Coifman, and P. Varaiya. The pems algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 536–541, 2001.

[13] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[14] M. Raptis. Inside seven - caltrans, district 7 - monthly newsletter. http://www.dot.ca.gov/dist07/Publications/Inside7/story.php?id=62, April 2011.

[15] P. Varaiya. Freeway performance measurement system, pems v3, phase 1: Final report. *Working Papers*, October 2001.

[16] P. Varaiya. California's performance measurement system: Improving freeway efficiency through transportation intelligence. http://www.techtransfer.berkeley.edu/newsletter/02-4/pems.php, 2002.

Appendix A

Predictor Evaluation Results - Part 1

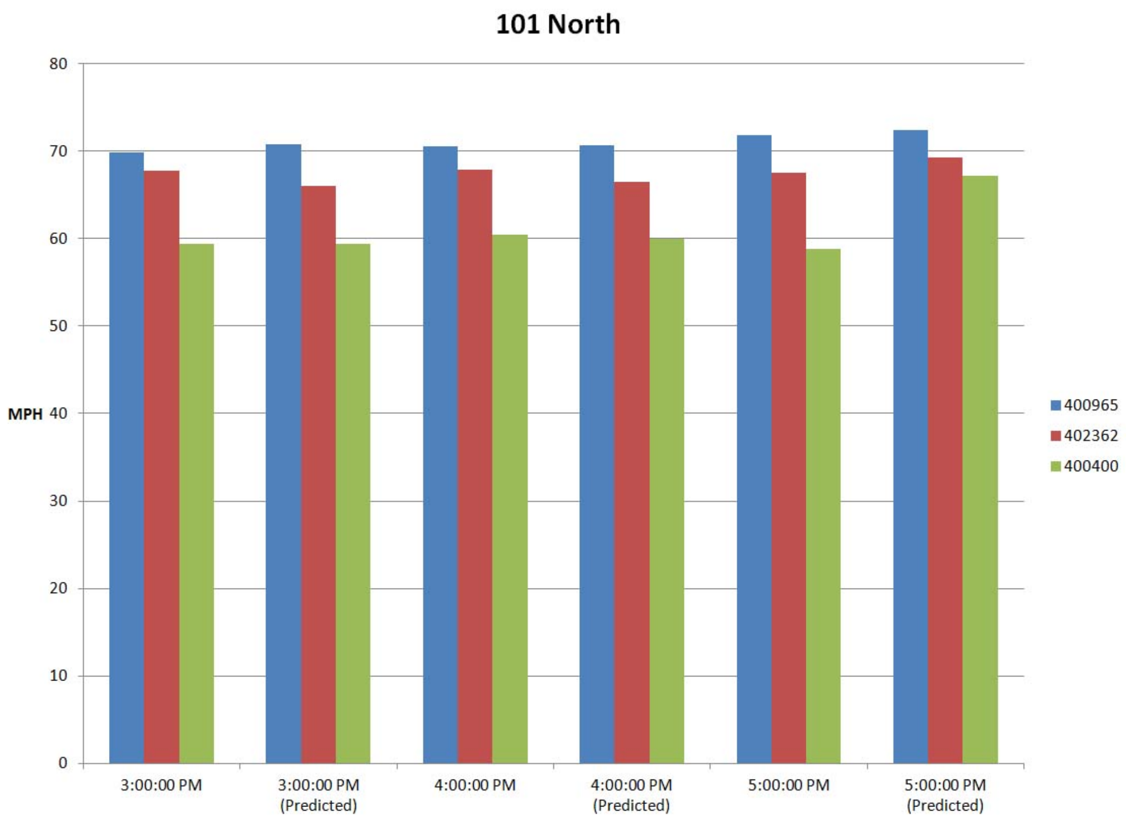


Figure A.1: Results for I-880 and US-101 on May 1, 2013.

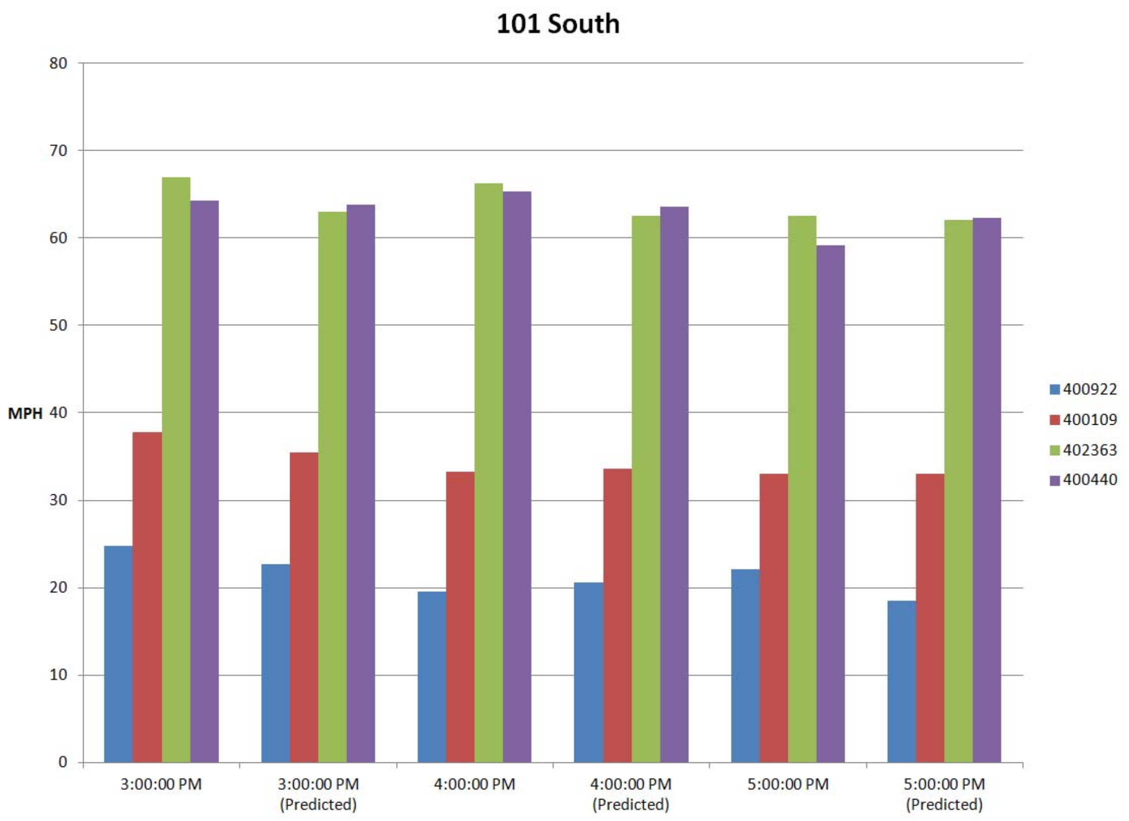


Figure A.2: Results for I-880 and US-101 on May 1, 2013.

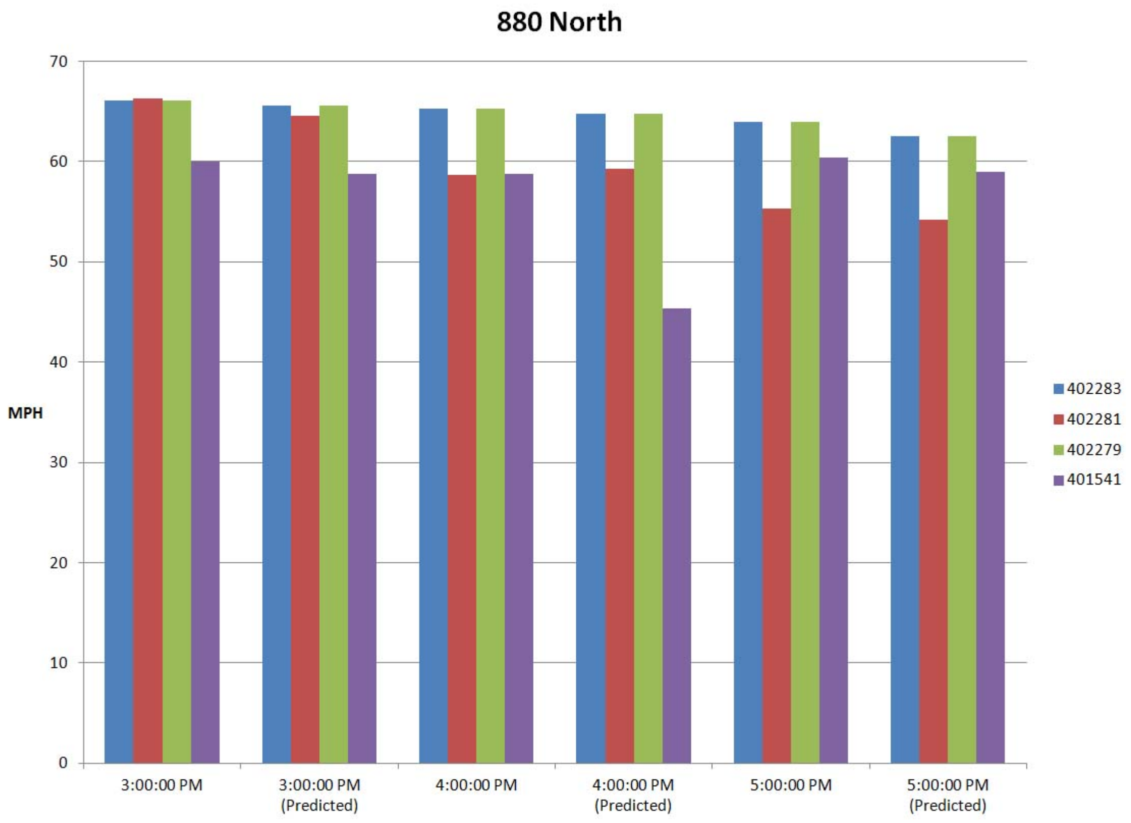


Figure A.3: Results for I-880 and US-101 on May 1, 2013.

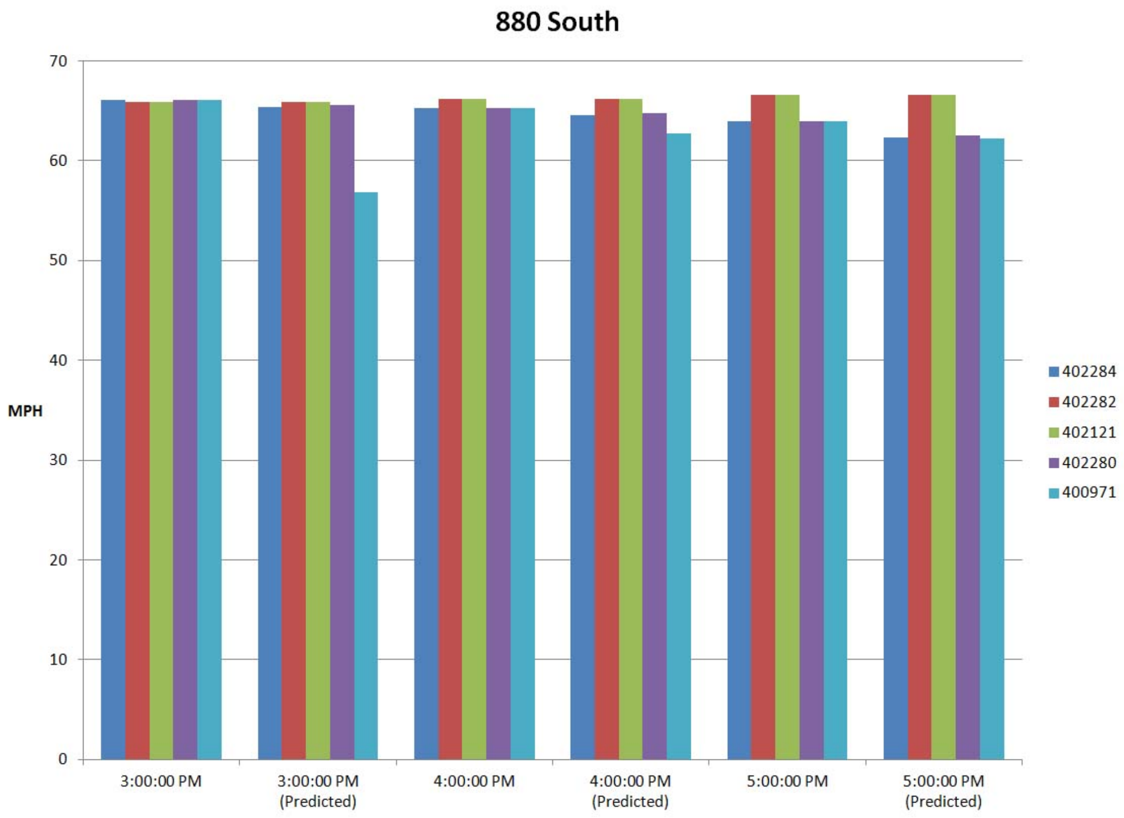


Figure A.4: Results for I-880 and US-101 on May 1, 2013.

