ALGORITHMS AND DATA STRUCTURES FOR THE

REAL-TIME PROCESSING OF TRAFFIC DATA

by

JEFFREY MILLER

_____

A Dissertation Presented to the
FACULTY AND THE SCHOOL OF ENGINEERING
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2007

Table of Contents

List of Tables

List of Figures

Abstract

This thesis is focused on gathering traffic information in a distributed fashion through an application running in each vehicle rather than through expensive hardware that must be embedded in roads at discrete locations. This allows for a new approach to gathering traffic data over a continuous-flow rather than at discrete locations, as is the case with existing technologies. Loop detectors and video cameras, among other devices, currently provide the primary means for gathering data, but the data is gathered at the locations of these devices. Using mobile technology, the vehicles can transmit their speed and location to a central system, allowing the fastest path of each vehicle from its current location to its destination to be maintained in real-time. Gathering the speed and location of each vehicle in real-time over a continuous flow will allow more novel applications, such as incident identification and traffic prediction, to be developed.

I present an analysis of the bandwidth that would be required compared to the bandwidth that is currently available over cellular networks, from which I conclude that the bandwidth currently available is sufficient to run this application. I present and analyze data structures and algorithms that gather the speed and location of each vehicle and then can generate the fastest path for each in an efficient manner. I build on the work of static shortest path algorithms and dynamic shortest path algorithms to produce a new class of algorithms called the Dynamic All-Pairs All-Paths (APAP) algorithms. I analyze the algorithmic running time of these algorithms, and show that the Dynamic APAP class of algorithms runs faster theoretically and the Dynamic APAP – Constant Update algorithm runs faster practically than existing algorithms.

I have created FreeSim (http://www.freewaysimulator.com), which is a freeway traffic simulator for Intelligent Transportation Systems (ITS), that allows individual vehicles to communicate autonomously with the roadway (V2R) or other vehicles (V2V). FreeSim's architecture is unique in that it supports constant communication between all vehicles and a central server that is monitoring traffic and sending advice about preferred paths to vehicle's destinations. FreeSim provides a testing environment that supports both user-generated and live traffic as input. It also supports plug-and-play algorithms for computing fastest/shortest paths or any other graph or traffic algorithms. Running FreeSim with live loop detector data gathered by the Los Angeles division of the California Department of Transportation, I have produced graphs comparing the time to traverse the fastest path between two nodes, the difference in time to traverse a path based on live speeds and optimal speeds, and the relationship between speed and volume of vehicles passing a point within a certain amount of time, among other graphs. I conclude by presenting future research questions.

Chapter 1

Introduction

Assume that a person would like to travel by car from downtown Los Angeles to his home in the San Fernando Valley suburb at 5:00pm on a Wednesday. How would he decide which highways and streets to take? He could use a street map to determine the route that appears to be the shortest or use electronic maps to determine the actual shortest route. A navigation system in his vehicle could aid in turn-by-turn directions based on his current location. Radio stations could help in providing traffic and incident updates based on data they have received from third-party organizations, telephone calls from drivers, and reports by helicopters flying over the freeway system. He could even look on the Internet before departing to determine the traffic conditions at that moment, though this may (and most likely will) change by the time he travels along the route. As traffic conditions change, however, the driver will not know if his fastest route has changed, and thus may be traveling along a route that will take significantly longer than other routes. Suppose instead the traveler could have access to up-to-the-minute traffic information covering all possible routes he might follow to get home. Could such data be gathered in time to be useful? Could the data be analyzed in time to be useful? In this thesis, I consider how communicating real-time traffic data with vehicles via wireless, mobile devices can be efficiently processed and used to dynamically determine and adjust optimal traffic paths.

Traffic poses a major issue in many cities around the world. The amount of time to travel from one location to another can vary significantly based on the current traffic conditions. A recent study performed by Texas A&M in 2004 [1] concluded that traffic on freeways in the United States costs drivers an additional 3.5 billion hours each year.

The study also noted that Los Angeles is the worst city for traffic in the United States, where, in 2002, drivers wasted an average of an additional 93 hours in traffic [1].

Each day in Los Angeles, at any given time, there may be more than 1 million vehicles traversing the freeway system. On a popular freeway section, such as where Interstate 405 meets Interstate 10, an average of almost 600,000 vehicles travel by that intersection during the day [2]. Traffic jams, accidents, construction, etc. can all create congestion, and a driver is often faced with the question of whether to get off the freeway and try an alternate path or to continue to fight the freeway traffic. The California Department of Transportation (CalTrans) and other agencies are constantly studying ways to improve traffic flow, e.g. car pool lanes and traffic flow lights. In this thesis, I consider how one might apply the latest mobile communication technologies and efficient data processing to help alleviate the problem.

Transportation organizations have attempted to collect speed data on a macroscopic level by gathering traffic flow information. For example, CalTrans has hardware devices called loop detectors embedded in freeways to track the number of vehicles passing that point and the amount of time that a vehicle is physically occupying the space over the detector. This data is made publicly available, though there is at least a 15-minute delay and no data is provided for areas between the hardware sensors. In addition, the loop detectors fail frequently and are not always serviced properly, and as of November 2006, only 52% of the loop detectors in Caltrans District 7 (the district including Los Angeles and Ventura counties) were operating correctly [59]. The Performance Measurement System (PeMS) project at Berkeley [3] has created an algorithm to approximate the speed of the traffic at each of the loop detectors based on

estimating the length of a vehicle and using the occupancy data provided by the loop detector [4]. Sigalert.com [5], Yahoo Maps [7], and Google Maps [27] all provide a user-friendly interface to the CalTrans and PeMS data and allow a user to determine the speeds at each of the sensor locations for a route consisting of a set of freeways. In addition, Sigalert.com allows a user to view a map of a region (such as Los Angeles – http://www.sigalert.com/map.asp?Region=Greater+Los+Angeles), where the speeds on the freeways are grouped as dots indicating 1-14 mph, 15-34 mph, 35-54 mph, and greater than 55 mph (see Appendix A.1). It also shows severe, moderate, and minor accidents, and any additional information about the incident that may have been provided by the California Highway Patrol. This data can be very useful, though there is a delay of approximately 15 minutes, if not more, because of communication with CalTrans and the CHP (http://www.sigalert.com/faq.asp). Furthermore, Sigalert.com only provides information about the general flow of traffic on the different freeways, but not about specific vehicles or specific lanes. Looking at the data from a microscopic point-of-view will open the door to much more traffic analysis that is not currently possible.

Some companies have created web sites that enable a user to map a route from a source address to a destination address. Using a map of the roads, Mapquest [6], Yahoo Maps [7], and Google Maps [27], among others, have created applications to provide a user with turn-by-turn directions, exact distance to travel, and an estimate of how long it will take, assuming the user is driving the speed limit the entire way. Mapquest also provides alternate routes, such as avoiding highways or the shortest overall distance. In all of these applications, it is nearly impossible to determine the exact amount of time it will take for a user to travel from his source location to his destination location because

the real-time traffic conditions are not taken into account. Practically speaking, a route that is the shortest distance from one point to another does not mean that it will be the fastest, and a route that is the fastest if driving the speed limit does not mean that it will be the fastest under certain traffic conditions.

In this thesis, I will attempt to design, simulate, and analyze a system that is aware of the speed and location of all vehicles in a freeway system at any given time. Using this information, I will develop algorithms that determine the fastest path from a vehicle's current location to its desired destination and identify traffic bottlenecks that affect traffic flow. Assuming that vehicles have the ability to continuously transmit speed and location data, I will first analyze the data transfer requirements of this system. I will then develop several algorithms for processing this data, and finally, using system simulation, I will analyze live traffic data and establish the feasibility of such a system.

Chapter 2

Related Work

There has been significant work done on monitoring traffic, both academically and professionally. Many transportation organizations use loop detectors and video cameras to monitor traffic flow and report the gathered information via some form of media (i.e. TV, radio, internet, etc.) [4]. More recent work academically has focused on using sensor networks to gather traffic data, since sensors are much less costly than the hardware-intensive loop detectors and can be installed virtually anywhere with very little overhead [18, 19]. However, the sensors are autonomous entities that run on a limited energy source, so as the batteries die, there will be a cost associated with repairing certain nodes. In addition, they still must be installed in the roadways.

Recently, there has been some research into traffic flow on highways [25], and traffic flow utilizing a mixed sequence of automated and manually driven vehicles [26]. It has been predicted that vehicles that drive themselves on the highways will increase the throughput and help aid in the traffic problems, though this will not be an overnight change, so considerations have to be taken as to mixed-mode operation of automated and manually driven vehicles. Algorithms taking advantage of the data that we currently have available from loop detectors include trying to find the fastest path [49] and detecting delay-inducing freeway incidents by finding the speeds at two neighboring loop detectors [50].

Some work has also been done in determining how much data collected from a freeway system is actually necessary to accurately route vehicles to reduce travel time [28]. If the amount of data can be reduced, then the actual running times of the algorithms will be reduced.

Some proposed architectures for vehicle communication include vehicle-to-vehicle (V2V) and vehicle-to-roadway (V2R) communication [31, 32]. Using ad-hoc networking for vehicle-to-vehicle communication reduces the amount of access point or base station hardware that is required to be installed along the side of a road. However, the delay and reliability of the network are affected by the lack of constant connectivity. Another architecture that has been suggested makes use of the peer-to-peer technology for transmitting data to vehicles [33]. With this model, each vehicle would transmit traffic information that it knows to all of the other nearby vehicles, and each vehicle maintains a graph of the entire network so that decisions about which path to take are made locally. With the extremely large amount of data that comprises a freeway system, however, this architecture may not be practical. On a related note to that research is that a peer-to-peer network may allow for constant bit rates to be transmitted to vehicles without placing a significant load on base stations [34]. This idea is proposed to allow for browsing the web and checking email, though privacy issues will arise if this data is being transmitted through other vehicles before reaching the desired host.

There is a private company called Airsage [20] that is attempting to gather traffic data via cellular phones, though they are not leveraging any Global Positioning System technology because of overhead and privacy issues. Airsage determines the speed and location of a user by how he moves from one cellular tower's zone to another. Unfortunately, this approach will not work for densely populated areas where there are a significant number of vehicles on surface streets that are immediately adjacent to highways, for the Airsage application does not know whether the cellular phone is on the highway or the surface street.

As far as gathering data from a Global Information System (GIS) using a cellular phone, a bill was passed in 1996 by the United States Government popularly known as E911. This bill requires cellular companies to be able to provide location information of cellular phones to within 50-300 meters of the exact location. The purpose of this bill is if a cellular user calls 911 and does not know his precise location. This idea has been extended to determine if this data can be used for generating traffic data [35, 36, 37]. Obviously the result that follows explains that as the GIS data becomes more accurate, so does the accuracy of the traffic data provided. Sprint [60] now offers phones with the capabilities of navigation systems to provide a driver with turn-by-turn directions on his cellular phone, which shows the current accuracy of GIS data within the Sprint cellular network.

Assuming that the data at discrete locations can be obtained (which is currently done by many transportation organizations), there has been some work done in estimating the amount of time that it takes to travel along a specific path [21]. The Berkeley PeMS project has even gone so far as to attempt to predict travel time based on historical data [22]. However, most of this analysis is based on only obtaining traffic data at specific points, as is the case with loop detectors [23, 24].

From the collection of loop detector data, algorithms for shortest paths can be used to determine the fastest path to get from one location to another. Shortest path algorithms in a graph have been studied for many years, with popular algorithms being developed by Floyd [9, Appendix B.1] and Dijkstra [11, Appendix B.2], and Bellman-Ford [29, 30, Appendix B.3]. A shortest path algorithm in a transportation model was provided in [41], though without considering the dynamic nature of traffic flow, the

actual running time is greatly affected. An incremental approach for the single source shortest path problem was considered in [40], and a parallel algorithm for determining the single source shortest path between multiple nodes simultaneously was considered in [39]. Since considering only a single source is not sufficient for our application, the all-pairs shortest path algorithm is more suitable [42, 43]. Johnson has a famous algorithm for determining the all-pairs shortest paths [10]. Optimizing these algorithms down to sub-cubic running times can be found in [44].

Moving more toward the dynamic behavior of a graph, the all-pairs shortest path problem can be applied to a graph that has changing weights and edge insertions and deletions [14]. A good description of the dynamic all-pairs shortest path problem is provided in [45, 46]. An approximation algorithm for the dynamic all-pairs shortest path problem is provided in [47] and for digraphs is provided in [15]. Building on those algorithms, faster dynamic all-pairs shortest path algorithms are provided by Dijdjev [48] and Demetrescu and Italiano [13], who also have recently given a practical running time analysis of the algorithms in [61].

Simulations also provide a large part of transportation research, for the algorithms mentioned above cannot be tested or accepted on a large scale if it cannot be proven that they are accurate. Computer applications for urban transportation planning were formally proposed in 1967 [51], and since then, many simulators have been developed. In [53], the authors discuss additional micro-attributes that should be included in a simulator, such as intermittent lane changes, driver aggressiveness, and driver alertness, among other things. In the airline industry, a division of American Airlines has created a simulator for baggage tracking, passenger monitoring, monorails, etc. for airport traffic

[58]. For freight traffic, dispatchers are able to relay traffic information to its drivers using DyFTS [55]. The manner in which the traffic data is gathered is not discussed though. As far as highway traffic is concerned, THOREAU [57] and a JINI simulator [37] have the ability to transmit traffic data to mobile devices in the individual vehicles, but nothing is discussed about gathering traffic data from them as well. CORSIM [54, 56] is a popular traffic simulator that allows the user to have an input file telling individual vehicles what paths to take and lane changes to make, though this has to be done before the simulation rather than in real-time. A more thorough summary of traffic simulators is provided in Chapter 4.1.

With the research that has been and is being conducted, there has been little analysis of traffic assuming that a complete simulation of the freeway system can be obtained at any given point in time. This is due to the fact that this data has never been available, though I believe that technology is fast reaching the point of being able to obtain this data and use it to aid in the traffic problems facing commuters. In addition, a simulation for gathering data from the vehicles has not been created, nor have algorithms for identifying incidents and re-routing vehicles around that location. Solutions to congestion have been proposed by changing the ramp metering [52] based on the speeds of the vehicles on the freeway at that point [17], though as the population continues to increase in urban areas, the traffic problems facing commuters are only going to get worse. By diverting people along other paths that may be longer though faster, the traffic problem can be ameliorated in the short-term while more long-term solutions can be developed.

Chapter 3

Architecture and Algorithms

3.1 My Approach

Current approaches for gathering traffic flow information occur primarily by using loop detectors, which are large pieces of hardware that must be embedded in the highway. These devices are quite expensive, and are not very reliable, as anywhere from 15% up to 50% of the loop detectors in CalTrans' District 7 (which encompasses the Los Angeles and Ventura counties) may not be working at any given time [8, 59].

As mobile and geographic positioning system technology continue to improve, it becomes possible to leverage the multi-billion dollar cellular infrastructure to gather traffic data from individual vehicles at all points on a highway rather than just at specific locations at which loop detectors are installed. For purposes of this thesis, I am assuming that vehicles will transmit their speed and location through a cellular link to a cellular tower, which will forward the data to my system. The system will gather all of the speed and location data, which can later be used to answer such questions as: what is the fastest path between two points and where is the location of an incident affecting the flow of traffic. The specifics of the underlying applications will be described in more detail later.

In this thesis, I am only going to consider driving directions that consist of highways. Highways are defined as roads that do not have any outside agent governing the flow (such as traffic signals or stop signs). Although surface streets may have to be traversed to get from a vehicle's current location to its ultimate desired destination, I restrict my analysis to highway segments. Considering surface streets may be a modification that can occur in the future as an extension to this thesis.

Although the approach that I describe in this section and the following ones can be applied to any highway system, my examples are taken specifically from the Los

13

Angeles freeway system, as depicted in Appendix A.2. Assume that a freeway system can be characterized as a directed graph $G=(V, E)$, where a vertex $v \in V$ is defined as an on-ramp or off-ramp of a highway and an edge $e \in E$ is defined as the section of the highway connecting two vertices, as shown in Appendix A.3. In the entire Los Angeles freeway system, there are a total of 1053 on-ramps and 1088 off-ramps, giving 2141 vertices with 2401 edges. The fact that the number of edges is close to the number of vertices can be explained by noticing that the only time that we see more than one outgoing or incoming edge from a vertex is at a freeway intersection.

Depicting the freeway system as a graph allows a path to be defined in the usual way:

A path P on a directed graph G=(V, E) is a sequence of vertices such that

$P(i,j) = \{v_i,...,v_j\}$ such that for $i \leq k < j$, $(v_k, v_{k+1}) \in E$

Note that one limitation in the above definition with respect to my application is that v_j can not be an on-ramp, as the destination of a path cannot be a vertex at which the vehicle cannot exit the highway. However, v_i may be an on-ramp or an off-ramp, for the source vertex of a path will change as a vehicle is moving along the highway. This means that the source vertex may be an off-ramp if the vehicle is closest to an off-ramp when the path is being determined.

There are two separate, though closely related, problems that need to be considered. The first problem is how to gather the speed and location of all of the

vehicles in the highway system. The second problem is how to use this data to optimally determine the fastest path between a vehicle's current location and its desired destination. For both of these problems, the first question that needs to be answered is whether current technology supports the bandwidth required.

For the speed gathering problem, I assume the location data will be retrieved through a global positioning system, which allows a vehicle to determine its location as a (latitude, longitude) pair. Currently, navigation systems exist in many cars, and they routinely track the (latitude, longitude) position of the vehicle. The speed of the vehicle can be retrieved either through the vehicle's computer system or by using triangulation of two geographical locations divided by the time to travel between the points. I am assuming that all cars will support this capability. These two pieces of data will then be transmitted to my system for storage and analysis. The transmission will occur over a cellular link, so any mobile device that has access to the Internet will be able to transmit this data to my system. I will not have to be concerned with users who are not on freeways transmitting data because my application will know the (latitude, longitude) pairs that are valid for the freeways. If a speed is transmitted with a (latitude, longitude) pair that is not within a freeway, the application will ignore it until surface streets are considered in the future.

Bandwidth Considerations

To determine the bandwidth required to transmit the (speed, location) pair, some assumptions must be made. For the speed, 1 byte will be sufficient, which will allow for speeds up to 256, which, when dealing with miles per hour or kilometers per hour, is well

over the speed at which vehicles travel today. For the location, 8 bytes will suffice for the (latitude, longitude) pair, assuming the latitude and longitude are represented as floating-point values. Any data that is transmitted will incur overhead because of transfer information, such as headers and footers. I have assumed 40 bytes of packet overhead, since that is the size of a TCP packet header. One variable that I will have in the speed gathering application is how often the (speed, location) pair will be transmitted from the vehicle to my system. I will assume that this data is transmitted every x seconds, and as determined by [2], there could be up to 1 million vehicles in the Los Angeles freeway system at any given time. Then, the bandwidth required by my system for the speed gathering application is:

```
BW_SG = ((1 + 8 + 40 bytes) / x seconds) * 10^6 vehicles

     = (49 bytes / x seconds) * 10^6 vehicles

     = (392 bits / x seconds) * 10^6 vehicles * 1 Mb / 2^20 bits

     = 373.84/x Mbps
```

Transmitting the (speed, location) data every second (x=1) would require approximately 374 Mbps. As I do not believe that it is necessary for every vehicle on the highway to transmit its speed every second, a more reasonable value is x=30, in which case the required bandwidth is only 12 Mbps, which can definitely be handled by existing systems.

For the fastest path application, the vehicle will transmit its current location and the desired destination location to the system and receive the fastest path from the source to the destination in return. The fastest path will be a sequence of vertices, as defined

earlier in this section.   The data will be transmitted in both directions over a cellular link, so any mobile device with Internet access will be able to transmit and receive the data.

To determine the bandwidth for the application that returns the fastest path, a few assumptions must be made about the size of certain pieces of data. Since a path consists of a sequence of vertices, the size of the vertex will be assumed to be 2 bytes, which is enough to represent each vertex by a unique identifier. The overhead for a packet will be the same as in the speed gathering application, which is 40 bytes. The length of the path returned is a variable that depends on the source and destination, though we will assume that there are p vertices in the path. With an updated fastest path being requested every y seconds by up to 1 million vehicles simultaneously [2], the bandwidth required by my system for this application is:

$$BW_{FP} = (2 \text{ bytes/vert} * p \text{ verts} + 40 \text{ bytes}) / y \text{ secs} * 10^6 \text{ vehicles}$$
$$= (16p \text{ bits} + 320 \text{ bits}) * 10^6 \text{ vehs} / y \text{ secs} * (1Gb / 2^{30} \text{ bits})$$

Considering in the absolute worst case that every vertex in the freeway system is returned in the path, there are 2141 total off-ramps and on-ramps in the Los Angeles freeway system. If every one of these ramps was returned every second (y=1), the bandwidth required is approximately 32 Gbps. More practically, in the worst case, only the off-ramps and on-ramps in one direction of each freeway will be returned (since it does not make much sense to go both east and west on the same freeway, passing the same point more than once), giving 1070 off-ramps and on-ramps. In addition, the fastest path does not need to be updated every second, but every 30 seconds will be sufficient. With p=1070 and y=30, the required bandwidth for the fastest path application in my system is

approximately 541 Mbps. With connection speeds reaching the 1 Gbps or higher, a single system is capable of handling a bandwidth of 541 Mbps. From the system point-of-view, the required bandwidth is well within ranges supported by today's technology.

Another consideration arises from the mobile devices and the cellular base stations. Do these devices have the necessary bandwidth to handle the data being transmitted and received? As far as mobile devices are concerned, the Telecommunications Standardization Sector (ITS) targets 2 Mbps for the 3G cellular standard. For receiving fastest paths from the system (as follows from the above formula for BW_{FP} with only 1 vehicle instead of 10^6 vehicles), each mobile device will require:

```
BWC = (16p bits + 320 bits) * 1 veh / y secs * (1 Kb / 1024 bits)
```

In the worst case, the number of vertices in the path was 2141 and the paths were transmitted every second (y=1), approximately 34 Kbps of bandwidth is needed. A 3G cellular network can easily handle this bandwidth. Even more encouraging is the fact that practically speaking, the required bandwidth will be significantly less than 34 Kbps.

With respect to the cellular base stations, cellular towers are typically connected to the backbone network by a T3/DS3 line, which has a bandwidth of approximately 45 Mbps. This bandwidth is used to transmit both voice and data, however, so I will assume that I can utilize 22.5 Mbps to transfer data. To determine the bandwidth that each cellular tower will be required to handle, I need to assume that there are *c* cellular towers

Transmit Period (x)	Required Server Bandwidth	Required Bandwidth / Cell Tower
1 sec	373 Mbps	764 Kbps
30 sec	12 Mbps	24 Kbps

Table 1 – Bandwidth summary for transmitting (speed, location) data to the system

supporting the Los Angeles freeway system.[1] Assuming there are 500 cellular towers responsible for the freeways in Los Angeles (c=500), and considering a worst case bandwidth of 32 Gbps, each cellular tower will require 32/500 Gbps, which is approximately 64 Mbps. This cannot be handled by current cellular backbone technology. However, as long as the transmission of path data occurs for $y > 2$ seconds, cellular towers will not have a problem transmitting the data. At $y = 3$ seconds, each cellular tower would require approximately 21 Mbps.

Tables 1 and 2 summarize the discussion in this section. Table 1 shows that transmission of speed and location data from every vehicle in a freeway system every 30 seconds is able to be handled by current cellular technology, and even transmissions every second do not over-extend current bandwidth limitations. Table 2 shows that transmitting fastest path data back to every vehicle in a freeway system every second cannot be handled by current bandwidth limitations of cellular systems. However, transmitting the paths every 30 seconds is definitely feasible.

---

[1] The actual number of cellular towers is unknown since the number and location of towers is proprietary information to cellular companies.

Transmit Period (y)	Length of Path	Required Server Bandwidth	Required Bandwidth / Cell Tower
1 sec	2141 ramps	32 Gbps	*64 Mbps*
3 sec	2141 ramps	11 Gbps	21 Mbps
30 sec	1070 ramps	541 Mbps	1 Mbps

Table 2 – Bandwidth summary for transmitting fastest path data to the vehicles

In short, current hardware technology supports the gathering of real-time speed and location data and the returning of fastest path data. The challenge does not lie in the bandwidth requirements, but rather in the maintenance and computation of fastest paths based on the received speed and location data.

