SIMULATING, RECONSTRUCTING, AND ROUTING METROPOLITAN-SCALE TRAFFIC

David Wilkie

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2015

Approved by:

Ming C. Lin

Ron Alterovitz

Anselmo Lastra

Dinesh Manocha

Jason Sewall

ABSTRACT

David Wilkie: Simulating, Reconstructing, and Routing Metropolitan-Scale Traffic.
(Under the direction of Ming C. Lin.)

Few phenomena are more ubiquitous than traffic, and few are more significant economically, socially, or environmentally. The vast, world-spanning road network enables the daily commutes of billions of people and makes us mobile in a way our ancestors would have envied. And yet, few systems perform so poorly so often. Gridlock and traffic jams cost 2.9 billion gallons of wasted fuel and costs over 121 billion dollars every year in the U.S. alone.

One promising approach to improving the reliability and efficiency of traffic systems is to fully incorporate computational techniques into the system, transforming the traffic systems of today into cyber-physical systems. However, creating a truly cyber-physical traffic system will require overcoming many substantial challenges. The state of traffic at any given time is unknown for the majority of the road network. The dynamics of traffic are complex, noisy, and dependent on drivers' decisions. The domain of the system, the real-world road network, has no suitable representation for high-detail simulation. And there is no known solution for improving the efficiency and reliability of the system.

In this dissertation, I propose techniques that combine simulation and data to solve these challenges and enable large-scale traffic state estimation, simulation, and route planning.

First, to create and represent road networks, I propose an efficient method for enhancing noisy GIS road maps to create geometrically and topologically consistent 3D models for high-detail, real-time traffic simulation, interactive visualization, traffic state estimation, and vehicle routing. The resulting representation provides important road features for traffic simulations, including ramps, highways, overpasses, merge zones, and intersections with arbitrary states.

Second, to estimate and communicate traffic conditions, I propose a fast technique to reconstruct traffic flows from in-road sensor measurements or user-specified control points for interactive 3D visualization and communication. My algorithm estimates the full state of the traffic flow from sparse sensor measurements using a statistical inference method and a continuum traffic model. This estimated state then drives an agent-based traffic simulator to produce a 3D animation of traffic that statistically matches the sensed traffic conditions.

Third, to improve real-world traffic system efficiency, I propose a novel approach that takes advantage of mobile devices, such as cellular phones or embedded systems in cars, to form an interactive, participatory network of vehicles that plan their travel routes based on the current, sensed traffic conditions and the future, projected traffic conditions, which are estimated from the routes planned by all the participants. The premise of this approach is that a route, or plan, for a vehicle is also a prediction of where the car will travel. If routes are planned for a sizable percentage of the vehicles using the road network, an estimate for the overall traffic pattern is attainable. If fewer cars are being coordinated, their impact on the traffic conditions can be combined with sensor-based estimations. Taking planned routes into account as predictions allows the entire traffic route planning system to better distribute vehicles and to minimize traffic congestion.

For each of these challenges, my work is motivated by the idea of fully integrating traffic simulation, as a model for the complex dynamics of real world traffic, with emerging data sources, including real-time sensor and public survey data.

Dedicated to my friends and family; this would not have possible without you.

ACKNOWLEDGMENTS

I want to thank the members of GAMMA Lab, especially Jason Sewall, Jur van den Berg, Sujeong Kim, Abhinav Golas, Ravish Mehra, my committee members, and my advisor, Ming Lin.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Few phenomena are more ubiquitous in the modern world than traffic, and few are more significant economically and socially. The vast, world-spanning road network enables the daily commutes of billions of people and makes us mobile in a way our ancestors could not imagine. And yet, few systems perform so poorly so often. Congestion and gridlock cost 2.9 billion gallons of wasted fuel and over 121 billion dollars every year in the U.S. alone. [Schrank et al., 2012].

One promising approach to improving the reliability and efficiency of traffic systems is to fully incorporate computational technology, transforming today's traffic systems into *cyber-physical* systems. However, creating truly cyber-physical traffic systems will require overcoming many substantial challenges. Currently, the state of traffic at a given time is unknown for a large majority of the road network, though recent research attempts to patch together discrete and disparate data sources for a fuller view. The dynamics of the system are also complex, noisy, and computationally costly to simulate for large networks or large numbers of vehicles: the system has both continuous aspects, the flow of traffic, and discrete aspects, such as the cycles of intersections; the dynamics can be approximated with physical laws, but ultimately emerge from decisions made by drivers. And the domain of the system, the road network, is complex topologically and geometrically with no widely available high-detail models.