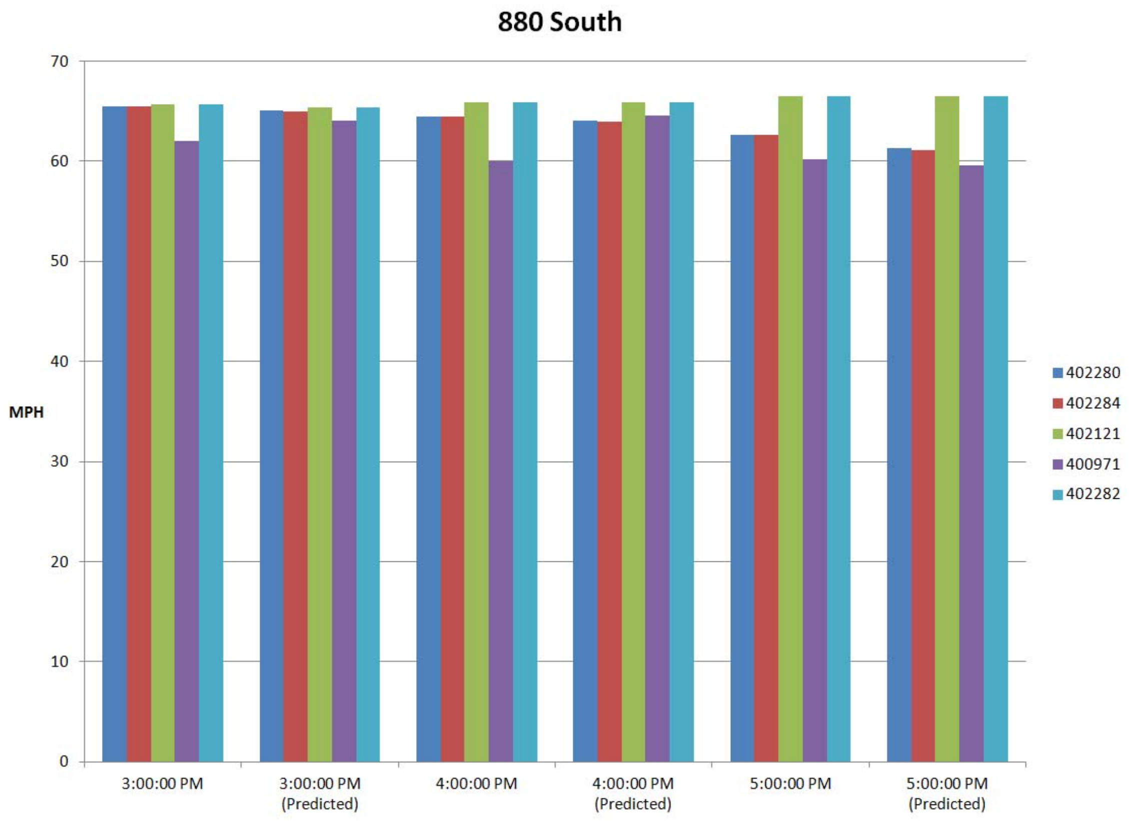


Figure A.5: Results for I-880 and US-101 on May 2, 2013.

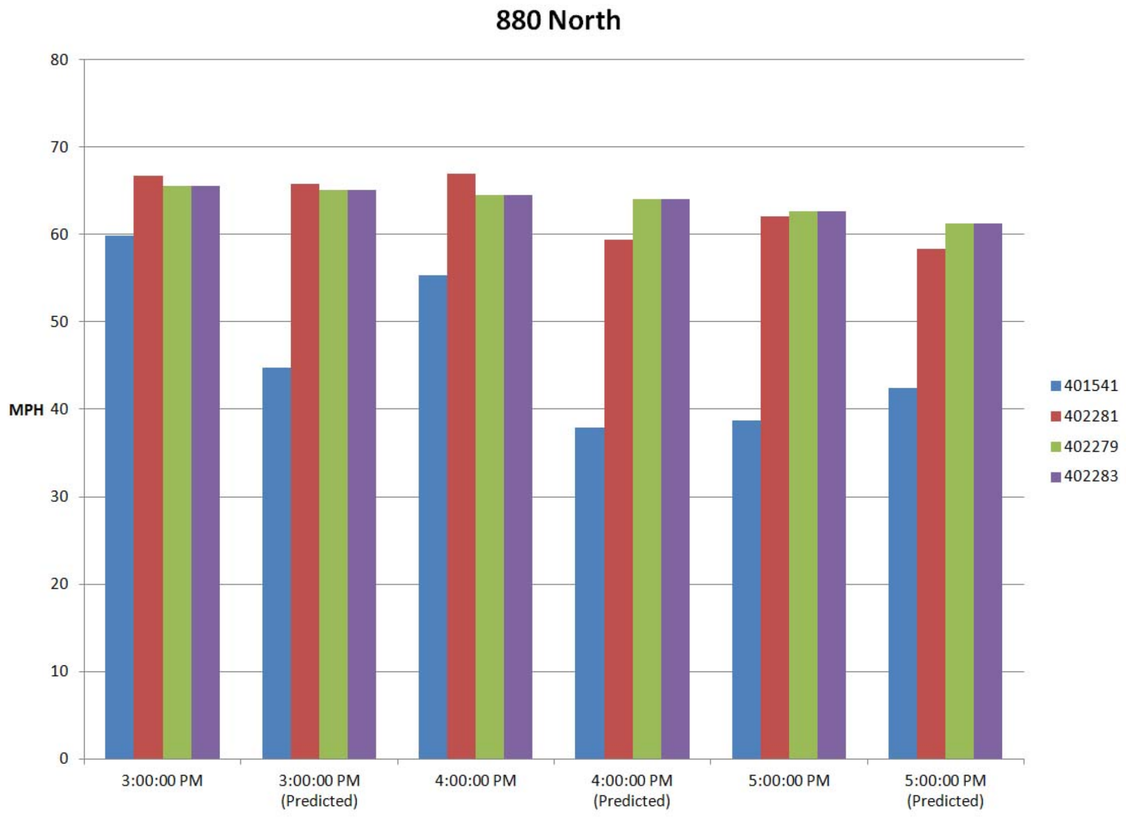


Figure A.6: Results for I-880 and US-101 on May 2, 2013.

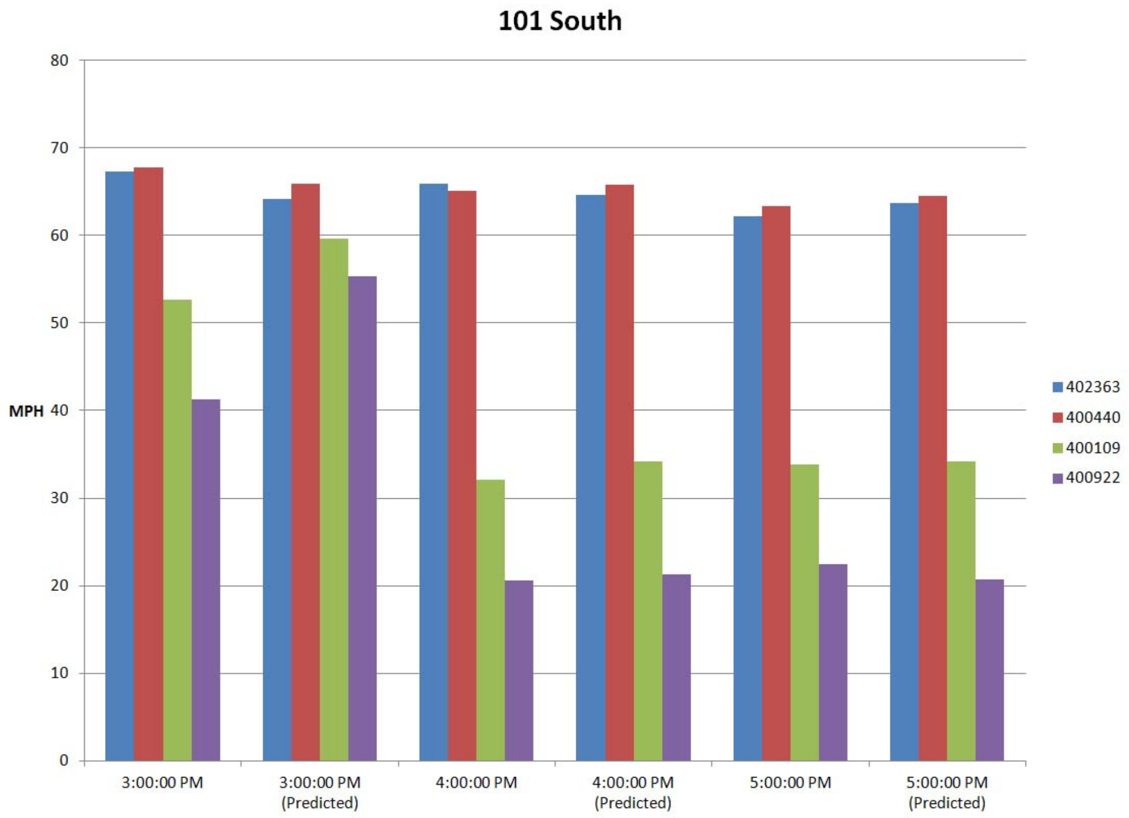


Figure A.7: Results for I-880 and US-101 on May 2, 2013.

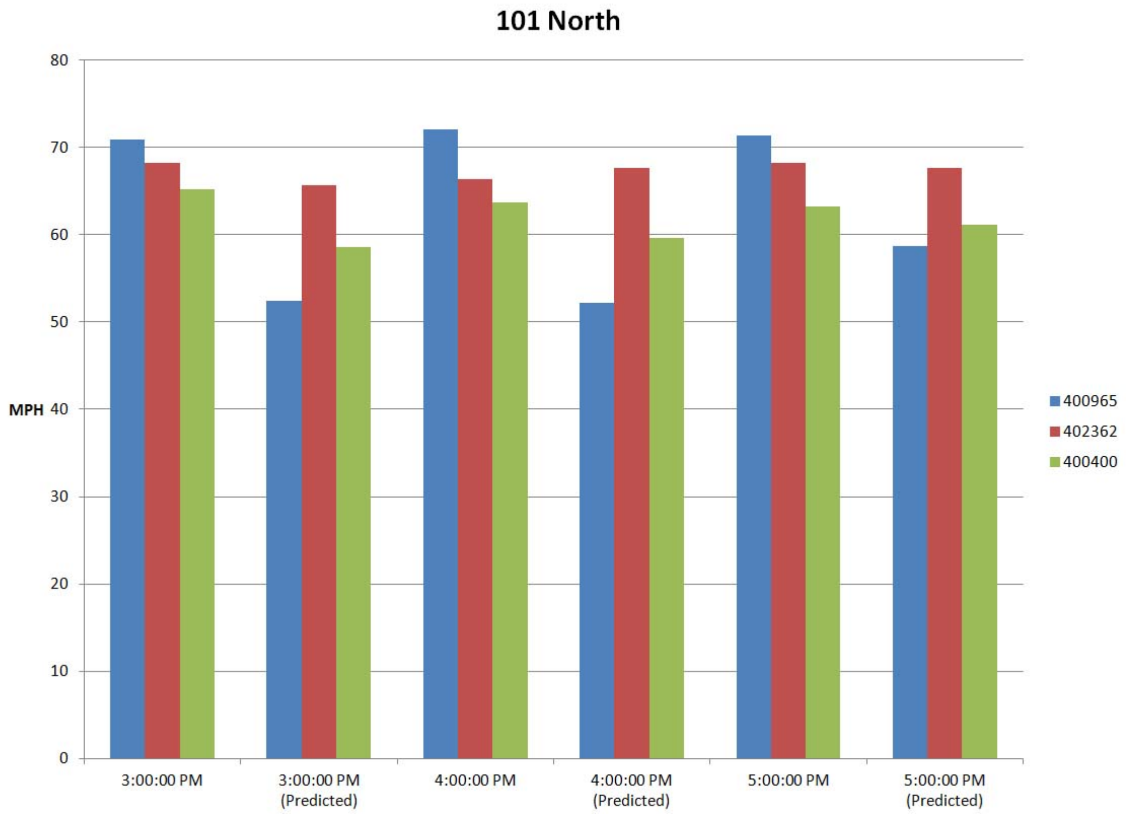


Figure A.8: Results for I-880 and US-101 on May 2, 2013.