3.2 Architecture Description

My proposed architecture for the speed gathering and fastest path applications is shown in Figure 1. The architecture needs to take into account the fact that, although these are distinct applications, the data is shared and some of the hardware components can be used in both applications. The difference lies in the functionality behind the application, which is implemented in two different sets of components in the architecture, called the Speed Gathering Computer Farm and the Fastest Path Computer Farm. The architecture I describe should not be construed as a physical architecture, but rather as a logical architecture. This architecture provides a convenient way of thinking about the applications I will describe, though alternative architectures may have advantages over mine.

The architecture can best be described by walking through a typical scenario of each application. For the speed gathering application, assume that each vehicle has a mobile device that is capable of sending data to cellular towers. The mobile device will retrieve its geographical location from a Global Positioning System, shown as GPS Satellites in the architecture. It will also retrieve the speed of the vehicle either from the vehicle's computer system or by triangulating a delta in distance by the amount of time elapsed. These two pieces of data will be transmitted via cellular towers to the Speed Gathering Computer Farm set of computers. The Speed Gathering computers will receive as input a (latitude, longitude) pair and the speed of the vehicle at that point. There is no output required, as the vehicle is not requesting any information. The Speed Gathering computers will map the (latitude, longitude) pair to a highway and then forward the (latitude, longitude, speed) triplet to the corresponding Highway Computer
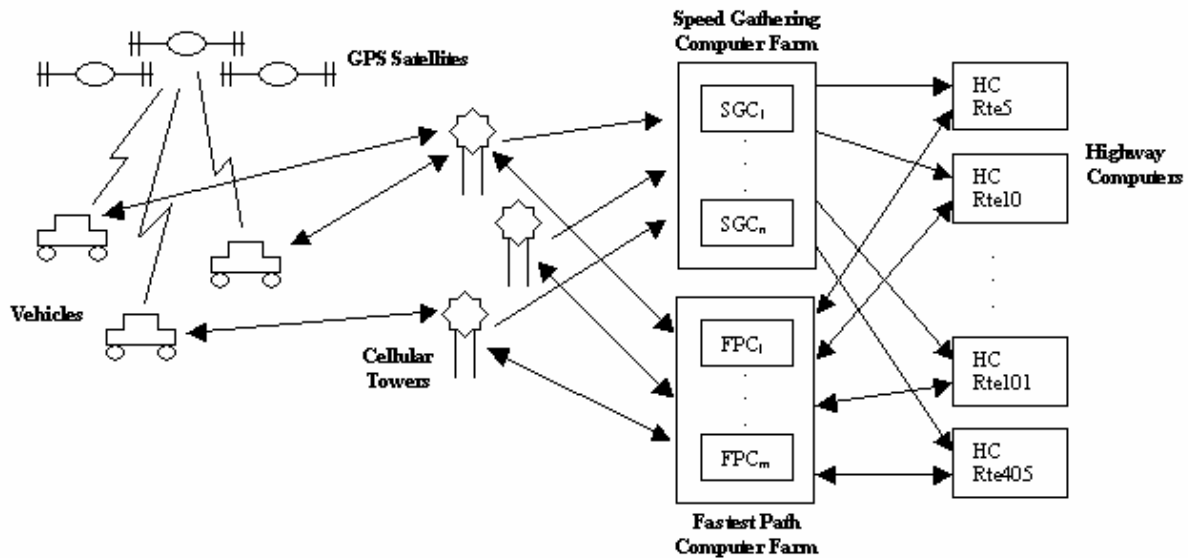
Figure 1 – Architecture for Speed Gathering and Fastest Path applications

(i.e. HC Rte10). The Highway Computer does not return any data back to the Speed Gathering computer.

For the fastest path application, we continue to assume that each vehicle has a mobile device capable of sending data to cellular towers and receiving data back. The mobile device will obtain its (latitude, longitude) from a Global Positioning System, and the driver will enter the desired destination. The vehicle's current location and the desired destination will be transmitted via a cellular link through a cellular tower to the Fastest Path Computer Farm. The Fastest Path computers will take as input the source (latitude, longitude) pair and the destination (latitude, longitude) pair. The Fastest Path computers will map the source and destination locations to (highway, on-ramp/off-ramp pairs), since all of the vertices in the graph are on-ramps or off-ramps. They will then determine all of the paths between the two locations and request the speed information from each Highway Computer traversed in each path. Based on the current speeds returned by the Highway Computers, the Fastest Path computers will determine the path

that will take the least amount of time from the source to the destination and return that path.

The algorithms developed for the Speed Gathering computers and the Fastest Path computers must be very efficient. With the potential of up to 1 million vehicles in the Los Angeles freeway system at a given point in time [2], if the algorithms are not very efficient, the latency will be so great that the system may appear to be unusable. Acceptable latency amounts are not yet known since this is based on user perception; however, a latency of more than several minutes seems unacceptable, as drivers will lose focus and traffic conditions may change.

The Highway Computers will store the (latitude, longitude, speed) data with the current time for all cars that have transmitted this data. They will then have to be able to answer queries and return the most current speeds requested by the Fastest Path computers. Let's assume that 10% of the 1,000,000 in the Los Angeles freeway system request path data at the same time, and let's assume that it requires 10 milliseconds to compute an optimal path. Ignoring transmission time,

```
10% * (1,000,000) * .01 seconds = 1000 seconds
```

1000 seconds is 16 minutes and 40 seconds, which is not acceptable for a driver to have to wait to receive an updated path. The data structures on the Highway Computers will have to be clever and efficient, as these computers contain the core data that is used by both applications every time a speed is transmitted or a fastest path is requested.

3.3 Algorithms

Considering the Los Angeles freeway system as a directed graph where the off-ramps and on-ramps are vertices and the segments of the freeways connecting those ramps are the edges, I will define the weights of the edges as the amount of time it will take to drive from one ramp to the next ramp. In other words, the amount of time to travel along that segment of the freeway in a vehicle will be the weight of the corresponding edge in the graph. As speeds are updated continuously in real-time, the amount of time to traverse an edge could change often. This implies that re-computation of the shortest path will have to be done frequently, and efficient algorithms to achieve this are crucial.

To compute the fastest path between two points in a directed graph, I discuss three classes of algorithms: the Naïve class, the Dynamic class, and a new class which I call the Dynamic All-Pairs All-Paths class. In section 3.3.1, I discuss the Naïve class of algorithms, in section 3.3.2, I discuss the Dynamic class, and in section 3.3.3, I discuss my Dynamic All-Pairs All-Paths class of algorithms.

3.3.1 Naïve Class of Algorithms

The Naïve class of algorithms includes Floyd-Warshall's Algorithm [9] and Johnson's Algorithm [10], which both compute the shortest path for all pairs of vertices in a graph. Floyd-Warshall's Algorithm takes advantage of "intermediate vertices", and the paths to which intermediate vertices are a part. An intermediate vertex of a simple path is any vertex that is not an end-point of the path. Floyd-Warshall's Algorithm uses a recursive approach that determines a minimum-weight path along each path from one

node to every other node in the graph. The running time of Floyd-Warshall's Algorithm is $O(V^3)$, as can be shown by analyzing the algorithm in Appendix B.1.

Johnson's Algorithm, on the other hand, re-weights the graph in a pre-processing step (if negative-weight edges exist), and uses the Bellman-Ford Algorithm [29,30] and Dijkstra's Algorithm [11] from every vertex for determining the shortest paths. Since my specific application does not contain any negative-weight edges, the pre-processing step of Johnson's Algorithm is irrelevant. Dijkstra's Algorithm iterates through all of the vertices that are neighbors of the initial vertex and determines the shortest path. It then iterates through all of the vertices that are neighbors of the initial node's neighbors and determines the shortest path to them. It continues in this fashion until the shortest path to all of the nodes in the graph has been found. The running time of Dijkstra's Algorithm, if implemented with a min-priority queue is $O(VlgV + E)$, giving a total running time for Johnson's Algorithm of $O(V^2lgV + VE)$, which is slightly faster than Floyd-Warshall's.[2] See Appendix B.2 for pseudo-code of Dikjstra's Algorithm, Appendix B.3 for pseudo-code of the Bellman-Ford Algorithm, and Appendix B.4 for pseudo-code of Johnson's Algorithm.

Analyzing these algorithms with respect to my specific problem requires the algorithm to be executed every time an edge update occurs, which is when an updated speed is received. Although other algorithms I will discuss may be faster for updates, re-running these algorithms every time a speed is received allows the fastest path to be determined in constant time, since the fastest paths will already have been computed for

---

[2] Both Johnson's and Floyd-Warshall's algorithms have good explanations and pseudo-code in [12].

all pairs of vertices when the optimal path is requested. These algorithms will be my basis for creating other algorithms that run faster or have other advantages over these.

3.3.2 Dynamic Class of Algorithms

The Dynamic All-Pairs Shortest Path class of algorithms were developed by Demetrescu and Italiano [13, 45], who created an algorithm that can determine the path with minimum cost between two vertices in a graph in constant time and update an edge in $O(V^2 log^3 V)$. This class of algorithms supports edge updates, deletions, and insertions by noticing that changing the weight of an edge to have infinite cost will make that edge never become part of a minimum cost path, which essentially removes that edge from the graph. Edge inserts behave just as edge updates do, though the weight of the edge will be changed from infinite (i.e. the edge did not previously exist) to its new value. With respect to my specific problem, the Dynamic All-Pairs Shortest Path algorithms execute more efficiently than the Naïve algorithms, though the edge update cost is still executing in polynomial time. The fastest path running time is constant, just as it was in the Naïve case. For more information on the Dynamic All-Pairs Shortest Path algorithms, refer to Demetrescu and Italiano [13], Klein and Subramanian [14], and King [15].

3.3.3 Dynamic All-Pairs All-Paths Class of Algorithms

The All-Pairs All-Paths class of algorithms takes advantage of the fact that the graph is rather static. With no edge insertions, all of the paths between all pairs of nodes can be pre-computed, which then does not require the application to take the extra step of determining all of the paths between two vertices when a request for a fastest path is

being answered. The pseudo-code for determining all of the paths between all pairs of vertices can be found in Appendix C.1. Based on the pseudo-code, the time to pre-compute the number of paths from one vertex to every other vertex is factorial with respect to the number of edges in the graph, or $O(E!)$. Then, to compute all of the paths between all pairs of vertices would require multiplying the time to compute the number of paths from one vertex to every other vertex ($O(E!)$) by the number of pairs of vertices, which is V^2. The overall running time to compute all paths between all pairs of vertices is then $O(V^2E!)$. Although this value is factorial with respect to the number of edges, it is really irrelevant since the pre-computation only occurs one time for the entire freeway system. Only when a change occurs (a freeway segment is added or removed) will this algorithm have to be re-executed.

For the Los Angeles freeway system as shown in Appendix A.2, there are 53 freeway intersections. This gives 53 * 53 = 2809 total pairs of intersections. Upon analysis of this graph, I have determined that there are 4,557,802,218 different paths between all pairs of vertices, giving an average of 1,534,613 paths between any two vertices. Although pre-computing all of these paths may save in computing time when a fastest path is requested, determining which of the 1.5 million paths between two nodes is the fastest is not a trivial problem[3]. The following three algorithms all use the pre-computation algorithm described above, though how they update edge weights and determine shortest paths differ. The first algorithm allows for constant edge updates whenever a new speed is received from a vehicle. The second algorithm allows for

---

[3] Although this is not considering all 2141 vertices in the Los Angeles freeway system, as depicted by Appendix A.3, the number of paths between any two vertices will be on the same order as calculated above. The additional vertices in Appendix A.3 mostly have two edges, which will not significantly add to the number of paths between two nodes, but rather just the number of edges in each path.

constant fastest path queries whenever a vehicle requests a fastest path. The third algorithm is a hybrid approach between the previous two, attempting to optimize the algorithms.

The *Constant Update Algorithm* sacrifices the speed of retrieving fastest paths for the amount of time it takes to update the weight of an edge. The way the algorithm works is by not maintaining the fastest paths at all times, but rather computing the fastest path when requested by a vehicle. When a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first[4]. If the average speed on that edge has changed by s miles per hour, the time to traverse that edge will be updated, which can be accomplished in constant time. To retrieve the fastest path, it is necessary to determine the time to traverse all m paths between the source vertex and the destination vertex, and select the path with the minimum time. Assuming in the worst case that each of the m paths will contain a maximum of V vertices, the overall running time becomes $O(mV)$. The data structures and pseudo-code for the Constant Update algorithm can be found in Appendix C.2.

The *Constant Query Algorithm* sacrifices the speed of updating an edge for the time to retrieve the fastest path. This algorithm always maintains the fastest path between all pairs of vertices by recalculating the fastest paths for all pairs of vertices that have a path containing the updated edge whenever an edge update occurs. When a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first. If the average speed on that

---

[4] The average number of vehicles passing a specific point in the LA freeway system in an hour is 2000 [17]. With 4 lanes on the freeway, this gives 667 cars every 5 minutes.

edge has changed by *s* miles per hour, the time to traverse that edge will be updated, and the fastest paths for all pairs of vertices that have a path containing that edge will be recalculated. In the worst case, all paths for all pairs of vertices contain that edge, which are V^2 pairs, with m paths for each pair, giving V^2m. To insert an updated edge (implemented as a binary search tree) will take *O(logm)*, giving an overall running time of $O(V^2\ mlogm)$. However, to retrieve a path merely requires extracting the minimum path from the binary search tree, which can be done in constant time. The data structures and pseudo-code for the Constant Query Algorithm can be found in Appendix C.3.

The *Hybrid Algorithm* attempts a compromise between the Constant Update and the Constant Query algorithms by maintaining the amount of time it takes to traverse each path whenever an edge update occurs. It does not maintain a sorted sequence of paths, however, as the Constant Query algorithm did. When a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first. If the average speed on that edge has changed by *s* miles per hour, the time to traverse that edge will be updated, and all paths that contain that edge will have their overall time updated. If the updated edge is contained within every path of every pair of vertices, this algorithm will take $O(V^2\ m)$. To retrieve the fastest path, the application will only have to compare the times to traverse all m paths from the source to the destination and return the path with the minimum time. There are only *m* paths for each pair of vertices, giving a running time of *O(m)*. The data structures and pseudo-code for the Hybrid algorithm can be found in Appendix C.4.

Table 3 contains a recapitulation of the running times discussed in this section. The important observation to note is that, depending on the value of *m,* the All-Pairs All-

	Pre-computation	Update Edge	Retrieve Fastest Path
Naïve (Johnson)	N/A	$O(V^2 \log V + VE)$	$O(1)$
Dynamic All-Pairs Shortest Path (Demetrescu, Italiano)	N/A	$O(V^2 \log^3 V)$	$O(1)$
All-Pairs All-Paths Pre-Computed – Constant Update	$O(V^2 E!)$	$O(1)$	$O(Vm)$
All-Pairs All-Paths Pre-Computed – Constant Query	$O(V^2 E!)$	$O(V^2 m \log m)$	$O(1)$
All-Pairs All-Paths Pre-Computed – Hybrid	$O(V^2 E!)$	$O(V^2 m)$	$O(m)$

Table 3 – Running time comparison for the five algorithms discussed in this section

Paths class of algorithms may execute faster than the other classes of algorithms. However, the value of m is an unknown that can be determined by the longest distance of an alternate path a driver would take if it were to save him time in traveling to his destination. If m is constant, the Constant Query and Hybrid algorithms have polynomial edge updates and constant fastest path retrievals. Also important to note is that the number of edge updates will greatly exceed the number of fastest path queries, which should be taken into account when considering which set of algorithms will run most efficiently for this problem. With this consideration, the Constant Update algorithm runs in constant time for edge updates and linear time for retrieving fastest paths if the number of paths to consider is constant, which, during simulation, proved to be the fastest algorithm practically.

Chapter 4

FreeSim Overview

4.1 Current Traffic Simulators

Many transportation applications currently exist and are still being developed. I have classified seven popular traffic applications (CORSIM/TSIS [64,65], FreeSim [62], MITSIM [66], PARAMICS [67], RENAISSANCE [68], VATSIM [69], and VISSIM [70]) into eight different categories: platform, source code availability, cost, transportation network, underlying method of use, vehicle model, data input manner, and data gathering manner. Tables 4.1 and 4.2 provide a summary of this data.

The "Application" field provides a list of the different traffic applications I am comparing with the organization that created the application in parentheses. CORSIM is the only application created for a government organization, which is the United States Federal Highway Administration (FHA). FreeSim, MITSIM, RENAISSANCE, and VATSIM were created by universities, and PARAMICS and VISSIM were created and are supported by private companies.

The "Platform" shows on which operating system the application will run. There is close to an even split between the Windows and Linux applications, and FreeSim allows execution on any operating system since the entire application is written in Flash and Java, which are platform independent.

The "Source Code Availability" is critical for determining how extensible the application is. FreeSim and MITSIM are both open-source, licensed under the GNU GPL [72] and the MITSIMLab Open Source License [71], respectively. The other applications are extensible, but only based on the application programming interface (API) provided by the developers of the software.

Application	Platform	Source Code Availability	Cost	Transportation Networks
CORSIM/TSIS (FHWA)	Windows	Closed	Paid	Free-Flowing (FRESIM), Regulated (NETSIM)
FreeSim (USC)	Any	Open	Free	Free-Flowing
MITSIM (MIT)	Linux	Open	Free	Free-Flowing, Regulated
PARAMICS (Quadstone Ltd.)	Linux, Solaris, Windows	Closed	Paid	Free-Flowing, Regulated
RENAISSANCE (Technical Univ. de Crete)	Windows	Closed	Paid	Free-Flowing
VATSIM (Ohio State Univ.)	Linux	N/A	N/A	Free-Flowing, Regulated
VISSIM (PTV)	Windows	Closed	Paid	Free-Flowing, Regulated, Railways

Table 4.1 – Simulator Characteristics

The "Cost" data happens to correspond directly with the "Source Code Availability" data, where applications that are open-source are also provided for free, whereas applications that do not provide source code require a fee for licensing.

The "Transportation Networks" field describes the types of roadways that are available in the applications. All of the applications support free-flowing roadways (such as freeways, highways, interstates, etc.) and CORSIM, MITSIM, PARAMICS, VATSIM, and VISSIM all support traffic-regulated roadways, such as roads with traffic signals, stop signs, toll booths, etc. VISSIM also allows railways, bicycle traffic, and pedestrians.

The "Underlying Method of Use" allows users to know where the application can be used. Although they all could be used in a driving emulator, VATSIM is the only tool specifically designed for that. The rest of the applications are considered simulators since they are attempting to model a transportation system and show the system based on a certain vehicle model. Maroto, et.al., provide a good overview of other driving emulators, as well as providing another driving emulator with a new driver model in [73].

Application	Underlying Method of Use	Vehicle Model	Data Input Manner	Data Gathering Manner
CORSIM/TSIS (FHWA)	Simulator	Micro	Discrete	Traditional
FreeSim (USC)	Simulator	Macro, Micro	Continuous, Discrete	Individual Vehicles, Traditional
MITSIM (MIT)	Simulator	Micro	Discrete Transformed	Traditional
PARAMICS (Quadstone Ltd.)	Simulator	Micro	Discrete	Traditional
RENAISSANCE (Technical Univ. de Crete)	Simulator	Macro	Discrete Transformed	Traditional
VATSIM (Ohio State Univ.)	Emulator	Micro	Simulated Continuous	Traditional
VISSIM (PTV)	Simulator	Macro (VISUM), Micro	Discrete	Traditional

Table 4.2 – Simulator Characteristics

The "Vehicle Model" specifies whether the application shows the transportation system from a bird's eye view (macroscopic) or as viewed by an individual vehicle (microscopic). RENAISSANCE is the only application that solely models the vehicles macroscopically, and FreeSim and VISSIM both allow for macroscopic and microscopic modeling of vehicles. An overall view of the transportation system can be seen, and an individual vehicle's progress can be tracked in those simulators.

The "Data Input Manner" describes whether the data used to determine the locations and speeds of the vehicles is done continuously or just by giving their speed at a discrete location and then having the vehicle maintain that speed until the next discrete location in its path. VATSIM simulates a continuous flow since the vehicles in the application change speeds using algorithms based on the vehicles surrounding them coupled with a psychological driver-behavior algorithm. MITSIM and RENAISSANCE use discrete data; however they extend the discrete data to be uniformly continuous through an algorithm such as the traffic state estimator algorithm. For more information on how MITSIM and RENAISSANCE convert discrete data into uniformly continuous

data, refer to [66] and [68], respectively. FreeSim allows the data being sent to the vehicles to be continuously updated by the application, and then each individual vehicle can decide whether or not to respond to the received information.

The "Data Gathering Manner" describes how the data is gathered while the simulation is executing. In FreeSim, the individual vehicles communicate their speed and location back to the application, which is then fed into the algorithms for updating the fastest paths for all of the vehicles. As in the other applications, the data is also gathered in a traditional manner, similar to the way in which transportation organizations use loop detectors, video cameras, or other devices measuring data at discrete locations.

Although each of the applications described have certain advantages over other applications, FreeSim provides three main advantages. First of all, FreeSim is open-source and free to download. Second, the data used to simulate the vehicles in the freeway system can be sent in a continuous or a discrete manner. And finally, which is the main reason I created my own simulator rather than using an existing one, FreeSim allows communication between each individual vehicle and a central system (V2R) or between vehicles (V2V). Much of the research concerning intelligent transportation systems assume that the vehicles are able to communicate with a central server or other vehicles, and FreeSim, unlike other simulators, has this ability built into the framework.

4.2 FreeSim Description

FreeSim is a traffic simulator that models free-flowing transportation systems as a weighted directed graph. The edges of the graph are the freeway segments that the user would like to monitor, and the nodes are the connections between the segments. The segments can be as short or as long as desired and as granular as individual lanes. There is no limit as to how many edges can emanate from a node, although in most freeway systems this number will not exceed more than about eight (i.e. a 4-freeway interchange like the 4-level interchange in Los Angeles that connects the 5, 10, 60, and 110 freeways). A freeway system is stored in a database with FreeSim, though there is a second program bundled with FreeSim for reading a list of nodes and a list of edges that define a freeway system from text files and populating the database accordingly. Multiple freeway systems can be stored in the same database with unique identifying names.

The graphical user interface for FreeSim renders a freeway system in a browser via the Adobe Flash 8 plug-in. Over a socket connection, the Flash front-end connects to a Java-based server program for all of the simulator functionality. The Flash interface merely provides a light-weight GUI for displaying the output of the simulator. Appendix E.1 gives a screenshot of FreeSim when the map consists of freeway intersections as nodes, and Appendix E.2 gives a screenshot when the map consists of loop detectors as nodes.

FreeSim allows shortest paths and fastest paths to be determined based on distance or current speeds, respectively. The shortest paths between all pairs of nodes are determined at start-up using Johnson's algorithm, although the fastest path must be

determined at query time since it will change based on the current speeds. When querying for shortest paths, the amount of time to traverse the shortest path at current speeds is also returned, which must be calculated at run-time. Fastest path algorithms can be created and executed in FreeSim quite easily by registering a Java class with the traffic simulator that implements two methods – one for retrieving a fastest path from a source node to a destination node, and one for updating the weight of an edge with a new speed. A graph data structure containing the freeway system is also accessible by the class. The method for retrieving the fastest path is called when a user or vehicle requests a fastest path, and the method for updating the weight of an edge with a new speed is called when the speed on an edge is updated to a speed for which the difference is greater than a certain threshold, as defined in [63]. FreeSim already has six fastest path algorithms implemented: Dijkstra's Algorithm [11], Bellman-Ford's Algorithm [29, 30], Johnson's Algorithm [10], and the three All-Pairs All-Paths Pre-Computed algorithms [63]. For the three non-dynamic algorithms (Dijkstra's, Bellman-Ford's, and Johnson's), the method for updating the weight of an edge does nothing, since the fastest path will be recomputed at query time. For the All-Pairs All-Paths Pre-Computed set of algorithms, the pseudo-code of the update edge methods can be found in Appendix C.

In the FreeSim interface, the speed on an edge can be updated, which causes the method for updating the weight of an edge to be called for all of the fastest path algorithms that have been registered with the traffic simulator. The speed and time to traverse each edge at that speed can be viewed by placing the mouse on the top of a node. Those speeds and times will change in real-time as the simulator receives speed updates from vehicles.

In addition to the user being able to find shortest and fastest paths and update the speed on a freeway segment, FreeSim also allows a user to have a series of commands executed simultaneously during a simulation. A file can be created that allows a user to specify different events that will occur at discrete times during the course of a simulation. Among these events are vehicles being inserted into the freeway system at a certain source node bound for a destination node, vehicles sending updated speeds for a specific edge, vehicles requesting a new fastest path based on the current speeds, vehicles requesting the current amount of time to traverse a specific path, and messages that allow a user to track the progress of a vehicle along its path.