This dissertation proposes new computational models and approaches that build toward achieving a cyber-physical traffic system. One component of such a system is traffic simulation, which allows us to efficiently model the complex dynamics of real-world traffic. Traffic simulation is currently an important tool for planning and evaluating road network designs and for creating interactive animations, but I will demonstrate that it can also be a powerful tool for estimating and communicating the state of traffic to achieve more efficient traffic flows, especially when combined with newly available data sources.

To make headway on a problem as broad and challenging as this, I have focused my work on three core challenges. First, for the road network domain, my approach is to create simulation-enabled representations from GIS road network models, which are now publicly available in vast databases. This involves constructing formal models atop noisy, human-authored data and creating 3D geometric representations. Second, to estimate and communicate traffic conditions, I present a method that filters sensor data, which is discrete in space and time, to create a continuous estimate of traffic conditions that can be used to create visualizations or predict traffic patterns. Third, to improve the efficiency of the traffic system, I present an approach that manages traffic flows via participatory routing, in which the plans for individual vehicles are aggregated to estimate future traffic conditions. For each of these challenges, my work is motivated by the idea of fully integrating traffic simulation, as a model for the complex dynamics of real world traffic, with emerging data sources, including real-time sensors and public databases.

1.1 Traffic Systems

In this dissertation, I address physical systems that are composed of road networks and vehicles, which I refer to as a *traffic system*. These systems can also include various types of sensing, communication, and computational components. In the following, I discuss the ways in which we can model these systems for both reasoning and computation.

1.1.1 Road Networks

One component of a traffic system is a road network, which can be modeled at varying levels of detail.

1.1.1.1 Directed Graph

A road network can be represented very simply as a directed graph for which edges, E, represent segments of roadway, and the vertices, V, represent intersections between roads and terminus points. Each edge can have attributes such as length, number of lanes, speed limit, etc.

However, a network such as this would not be suited for any visual analysis or highly detailed simulation, as the geometric details of the road are lost. It also could not handle problems that require more geometric information, such as determining which roads correspond to a sequence of global-positioning system (GPS) coordinates.

1.1.1.2 Polyline Road Networks

To address these shortcomings, the road network needs to be embedded in a space, typically a 2D Cartesian space, which the latitude-longitude points often used to define the roads can be projected into. In this representation, the vertices, V^i, i.e. intersections, have coordinates indicating their positions, and each edge e, i.e. each road segment, has additional vertices, V_e^g, that approximate the geometry of the road. The edges and vertices can have additional attributes, A, as above.

We will call the above a *polyline road network*, and it is the most common representation of road networks. Representations such as this are used for nearly all geographic-information system (GIS) applications, including for online mapping and vehicle routing, and are available for many countries in public databases.

This representation also has its shortcomings. For one, lanes are not individually represented. This limits the level of detail at which simulations can be done and limits the level of analysis. Second, the polyline (or polygonal chain) geometry can be a source of significant error – real world cars "cut corners", following curvilinear paths. This is exacerbated at intersections, at which simulated cars could take unrealistic 90°turns. Third, intersections in this representation are modeled as 0-degree objects, when in reality they have a geometric extent and can encompass complicated topological relationships. Due to this representation, inelegant solutions arise, such as modeling complex intersections as multiple vertices in an attempt to capture their geometric and topological details.

1.1.1.3 Lanes, Geometry, and Intersections

To address these issues, additional details can be added to the representation. First, individual lanes can be represented as full-fledged objects. These lanes need geometric representations, which can either be distinct or can reference the geometry of an underlying road segment. Second, intersections also need a more detailed representation: their geometric extents and topological relations need to be captured. This includes not only a bounding geometry but *internal lanes* that connect the incoming and outgoing lanes. Third, a more detailed geometric representation can be used, such as a higher number of vertices or curves. Finally, a model is needed for each intersection that defines the behavior of the intersection, whether the intersection uses a traffic light logic, stop sign, or some other rules of the road.

1.1.2 Traffic state

The other essential component of the traffic system is the vehicles – all the cars, trucks, motorcycles, etc. This aspect of the system is time-varying[1] and can be represented as either continuous, i.e. density and velocity fields, or discrete, i.e. individual vehicles. Beyond the vehicles that are on the roads at a particular time, some representation is needed for the boundary conditions of the system – the sources and sinks for vehicles.