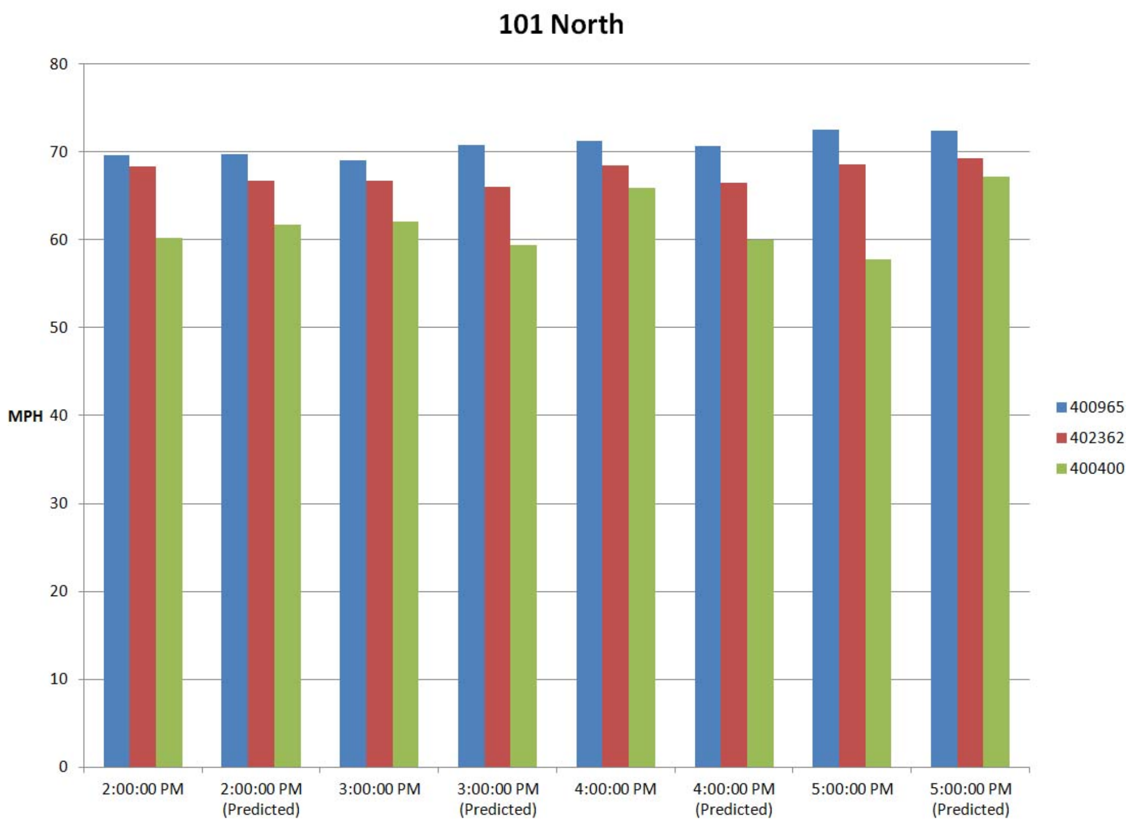


Figure A.9: Results for I-880 and US-101 on May 8, 2013.

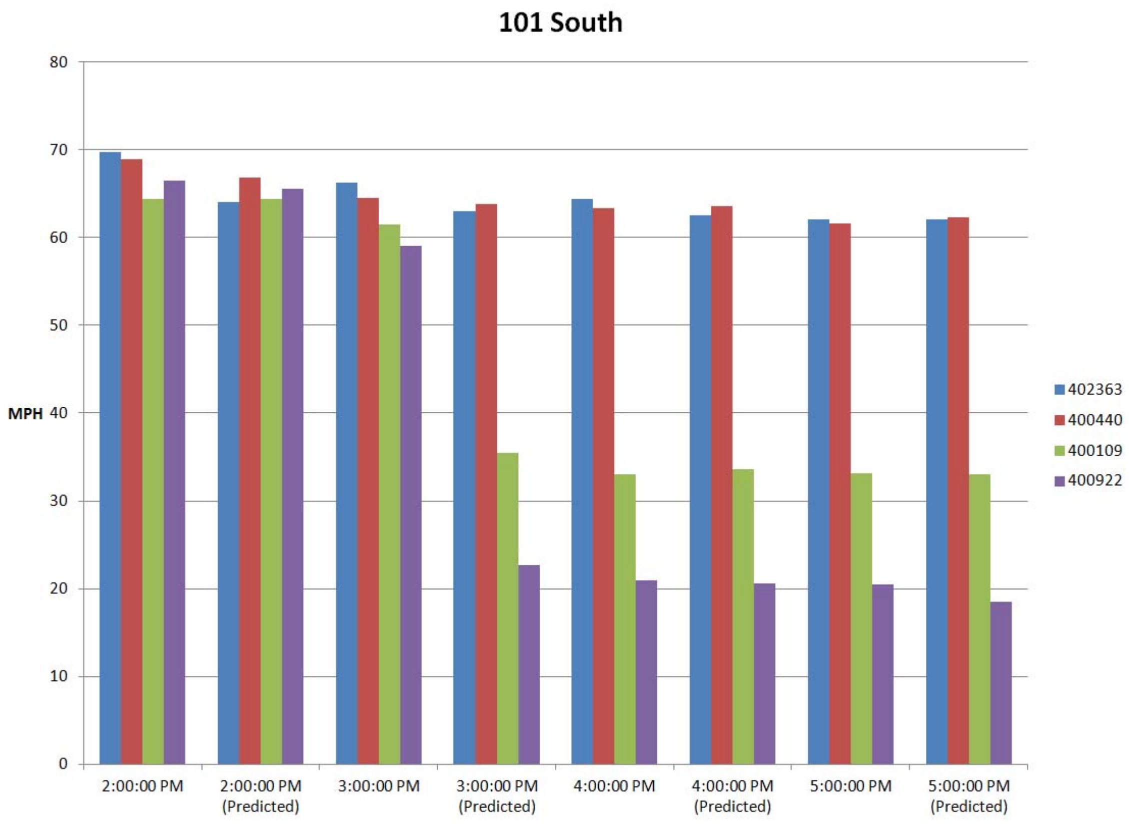


Figure A.10: Results for I-880 and US-101 on May 8, 2013.

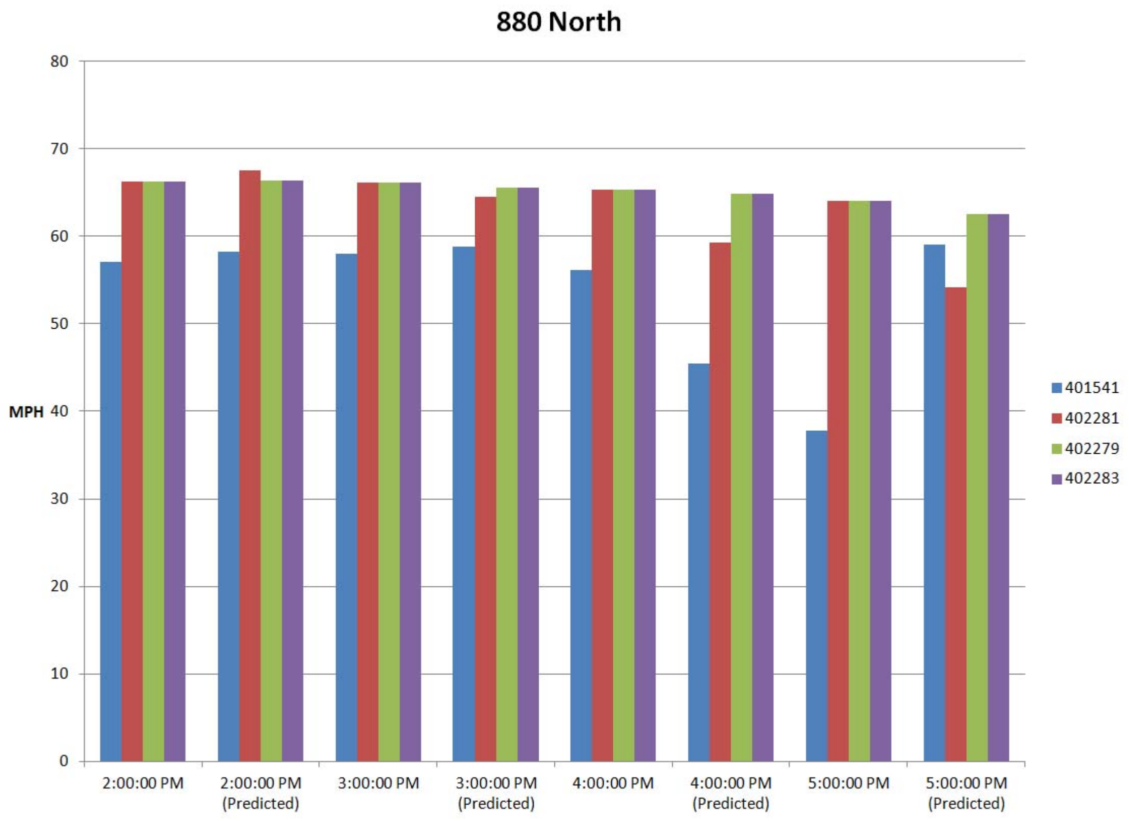


Figure A.11: Results for I-880 and US-101 on May 8, 2013.

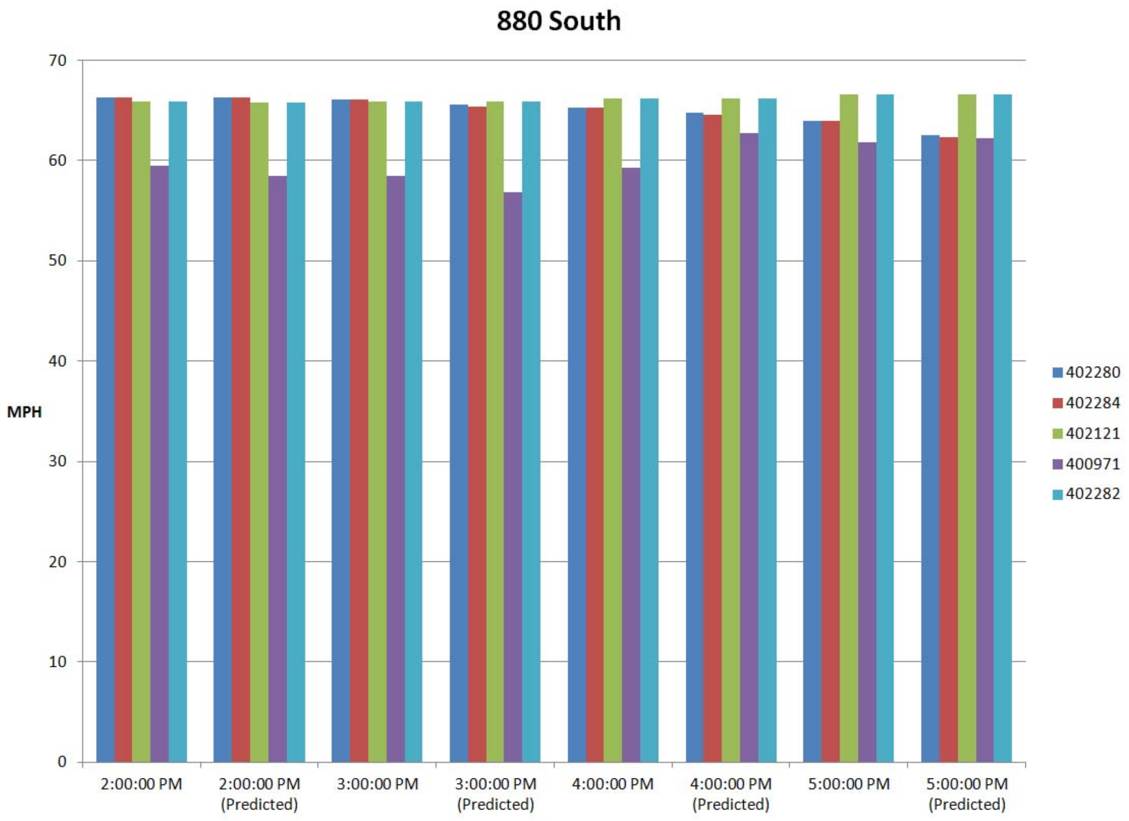


Figure A.12: Results for I-880 and US-101 on May 8, 2013.