One possible use of this input file is to load the freeway system with live speed data, such as that gathered by a transportation organization. Appendix D.2 shows sample loop detector data gathered from the California Department of Transportation. The data obtained by loop detectors can be used to obtain the speed of the vehicles crossing that detector during a given time period [4, 24], which can then be used to create the input file to be executed by FreeSim. Appendix E.2 shows a sample input file for FreeSim based on Caltrans' loop detector data. This gives FreeSim the ability to simulate the traffic within a freeway system based on live data. With this approach, FreeSim does not generate any traffic data of its own, which removes the concern that it does not accurately simulate real traffic within a freeway system. Based on having an accurate simulation of real traffic data, there are many potential applications and research questions that can be answered.

4.3 Research Questions Addressed by FreeSim

FreeSim has been executed with user-generated data and live data gathered from the California Department of Transportation (CalTrans). Any research based on vehicle travel times would be relatively easy to conduct within FreeSim, such as determining the optimal time of the day to travel based on the speeds or occupancy of the freeway system or observing the change in time to travel based on the time of the day at which a vehicle departs from a location. Determining the optimal speed of vehicles to provide the maximum throughput for a specific freeway system could also be measured, as was reported in [17] for the Los Angeles freeway system, and extended in Chapter 5.2 of this thesis.

In addition, any research concerned with the distance traveled during a certain time frame based on changing speeds could be determined. Using the amount of time a vehicle is traveling with the distance traveled during a certain time frame, coupled with average gas mileage of vehicles and price per gallon of gas, the number of gallons of fuel and the amount of money that could have been saved by avoiding traffic could be calculated. All of this data can be used to aid in improving the public transportation systems of a city by providing alternate routes during times of heavy traffic or by giving the systems more accurate arrival and departure times.

Based on the live data gathered by a transportation organization, the added traffic due to an incident (such as a collision, road construction, etc.) can be determined. Alternatively, as an event that attracts a significant number of vehicles begins or ends (such as a sporting event), the effect on the traffic of a freeway system of inserting or removing that many vehicles at a certain node can be measured. Similarly, if a new

building were proposed to be built that would attract a number of vehicles each day during a certain time interval, the impact of the added traffic within a freeway system could be simulated. As new freeways are proposed, the overall improvement in the flow of traffic in the entire freeway system could be measured.

From a more theoretical standpoint, graph and traffic algorithms can be tested on user-generated or real traffic data. Shortest path algorithms with changing edge weights (also known as dynamic shortest path algorithms) can be implemented through an API in FreeSim. Traffic prediction algorithms, such as those in [74] and [75], can not only be tested, but can also be verified against live traffic data. Incident identification algorithms can be executed to determine how long it takes to determine that an incident has occurred based on real traffic data.

Focusing more on ITS technology, figuring out how many (or what percentage of) vehicles need to transmit their speed and location to a central traffic server to be able to accurately route vehicles along fastest paths can be determined. With individual vehicles having the ability to communicate as autonomous entities, aggregation algorithms or peer-to-peer approaches can be implemented among the vehicle objects. Having more granular information such as speeds of vehicles in specific lanes gives the ability to determine if lanes are traveling faster than other lanes and therefore saving time for those vehicles.

Although I can not possibly enumerate all of the questions that can be answered using a traffic simulator, this at least provides a few of the open research questions that can be solved using FreeSim.

Chapter 5

Traffic Analysis with FreeSim

5.1 Fastest and Shortest Path Analysis with Caltrans Data

The California Department of Transportation (Caltrans) has approximately 9,000 main line loop detectors in District 7, which includes the Los Angeles and Ventura counties, in addition to having around 450 video cameras installed throughout the freeway system. As of November 2006, only approximately 53% of the loop detectors were functioning properly [59]. I gathered 30-second loop detector data from Caltrans for Friday, November 3, 2006 between the hours of 7:00a.m. and 10:00a.m. from the Northwest section of the Los Angeles freeway system. The exact 768 loop detectors used are represented in the FreeSim screenshot in Appendix E.2. Note that there are only 96 loop detector stations shown in the simulator, but each loop detector station consists of approximately 8 loop detectors, one for each lane in each direction, with an average of 4 lanes in each direction.

The 30-second data provided by a loop detector is the volume of vehicles that have passed that detector within a 30-second period (volume) and the amount of time that a vehicle occupied the space above the loop detector during a 30-second period (occupancy). As described in [4], from the volume V (# cars to pass the loop detector in the given time period), occupancy O (% of time that a vehicle is occupying the loop detector in the given time period), and time period T (30 seconds in this case), and estimating the average length of a vehicle L (feet / vehicle), the speed S (feet / second) can be calculated as:

```
S = L * V / (O * T)
```

Based on the calculated speed, I analyzed the data from Caltrans to determine how the amount of time to travel from one node to another changed with the changing

Path #	Path	Distance (miles)	Time to Traverse at Optimal Speeds (mm:ss)
Path 1	5S – 170S – 101S	15.63	14:26
Path 2	405S – 101S	21.65	19:59
Path 3	405S – 118E – 5S – 170S – 101S	18.40	16:59
Path 4	405S – 118E – 5S – 134W – 101S	27.40	25:18
Path 5	5S – 134W – 101S	24.63	22:44
Path 6	5S – 118W – 405S – 101S	22.16	20:27

Table 5 – Paths from 5S at San Fernando 1 to 101S at Vermont for map in Appendix E.2

traffic conditions. I chose a start node of the 5S at San Fernando 1, which is the node in the top-left corner of the map, and an end node of the 101S at Vermont. Between these two nodes based on the data that I gathered from Caltrans, there are six different paths, which can be seen in Table 5. These are the six shortest paths between the two nodes, and the distances have about a 12 mile difference between the shortest and the longest paths and the time to traverse at speed limit (which is 65 mph for most freeways in the Los Angeles freeway system) vary by about 10 minutes.

The first analysis I performed was based on determining how the time to traverse the shortest path (Path 1) changed with the changing traffic conditions. Graph 1 shows the amount of time to traverse the shortest path between the hours of 7:00a.m. and 10:00a.m. on November 3, 2006. I performed an average for 5 minutes of data since the 30 second data was very jittery. I have included a graph of the 30-second data in Appendix F.1.

As can be seen in Graph 1, the amount of time to traverse the shortest path between 7:00a.m. and 7:45a.m. remains steady around 24 minutes. From 7:45a.m. to 8:45a.m., the time to traverse is between 24 and 29 minutes, and then drops steadily close to its optimal time around 15 minutes by 10:00a.m. This clearly shows what is popularly known as "rush hour" between the hours of 7:00a.m. and 9:00a.m. on a weekday in Los
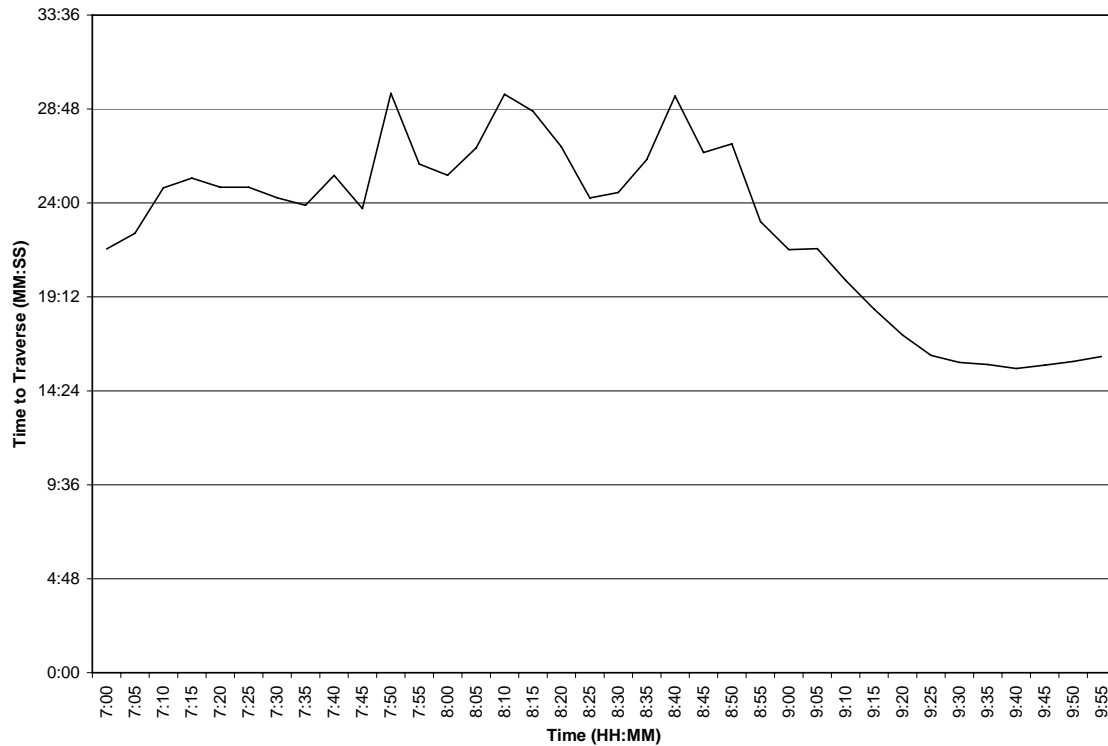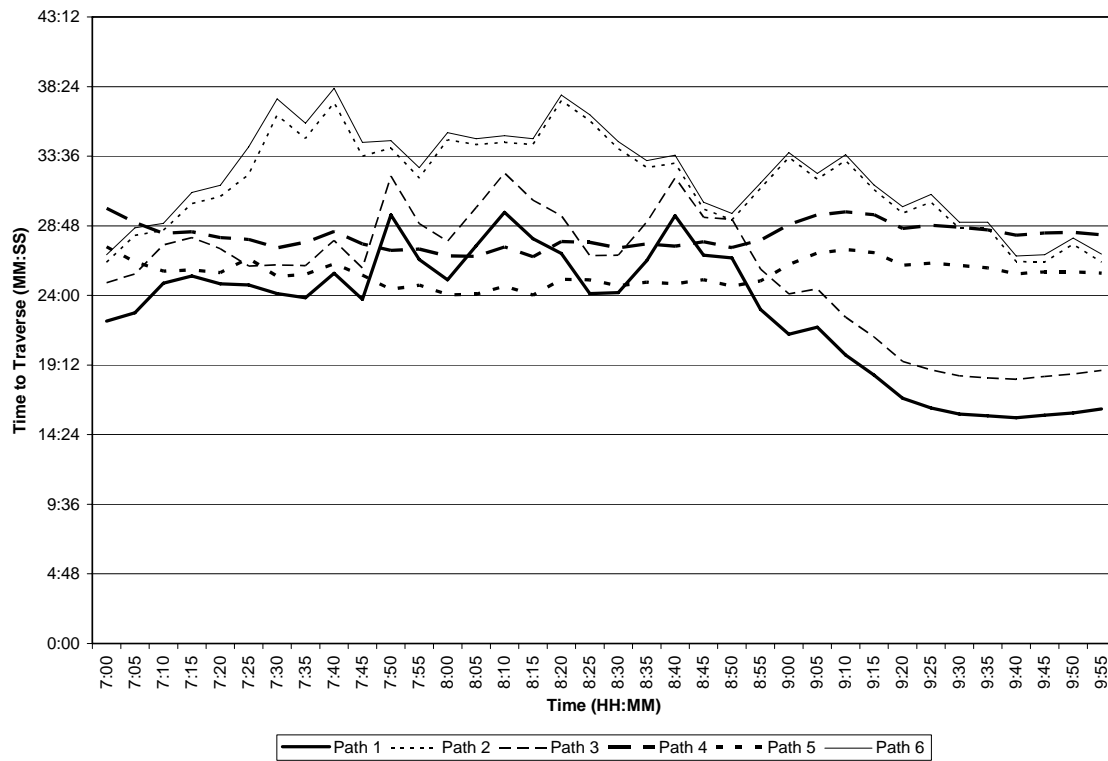
Graph 1 – Time to traverse the shortest path (Path 1) on 2006-11-03 with 5 minute averages

Angeles. Instead of always taking the shortest path, however, a driver can save time if he takes alternate paths at certain times, as can be seen in Graph 2.

Graph 2 shows the time it takes to traverse the fastest path instead of just the shortest path. The fastest path jumps between Path 1 and Path 5, which is not the second shortest path. The reason for this is based on the traffic conditions on the 405S, which is included in the other four paths. As can be seen, the fastest path stays steady around 24 minutes until 8:45a.m., then drops steadily to close to the time to traverse the shortest path under optimal speeds. So between 7:00a.m. and 9:00a.m., it would definitely benefit a driver to know which route is the fastest route at that time, since it is not always the shortest route. Appendix F.2 shows the 30-second data for the time to traverse the fastest path.
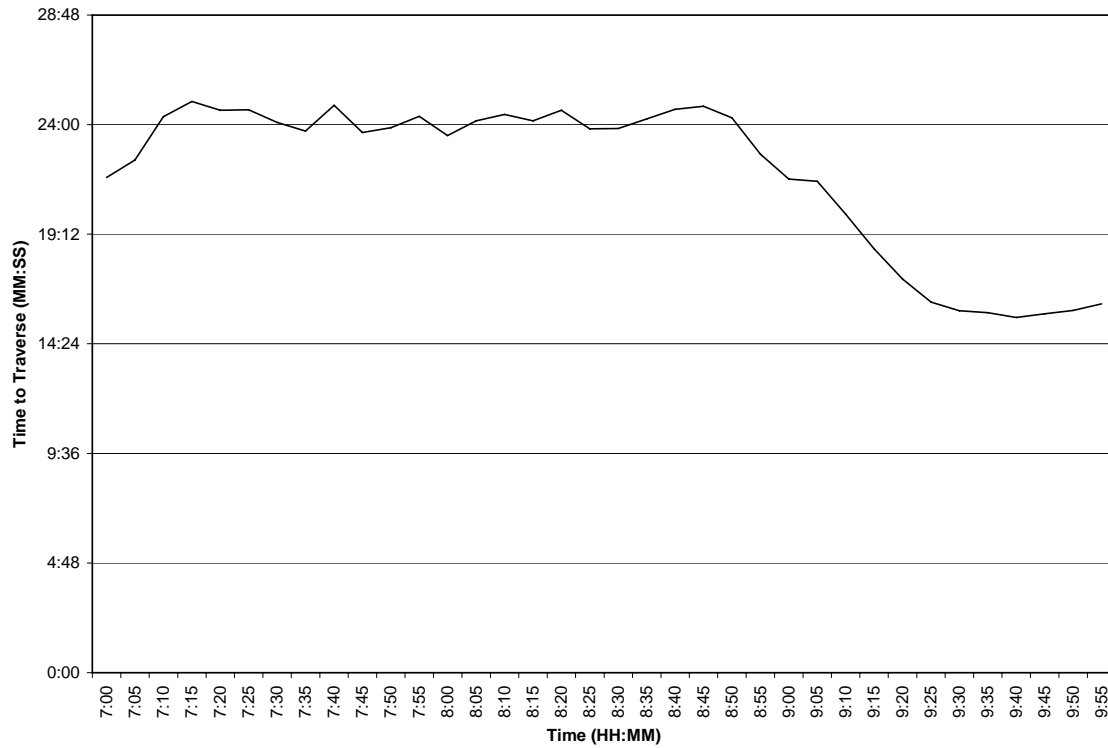
Graph 2 – Time to traverse fastest path on 2006-11-03 with 5 minute averages


Graph 3 – Time to traverse each path from Table 5 on 2006-11-03 with 5 minute averages

45
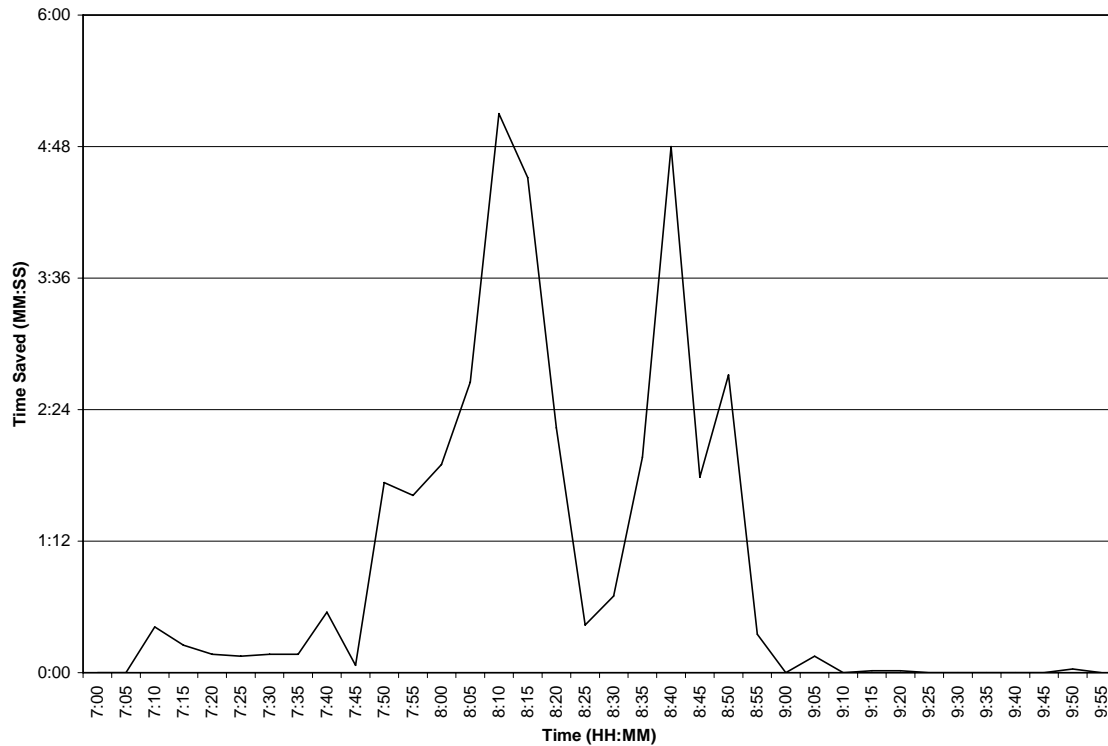
Graph 4 – Difference in time to traverse fastest path instead of shortest path on 2006-11-03 with 5 minute
averages

Graph 3 shows the amount of time to traverse each of the six paths in Table 5
with 5-minute averages from the 30-second data. The fastest path is always either path 1
or path 5, where path 1 is the fastest outside of the 7:45a.m. to 8:45a.m. time period,
which can be referred to as "rush hour", and path 5 is the fastest during that time. Path 1
does improve for approximately 10 minutes around 8:25a.m., but then quickly jumps
back up to being slower than path 5. Paths 2 and 6 are the slowest due to the long stretch
of the 405S freeway that they share which was extremely congested during the sample
period. Appendix F.3 shows the 30-second data for time to traverse each path.

Graph 4 shows the difference in the amount of time to traverse the fastest path
compared to the amount of time to traverse the shortest path with 5-minute averages from
the 30-second data. This graph can be used to determine the overall time savings that a
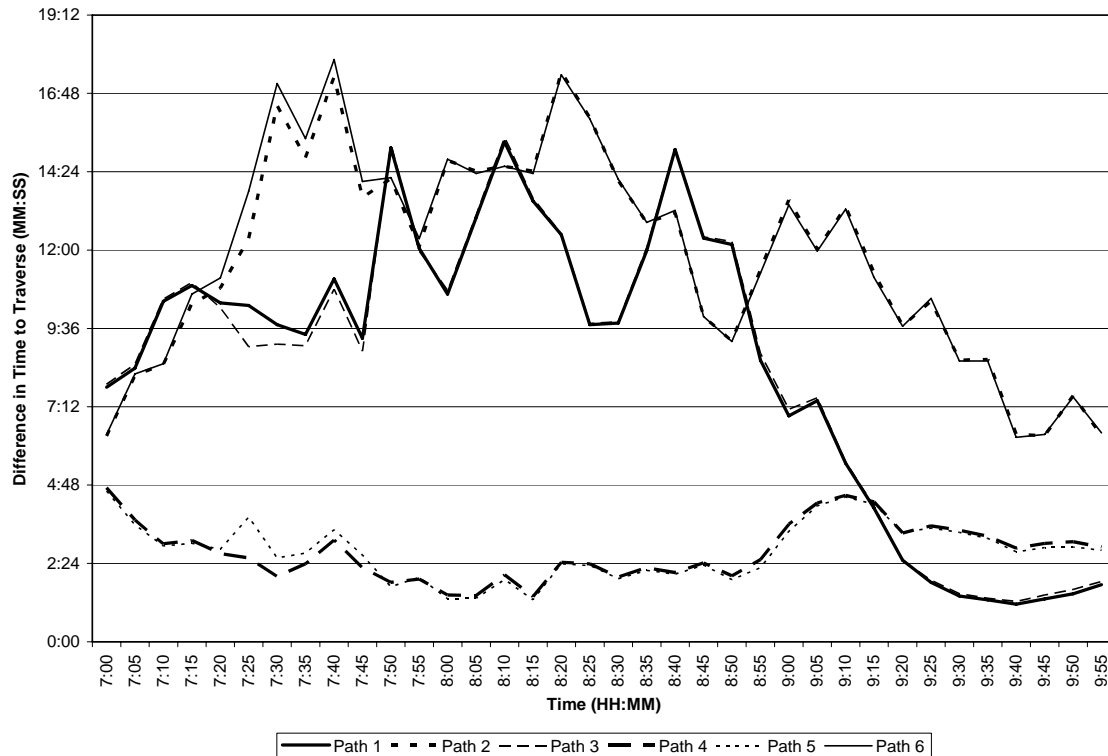
Graph 5 – Difference in time to traverse each path under current speeds and optimal speeds

driver could experience if he were to use a system that routed him on the fastest path rather than always taking the shortest path. Between 7:45a.m. and 8:45a.m., a driver can save as much as 5 minutes on his commute, which is rather significant given the fact that the shortest path would take him less than 15 minutes under optimal traffic conditions. A 33% savings can occur if a driver were to take the fastest path instead of the shortest path on his commute. The drop at 8:25a.m. shows that the shortest path re-emerged as the fastest path for about 10 minutes, then became slower again around 8:35a.m., as was also seen in Graph 3 above. Appendix F.4 shows the 30-second data for the difference in time to traverse the fastest path compared to the amount of time to traverse the shortest path.

Another analysis I performed on the amount of time to traverse paths was based on the difference in the amount of time it took to traverse each path under the current speeds and the optimal speeds. The bigger the difference implies that there is more

traffic on that path than a path with a smaller difference. Graph 5 shows the difference in time to traverse each of the paths based on 5-minute averages of the 30-second data. As can be assumed, paths 2 and 6 contain the largest difference in time on average, which are the two paths that have the largest stretch of the 405S. Paths 4 and 5 contain the smallest difference in time on average, which both consist of a long stretch of the 5S between the 118 and the 134 freeways. The shortest path (path 1) actually has a substantial delay, with a difference in time to traverse of between 9 and 15 minutes during rush hour. With the path taking less than 15 minutes at optimal speeds, a driver's commute could be more than doubled if he only takes the shortest path regardless of the current speeds. As the traffic begins to subside around 9:00a.m., the shortest path has less of a delay than all of the other paths, and therefore becomes the fastest path again. The difference in time to traverse each path under current speeds compared to optimal speeds without averaging the 30-second data is provided in Appendix F.5.

5.2 Volume to Speed Evaluation

The loop detectors that have been installed in the freeways by Caltrans report the volume of vehicles to pass the detector in a 30-second period and the amount of time that a vehicle is occupying the space above the loop detector. As mentioned in Chapter 5.1, [4] describes how to compute the speed based on the volume, occupancy, estimated length of a vehicle, and amount of time. Similarly, is it possible to determine the speed without having the occupancy? If so, then a question like the following could be answered: What will the traffic impact be if a new building is built that will attract 5000 more vehicles between the hours of 7:00a.m. and 9:00a.m. on weekdays?

The data from the loop detectors is also broken down by individual lanes, and then summarized for each location. The first analysis I performed on the data in relation to volume was to determine the average speed of the vehicles based on the volume of vehicles passing a loop detector in each lane in a 30-second period. I aggregated the data for each individual lane to produce Graph 6.