At any point in time, we can refer to the *state* of the traffic system, which is the specification of all the time-varying values, such as how many cars are on the road; how fast the cars are moving; and how many cars are entering the system. The representation of the state is dependent on how the dynamics of the traffic system are modeled. There are two primary approaches. First, every vehicle can be represented. This is the most direct

[1]Except at very severe traffic jams.

approach: for every car in the real, there is a virtual car in the model with a known position along a lane, a velocity, and perhaps other parameters. This type of representation is referred to as *microscopic* in the literature or *agent-based* if the car models encapsulate some decision making ability. Second, the vehicles can be represented as average quantities over some spatial discretization; i.e. a lane can be divided into cells, and each of these cells can contain a density value and a velocity value that represent the average of these statistics in the real world. This type of representation is called *macroscopic*.

1.1.3 Boundary Conditions

Real world road networks span whole continents. Working with networks this large is impractical for most applications, so some region of interest must be specified. In reality, cars pass into and out of this region, and how this flow is represented depends on the representation of the system dynamics, but examples include constant density and velocity values or a rate at which cars are spawned at the boundary points. Internal boundaries, such as large parking lots, can also be modeled in this manner.

1.1.4 Dynamics

Finally, our model requires a definition for the system dynamics, i.e. how the cars actually move. Like the state representation, there are two broad categories for these models – microscopic and macroscopic. For the former, individual vehicles are simulated, typically by calculating an acceleration from the state of the vehicle and the vehicle ahead of it along the lane. For the latter, differential equations define the evolution of the density and velocity fields over time. In either case, the model of the dynamics can be embodied in a simulation, which, if given initial and boundary conditions, can model the evolution of the traffic dynamics over time.

1.2 Sensor Data

Even a perfect model of a traffic system is only useful for real world applications if the system's state is known, including the boundary conditions. But this is also a difficult problem: traffic patterns vary day to day, hour to hour, and minute to minute; the distributions of origins and destinations are largely unknown; few roads have any sensors; and the sensor data that exists is difficult to fuse, not widely available, and noisy.

However, two emerging trends offer promise that a much fuller and clearer view of traffic will be available in the near future. The first of these trends is the networking and aggregation of existing sensor systems, allowing for a more complete view of the traffic system and real time updates of traffic conditions. The second trend is the proliferation of mobile devices. These devices enable traffic applications such as routing by vehicle drivers, and, due to their networked nature, enable the drivers to report, actively or passively, on the nearby traffic conditions. Together, these two trends promise accurate and widespread sensing, implying that we can know the state of traffic at any moment in time. And knowing the state of a system is the first step toward optimizing it.

There are three principal types of traffic sensors: loop detectors, cameras, and mobile devices. These sensors can be divided into two categories, *stationary sensors*, such as the loop detectors and cameras, and *mobile sensors*, such as the mobile devices. This distinction is important as the categories provide different types of data: stationary sensors describe most or all of the traffic state at a fixed location, i.e. they provide *Eulerian measurements*, and mobile sensors report measurements along each vehicle's trajectory, i.e. they provide *Lagrangian measurements*.

1.2.1 Loop Detectors

The first type of sensor is the loop-detector. A loop detector is an induction loop that can be used to sense the presence of a vehicle. Double loop-detectors can also be used to calculate the velocity of the passing vehicle.

Loop detectors are installed permanently on many highways and are often deployed temporarily on primary and secondary roads to gather survey data on traffic patterns. As such, loop detectors are currently the most common and widely available form of traffic sensor.

Loop detectors can record every vehicle that passes, their speeds, and other attributes. It can break this flow data down by individual lanes. They thus provide a wealth of traffic data. However, a loop detector only provides data at its location. The next sensor on a highway may be miles away. Between the detectors, the state of traffic can only be estimated. Most roads do not have this level of sensing, as permanent loop detectors are placed almost exclusively on highways.

1.2.2 Traffic Cameras

Another form of traffic sensor is the video camera. Cameras provided a real-time view of a region of the road network, capturing more context than loop detectors. They can also be relatively inexpensive.

However, cameras do not directly measure the traffic state. That is, they do not provide a measure of the vehicles' velocities or how many vehicles there are. This data can be obtained from a video, but it requires tracking each car and determining the position of the camera relative to the vehicles. For these reasons, cameras are not widely used to gather statistical data on traffic.