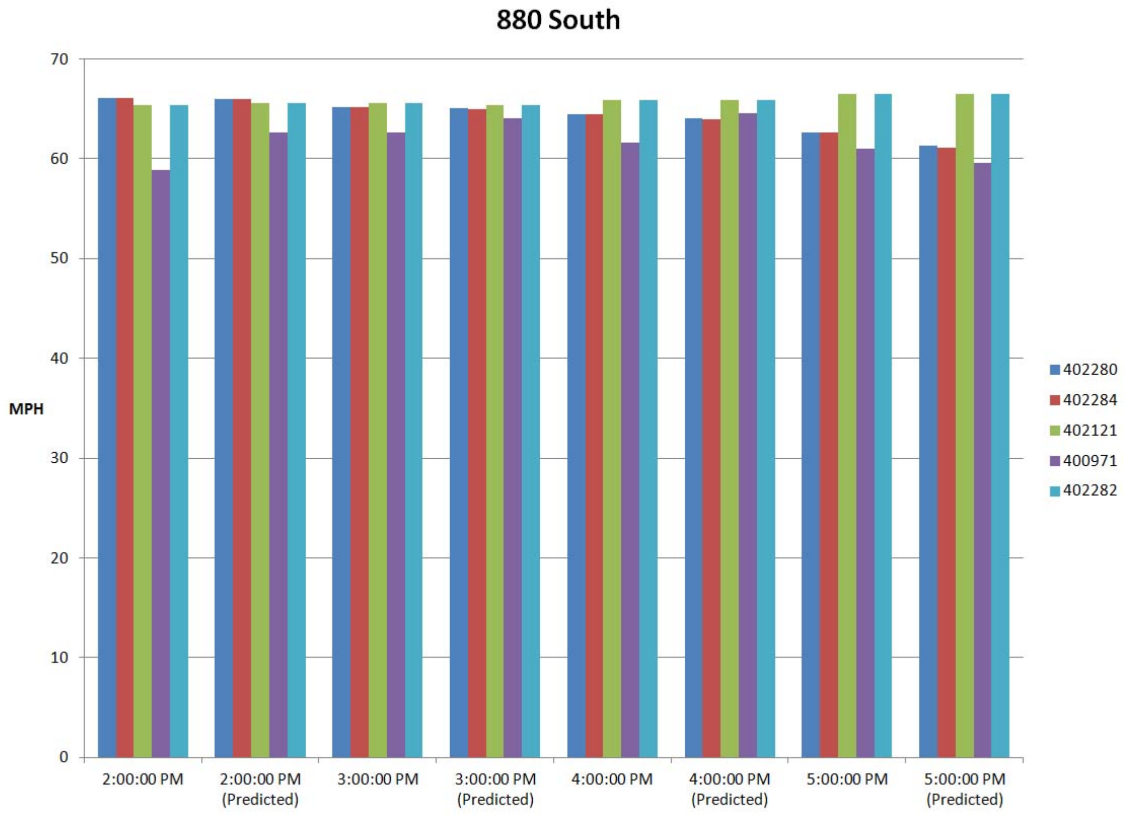


Figure A.13: Results for I-880 and US-101 on May 9, 2013.

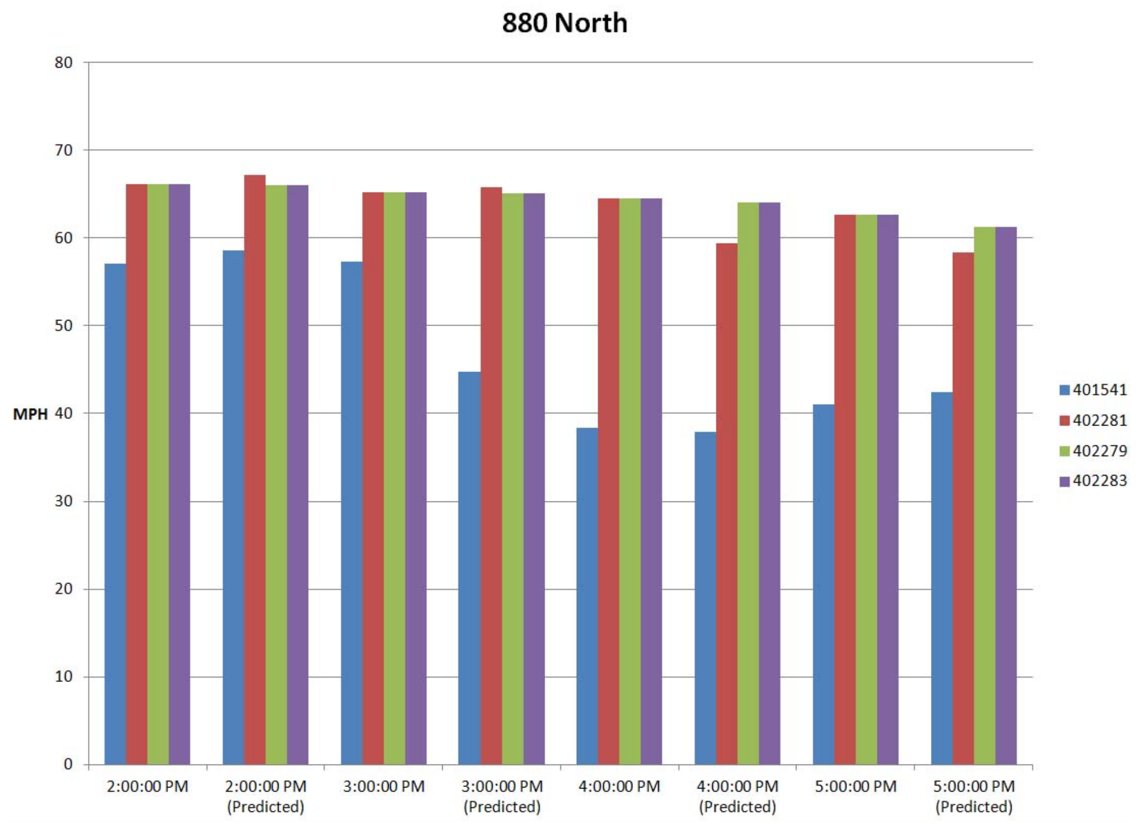


Figure A.14: Results for I-880 and US-101 on May 9, 2013.

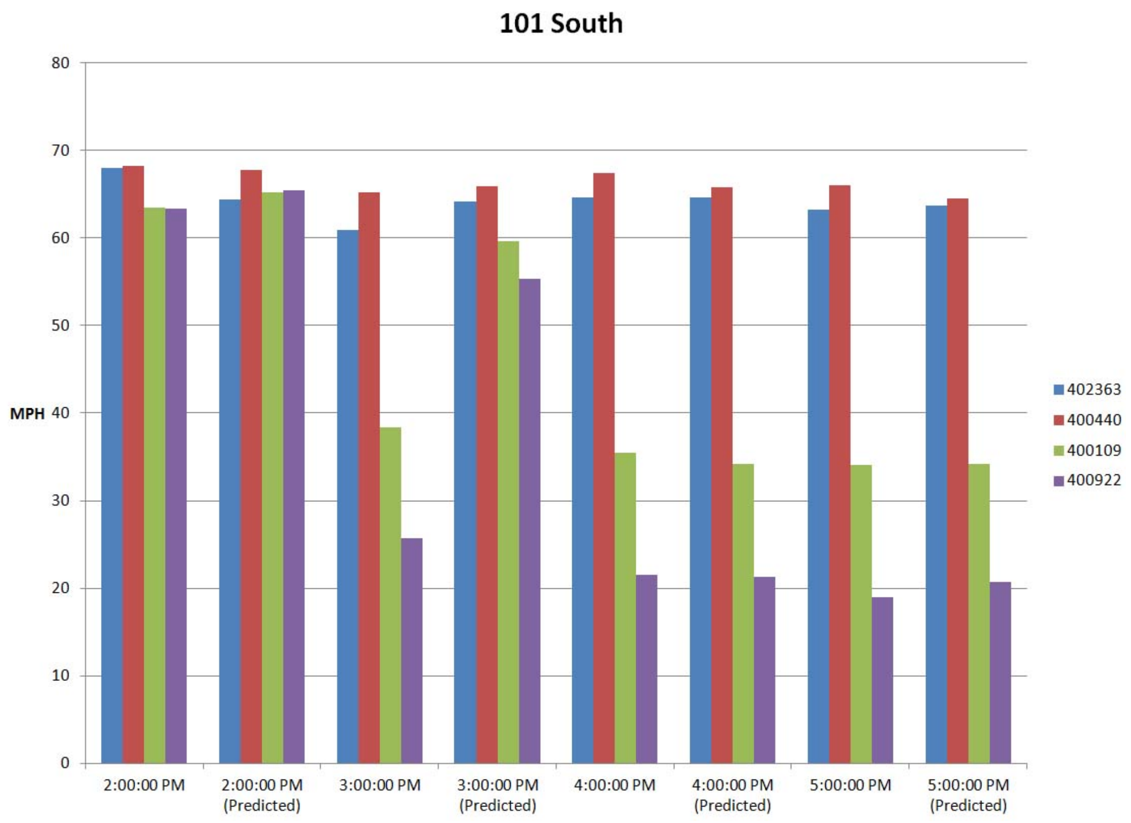


Figure A.15: Results for I-880 and US-101 on May 9, 2013.

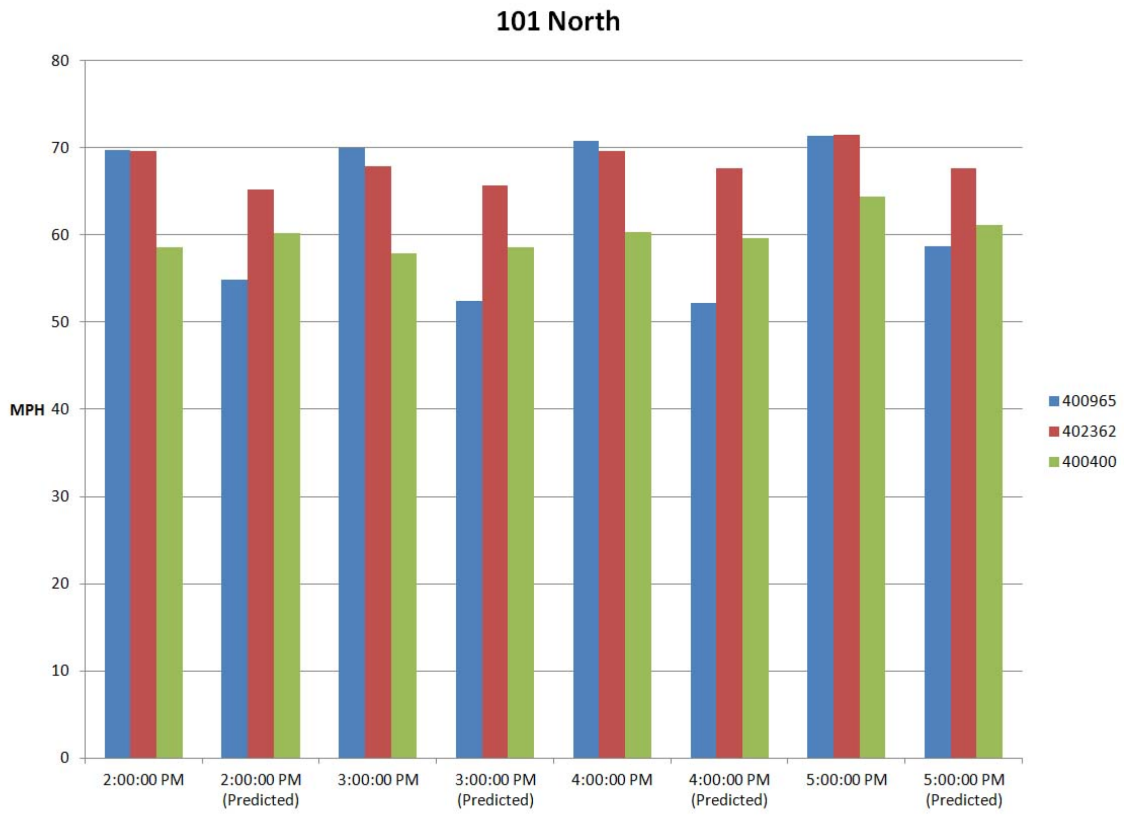


Figure A.16: Results for I-880 and US-101 on May 9, 2013.