In Graph 6, the data when the volume is very low may not be an accurate representation of the speed at that point since the sample set is so small. Conversely, as the volume becomes very high (i.e. larger than 23), the data may not be as accurate either. Assuming the volume for a 30-second period is 25, one vehicle would have to pass that loop detector every 1.2 seconds, which would require the vehicles to be traveling at least 50 mph with very little spacing between them. However, the data for the volume between 8 and 18 vehicles / 30 seconds shows that the average speed of all of the lanes are very close to each other, differing by less than 10 mph at any given volume. From this data, the lane in which a driver travels will not significantly affect his commute over
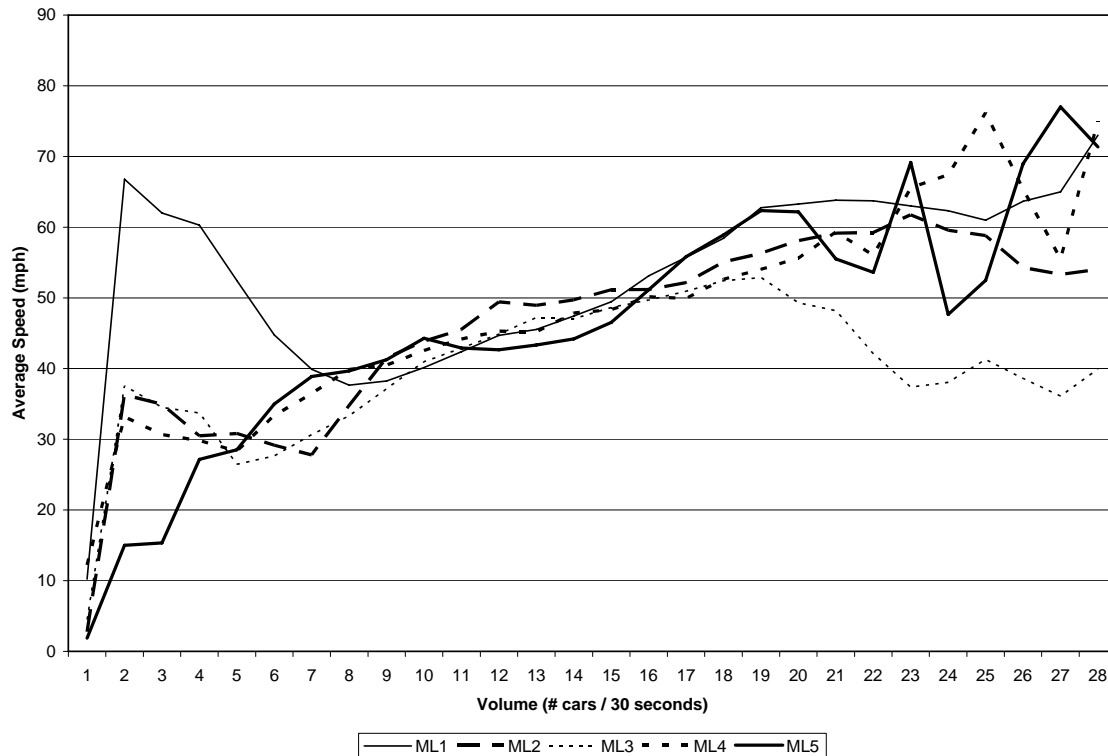
Graph 6 – Average speed per lane for the volume of vehicles passing a loop detector every 30 seconds

the long-run. Of course, the lane is significant if there is an accident or a lane closure, although if the sample is large enough, these incidents will not affect the average speeds of the lanes significantly.

Averaging the volumes for all of the lanes for different speeds produced a graph that is provided in Appendix F.6. I smoothed the data in this curve by averaging the volumes for every 5 mph, which produced Graph 7. This graph shows that in a 30-second period, the number of vehicles that pass a certain point increases until 30 mph, then decreases until 45 mph, then increases until 65 mph, then decreases for speeds greater than 65 mph. This can be explained by the spacing of the vehicles. When vehicles are driving slower, the drivers feel safer driving closer to the vehicle in front, but at higher speeds, drivers allow a greater distance between their vehicle and the vehicle in front of them.
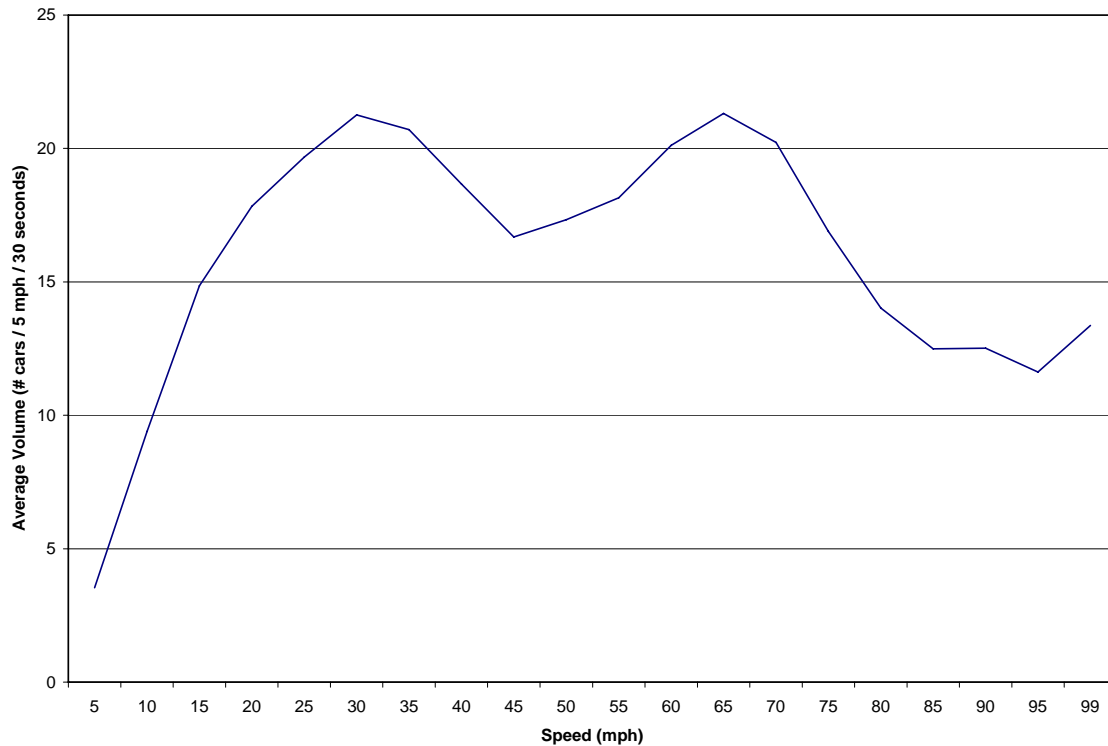
Graph 7 – Average 30-second volumes for all lanes at different speeds

Looking purely at the throughput (or volume) of vehicles, in [17], the authors claim that the maximum throughput in Los Angeles occurs at 60 mph; however, based on the data that I gathered from Caltrans between 7:00a.m. and 10:00a.m. on November 3, 2006, the maximum throughput can be achieved between 30 and 35 mph and at 65 mph. This difference may be explained by the fact that the authors in [17] gathered data from midnight to noon on September 1, 2000, which was a Friday, whereas my data was gathered solely during a heavy-commute time. Therefore, to maximize the throughput during heavy traffic times, drivers can travel around 30 mph or 65 mph, which will allow approximately 21 vehicles to pass a particular point during a 30-second period. If a freeway is experiencing the maximum throughput at 30 mph, each vehicle will have 62 feet of space before the vehicle in front of it. The 62 feet includes the length of the vehicle, which Caltrans estimates to be 18 feet on average [59]. That gives the vehicle 44
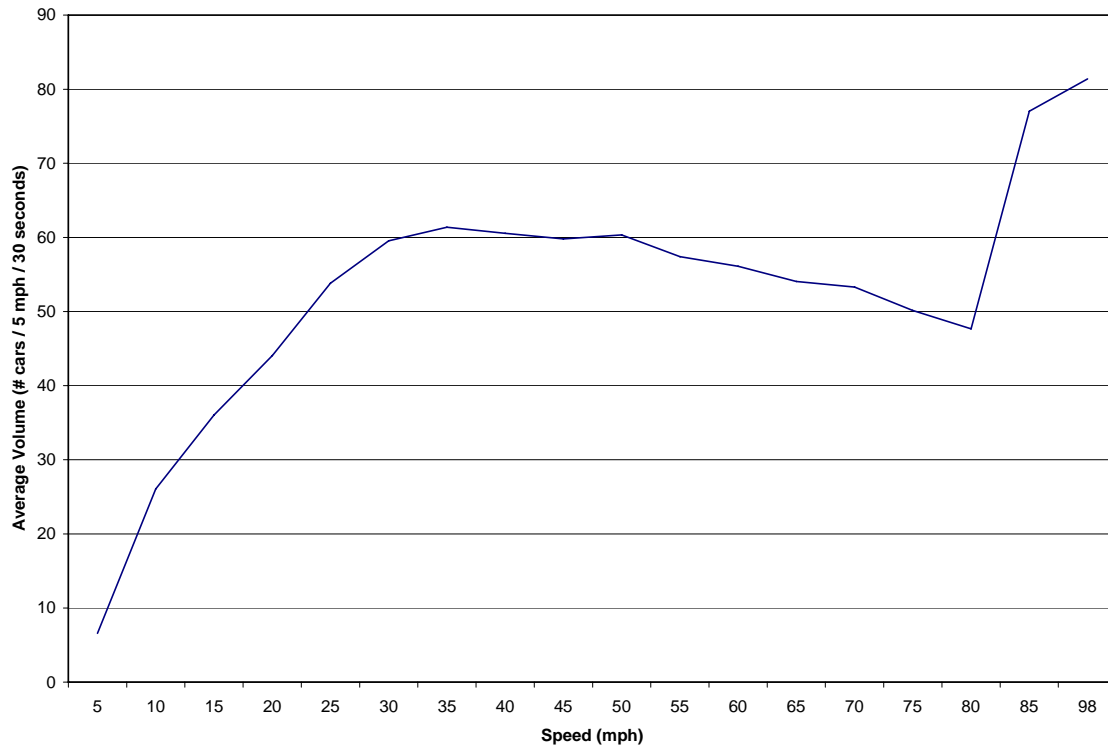
Graph 8 – Average 30-second volumes for summary data at different speeds

feet between itself and the vehicle in front of it. The United Kingdom Highway Code [76] suggests 75 feet of space to stop while traveling 30 mph. For a vehicle traveling 65 mph on a freeway that is experiencing the maximum throughput, each vehicle will have 118 feet of space before the vehicle in front of it. For a vehicle traveling 55 mph, the California Driver Handbook for 2007 suggests approximately 400 feet to stop [77]. In both of these cases, drivers in the Los Angeles area drive closer to the vehicle in front of them than is recommended by transportation organizations.

Considering just the summary data from the loop detectors, the average volume for every 5 mph at different speeds is shown in Graph 8. This graph shows that the average maximum throughput occurs between 30 and 50 mph, which can be attributed to the fact that vehicles that are traveling slower can travel closer to each other, although if they are traveling too slow, not as many vehicles will pass a point in a 30-second period.

As the speeds become greater than 70 mph, the data set was is too small to conclude that the values for the average volume are accurate. The reason for the wider range of maximum throughput with summary data as opposed to individual lane data is due to the fact that the maximum throughput for individual lanes occurs at either 30 mph or 65 mph, but when aggregating all the data together as summary data, those values are averaged, and a wider range of maximum throughput results. A graph of the average volume not aggregated by 5 mph increments is included in Appendix F.7.

Chapter 6

Conclusion and Future Work

In this thesis, I propose an architecture for gathering traffic data from individual vehicles rather than from hardware devices installed at discrete locations within a highway system. The vehicles report their speed and location to a central server that aggregates all the data and can act as an oracle that other vehicles can consult to determine the fastest path from their current location to their desired destination. To optimally produce fastest paths, I developed a new class of algorithms known as the Dynamic All-Pairs All-Paths algorithms. These algorithms execute faster than current shortest path and dynamic path algorithms because they take into account the static nature of freeway systems represented as graphs. All of the paths between all pairs of nodes in the graph can be computed off-line before the algorithms execute and then used to produce fastest paths more efficiently. The Dynamic All-Pairs All-Paths – Constant Update algorithm executes the fastest practically because updates to edges occur much more frequently than path queries.

I created a traffic simulator called FreeSim to test these and other algorithms. The current traffic simulators do not allow individual vehicles to autonomously communicate with each other or with the roadway. FreeSim is suited for Intelligent Transportation Systems (ITS) applications which assume that vehicles will be able to communicate continuously with other vehicles (V2V) or stationary objects (V2R), as has been developed in the IEEE 1609 suite of Wireless Access in Vehicular Environments (WAVE) Communications standards [78]. In addition, FreeSim is open-source and free to download from http://www.freewaysimulator.com.

To test fastest or shortest path algorithms, FreeSim has an Application Programming Interface (API) that allows programmers to quickly deploy their code and

test the efficiency against other fastest path algorithms. FreeSim also allows live data to be used as input to the simulator, such as that gathered by a transportation organization. I tested FreeSim with data gathered by loop detectors in Caltrans' District 7 (which includes Los Angeles and Ventura counties), then analyzed the results.

I computed the amount of time it would take to traverse different paths from a source node to a destination node, and determined that the shortest path is not always the fastest path from one location to another, especially during rush hour traffic. During increased traffic times, the second shortest path also did not prove to be the fastest path, but the fastest path changed regularly. This proves that there is a need for a system to route drivers along their fastest paths based on the current traffic conditions.

The shortest path I considered would take a driver just under 15 minutes while traveling at the speed limit. During the hours between 7:00a.m. and 10:00a.m. on Friday, November 3, 2006, a driver could have saved up to 5 minutes if he took a path other than the shortest path. Although this may not seem like a lot, this is a time savings of approximately 33%. Not surprising to Los Angeles drivers, the reason the second shortest path was not the second fastest path was because it included the 405S, which is a very heavily-trafficked freeway during rush hour.

I then evaluated the volume of vehicles passing each loop detector compared to the speed of the vehicles at each loop detector. I determined that there are two different speeds at which Los Angeles will experience maximum throughput on the freeway system during rush hour – at 30 mph and 65 mph. At both of these speeds, there are approximately 21 vehicles passing the loop detector in a 30 second period. At 30 mph, each vehicle will have 44 feet between itself and the vehicle in front of it. At 65 mph,

each vehicle will have 118 feet between itself and the vehicle in front of it. These following-distances are far less than what is recommended by departments of transportation worldwide.

The research on intelligent transportation systems is far from reaching its peak. As long as there is traffic, there will be research attempting to improve it. I would like to answer the question of how a new building will affect the flow of traffic in a freeway system, or how an athletic event that attracts a certain number of vehicles will affect the speed and throughput of a freeway system. Also, if the data is being gathered from individual vehicles in a continuous manner rather than by hardware devices at discrete locations, I would like to determine what percentage of vehicles need to be transmitting their speed and location to still be able to accurately route vehicles along fastest paths. If this percentage is less than 100% (which I believe it is), aggregation of this data could occur in a distributed fashion with the vehicles communicating with each other. That would lower the required bandwidth for the application and improve the response time from the system.

Traffic predication algorithms have already been proposed, and I would like to implement those in FreeSim and determine if there are any improvements that can be made. To more accurately determine the amount of time to travel along a path, prediction algorithms could aid in noting the changing traffic patterns based on the time of the day, the day of the week, the weather, lane conditions, event times, etc.

Further, I would like to try to tie in the real-time data gathered by Caltrans or another department of transportation into FreeSim to provide commuters with real-time routing based on the current traffic data. FreeSim would then provide the majority of the

functionality of sites such as Sigalert.com [5], although it would have the added advantage of being open-source and allowing researchers and programmers all around the world to test their algorithms against live, real-time traffic data.

Traffic is definitely a problem that not only affects urban areas, but is now extending to more suburban populations as well. Adding lanes or building more freeways is a very costly solution that takes many years to complete. Alternative solutions are necessary to deal with the traffic problems of today, and intelligent transportation systems attempt to solve the problem. My research focuses on providing a simulator application known as FreeSim for analyzing traffic data and validating algorithms and applications before deploying in a large-scale transportation system.
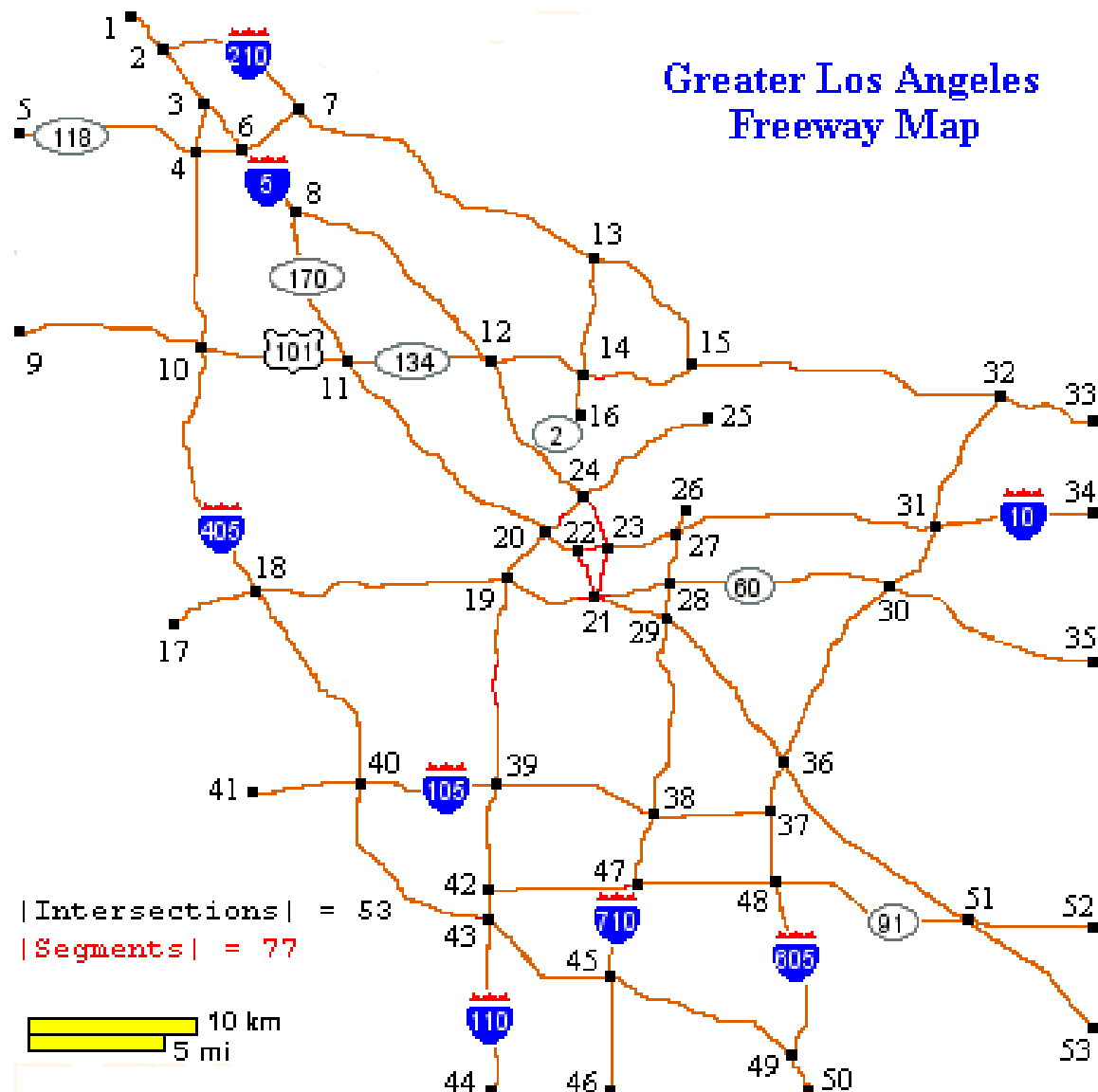
Appendix A
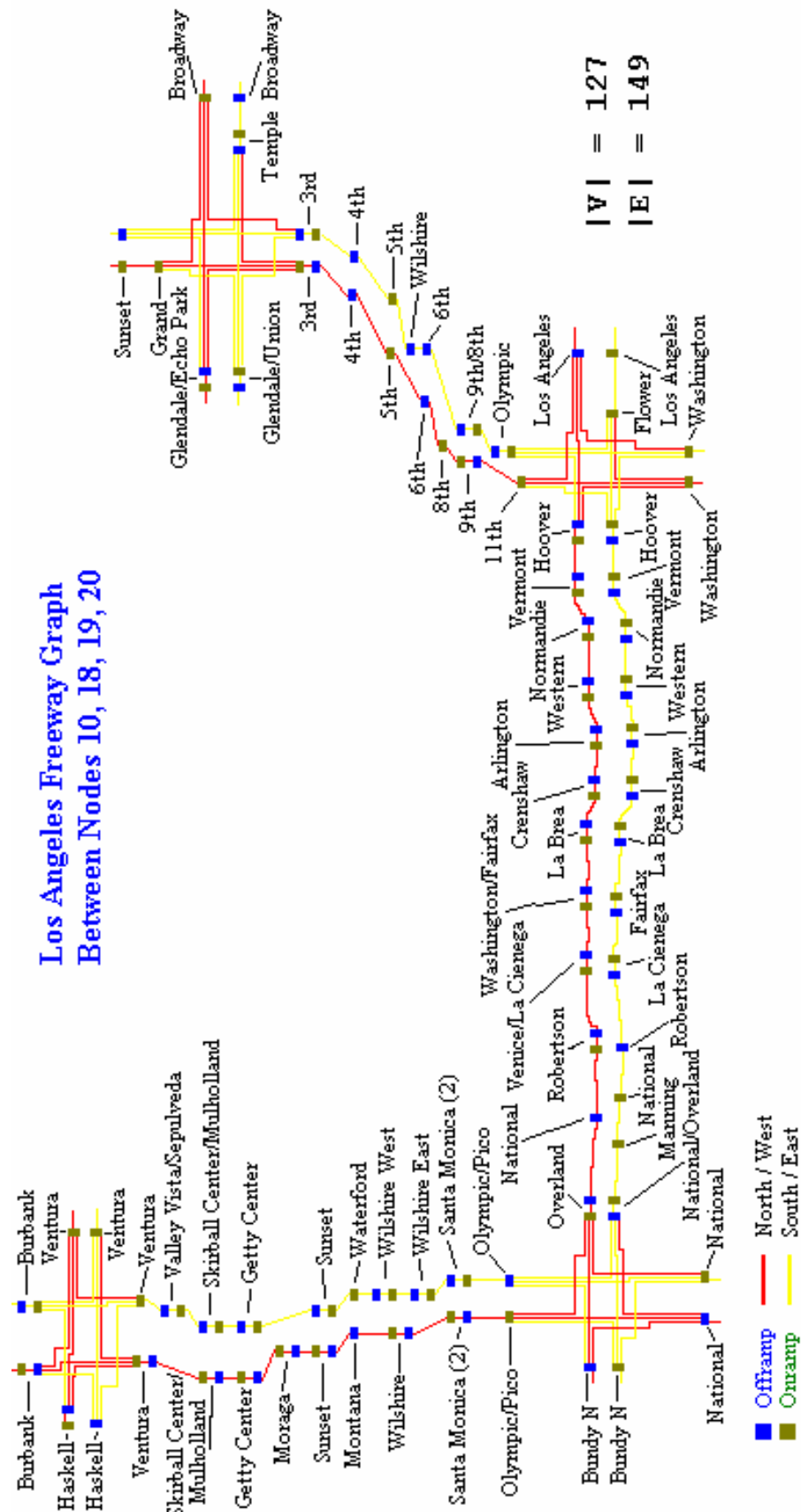

Maps

A.1 Sigalert.com Screenshot [5]

Greater Los Angeles
Freeway Map

|Intersections| = 53
|Segments| = 77

10 km
5 mi

A.3 Los Angeles Freeway Graph with Off-Ramps and On-Ramps

Los Angeles Freeway Graph
Between Nodes 10, 18, 19, 20

|V| = 127
|E| = 149

— North / West
— South / East

■ Offramp
■ Onramp

Appendix B

Naïve and Dynamic All-Pairs Shortest Path Algorithms

B.1 Floyd-Warshall's Algorithm

```
// W is a matrix of size nxn where
// w_ij = 0 if i=j, w_ij = w(i,j) if (i,j) ε E, wij = ∞ if (i,j) E
// D is a matrix of size nxn that contains the shortest path
// weights during each iteration of the loop
// D^(n) will contain the shortest paths between all vertices
FLOYD-WARSHALL(W) {
  n = num_rows(W)
  D^(0) <- W̄
  for(k=1..n) {
    for(i=1..n) {
      for(j=1..n) {
        d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
      }
    }
  }
  return D^(n)
}
```