1.2.3 GPS-Enabled Mobile Devices

The third type of sensor is the mobile device, such as a cell phone or dashboard computer, that is networked and GPS-enabled. These devices are capable of sending periodic updates of the position of the vehicle. Some can also send direct velocity measurements, and for those that cannot the velocity can be estimated using the distance the vehicle traveled through the road network and the time interval between updates. The strength of this type of sensor is its potential to be ubiquitous – every driver could carry a device, reporting on traffic conditions throughout the road network.

However, measurements from devices such as this are not yet widely available to the public or to researchers. The datasets that are available point to some weaknesses. First, it is unlikely that a high percentage of cars will regularly use a mobile device that reports on the traffic conditions: the average user only needs a mobile device when traveling somewhere unfamiliar; sending updates from a persistently running device requires the device to be plugged in or it drains the battery; and positional updates clearly have privacy implications. A consequence of this is that data is currently available only in limited areas, and many of the available datasets were collected from taxi fleets. Second, as only a small percentage of vehicles can be expected to report the conditions, the data from these sensors is very sparse in addition to being noisy. Third, these devices are not capable of sensing the total number of vehicles on the road. And fourth, mobile sensors move with the vehicles, meaning that a persistent view of a single area of the road network can be difficult to obtain.

1.3 Thesis Statement

My work focuses on integrating computational models of traffic systems, described above, with newly available and emerging data sources, including both sensor data and publicly available databases. This bring me to my thesis statement:

It is possible to integrate traffic simulation with emerging data sources, including mobile sensing, sensor networks, and public databases, to enable novel traffic system computations, including simulating traffic throughout whole cities, reconstructing detailed traffic conditions, and performing participatory multi-vehicle route planning.

To support this thesis, I present my approach to creating road network models, estimating and communicating traffic conditions, and optimizing the overall traffic flow via participatory routing. Finally, I discuss a prototype implementation of this system.

1.4 Main Results

My main results include the following.

1.4.1 Automatic Road Network Creation.

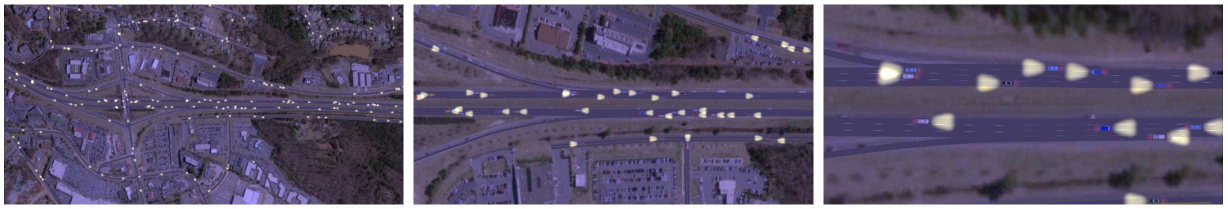


Figure 1.1: A road network created by my method with simulated traffic.

An immense amount of data exits for road networks in the form of Geographic Information System (GIS) databases, but the data in this form is not suitable for simulation. It lacks geometric details, lanes, relations between lanes at intersections, and contains human error and noise. In Chapter 2, I propose a method to create metropolitan-scale road networks suitable for simulation and animation from this publicly available GIS data. The resulting networks will be suitable for real-time, 3D simulation and animation. The method presents approaches to formalizing the road network, filtering common noise and errors, creating intersection geometry, and assigning heights and an ordering to highway overpasses and underpasses.