Appendix A

Predictor Evaluation Results - Part 2

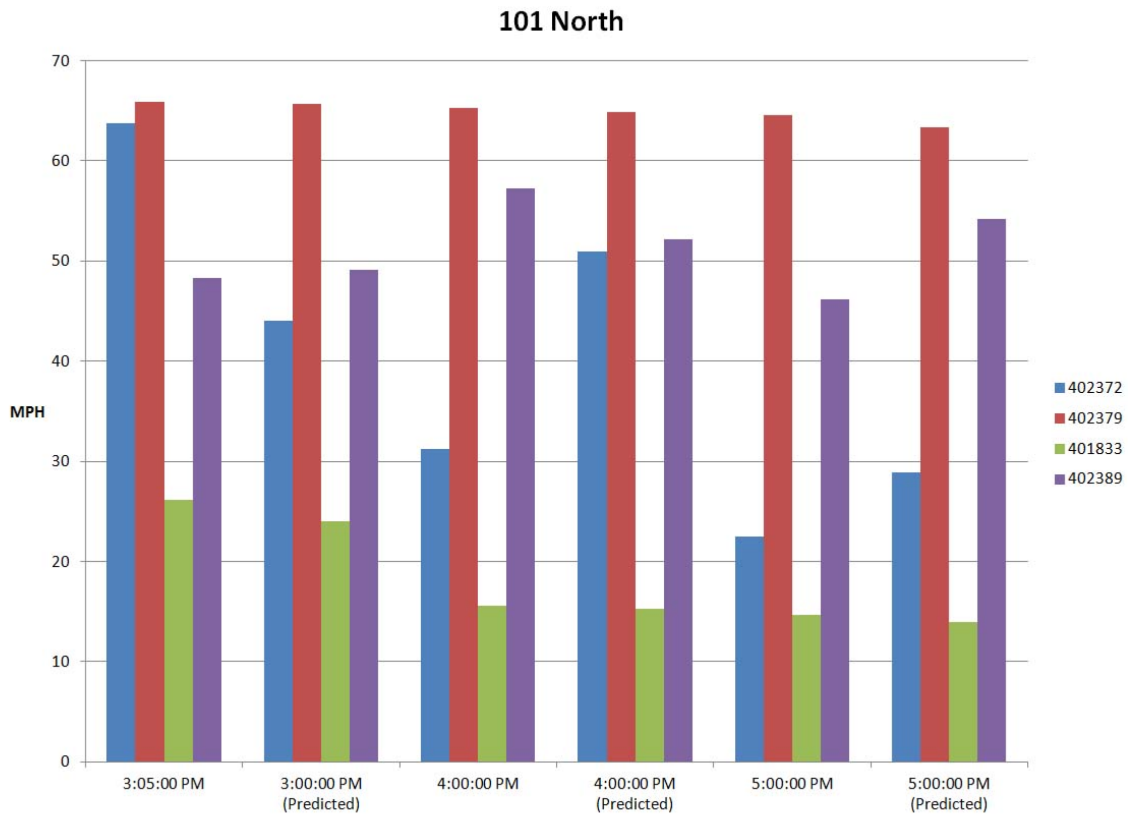


Figure A.1: Results for I-280 and US-101 on May 14, 2013.

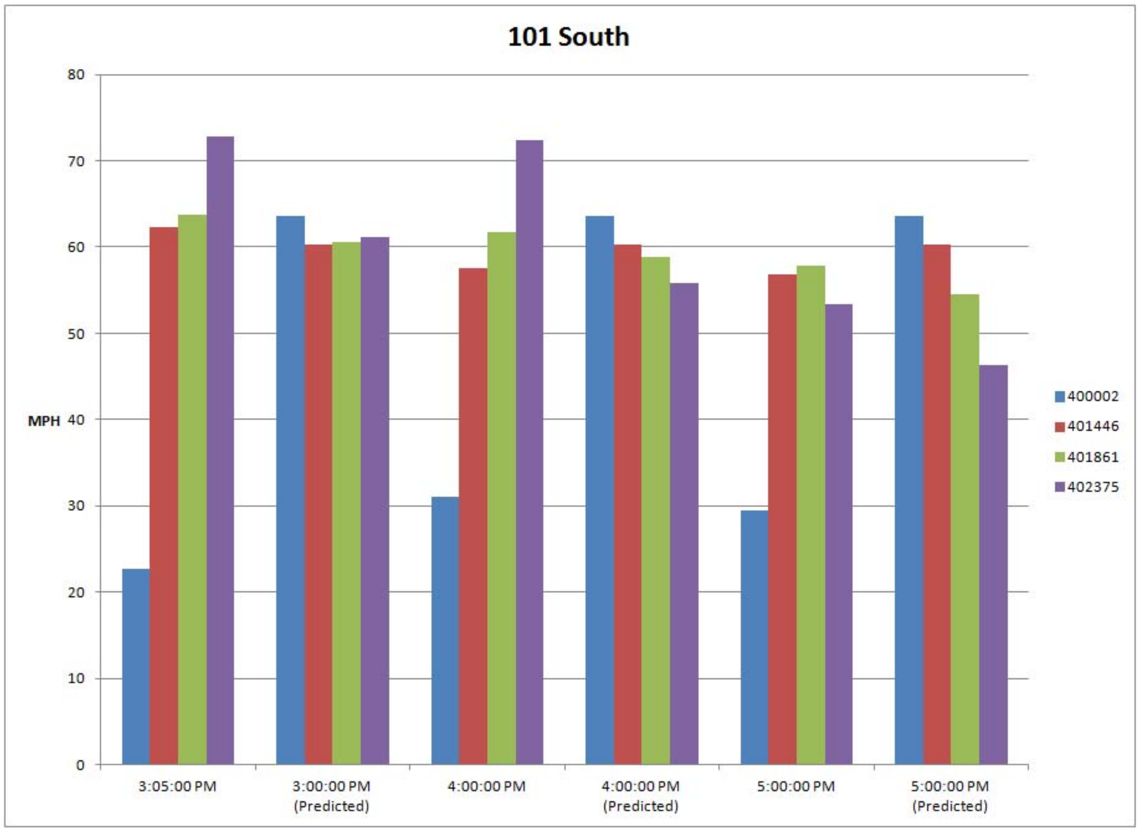


Figure A.2: Results for I-280 and US-101 on May 14, 2013.

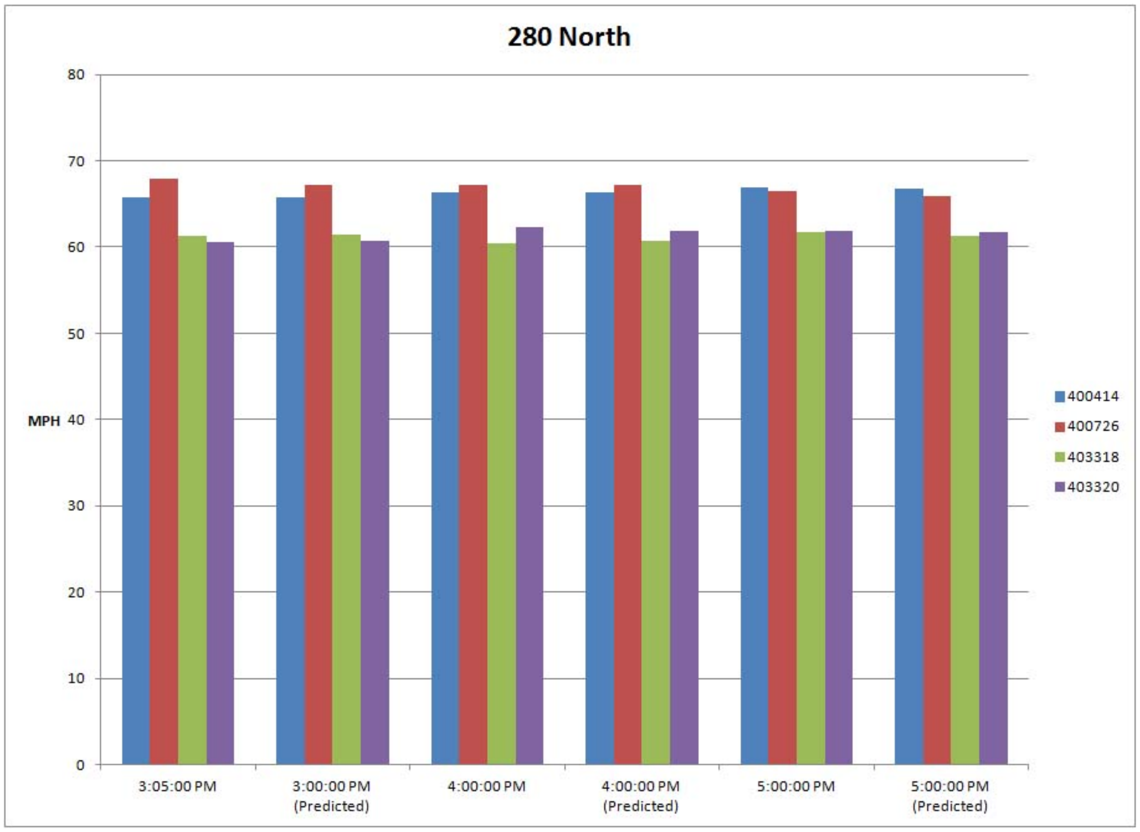


Figure A.3: Results for I-280 and US-101 on May 14, 2013.

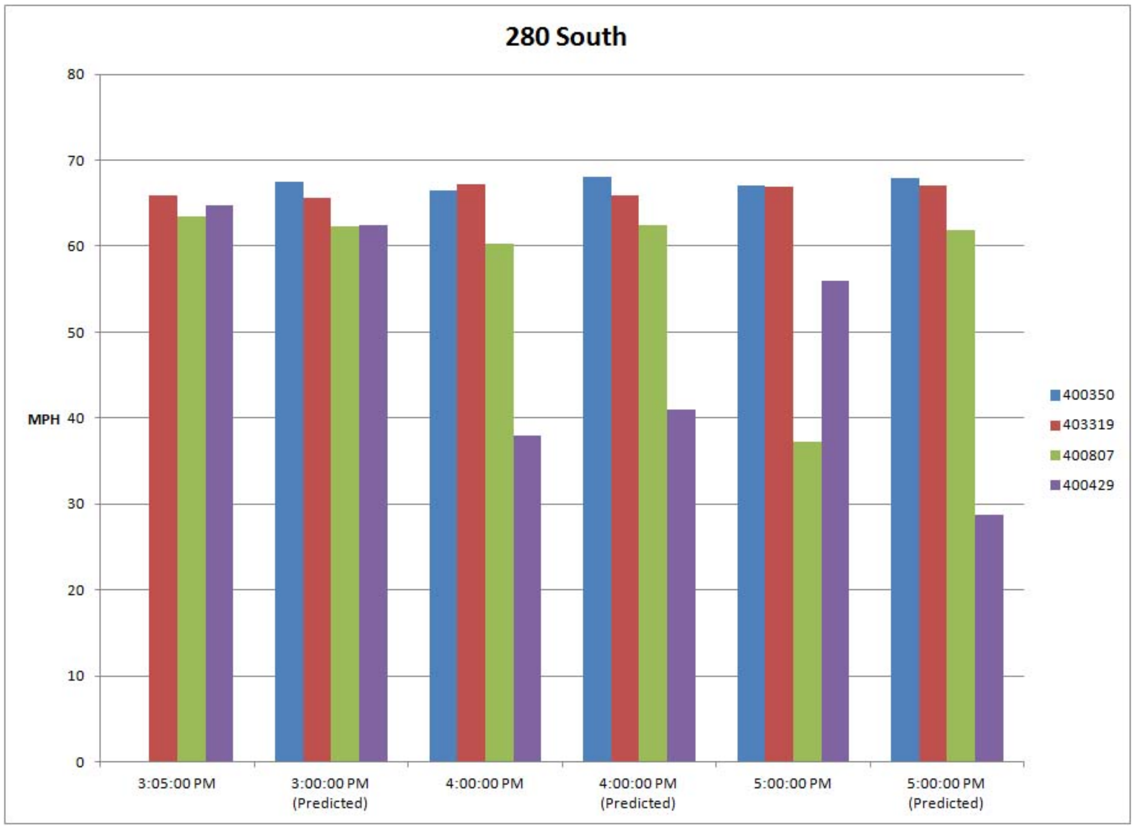


Figure A.4: Results for I-280 and US-101 on May 14, 2013.