B.2 Dijkstra's Algorithm

```
// G is a directed graph with vertices V and edges E
// w is the set of weights on the edges
// s is the vertex from which to find the shortest paths to every
// other vertex
// DIJKSTRA produces the paths with minimum weight to every node
// in the graph from source s with the predecessor node
DIJKSTRA(G, w, s) {
  INITIALIZE-SINGLE-SOURCE(G, s)
  S <- Φ
  Q <- V[G]
  while (Q ≠ Φ) {
    u = EXTRACT-MIN(Q)
    for (each vertex v ∈ adjacency_vertices(u)) {
      RELAX(u, v, w)
    }
  }
}


// G is a directed graph with vertices V and edges E
// s is the vertex to initialize
// d[v] is an upper bound on the weight of the shortest path from
// source s to destination v
// Π[v] is the predecessor node that must be traversed in the
// shortest path from source s to destination v
INITIALIZE-SINGLE-SOURCE(G, s) {
  for (each vertex v ∈ V[G]) {
    d[v] <- ∞
    Π[v] <- null
  }
  d[s] = 0
}


// source vertex u, destination vertex v, set of weights w
// this function determines if there is a shorter way to get
// to v by going through node u
// if so, update the shortest path weight d[v] and the
// predecessor node
RELAX(u, v, w) {
  if (d[v] > d[u] + w(u, v)) {
    d[v] = d[u] + w(u, v)
    Π[v] = u
  }
}
```

B.3 Bellman-Ford Algorithm

```
// G is a directed graph with vertices V and edges E
// w is the set of weights on the edges
// s is the vertex from which to find the shortest paths to every
// other vertex
// BELLMAN-FORD produces the paths with minimum weight to every
// node in the graph from source s with the predecessor node
BELLMAN-FORD(G, w, s) {
  INITIALIZE-SINGLE-SOURCE(G, s)
  for (i=1..|V[G]|-1) {
    for (each edge (u, v) ∈ E[G]) {
      RELAX(u, v, w)
    }
  }
  for (each edge (u, v) ∈ E[G]) {
    if (d[v] > d[u] + w(u, v)) {
      return false // graph contains negative weight cycles
    }
  }
  return true
}
```

B.4 Johnson's Algorithm

```
// G is a graph with vertices V and edges E
// returns D, which is a |V|x|V| matrix where d_ij contains
// the shortest distance between nodes i and j
JOHNSON(G) {
  Compute G' where V[G'] = V[G] ∪ {s} where s is a new node
                  E[G'] = E[G] ∪ {(s, v) : v ∈ V[G]}
                  W(s, v) = 0 ∀ v ∈ V[G]
  If (BELLMAN-FORD(G', w, s)) {
    return // negative weight cycles exist
  }
  for (each vertex v ∈ V[G']) {
    h(v) = δ(s, v)
    for (each edge (u, v) ∈ E[G']) {
      w'(u, v) = w(u, v) + h(u) - h(v)
    }
    for (each vertex u ∈ V[G]) {
      // the next line will compute δ'(u, v) ∀ v ∈ V[G]
      DIJKSTRA(G, w', u)
      For (each vertex v ∈ V[G]) {
        d_uv = δ'(u, v) + h(v) - h(u)
      }
    }
  }
  return D
}
```

Appendix C

All-Pairs All-Paths Data Structures and Code

C.1 All-Pairs All-Paths Pseudo-Code

```
ALL-PAIRS-ALL-PATHS() {
  for all v in V[G] {
    for all u in V[G] {
      if (u != v) {
        ALL-PATHS(v, v, u)
      }
    }
  }
}

// adapted from [16]
ALL-PATHS(origSrc, src, dest) {
  w = w(src, dest) ? w(src, dest) : 0;
  if (w > 0) {
    add (src, dest) to path p
    add p to path vector
    remove last edge from p
  }
  for all v such that (src, v) ε E[G] {
    if (w(src, v) > 0) {
      add (src, v) to path p
      w(src, v) = -w(src, v)
      ALL-PATHS(origSrc, v, dest)
      w(src, v) = -w(src, v)
    }
  }
}
```

C.2 All-Pairs All-Paths Pre-Computed – Constant Update Data Structures and Pseudo-Code

```
Edge {
  int node1; // start node of edge
  int node2; // dest node of edge
  float time; // time to traverse edge
  // list of paths to which this edge belongs
  Path [] paths; // worst case size - O(V2m)

  // update function passing new time to traverse edge
  void update(float newTime) {
    this.time = newTime;
  }
}

Path {
  Edge [] edges; // list of edges that make up the path
  float time; // time to traverse path
}

PathList {
  int sourceNode; // source node for paths in this structure
  int destNode; // destination node for paths in this structure
  Path [] paths; // list of all paths from source to dest node

  // function returning fastest path from source to dest node
  Path getFastestPath() {
    int minPath = 0;
    for (int i=0; i < paths.size; i++) {
      paths[i].time = 0;
      for (int j=0; j < paths[i].edges.size; j++) {
        paths[i].time += paths[i].edges[j].time;
      }
      if (paths[i].time < paths[minPath].time) {
        minPath = i;
      }
    }
    return paths[minPath];
  }
}
```

C.3 All-Pairs All-Paths Pre-Computed – Constant Query Data Structures and Pseudo-Code

```
Edge {
  int node1; // start node of edge
  int node2; // dest node of edge
  float time; // time to traverse edge
  // list of paths to which this edge belongs
  Path [] paths; // worst case size - O(V2m)

  // update function passing new time to traverse edge
  void update(float newTime) {
    float diff = this.time - newTime;
    this.time = newTime;
    for (int i=0; i < paths.size; i++) {
      paths[i].pathList.bst.remove(paths[i]);
      paths[i].time -= diff;
      paths[i].pathList.bst.insert(paths[i]);
    }
  }
}

Path {
  Edge [] edges; // list of edges that make up the path
  float time; // time to traverse path
  PathList pathList; // PathList to which this path belongs
}

PathList {
  int sourceNode; // source node for paths in this structure
  int destNode; // destination node for paths in this structure
  Path [] paths; // list of all paths from source to dest node
  BinarySearchTree bst; // BST to store paths in a logical order

  // function returning fastest path from source to dest node
  Path getFastestPath() {
    return bst.minimum();
  }

}
```

C.4 All-Pairs All-Paths Pre-Computed – Hybrid Data Structures and Pseudo-Code

```
Edge {
  int node1; // start node of edge
  int node2; // dest node of edge
  float time; // time to traverse edge
  // list of paths to which this edge belongs
  Path [] paths; // worst case size - O(V2m)

  // update function passing new time to traverse edge
  void update(float newTime) {
    float diff = this.time - newTime;
    this.time = newTime;
    for (int i=0; i < paths.size; i++) {
      paths[i].time -= diff;
    }
  }
}


Path {
  Edge [] edges; // list of edges that make up the path
  float time; // time to traverse path
}


PathList {
  int sourceNode; // source node for paths in this structure
  int destNode; // destination node for paths in this structure
  Path [] paths; // list of all paths from source to dest node

  // function returning fastest path from source to dest node
  Path getFastestPath() {
    int minPath = 0;
    for (int i=0; i < paths.size; i++) {
      if (paths[i].time < paths[minPath].time) {
        minPath = i;
      }
    }
    return paths[minPath];
  }
}
```

Appendix D

CalTrans Available Data

D.1 Annual Average Daily Traffic - Santa Monica 10 Freeway 2004 Traffic Counts

Postmile	Description	West Peak Hour	West Peak Month	West AADT	East Peak Hour	East Peak Month	East AADT
2.16	JCT. RTE 1,LINCOLN BOULEVARD				11700	159000	157000
3.21	20TH STREET/ CLOVERFIELD BLVD	11700	159000	157000	14400	199000	193000
4.24	CENTINELA AVENUE/PICO BOULEVARD	14400	199000	193000	14600	208000	203000
4.51	BUNDY DRIVE	14600	208000	203000	17200	250000	246000
5.45	JCT. RTE. 405,SAN DIEGO FREEWAY	17200	250000	246000	19700	272000	264000
6.4	OVERLAND AVENUE	19700	272000	264000	17200	267000	261000
7.21	MANNING AVENUE/ NATIONAL BOULEVARD	17200	267000	261000	19000	286000	280000
7.92	ROBERTSON BOULEVARD	19000	286000	280000	19200	281000	275000
8.97	JCT. RTE 187; VENICE/LA CIENEGA BOULEVARDS	19200	281000	275000	18200	283000	279000
10.43	LA BREA AVENUE	18200	283000	279000	19500	295000	291000
11.39	CRENSHAW BOULEVARD	19500	295000	291000	20200	312000	307000
12.32	ARLINGTON AVENUE	20200	312000	307000	20300	322000	316000
12.82	WESTERN AVENUE	20300	322000	316000	20700	332000	325000
13.3	NORMANDIE AVENUE	20700	332000	325000	20700	334000	328000
13.8	VERMONT AVENUE	20700	334000	328000	21200	340000	332000
14.25	HOOVER STREET	21200	340000	332000	21100	336000	328000
14.84	JCT. RTE. 110, HARBOR FREEWAY	21100	336000	328000	16400	241000	236000

D.2 Loop Detector Data


```
Runtime:  01-08-2007,   10:21                              TRAFFIC  DATA  REPORT

                                                          30 Second Loop Data
                                        FROM:  11-03-2006          TO:  11-03-2006
                                               07:00:00                 07:15:00


VDS DESCRIPTION:                          SHERMAN WAY              Main Line /
              LA-405-S,    PM: 42.26                               HOV
                                          1

NOV-03-
2006        HOV 1         ML 1         ML 2         ML 3         ML 4       ML STATION
FRIDAY                                                                      TOT   AVG   EST
            VOL OCC SPD ST VOL OCC SPD ST VOL OCC SPD ST VOL OCC SPD ST VOL OCC SPD ST  VOL   OCC   SPD


07:00:30  14 21.1 23 1   10 9.7 35 1   14 12.9 37 1   13 18.2 29 1   13 29.9 21 1   50 17.7 30
07:01:00  13 18.8 23 1   13 12.0 37 1   14 13.9 34 1   12 16.5 30 1   12 18.8 30 1   51 15.3 33
07:01:30  16 16.7 33 1   16 16.3 33 1   15 16.2 31 1   12 15.2 32 1   13 19.2 32 1   56 16.7 32
07:02:00  14 12.0 40 1   12 16.5 25 1   14 14.2 34 1   10 11.6 35 1   11 19.8 26 1   47 15.5 30
07:02:30  13 15.2 29 1   13 15.1 29 1   15 15.0 34 1   12 20.0 25 1   14 23.2 29 1   54 18.3 29
07:03:00  14 25.8 18 1   12 11.0 37 1   13 12.8 35 1   14 17.4 33 1   15 39.0 18 1   54 20.1 30
07:03:30  13 33.0 13 1   13 11.8 37 1   11 8.9 42 1    9 14.5 25 1    10 33.5 14 1   43 17.2 31
07:04:00  12 23.4 17 1   14 15.7 30 1   15 13.1 39 1   10 14.5 28 1   13 15.4 40 1   52 14.7 35
07:04:30  13 25.0 18 1   13 14.8 30 1   14 13.3 36 1   13 17.0 31 1   17 17.9 45 1   57 15.7 36
07:05:00  12 34.3 12 1   12 11.2 37 1   12 12.8 32 1   18 18.8 39 1   14 24.6 27 1   56 16.8 34


 5mt:  134          128          137          123          132          520
07:05:30  13 19.5 23 1   12 13.6 30 1   16 18.4 30 1   11 16.0 28 1   13 23.4 27 1   52 17.8 29
07:06:00  14 19.3 25 1   14 15.5 31 1   15 14.7 35 1   15 17.4 35 1   10 12.8 37 1   54 15.1 34
07:06:30  11 9.8 38 1    14 13.1 36 1   14 15.7 30 1   12 13.8 36 1   14 18.1 37 1   54 15.2 35
07:07:00  15 15.5 33 1   14 17.1 28 1   16 17.3 32 1   11 20.9 22 1   14 20.1 33 1   55 18.8 29
07:07:30  9 8.4 36 1     13 19.7 23 1   13 18.2 24 1   10 20.4 20 1   12 35.8 16 1   48 23.5 21
07:08:00  9 11.1 28 1    5 43.9 4 1     3 52.9 2 2     6 39.8 6 1     5 47.9 5 1     19 46.1 5
07:08:30  11 12.0 31 1   11 22.2 17 1   10 22.9 15 1   10 27.9 15 1   8 44.4 9 1     39 29.3 14
07:09:00  9 9.8 31 1     14 18.9 25 1   14 19.2 25 1   12 18.5 26 1   10 30.1 16 1   50 21.7 24
07:09:30  10 7.6 45 1    14 16.1 30 1   10 11.9 29 1   12 22.9 21 1   14 32.6 20 1   50 20.9 25
07:10:00  14 11.3 42 1   14 13.0 37 1   14 18.2 26 1   11 24.0 19 1   9 40.0 11 1    48 23.8 25
 5mt:  115          125          125          110          109          469
```

75

Appendix E

FreeSim

E.1 FreeSim Screenshot with Freeway Intersection Nodes

E.2 FreeSim Screenshot with Loop Detector Nodes

## E.3 Sample FreeSim Input File from Caltrans Data

Caltrans Northwest Loop Detectors

```
# VEHICLE_COMMAND_NEW_VEHICLE|VEHICLE_ID|TIME_TO_EXECUTE|START_NODE|
START_FREEWAY|DEST_NODE|DEST_FREEWAY
70|0|10|S FERNANDO 1|LA-5-S|VERMONT|LA-101-S

# VEHICLE_COMMAND_GET_TIME_TO_TRAVERSE|VEHICLE_ID|TIME_TO_EXECUTE|
START_NODE|START_FREEWAY_NAME|...|END_NODE|END_FREEWAY_NAME
76|0|10|S FERNANDO 1|LA-5-S|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|CHATSWORTH|LA-5-S|RTE 118 CN|LA-5-
S|OSBORNE 2|LA-5-S|ARLETA|LA-170-S|ROSCOE 2|LA-170-S|ROSCOE 1|LA-170-S|SHERMAN WAY|LA-170-S|VICTORY
2|LA-170-S|VICTORY 1|LA-170-S|OXNARD|LA-170-S|BURBANK|LA-170-S|MAGNOLIA|LA-170-S|VENTURA|LA-101-
S|LANKERSHIM|LA-101-S|UNIVERSAL 2|LA-101-S|BARHAM|LA-101-S|CAHUENGA 2|LA-101-S|CAHUENGA 1|LA-101-
S|ARGYLE|LA-101-S|HOLLYWOOD|LA-101-S|SUNSET|LA-101-S|WESTERN|LA-101-S|SANTA MONICA|LA-101-
S|NORMANDIE|LA-101-S|MELROSE|LA-101-S|VERMONT|LA-101-S

# VEHICLE_COMMAND_UPDATE_EDGE|VEHICLE_ID|TIME_TO_EXECUTE|START_NODE|START_FREEWAY_NAME|
DEST_NODE|DEST_FREEWAY_NAME|NEW_SPEED
72|0|10|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|78
72|0|10|CHATSWORTH|LA-5-S|RTE 118 CN|LA-5-S|69
72|0|10|RTE 118 CN|LA-5-S|OSBORNE 2|LA-5-S|67
72|0|10|RTE 118 CN|LA-5-S|DEVONSHIRE 2|LA-405-S|67
72|0|10|OSBORNE 2|LA-5-S|SHELDON|LA-5-S|41
72|0|10|OSBORNE 2|LA-5-S|ARLETA|LA-170-S|36
72|0|10|SHELDON|LA-5-S|LANKERSHIM|LA-5-S|36
72|0|10|PENROSE|LA-5-S|SUNLAND|LA-5-S|56
72|0|10|SUNLAND|LA-5-S|ROSCOE|LA-5-S|43
72|0|10|ROSCOE|LA-5-S|HOLLYWOOD WAY|LA-5-S|45
72|0|10|HOLLYWOOD WAY|LA-5-S|BUENA VISTA|LA-5-S|24
72|0|10|BUENA VISTA|LA-5-S|LINCOLN|LA-5-S|26
72|0|10|LINCOLN|LA-5-S|BURBANK 2|LA-5-S|40
72|0|10|BURBANK 1|LA-5-S|VERDUGO|LA-5-S|65
72|0|10|VERDUGO|LA-5-S|ALAMEDA 2|LA-5-S|56
72|0|10|WESTERN 2|LA-5-S|WESTERN 1|LA-5-S|72
72|0|10|WESTERN 1|LA-5-S|VICTORY TR|LA-5-S|23
72|0|10|RT 405 CN|LA-101-S|SEPULVEDA|LA-101-S|68
72|0|10|SEPULVEDA|LA-101-S|VAN NUYS|LA-101-S|69
72|0|10|VAN NUYS|LA-101-S|WOODMAN|LA-101-S|65
72|0|10|LAUREL CANYON|LA-101-S|VENTURA|LA-101-S|72
72|0|10|FOREST LAWN|LA-134-W|BUENA VISTA|LA-134-W|58
72|0|10|ARLETA|LA-170-S|ROSCOE 2|LA-170-S|65
72|0|10|VICTORY 1|LA-170-S|OXNARD|LA-170-S|56
72|0|10|OXNARD|LA-170-S|BURBANK|LA-170-S|22
72|0|10|BURBANK|LA-170-S|MAGNOLIA|LA-170-S|14
72|0|10|MAGNOLIA|LA-170-S|VENTURA|LA-101-S|18
72|0|10|DEVONSHIRE 2|LA-405-S|RTE 118 CN|LA-5-S|65
72|0|10|DEVONSHIRE 2|LA-405-S|DEVONSHIRE 1|LA-405-S|66
72|0|10|DEVONSHIRE 1|LA-405-S|NORDHOFF OFF|LA-405-S|58
72|0|10|NORDHOFF OFF|LA-405-S|NORDHOFF ON|LA-405-S|44
72|0|10|NORDHOFF ON|LA-405-S|ROSCOE|LA-405-S|45
72|0|10|ROSCOE|LA-405-S|STAGG ST|LA-405-S|30
72|0|10|STAGG ST|LA-405-S|SHERMAN WAY 2|LA-405-S|28
72|0|10|SHERMAN WAY 2|LA-405-S|SHERMAN WAY 1|LA-405-S|36
72|0|10|SHERMAN WAY 1|LA-405-S|VICTORY 2|LA-405-S|30
72|0|10|VICTORY 1|LA-405-S|OXNARD ST|LA-405-S|20
72|0|10|OXNARD ST|LA-405-S|BURBANK 2|LA-405-S|23
72|0|10|BURBANK 2|LA-405-S|BURBANK 1|LA-405-S|20
72|0|10|BURBANK 1|LA-405-S|RT 405 CN|LA-101-S|21
72|0|10|BURBANK 1|LA-405-S|VENTURA|LA-405-S|22
72|0|10|WOODCREST|LA-405-S|ROYAL RIDGE|LA-405-S|31
72|0|10|BEL AIR CR|LA-405-S|GETTY / SEPULVED|LA-405-S|30
72|0|10|GETTY / SEPULVED|LA-405-S|MORAGA|LA-405-S|23
72|0|10|MORAGA|LA-405-S|SUNSET WB / CHUR|LA-405-S|36
```
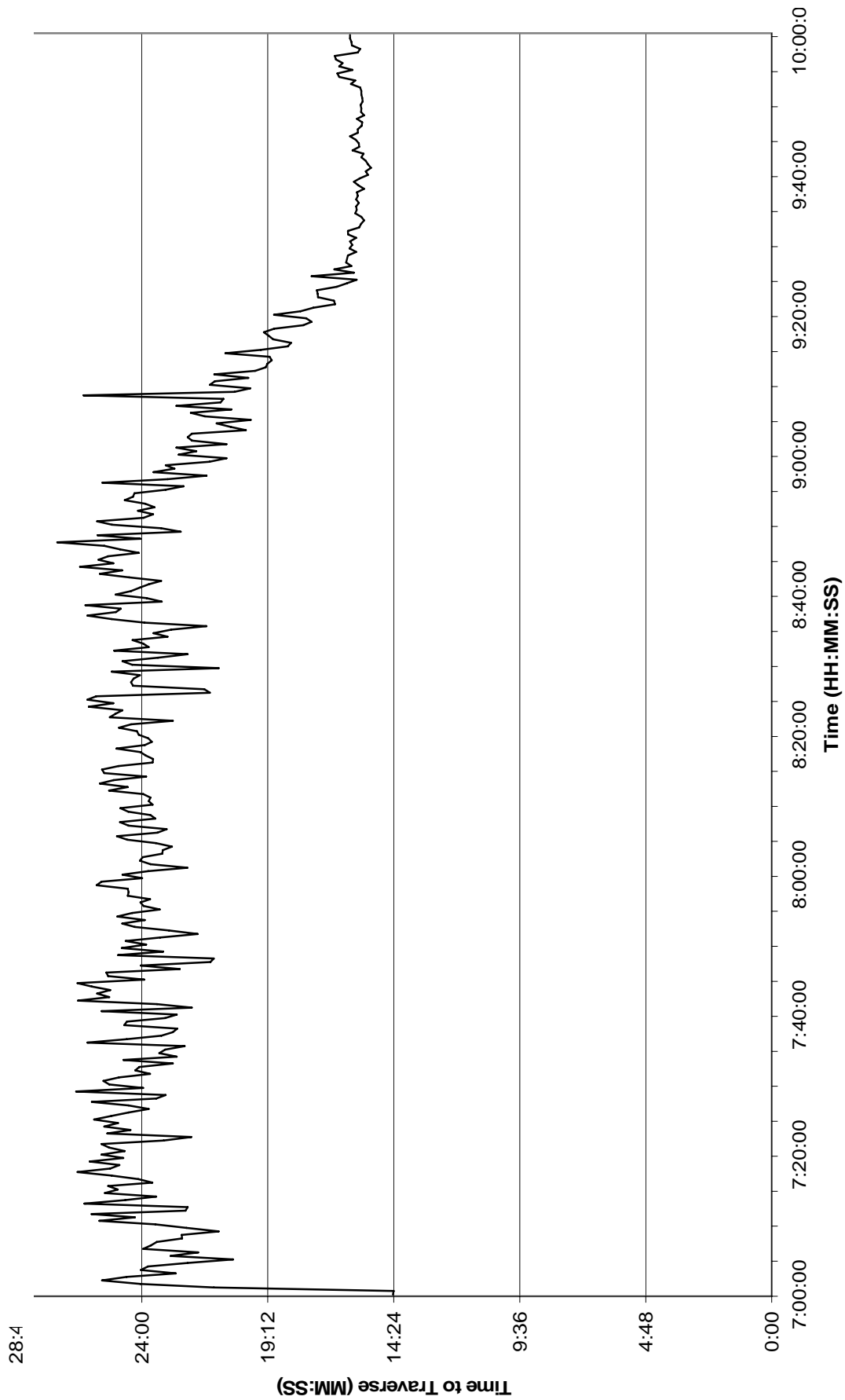
Appendix F