1.4.2 Detailed Traffic Reconstruction.

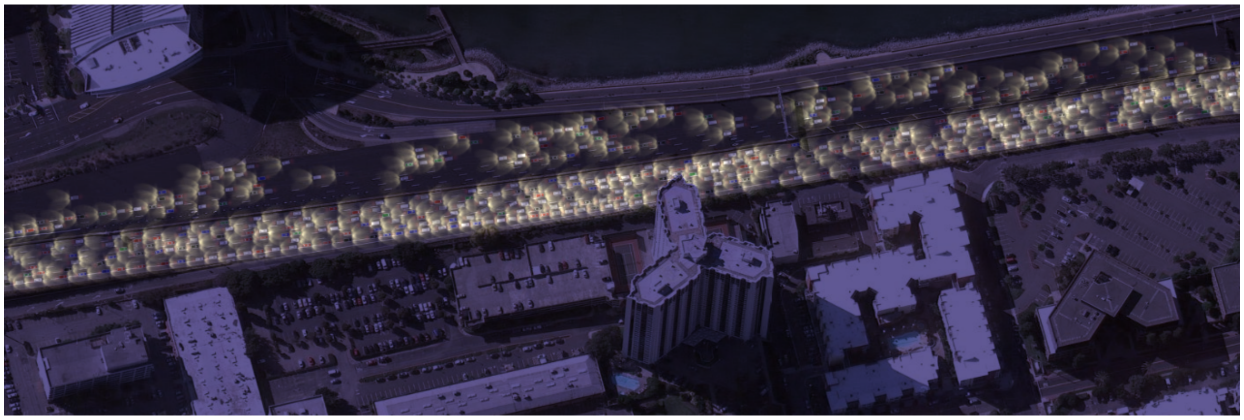


Figure 1.2: An animation of a reconstructed traffic state.

The most common traffic sensors currently are loop detectors, which are embedded on many highways throughout the world. Recently, sensor networks of loop detectors have been created in same areas, enabling real-time data for many important roads. However, this data is discrete in time, due to aggregation periods, and space, as sensors can be a mile or farther apart. In Chapter 3, I propose a method to create traffic reconstructions from loop detector networks such as this, i.e. the method will estimate the time-varying state of traffic and create a corresponding simulation. Unlike previous work, my method uses the state-of-the-art, second-order ARZ traffic model [Aw and Rascle, 2000, Zhang, 2002]. The resulting simulation will recreate traffic that is qualitatively and quantitatively similar to the sensed real-world conditions.

1.4.3 Self-Aware Route Planning.

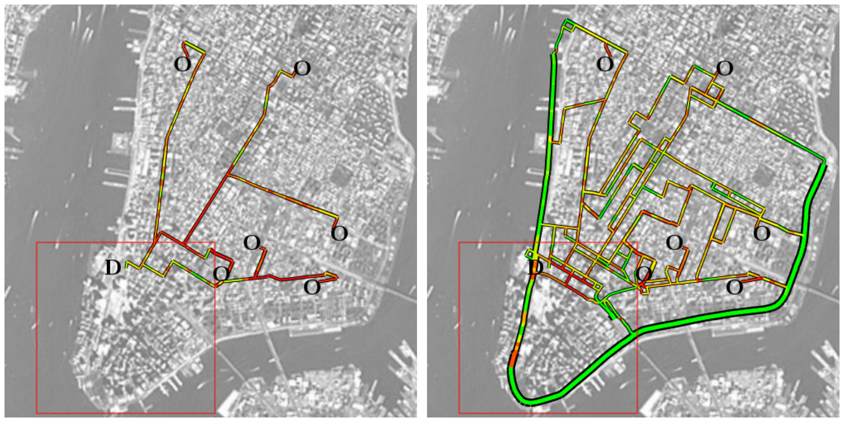


Figure 1.3: On the left, traffic from shortest path routes. On the right, traffic from my routing system.

Even with the knowledge of the state of the traffic system, it can be a challenge to optimize the overall performance. One possibility is to route vehicles using the knowledge you have about traffic conditions, but if a large percentage of vehicles are routed, they themselves determine the future traffic conditions. In Chapter 4, I propose a method that can efficiently route thousands of vehicles in a manner that minimizes congestion and travel time. By utilizing my method for estimating a vehicle's effect on the traffic density field, my method allows the route planner to take other vehicles into account without any explicit coordination. In realistic metropolitan-scale simulations, my method achieves a travel time speedup of over 2x over shortest path routing.