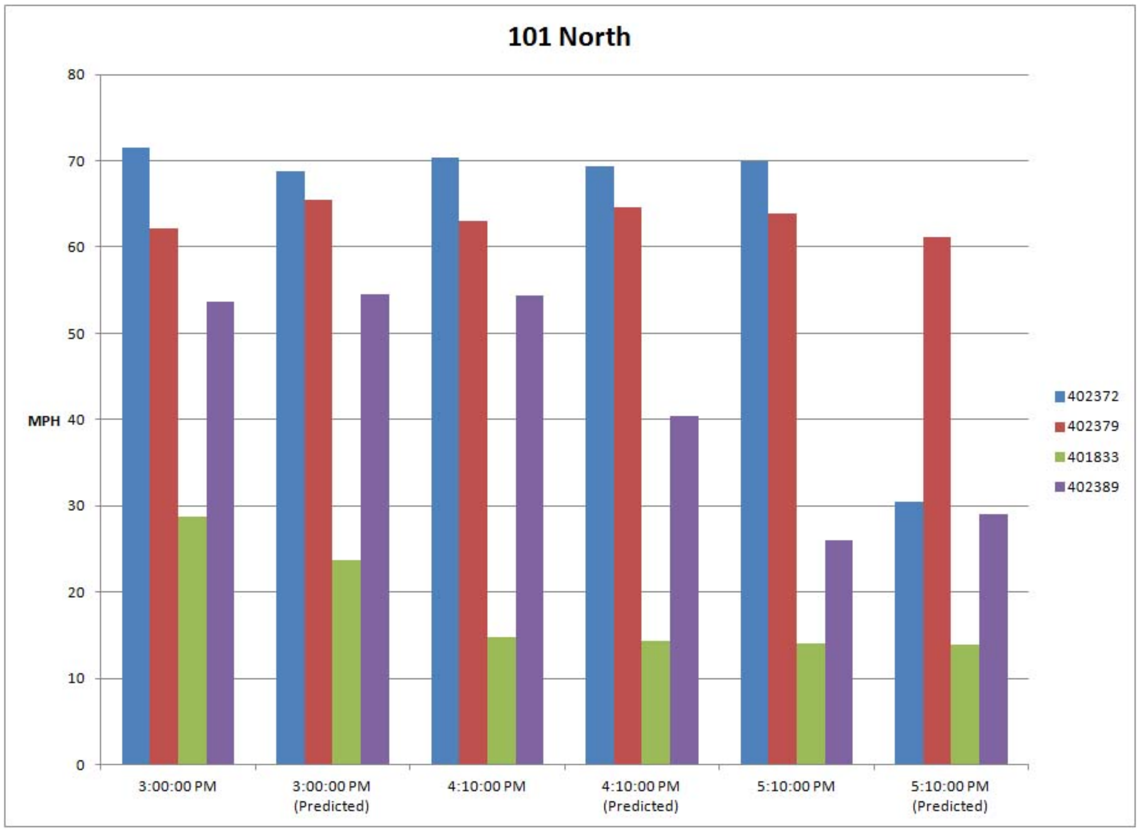


Figure A.5: Results for I-280 and US-101 on May 15, 2013.

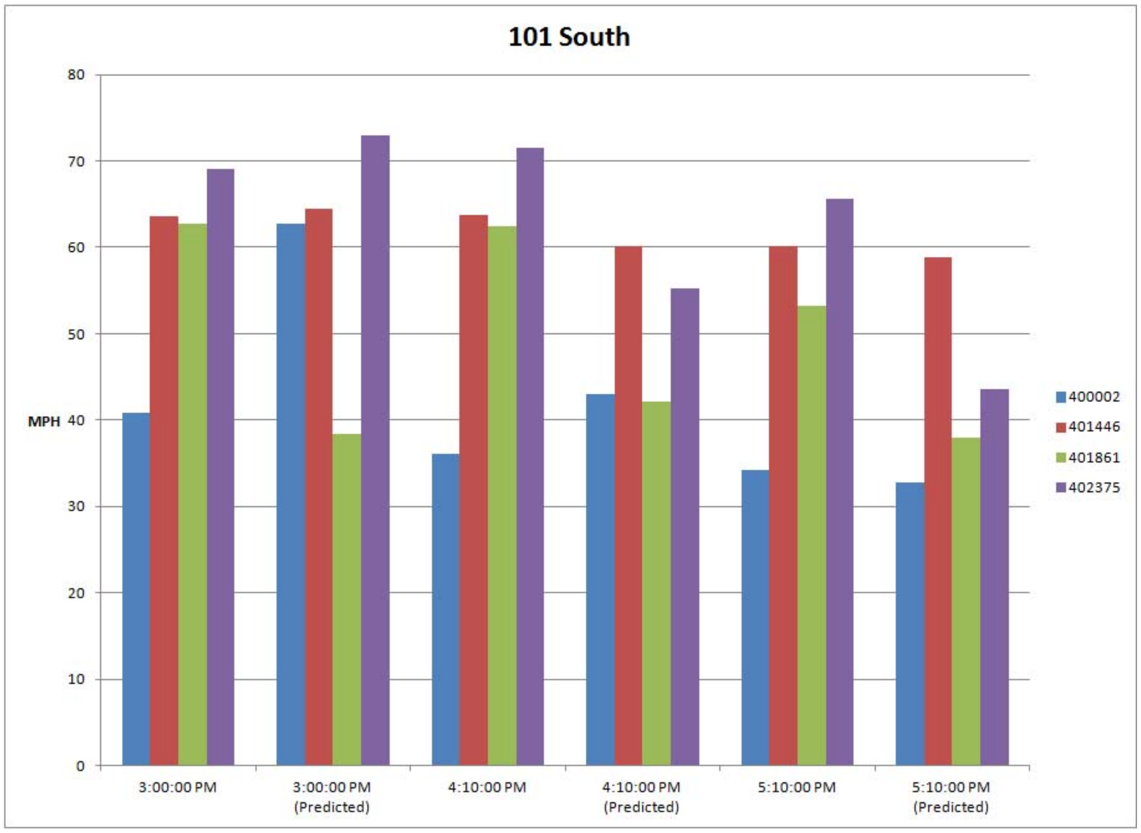


Figure A.6: Results for I-280 and US-101 on May 15, 2013.

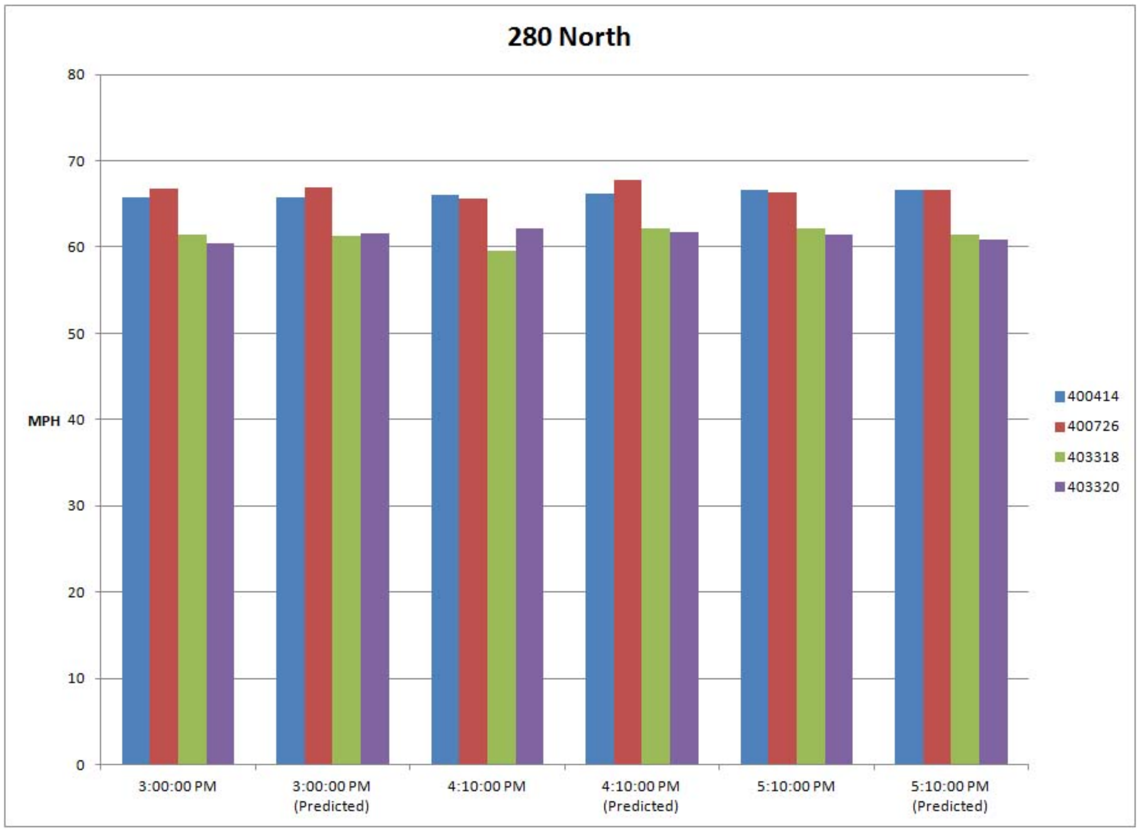


Figure A.7: Results for I-280 and US-101 on May 15, 2013.

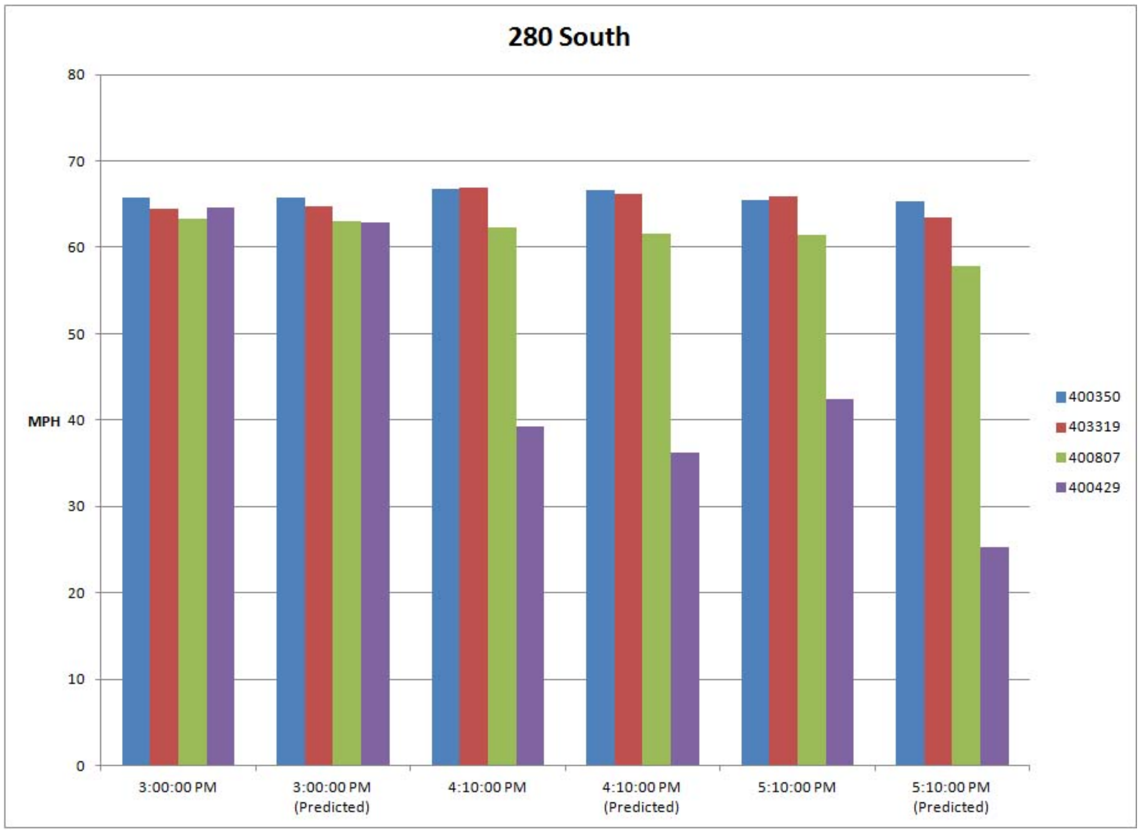


Figure A.8: Results for I-280 and US-101 on May 15, 2013.