Travel Time Graphs from Caltrans Data

F.1 Time to Traverse Shortest Path with 30-Second Data – from 5S at San Fernando 1 to 101S at Vermont on 2006-11-03
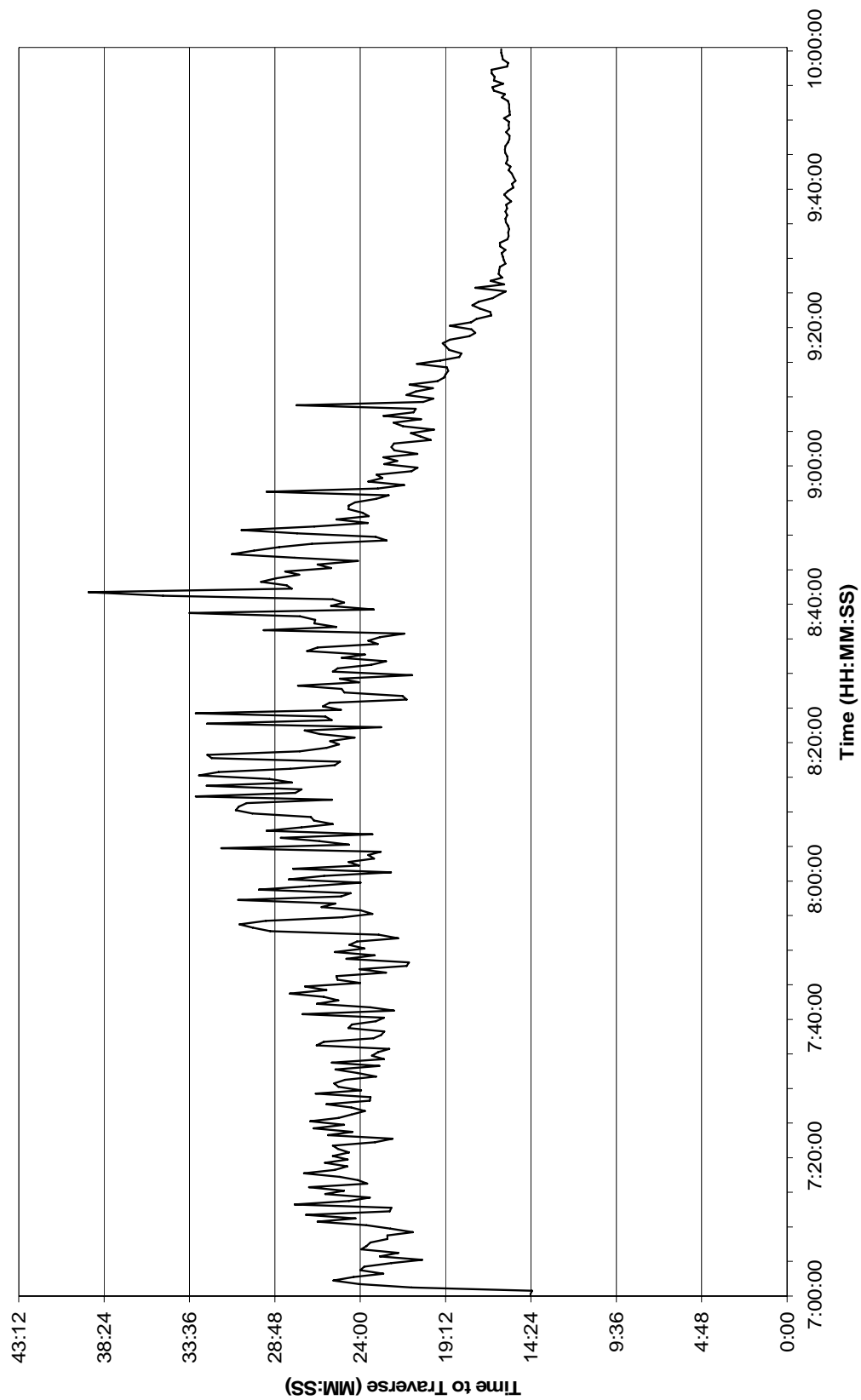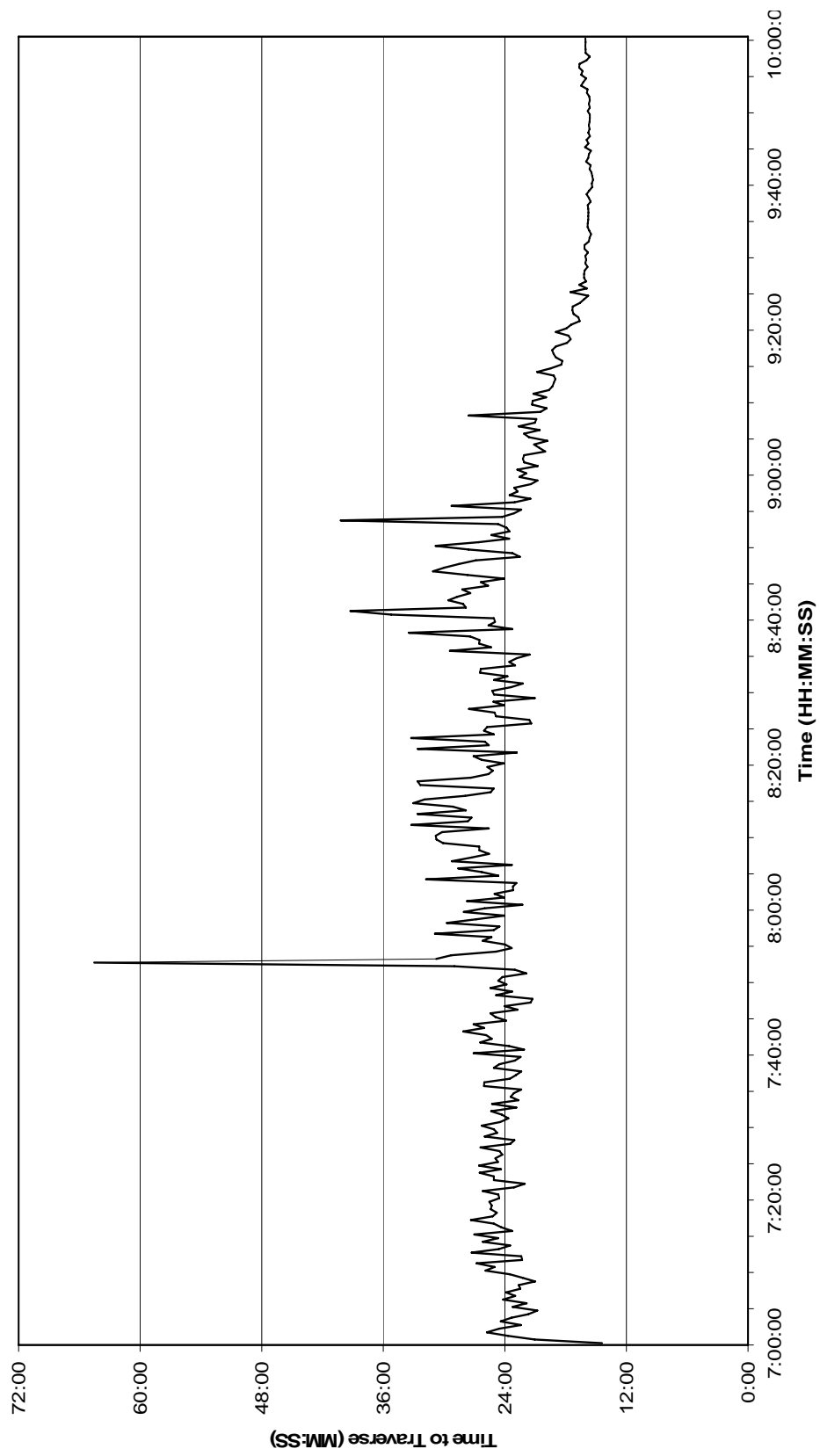
F.2 Time to Traverse Fastest Path with 30-Second Data – from 5S at San Fernando 1 to 101S at Vermont on 2006-11-03
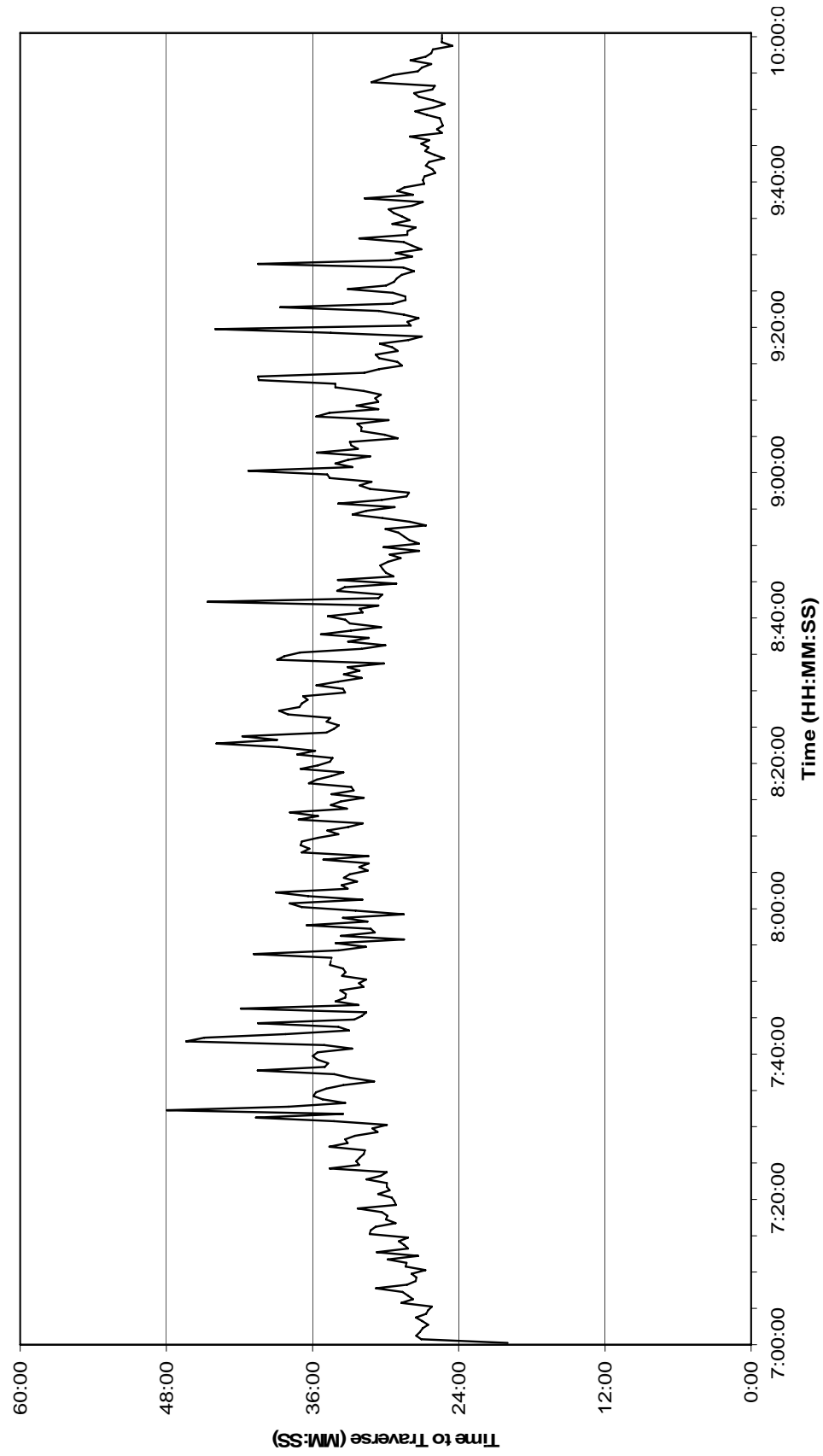
F.3 Time to Traverse Each Path with 30-Second Data – from 5S at San Fernando 1 to 101S at Vermont on 2006-11-03
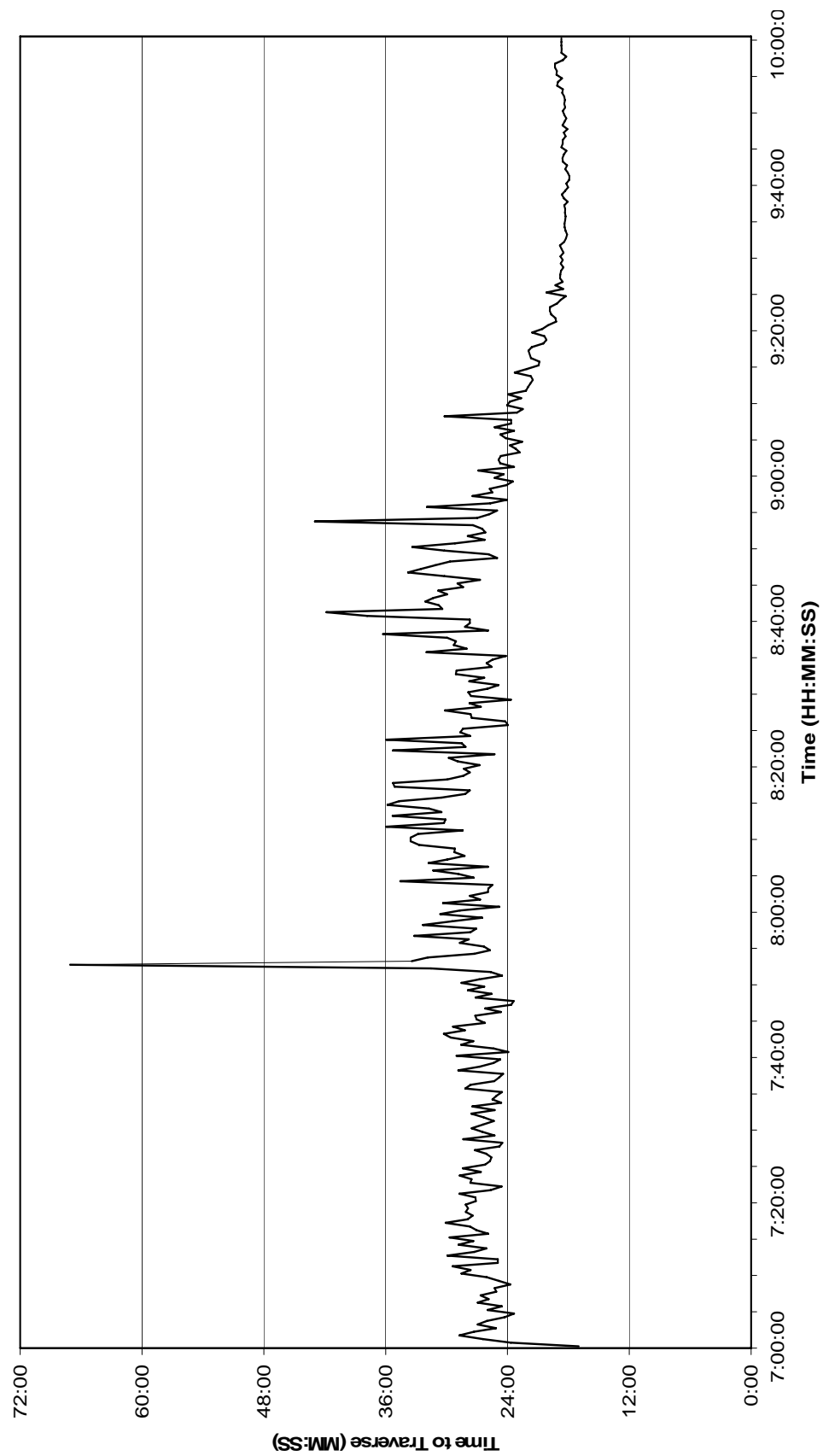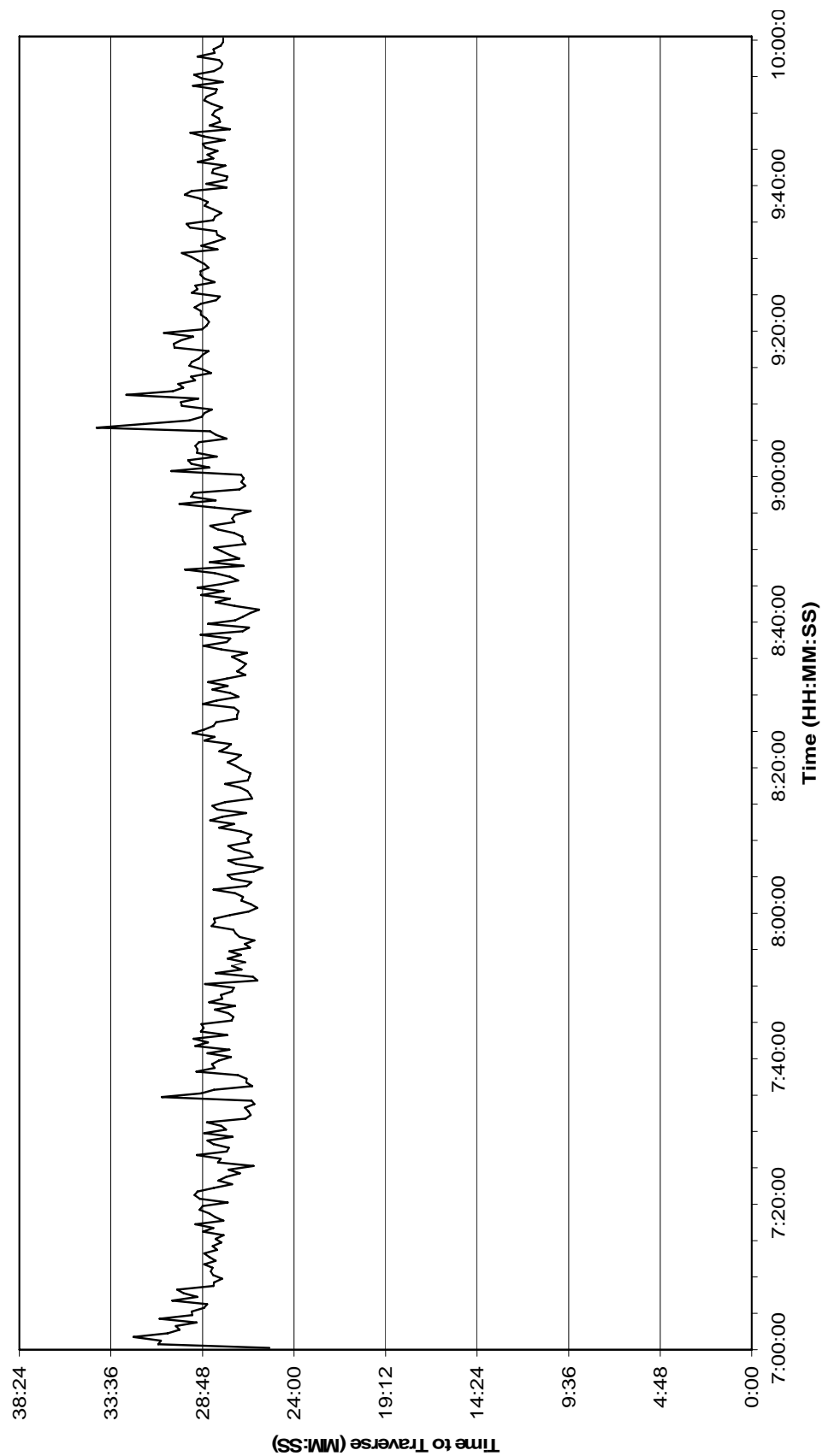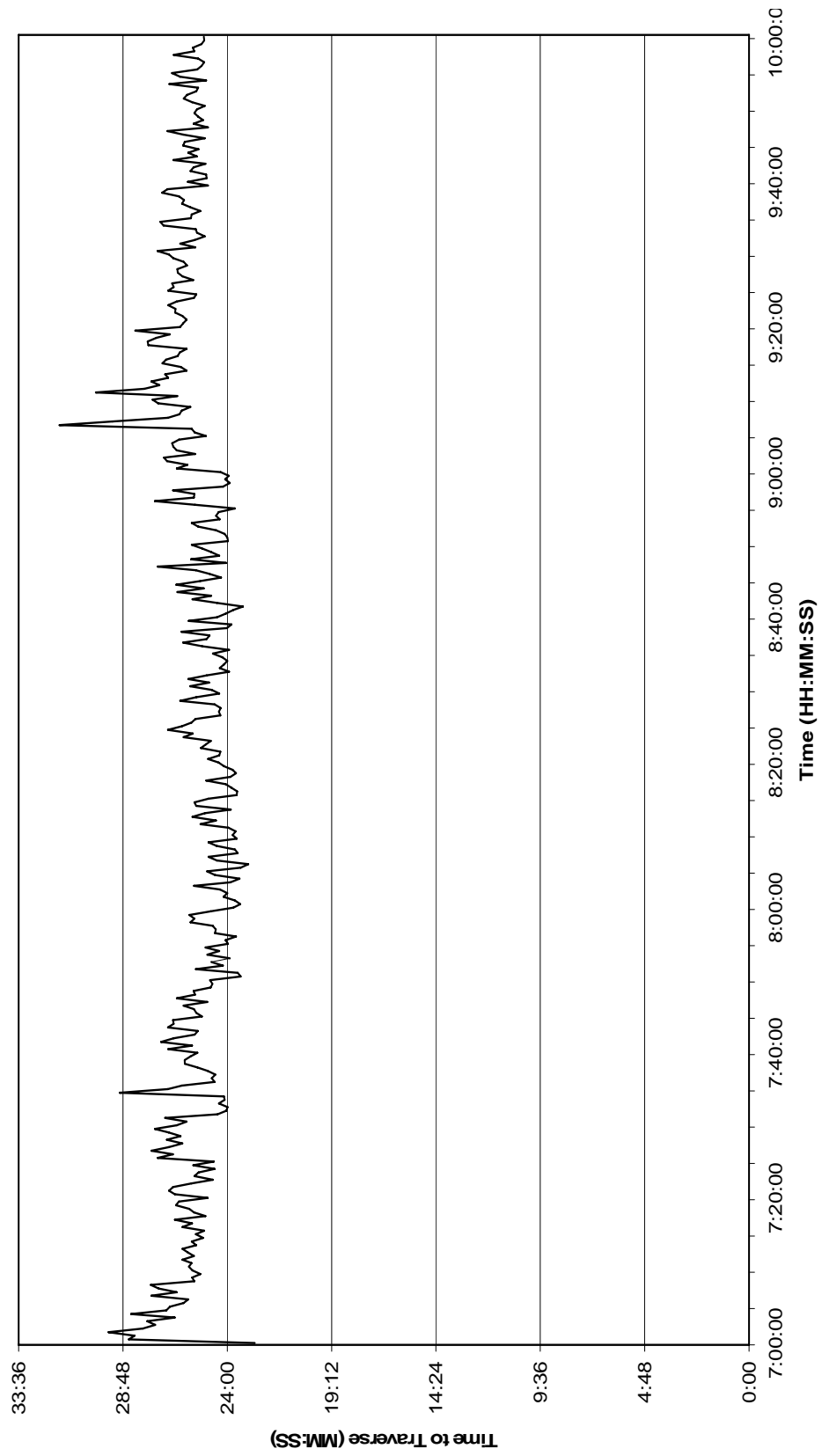
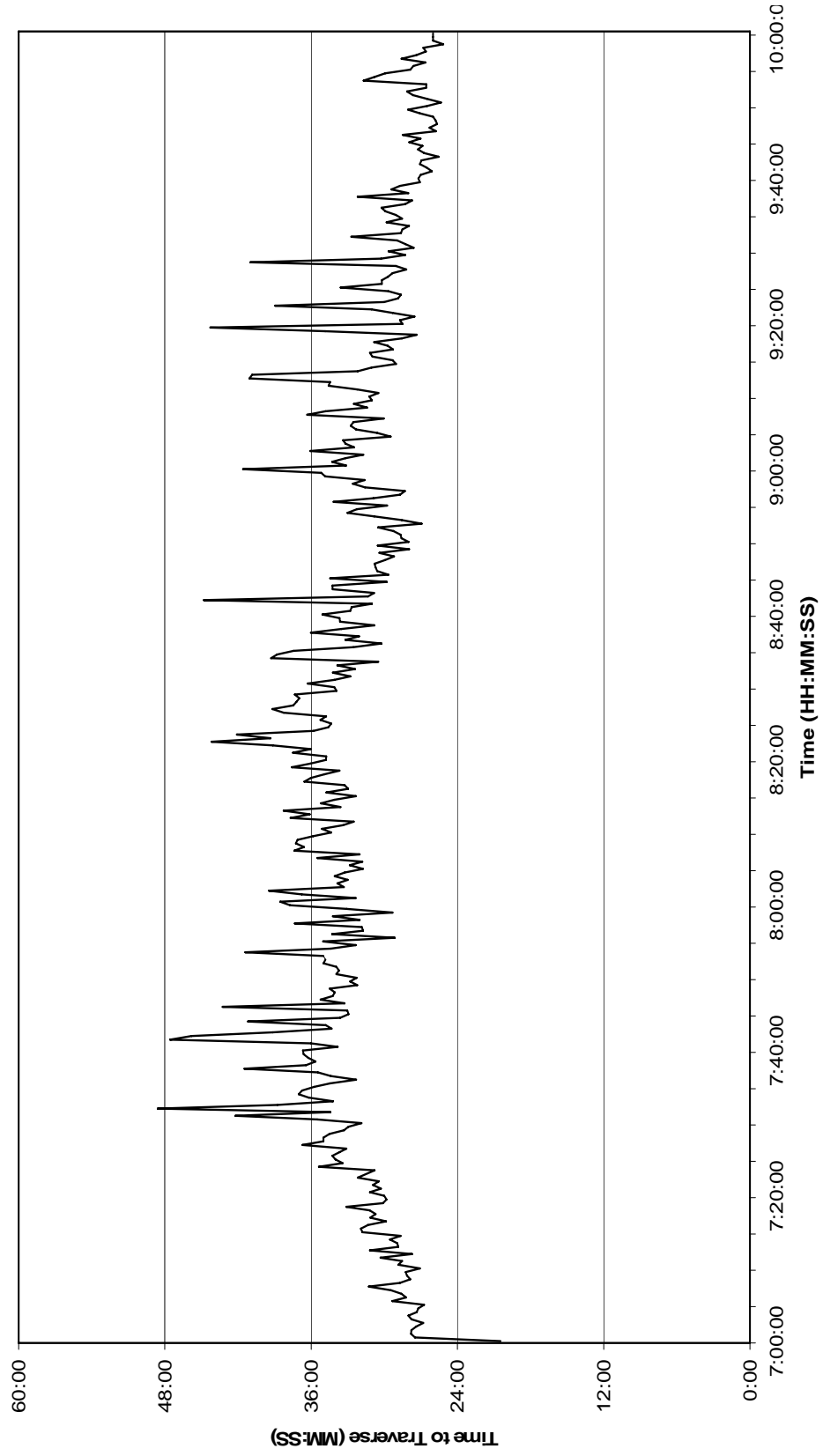Path 1 – 5S – 170S – 101S