1.4.4 Mobile System for Participatory Routing.

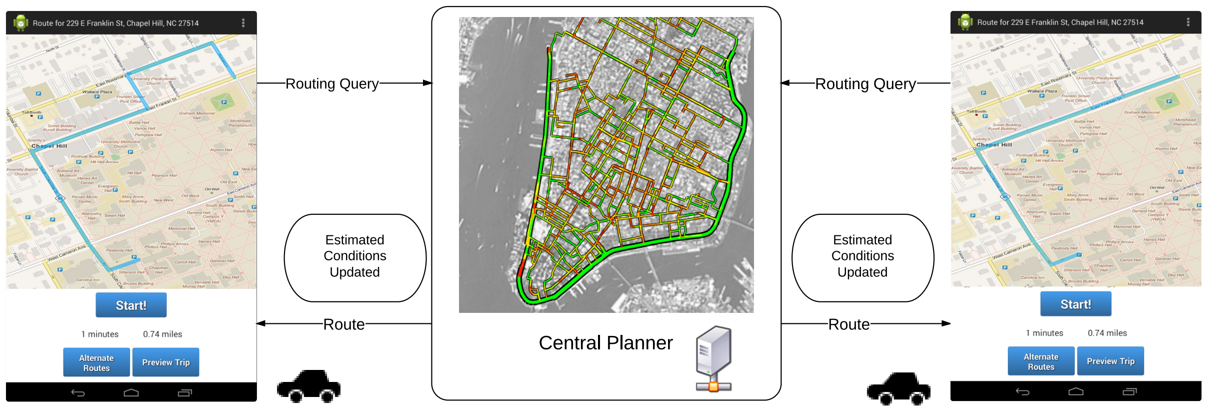


Figure 1.4: A use case diagram for my routing system.

Finally I propose an architecture, design, and prototype implementation of a system that implements the self-aware routing technique. The prototype server can handle 1000 near simultaneous routing requests in under 15 seconds, with an average response time of under a second. Each route planned is used by the system to update its estimation of the traffic conditions. The mobile client features a novel use of destination prediction for ease-of-use and a UI similar to existing navigation systems, with a local map of the road network, highlighted route, and turn-by-turn directions.

1.5 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, I describe my approach to creating simulation-capable road networks. In Chapter 3, I describe my approach to creating visualizations of traffic conditions from sensor data. In Chapter 4, I present a theoretical framework for route planning that considers the effect on traffic of the previously planned routes. In Chapter 5, I expand this method to account for real world conditions and demonstrate empirically that such a system could result in significant improvements to traffic system efficiency. Finally in Chapter 6, I present some concluding remarks and discuss some possible future research directions.

CHAPTER 2: ROAD NETWORKS

2.1 Introduction

Simulation is an important tool for addressing the challenges of traffic system state estimation, optimization, and visualization. However, traffic simulations take place on a complex domain, the road network. The acquisition and representation of this domain present numerous challenges: the domain must match the real-world for the simulation to be useful; it must enable efficient simulation to handle large cities; and the underlying structure is complex and detailed, featuring both topological and geometric elements.

Traffic simulation describes large numbers of vehicles on a traffic network by taking advantage of the reduced dimensionality typically found on road networks: vehicles follow roads, and their motion can be described with few degrees of freedom. Traffic simulations take place on a network of lanes, and this network needs to be represented with all its details, including the number of lanes on a road, intersections, merging zones, and ramps. Digital representations of real-world road networks are commonly available, but the level of detail and noisiness of these data makes them not usable for many traffic simulation applications.

The work presented in this Chapter is primarily aimed at augmenting freely-available data sets with sufficient detail to allow for useful vehicle motion synthesis. I introduce an efficient approach for automatically transform geographic information system (GIS) data, i.e. polyline roads and associated metadata, into functional road models for large-scale traffic simulations. The resulting representation consists of two tightly integrated components, (1) a lane-centric topological representation of complex road networks and (2) an arc-road representation for geometric modeling of the road networks. The resulting model has the following characteristics:

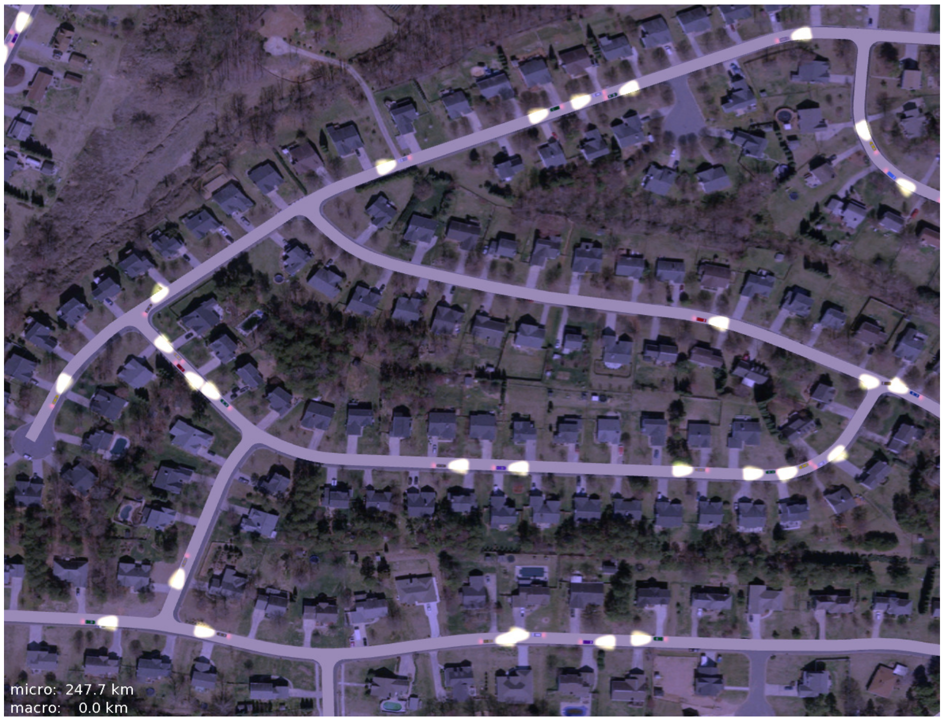


Figure 2.1: A road network generated directly from GIS data by my method. The road network has been overlaid on top of a satellite image. Note that the cars on the road network are animated using a traffic simulator running on my road network representation.

- It provides a road network representation with the necessary details for traffic simulation and realistic visualization using GIS data as input;
- The resulting road models are C^1 continuous and well-defined across the entire simulation domain;
- It is computationally efficient for performing geometric operations, such as computing the distance between cars, location-based queries, *etc.*

I demonstrate the effectiveness of the high-detail road networks automatically created by my approach using two different contemporary traffic simulation models, the continuum based method of Sewall et al. [Sewall et al., 2010a] and the agent-based simulation method of Treiber et al. [Treiber et al., 2000]. I use these simulators to create traffic visualizations on realistic road networks overlaid on satellite images. In Fig. 5.1, we can see an example road network generated by my method seamlessly overlaid on a satellite photograph and used for a real-time traffic simulation and visualization.

Challenges. To accomplish this, I attack numerous challenges. First, constructing the intersection, ramp, and road geometries presents numerous special and degenerate cases, typical of geometric computation. My method is designed to automatically handle as many of these cases as possible. Second, GIS data of road networks are not intended to be used for simulation. I reformulate these networks in order to extrapolate a network on which simulation can be done. Third, the data as available requires filtering in order to be processed; while this is not the main focus of my work, it is a challenge that I address in this Chapter. Fourth, these networks are large in scale, and so efficient algorithms and implementations are required. Fifth, designing the interactions of the system itself is a challenge as this project is a combination of multiple systems: a road network importer, a road network representation, a simulation system, and a visualization system. Finally, there are algorithmic challenges in capturing details such as overpasses and in defining arc roads.

The Chapter is organized as follows. In Section 2, I discuss existing road network representations, both commercial and public domain, and prior work in representing roads. In Section 3, I discuss the specific requirements that traffic simulation imposes on a road network representation and give an overview of my approach. In Section 4, I discuss the topological processing I do in order to create a road network. In Section 5, I discuss the handling of overpasses and underpasses. In Section 6, I discuss our results and validation. In Section 7, I present my concluding remarks.

2.2 Related Work

Digital representations of traffic networks have been widely used for tasks such as civil planning, consumer-level GPS systems, simulation, and visual applications like maps, games, films, and virtual environments, yet each application requires different types of information about the road network. For many display and routing applications, simple graphs with edge metadata are sufficient; for other applications, such as traffic simulation or driving in a virtual world, geometric details about the lanes that constitute the network, their topological arrangement, the layout of intersections, traffic-light timing behavior, road surfaces, and other information are needed.

2.2.1 GIS Data, Tools, and Software Systems

While digital road networks are widely available, the amount of detail varies widely across sources. Data for North America and Europe are freely available from the U.S. Census Bureau's TIGER/Line® database [U.S. Census Bureau, 2010] and 'crowd-sourced' community projects like OpenStreetMaps [OpenStreetMap community, 2010], but these data sets contain polyline roads with minimal attributes — information about lanes and intersection structure is wholly missing. Commercially-available data sets, such as those provided by NAVTEQ [NAVTEQ, 2010], often contain some further attributes, such as the lane arrangements at intersections, but they are expensive to obtain, the techniques used are not known, and they do not capture all of the desired detail.