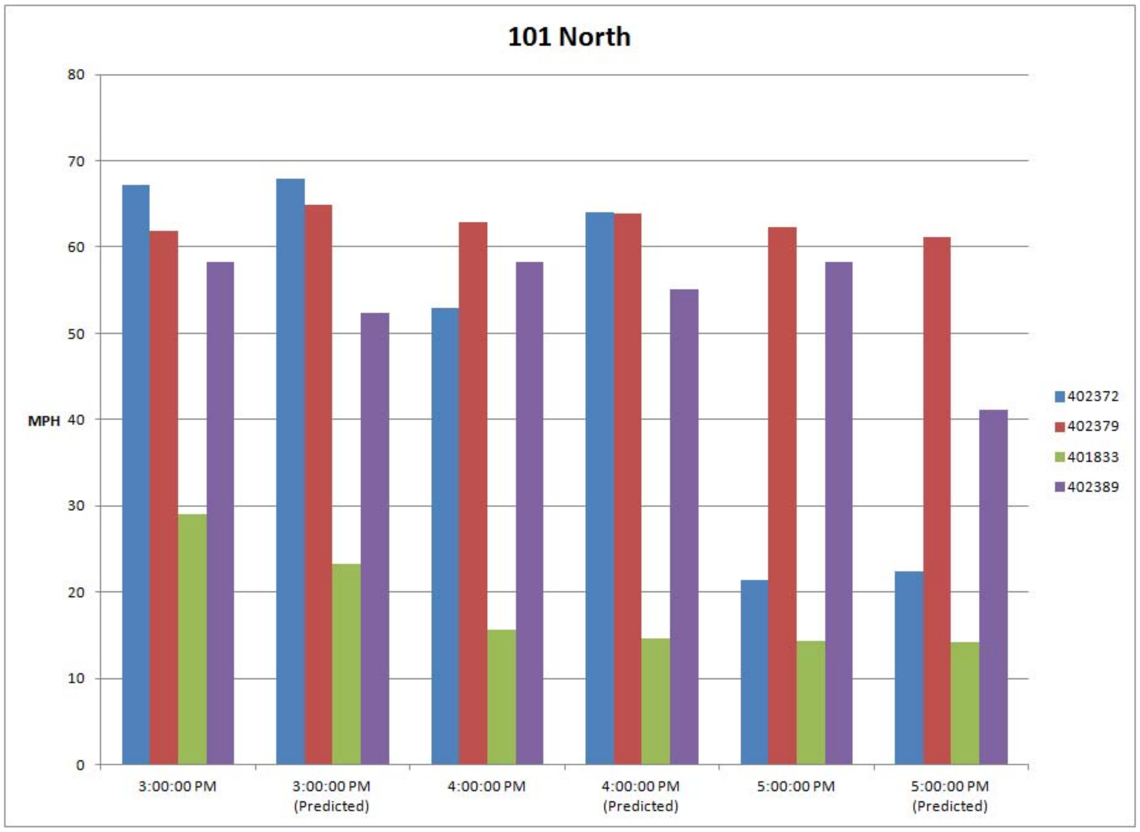


Figure A.9: Results for I-280 and US-101 on May 16, 2013.

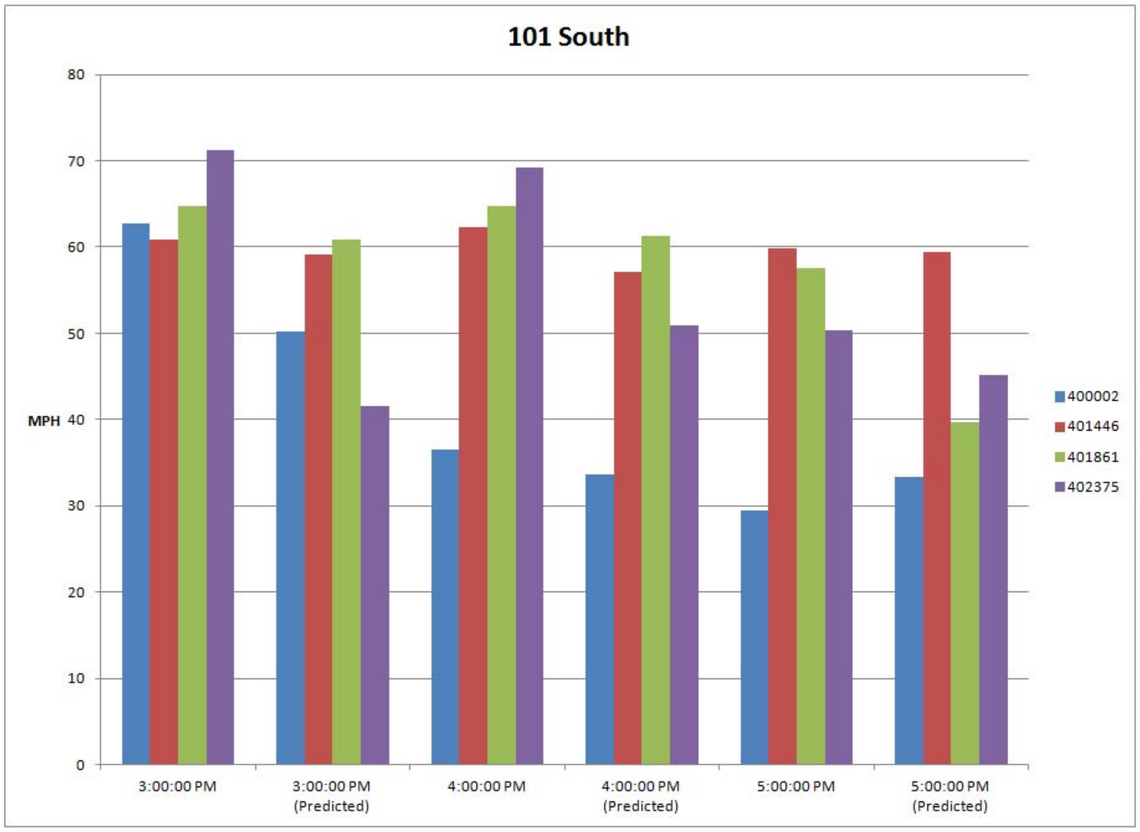


Figure A.10: Results for I-280 and US-101 on May 16, 2013.

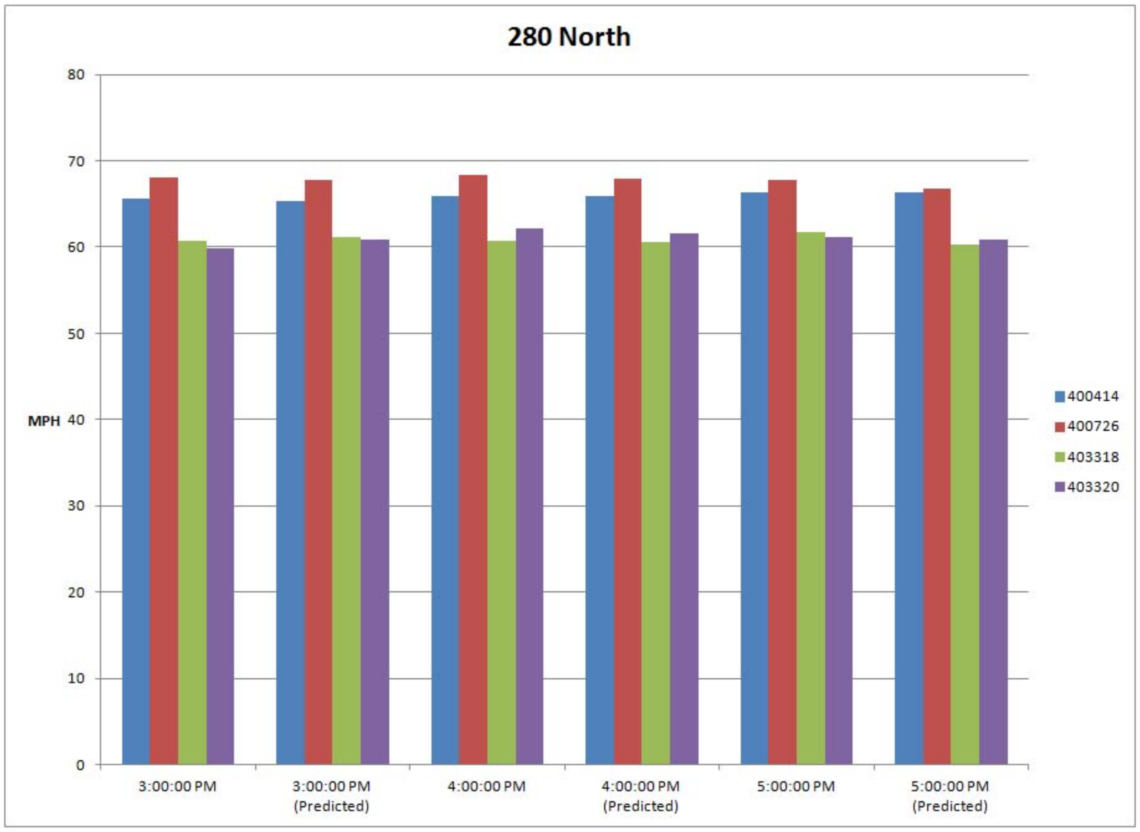


Figure A.11: Results for I-280 and US-101 on May 16, 2013.

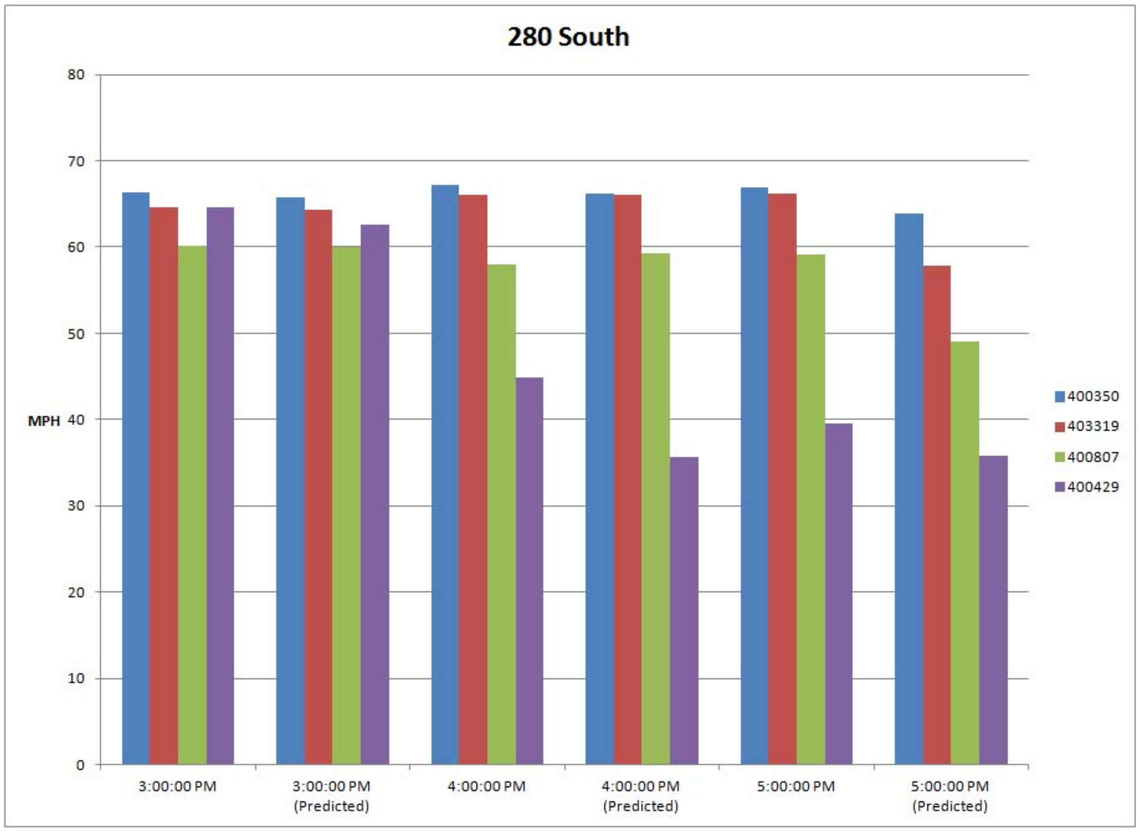


Figure A.12: Results for I-280 and US-101 on May 16, 2013.