Path 2 – 405S – 101S

Path 3 – 405S – 118E – 5S – 170S – 101S

Path 4 – 405S – 118E – 5S – 134W – 101S

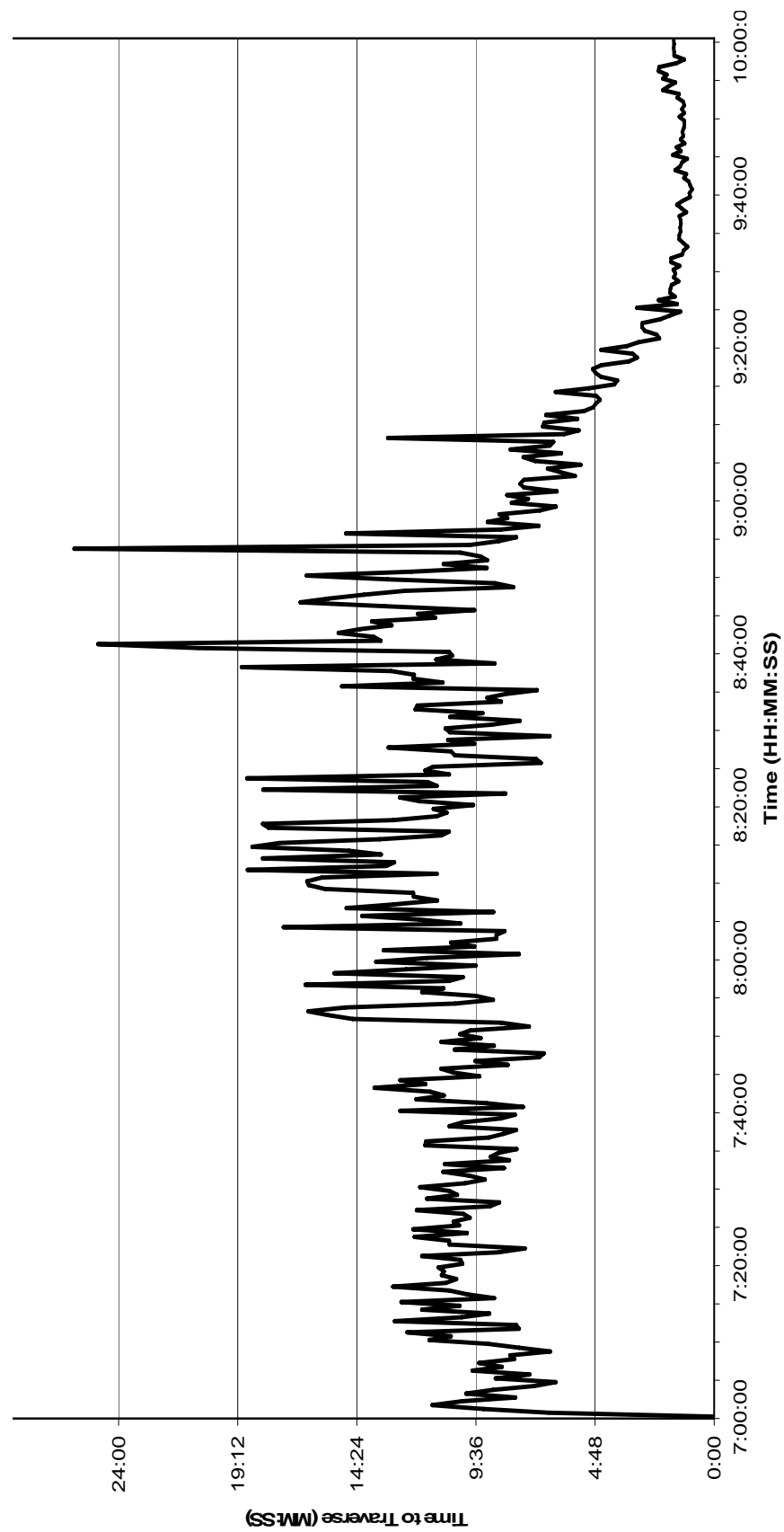Path 5 – 5S – 134W – 101S

Path 6 − 5S − 118W − 405S − 101S

F.4 Time Saved by Traversing Fastest over Shortest Path with 30-Second Data – from 5S at San Fernando 1 to 101S at Vermont on
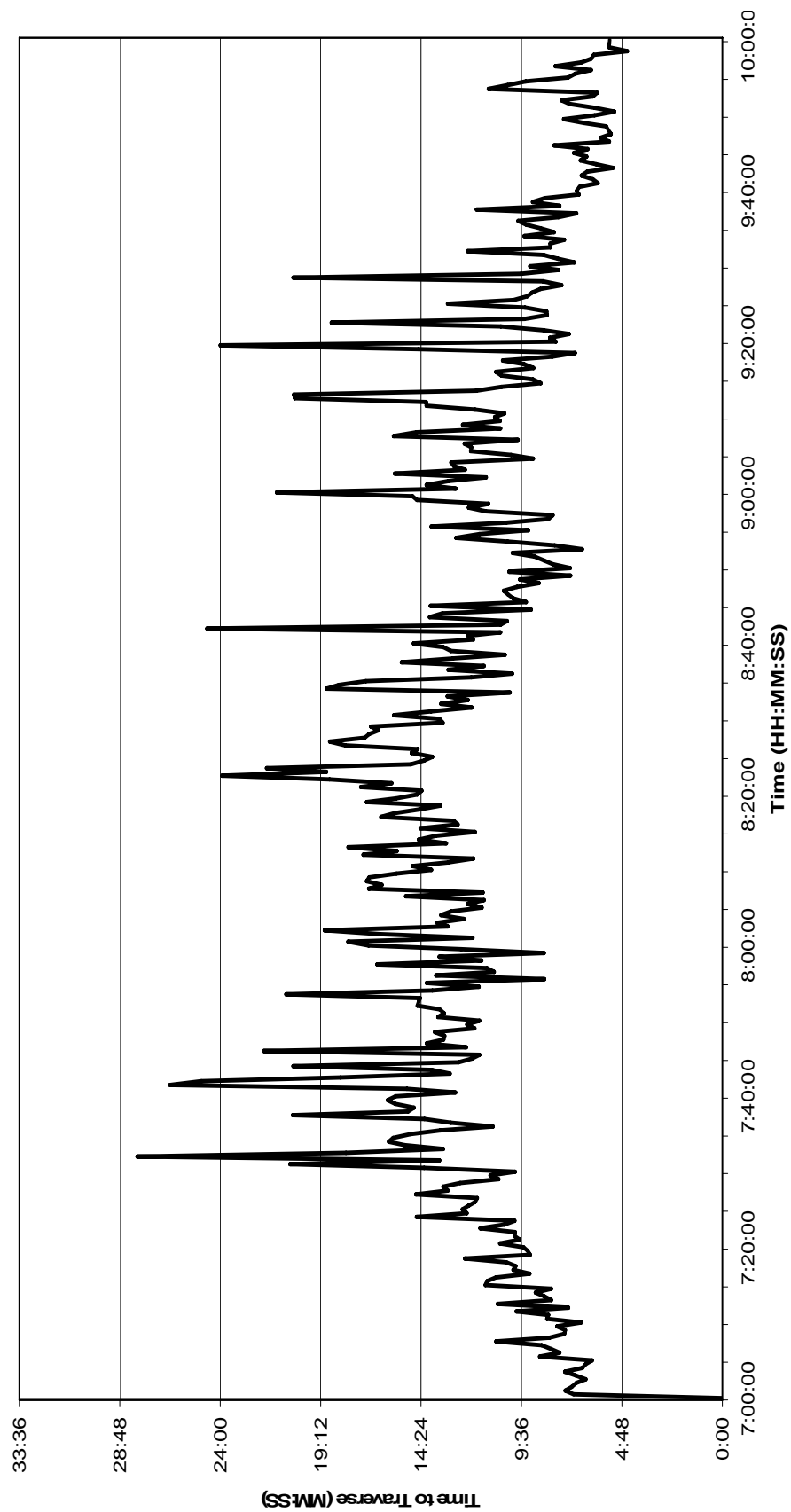

Time (HH:MM:SS)

Time Saved (MM:SS)
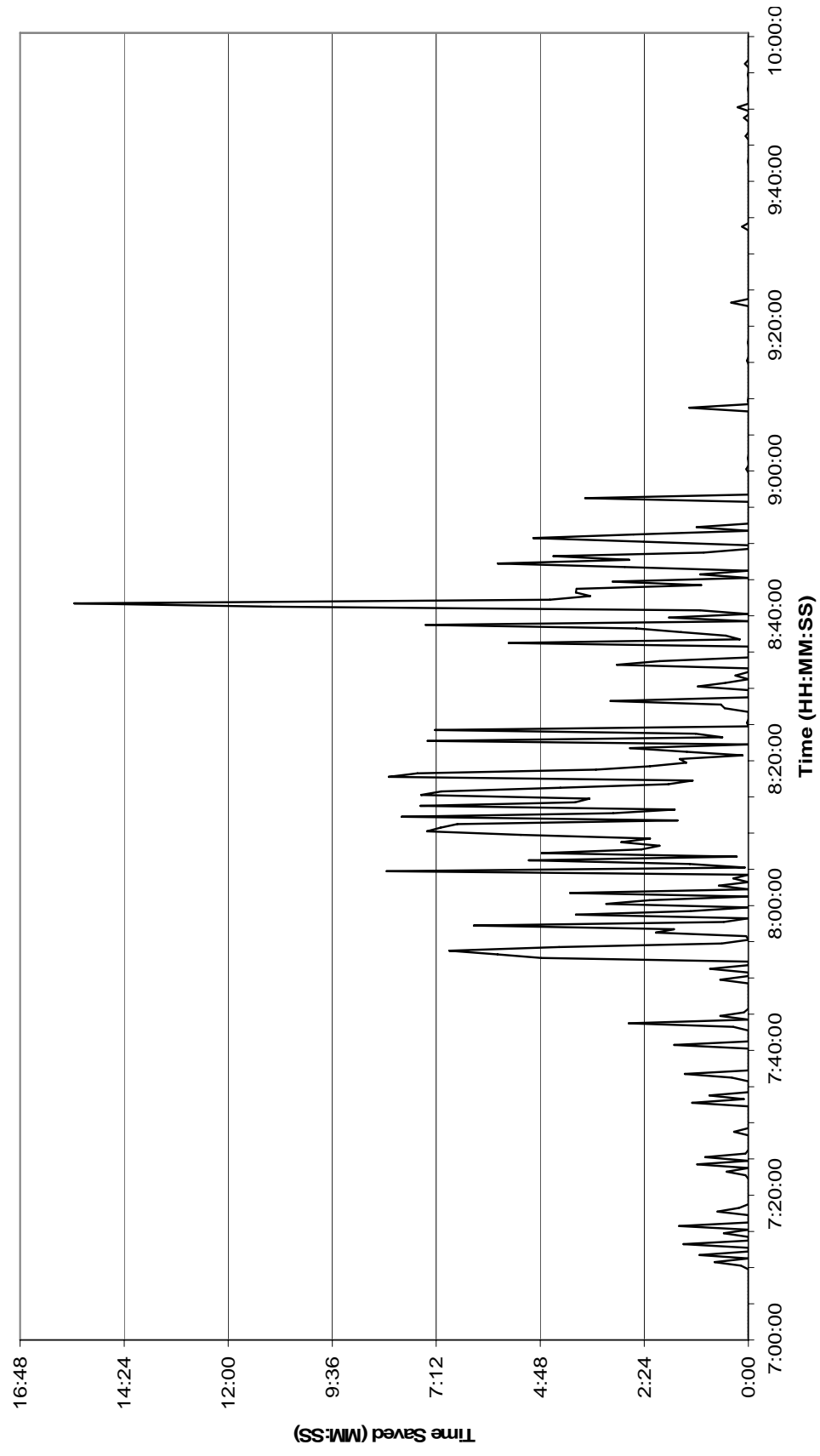
F.5 Difference in Time to Traverse All Paths under Current Conditions and Optimal Conditions – from 5S at San Fernando 1 to 101S
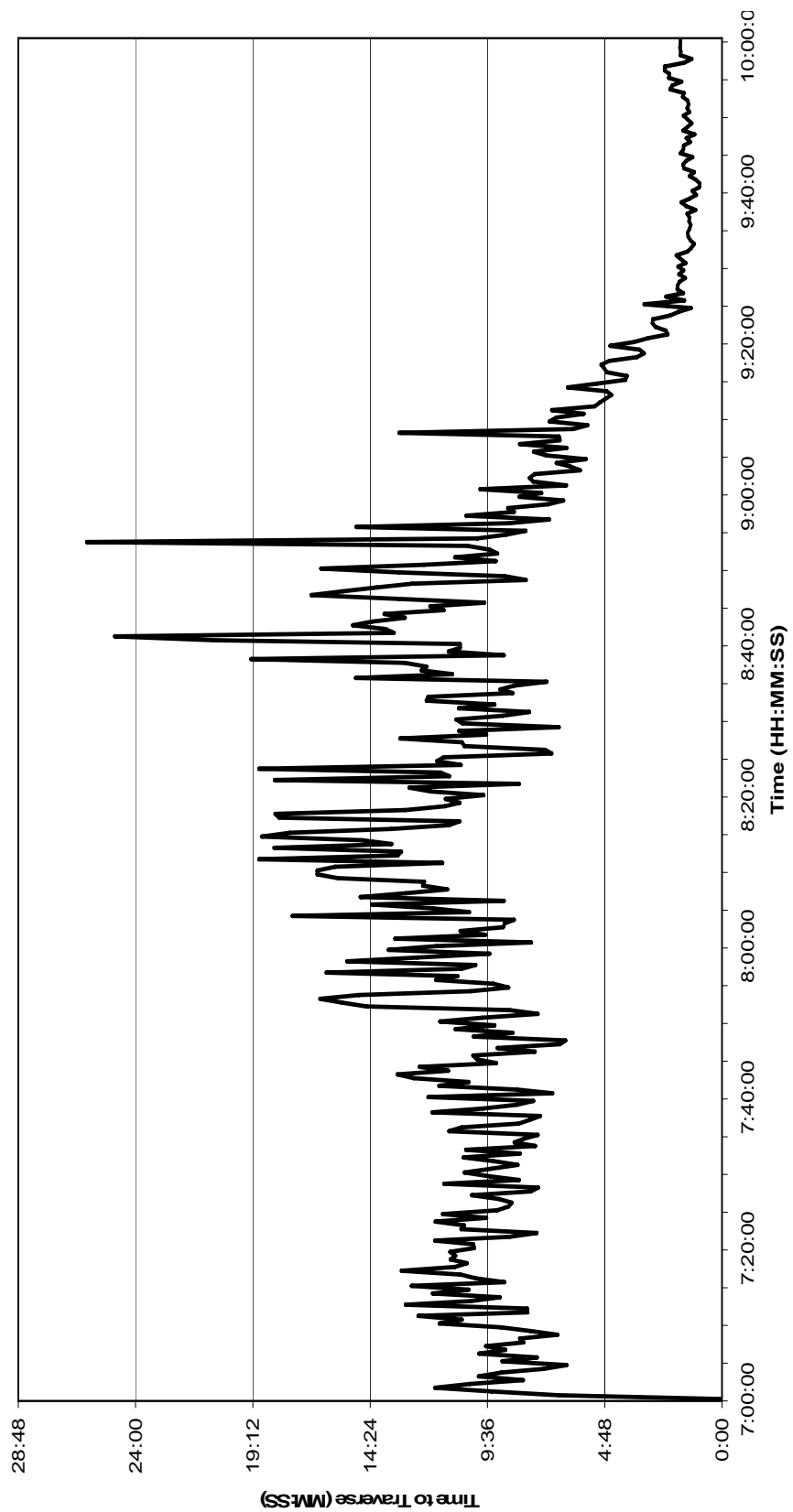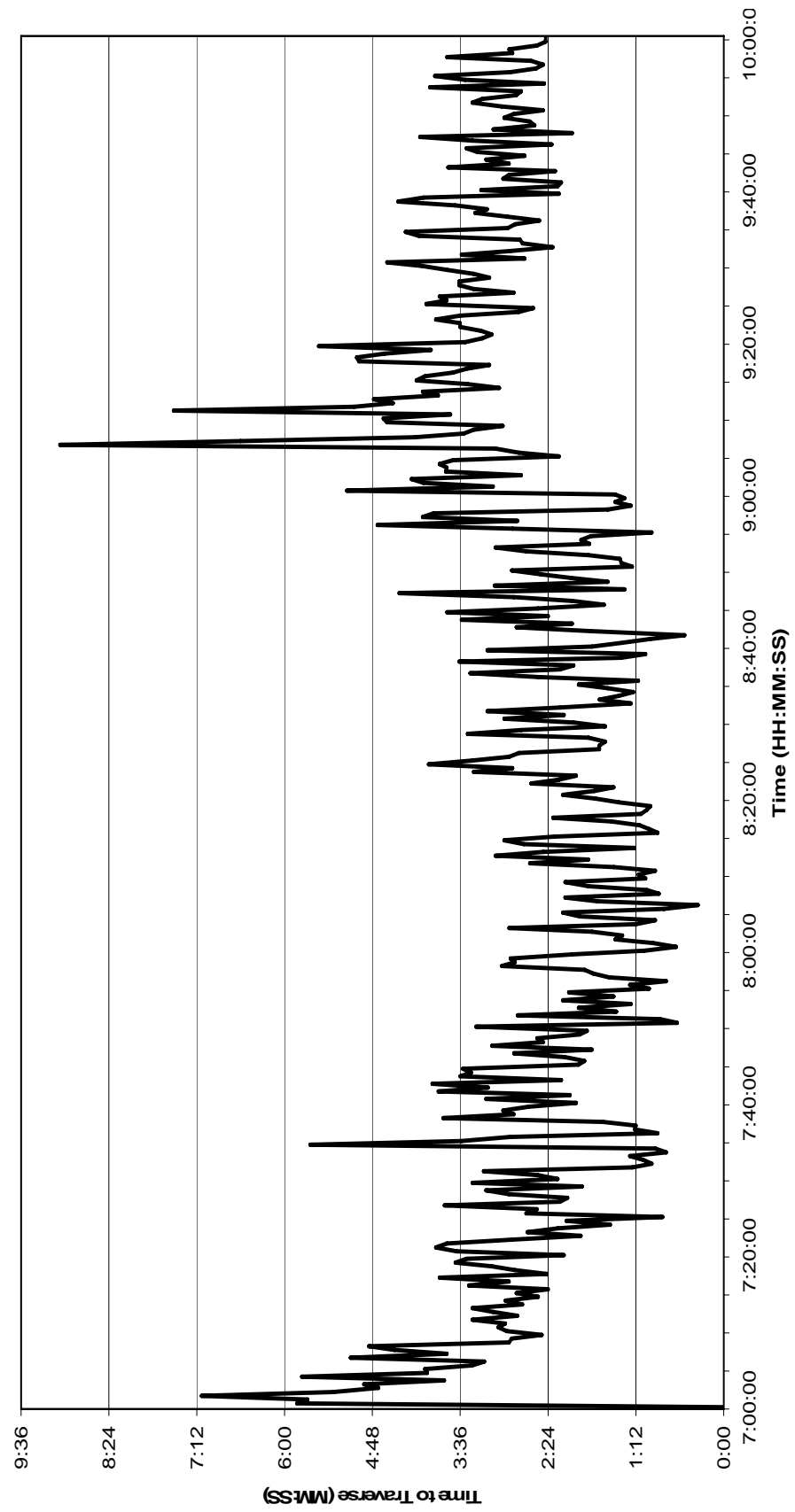
at Vermont on 2006-11-03