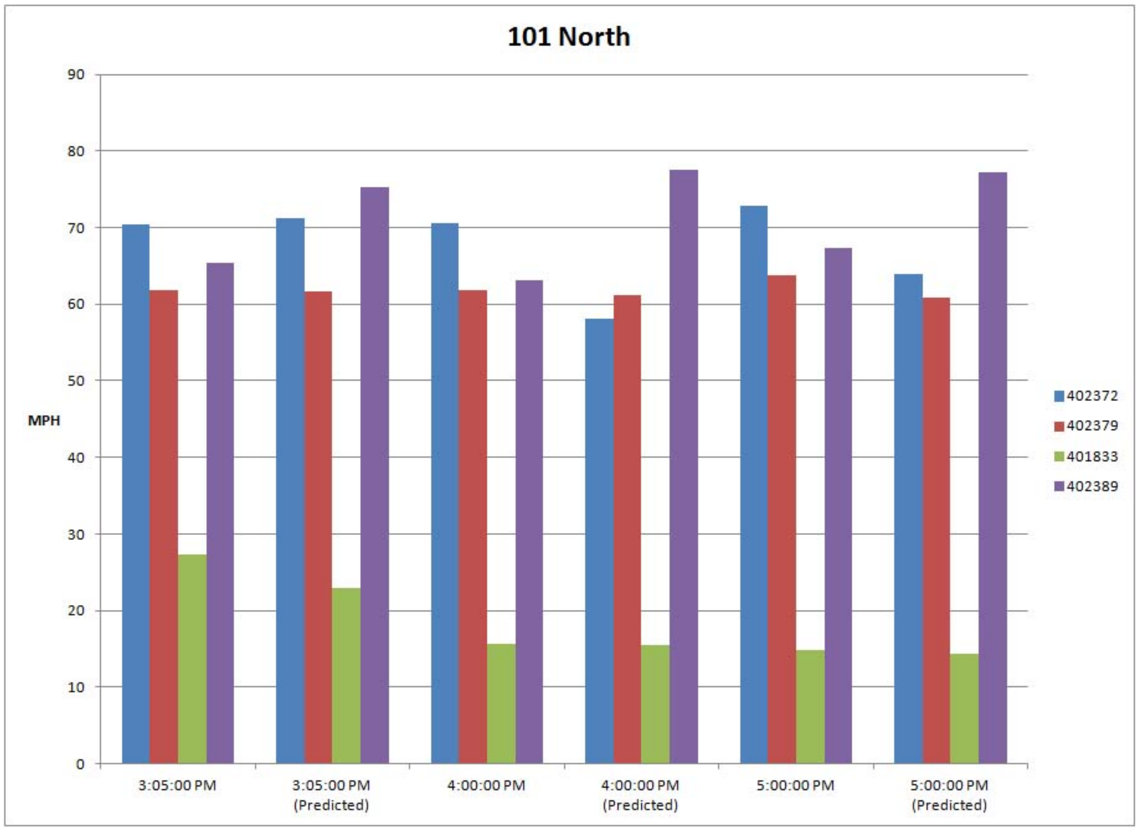


Figure A.13: Results for I-280 and US-101 on May 17, 2013.

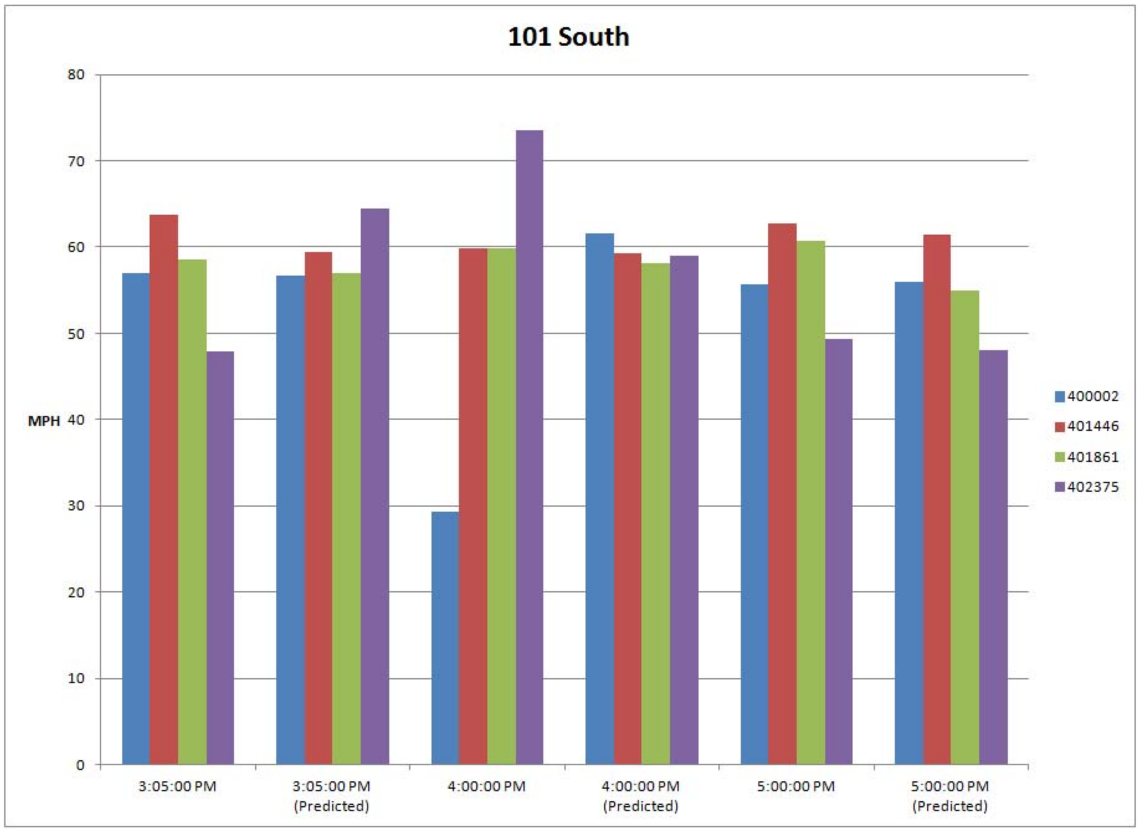


Figure A.14: Results for I-280 and US-101 on May 17, 2013.

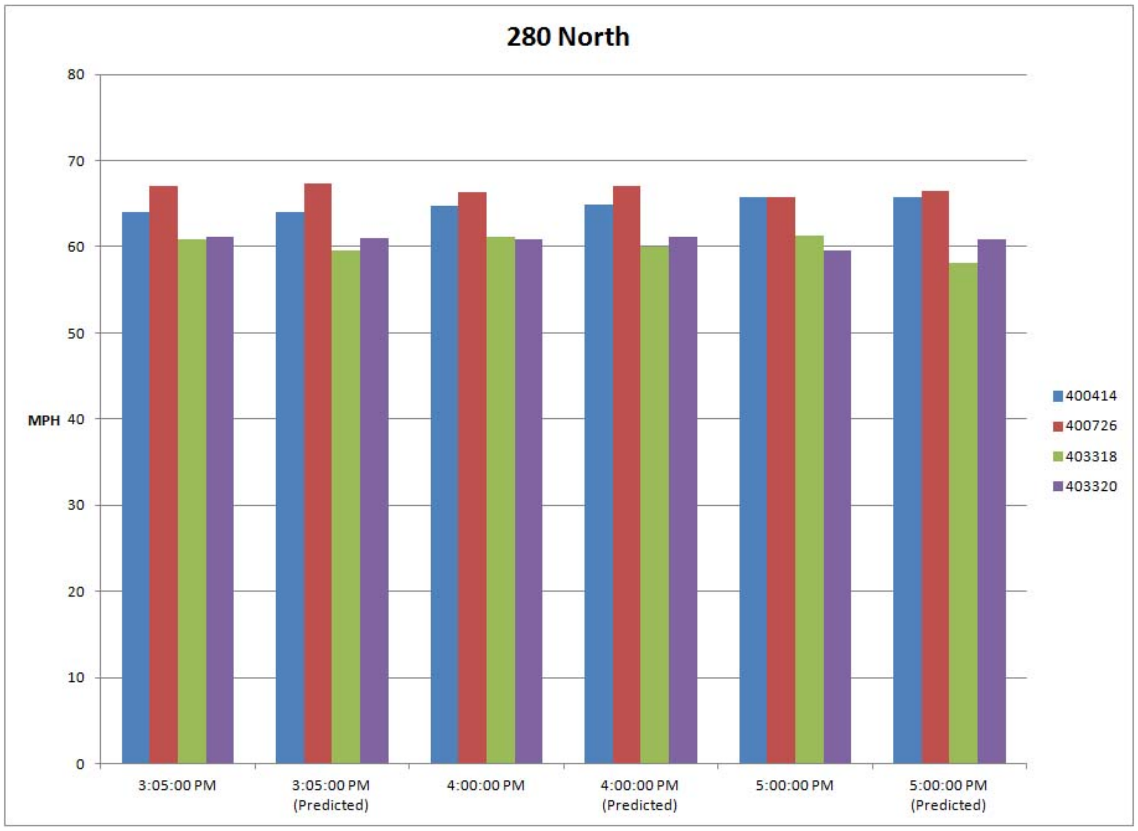


Figure A.15: Results for I-280 and US-101 on May 17, 2013.

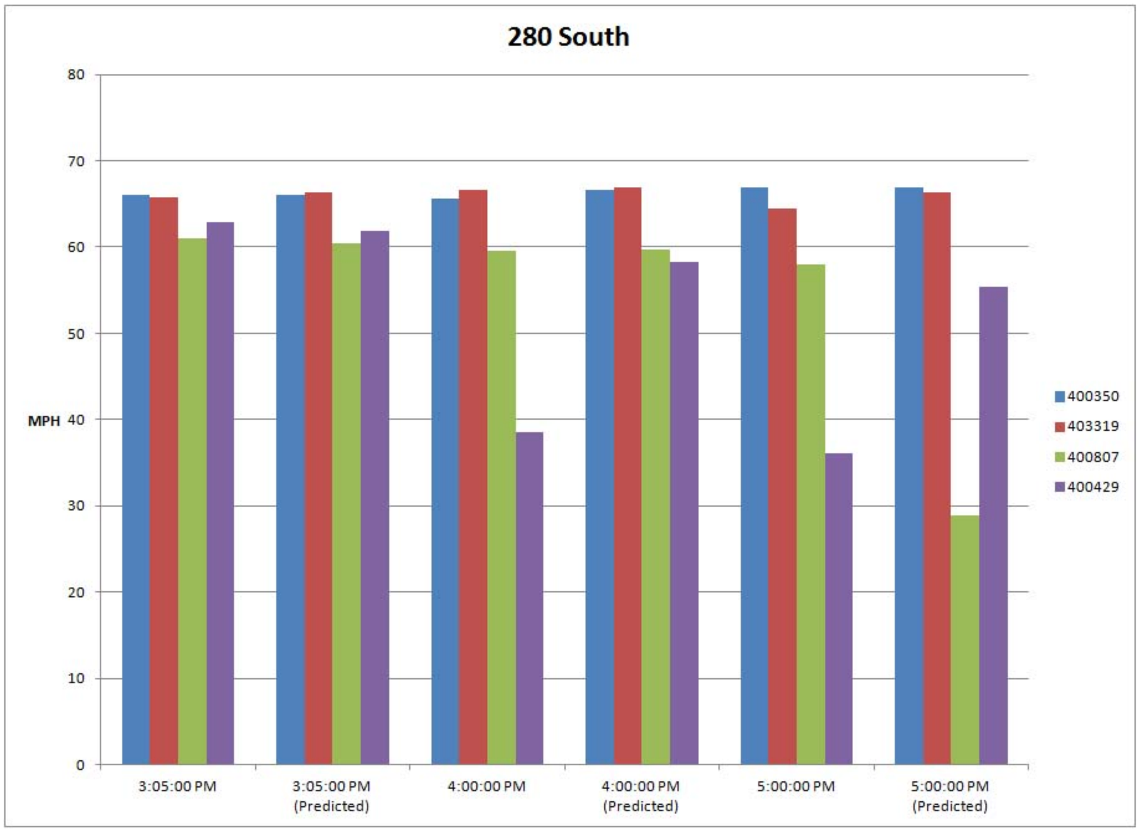


Figure A.16: Results for I-280 and US-101 on May 17, 2013.