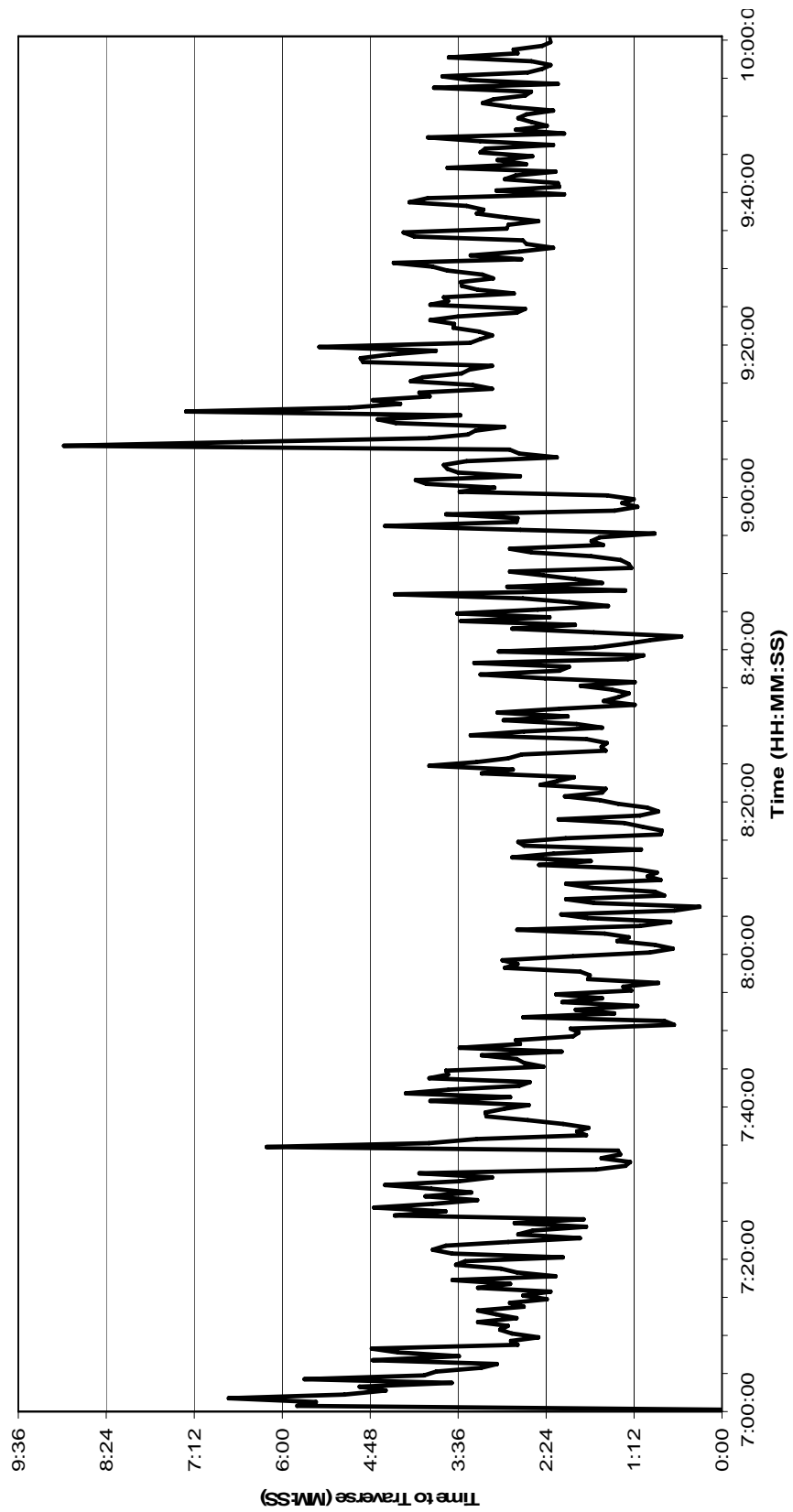Path 1 – 5S – 170S – 101S

Path 2 – 405S – 101S

Path 3 − 405S − 118E − 5S − 170S − 101S

Time (HH:MM:SS)
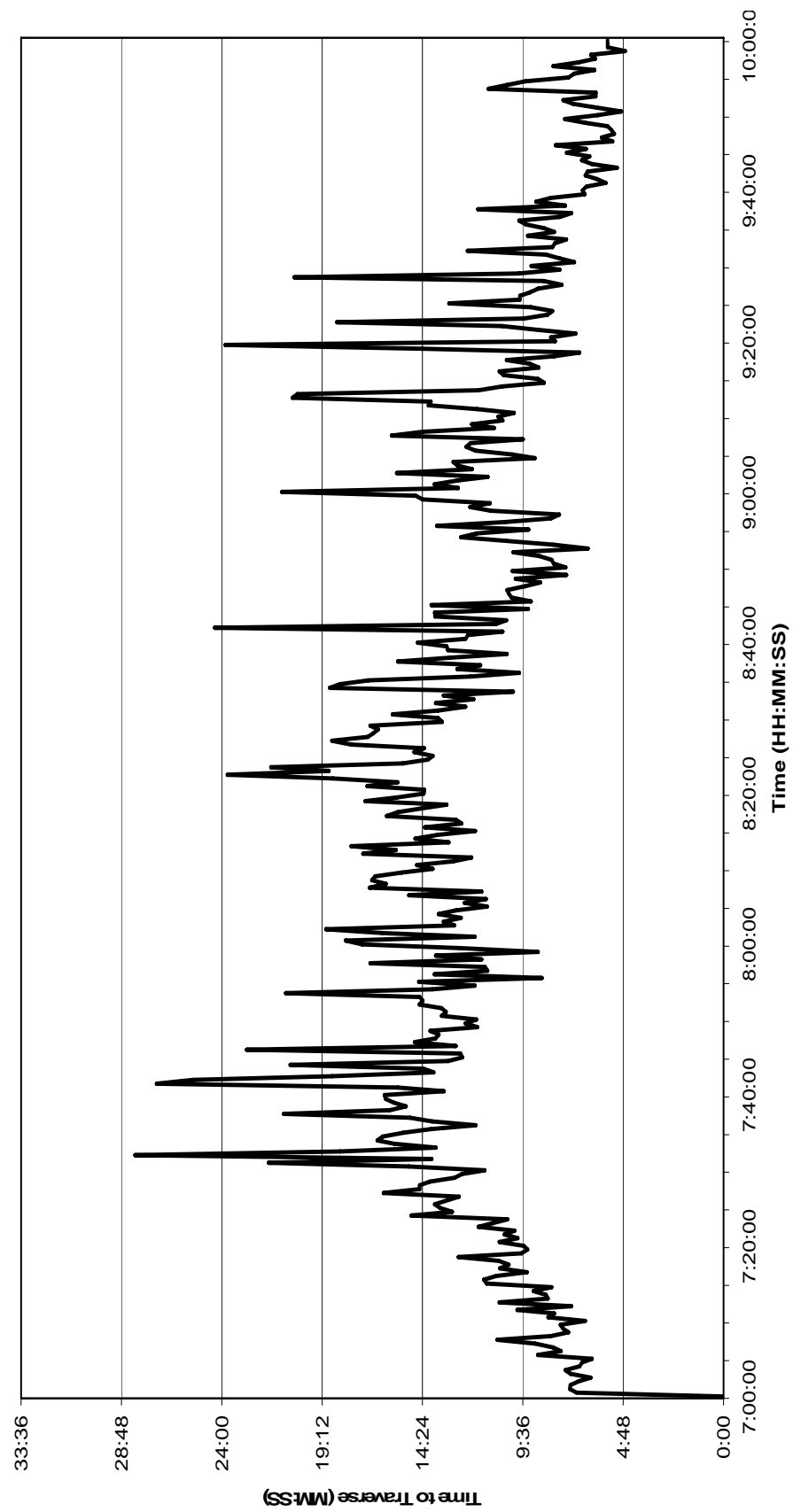
Time to Traverse (MM:SS)
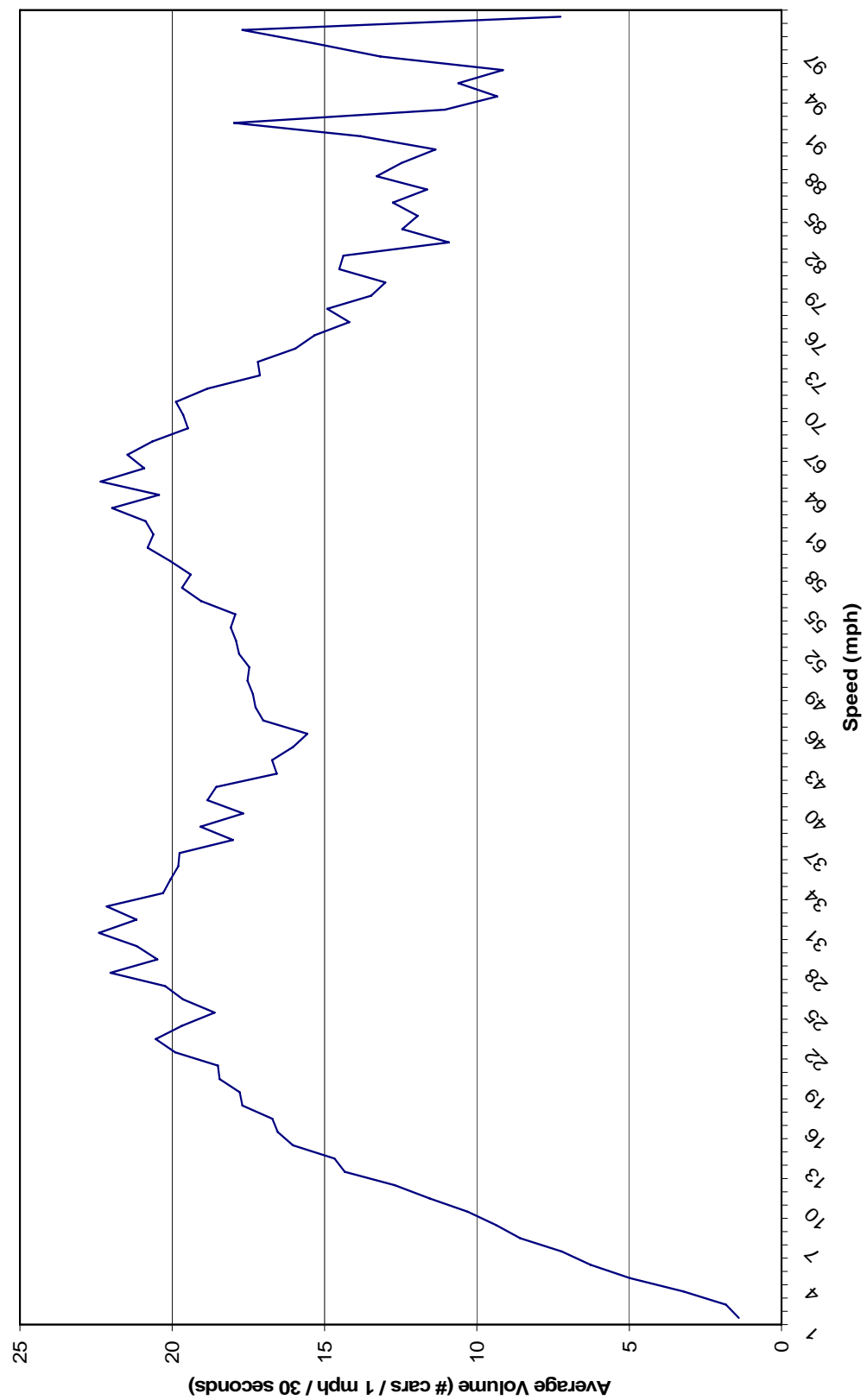
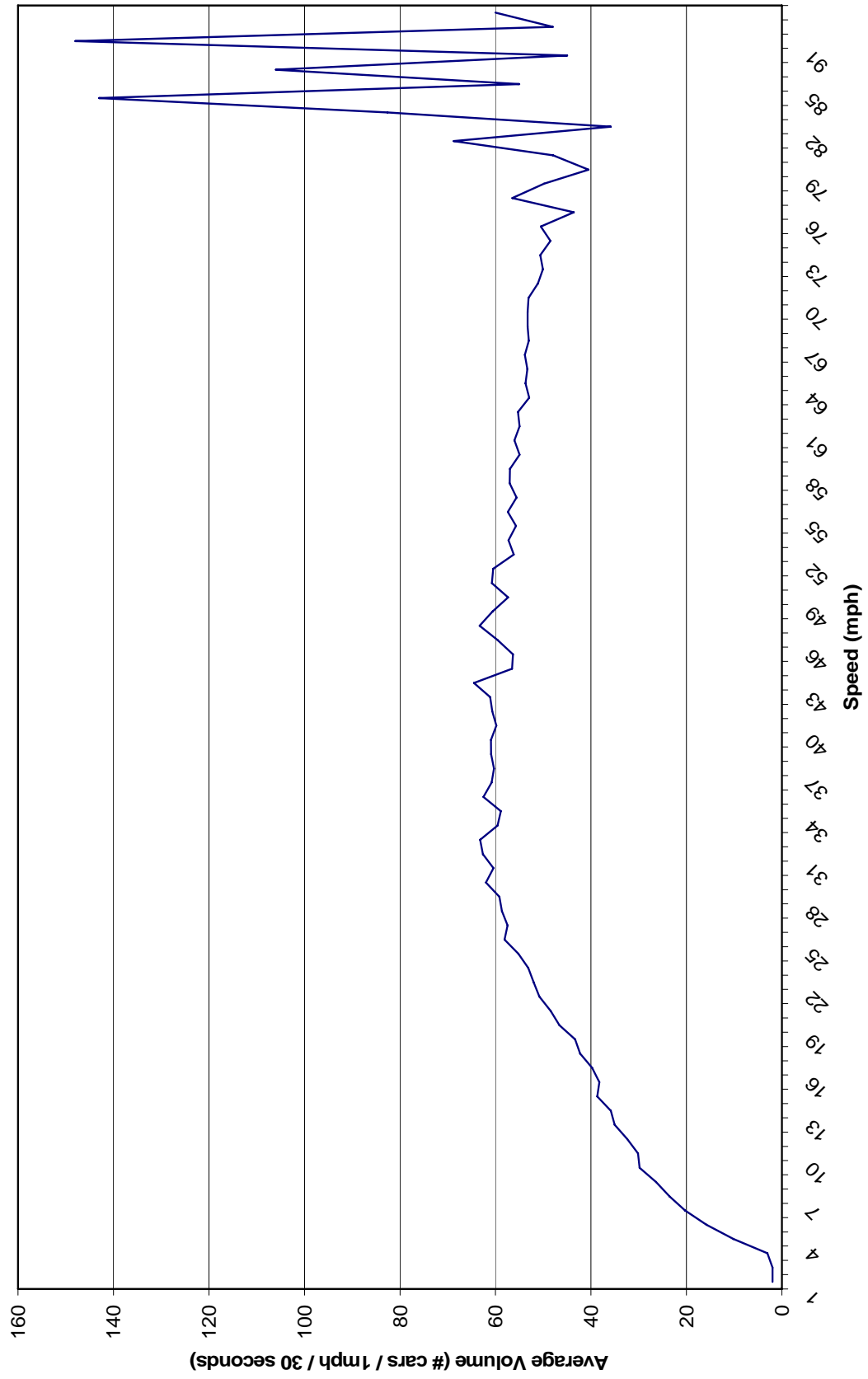Path 4 – 405S – 118E – 5S – 134W – 101S

Path 5 – 5S – 134W – 101S

Path $6 - 5S - 118W - 405S - 101S$

F.6 Average Volume for All Lanes at Different Speeds

F.7 Average Volume for Summary Data at Different Speeds

Average Volume (# cars / 1mph / 30 seconds)

Speed (mph)

Appendix G

References

[20] Airsage. http://www.airsage.com.

[16] Antonakos, James L., Kenneth C. Mansfield.   Practical Data Structures Using C/C++. Prentice Hall College Div, January 1999.

[59] Aragaki, Jeff. Project Manager of Intelligent Transportation Systems Division in Caltrans District 7. Personal Communication, November 3, 2006.

[29] Bellman, Richard. "On a routing problem." Quarterly of Applied Mathematics, Volume 16, Number 1, 1958.

[78] Berger, Ivan. "Standards for Car Talk." IEEE - The Institute, Volume 31, Number 1, March 2007.

[25] Bickel, Peter, Chao Chen, Jaimyoung Kwon, John Rice, Pravin Varaiya, Erik van Zwet. "Traffic Flow on a Freeway Network." *Proceeding of MSRI Workshop on Nonlinear Estimation and Classification*, March 19-29, 2001.

[56] Brummer, Daniel, Gary Cross, Jack Levis, Catherine McGhee, Dudley Whitney. "Toward Increased Use of Simulation in Transportation." *Proceedings of the 30th Conference on Winter Simulation*, 1998.

[77] California Driver Handbook 2007. http://www.dmv.ca.gov/pubs/dl600.pdf.

[2] CalTrans 2003 AADT – Average Annual Daily Traffic, 2003.
http://www.dot.ca.gov/hq/traffops/saferesr/trafdata/.

[67] Cameron, Gordon, Brian J.N. Wylie, David McArthur. "Paramics: Moving Vehicles on the Connection Machine." *Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*, November 14-19, 1994.

[36] Cayford, Randall, Tigran Johnson. "Operational Parameters Affecting the Use of Anonymous Cell Phone Tracking for Generating Traffic Information." *Transportation Research Board 2003 Annual Meeting*, 2003.

[52] Chen, Chao, Zhanfeng Jia, Pravin Varaiya. "Causes and Cures of Highway Congestion." *IEEE Control Systems Magazine*, Volume 21, Issue 4, December 2001.

[22] Choe, Tom, Alexander Skabardonis, Pravin Varaiya. "Freeway Performance Measurement System (PeMS): An Operational Analysis Tool." *81st Annual Meeting Transportation Research Board*, January 2002.

[54] Clark, Jim, Gene Daigle. "The Importance of Simulation Techniques in ITS Research and Analysis." *Proceedings of the 29th Conference on Winter Simultation*, 1997.

[42] Cohen, Edith, Uri Zwick. "All-Pairs Small-Stretch Paths." *Symposium on Discrete Algorithms*, 1997.

[18] Coleri, Sinem, Sing Yiu Cheung, Pravin Varaiya. "Sensor Networks for Monitoring Traffic." *42nd Annual Allerton Conference on Communication, Control, and Computing*, September 2004.

[12] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms – 2nd Edition. The MIT Press, 2001.

[64] Corsim. http://ops.fhwa.dot.gov/trafficanalysistools/ corsim.htm

[61] Demetrescu, Camil, Giuseppe F. Italiano. "Experimental Analysis of Dynamic All Pairs Shortest Path Algorithms." ACM Transactions on Algorithms. Volume 2, Number 4, October 2006.

[13] Demetrescu, Camil, Giuseppe Italiano. "A New Approach to Dynamic All Pairs Shortest Paths." ACM Symposium on Theory of Computing, June 2003.

[45] Demetrescu, Camil, Stefano Emiliozzi, Giuseppe F. Italiano. "Experimental Analysis of Dynamic All Pairs Shortest Path Algorithms." *Symposium on Discrete Algorithms*, 2004.

[11] Dijkstra, E.W. "A note on two problems in connexion with graphs." Numerische Mathematik, 1959.

[19] Ding, Jiagen, Sing-Yiu Cheung, Chin-Woo Tan, Pravin Varaiya. "Signal Processing of Sensor Node Data for Vehicle Detection." *IEEE 7th International Intelligent Transportation Systems Conference*, October 2004.

[48] Djidjev, H.N., G.E. Pantziou, C.D. Zaroliagis. "Improved Algorithms for Dynamic Shortest Paths." Algorithmica, Volume 28, 2000.

[70] Fellendorf, Martin. "VISSIM: A Microscopic Simulation Tool to Evaluate Actuated Signal Control Including Bus Priority." *64th Institute of Transportation Engineers Annual Meeting*, October 1994.

[58] Fishburn, Paul T., Javad Golkar, Kevin M. Taaffe. "Simulation of Transportation Systems." *Proceedings of the 27th Conference on Winter Simulation*, 1995.

[9] Floyd, Robert. "Algorithm 97 (SHORTEST PATH)." Communications of the ACM, 1977.

[30] Ford Jr., Lestor R., D.R. Fulkerson. Flows in Networks. Princeton University Press, 1962.

[33] Ghandeharizadeh, Shahram, Bhaskar Krishnamachari. "C2P2: A Peer-to-Peer Network for On-Demand Automobile Information Services." *1ˢᵗ International Workshop on Grid and Peer-to-Peer Computing*, August 2004.

[72] GNU General Public License. http://www.gnu.org/ licenses/gpl.html.

[27] Google Maps. http://maps.google.com.

[8] Howe-Steiger, Linda. "Staying in the Loop." *Fall 2001 Tech Transfer Newsletter*, UC Berkeley, 2001.

[4] Jia, Zhanfeng, Chao Chen, Ben Coifman, Pravin Varaiya. "The PeMS algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors." *IEEE 4th International Intelligent Transportation Systems Conference*, February 12, 2001.

[10] Johnson, Donald. "Efficient Algorithms for Shortest Paths in Sparse Networks." Journal of the ACM, 1977.

[28] Kim, Seongmoon, Mark E. Lewis, Chelsea C. White, III. "State Space Reduction for Nonstationary Stochastic Shortest Path Problems With Real-Time Traffic Information." IEEE Transactions on Intelligent Transportation Systems, Volume 6, Number 3, September 2005.

[15] King, Valerie. "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs." IEEE Symposium on Foundations of Computer Science, 1999.

[14] Klein, P.N., S. Subramanian. "A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs." Algorithmica, Volume 22, 1998.

[23] Kwon, Jaimyoung, Benjamin Coifman, Peter Bickel. "Day-to-Day Travel Time Trends and Travel Time Prediction from Loop Detector Data." Transportation Research Board #1717, 2000.

[26] Li, Kim, Petros Ioannou. "Modeling of Traffic Flow of Automated Vehicles." IEEE Transactions on Intelligent Transportation Systems, Volume 5, Number 2, June 2004.

[50] Lin, Wei-Hua, Carlos F. Daganzo. "A Simple Detection Scheme for Delay-Inducing Freeway Incidents." *Transportation Research*, Volume 31A, April 1996.

[6] Mapquest. http://www.mapquest.com.

[73] Maroto, Joaquin, Eduardo Delso, Jesus Felez, Jose Ma. Cabanellas. "Real-Time Traffic Simulation With a Microscopic Model." IEEE Transactions on Intelligent Transportation Systems. Volume 7, Number 4, December 2006.

[62] Miller, Jeffrey, Ellis Horowitz. "FreeSim – A Free Real-Time Traffic Simulator." Submitted to *IEEE 10th International Intelligent Transportation Systems Conference*, September 2007.

[63] Miller, Jeffrey, Ellis Horowitz. "Algorithms for Real-Time Gathering and Analysis of Continuous-Flow Traffic Data." *IEEE 9th International Intelligent Transportation Systems Conference*, September 17-20, 2006.

[71] MITSIMLab Open Source License. http://mit.edu/its/MITSIMLabOSnew.html# license.

[31] Moustafa, Hasnaa, Gilles Bourdon, Yvon Gourhant. "AAA in Vehicular Communication on Highways with Ad hoc Networking Support: A Proposed Architecture." *2nd ACM International Workshop on Vehicular Ad Hoc Networks*, September 2005.

[34] Ng, Chong-Keat, Chao Chen. "Wireless Content Delivery and User Profiling." *IEEE 4th International Intelligent Transportation Systems Conference*, February 2001.

[65] Owen, Larry E., Yunlong Zhang, Lei Rao, Gene McHale. "Traffic Flow Simulation Using CORSIM." *Proceedings of the 2000 Winter Simulation Conference*, December 10-13, 2000.

[41] Pallottino, Stefano, Maria Grazia Scutella. "Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects." *University of Pisa Technical Report TR-97-06*, April 1997.

[49] PeMS Group. "Freeway Performance Measures – Calculations with Loop Detectors." August 22, 1999.

[3] Performance Measurement System (PeMS). http://pems.eecs.berkeley.edu.

[24] Petty, Karl F., Peter Bickel, Jiming Jiang, Michael Ostland, John Rice, Ya'acov Ritov, Frederic Schoenberg. "Accurate estimation of travel times from single-loop detectors." Transportation Research, Part 1 (Policy and Practice), Volume 32A, #1, January 1998.

[38] Peytchev, Evtim, Christophe Claramunt. "Experiences in Building Decision Support Systems for Traffic and Transportation GIS." *ACM Global Information Systems Conference*, November 2001.

[75] Quek, Chai, Michel Pasquier, Bernard Boon Seng Lim. "POP-TRAFFIC: A Novel Fuzzy Neural Approach to Road Traffic Analysis and Prediction." IEEE Transactions on Intelligent Transportation Systems, Volume 7, Number 2, June 2006.

[40] Ramalingam, G., Thomas Reps. "An Incremental Algorithm for a Generalization of the Shortest-Path Problem." *Journal of Algorithms*, Volume 21, Issue 2, September 1996.

[69] Redmill, Keith A., Umit Ozguner. "VATSIM: A Vehicle and Traffic Simulator." *IEEE 2nd Inetrnational Intelligent Transportation Systems Conference*, October 5-8, 2001.

[21] Rice, John, Erik van Zwet. "A Simple and Effective Method for Predicting Travel Times on Freeways." *IEEE 5th International Intelligent Transportation Systems Conference*, August 2002.

[47] Roditty, Liam, Uri Zwick. "Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs." *Proceedings of the 45th Foundations of Computer Science Conference*, October 2004.

[46] Roditty, Liam, Uri Zwick. "On dynamic shortest paths problems." *Proceedings of the 12th European Symposium on Algorithms*, 2004.

[39] Romeijn, H. Edwin, Robert L. Smith. "Parallel Algorithms for Solving Aggregated Shortest Path Problems." *Computers and Operations Research*, Volume 26, Issues 10-11, September 1999.

[53] Sas, Sanjoy, Betty A. Bowles, Chris R. Houghland, Steven J. Hunn, YunLong Zhang. "A Knowledge Based Model of Traffic Behavior in Freeways." *ACM Symposium on Applied Computing*, 1999.

[37] Schaefer, Lisa. "Architecture Using JINI Technology for Simulation of an Agent-Based Transportation System." *Proceedings of the 2001 Winter Simulation Conference*, 2001.

[51] Schofer, Ralph E., Franklin F. Goodyear. "Electronic computer applications in urban transportation." *Proceedings of the 1967 22nd National Conference*, 1967.

[1] Shrank, David, Tim Lomax. "2004 Urban Mobility Report." *Texas Transportation Institute's Annual Urban Mobility Report*, September 2004.

[43] Siedel, Raimund. "On the All-Pairs-Shortest-Path Problem." *24th Annual ACM Symposium on Theory of Computing*, 1992.

[5] Sigalert.com. http://www.sigalert.com.

[60] Sprint Website. http://www.sprint.com.

[44] Takaoka, T. "Subcubic Cost Algorithms for the All Pairs Shortest Path Problem." <u>Algorithmica</u>, Volume 20, 1998.

[76] United Kingdom Highway Code. http://www.highwaycode.gov.uk/09.htm.

[74] Vlahogianni, Eleni, Matthew Karlaftis, John Golias, Nikolaos Kourbelis. "Pattern-Based Short-Term Urban Traffic Predictor." *IEEE 9th International Intelligent Transportation Systems Conference*, September 17-20, 2006.

[57] Wang, Paul T.R., Richard A. Glassco. "Enhanced THOREAU Traffic Simulation for Intelligent Transportation Systems (ITS)." *Proceedings of the 27th Conference on Winter Simulation*, 1995.

[68] Wang, Yibing, Markos Papageorgiou, Albert Messmer. "RENAISSANCE: A Real-Time Freeway Network Traffic Surveillance Tool." *IEEE 9th International Intelligent Transportation Systems Conference*, September 17-20, 2006.

[32] Wu, Hao, Mahesh Palekar, Richard Fujimoto, Jaesup Lee, Joonho Ko, Randall Guensler, Michael Hunter. "Vehicular Networks in Urban Transportation Systems." *Digital Government Conference*, May 2005.

[55] Xu, Jinghua, Kathleen L. Hancock, Frank Southworth. "Dynamic Freight Traffic Simulation Providing Real-Time Information." *Proceedings of the 35th Conference on Winter Simulation*, 2003.

[7] Yahoo Maps. http://maps.yahoo.com.

[66] Yang, Qi, Haris N. Koutsopoulos. "A Microscopic Traffic Simulator for Evaluation of Dynamic Traffic Management Systems." *Transpn Res., Part C*, Volume 4, Number 3, 1996.

[35] Yim, Youngbin. "The State of Cellular Probes." *California Partners for Advanced Transit and Highways Research Report UCB-ITS-PRR-2003-25*, July 2003.

[17] Zhanfeng, Jia, Pravin Varaiya, Chao Chen, Karl Petty, Alex Skabardonis. "Maximum throughput in LA freeways occurs at 60 mph." *Berkeley Technical Report*, January 16, 2001.