Numerous methods have been proposed for automatic and semi-automatic GIS road extraction from aerial and satellite images. Extensive surveys include [Mena, 2003], [Park et al., 2002], and [Fortier et al., 1999]. These methods are complimentary to my work: the GIS network I assume as input could be the product of a satellite image extraction method.

Procedural modeling of cities and roads have been an active area of research interest in computer graphics. For example, recent work by [Galin et al., 2010] and [Chen et al., 2008], among a notable body of investigation, have enabled the generation of detailed, realistic urban layouts and roads for visualization.

Commercial procedural city modelling software is also available. For example, consider the intersection geometry generated by CityEngine® shown in Figure 2.2. Here, the intersection is modelled as a square connected to neighboring rectangles with narrow triangles.

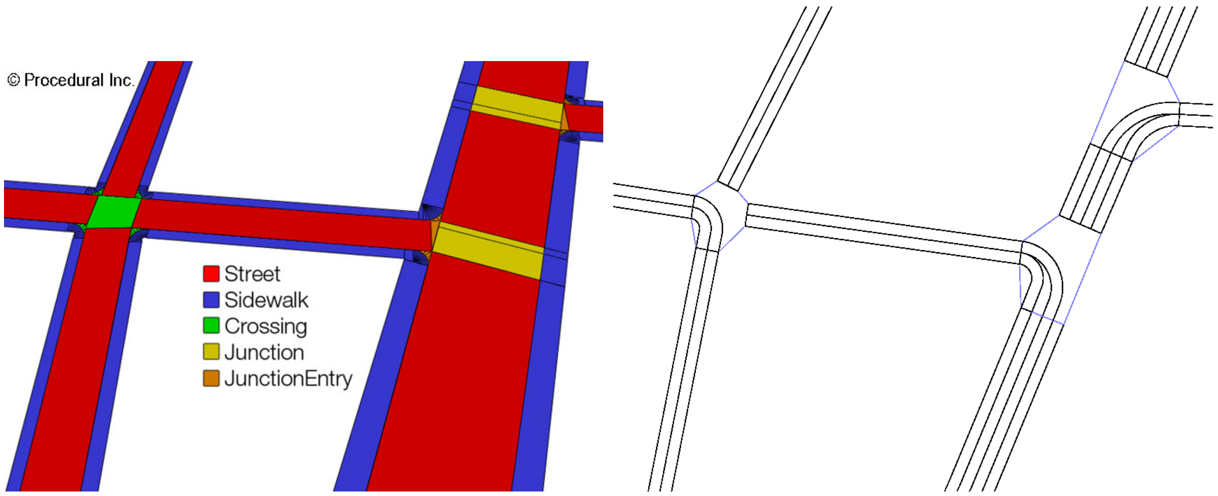


Figure 2.2: Geometry for a simple road network created by CityEngine®[Prodecural Inc., 2011] is shown in (a). In (b), I show the geometry created by our method for a similar road network. Note that only the lanes in the intersections that currently have a green light are shown.

In this work, I construct the geometry for every lane, not just the roads; the lane connections are C^1 continuous, and the geometry defines all the needed parameters for vehicle animation, including orientation and steering angle.

2.2.2 Geometric Representation

Numerous spatial representations of curves have been developed over the years — see the comprehensive books by Farin [Farin, 1996] and Cohen et al. [Cohen et al., 2001]. However, road networks and traffic behavior have specific requirements: existing curve representations are not the best suited for modeling road networks to support real-time traffic simulations.

For example, the popular NURBS formulation [Piegl and Tiller, 1997], despite of its generality of representations, is costly in space and efficiency. In particular, many splines do not readily admit arc-length parametrizations: those must be obtained using relatively expensive numerical integration techniques for establishing vehicle positions and for describing quantities of vehicles on each lane in traffic simulators.

Willemsen et al. [Willemsen et al., 2006] describe *ribbon networks*, specifically discussing the need for 'fattened' splines to describe road shapes, and our technique is potentially complimentary to the modeling technique for road networks they present. However, they use the representation of Wang et al. [Wang et al., 2002], which is only approximately arc-length parametrized and requires iterative techniques for evaluation. In contrast, our method only needs a simpler and much cheaper direct evaluation.

van den Berg and Overmars [van den Berg and Overmars, 2007] proposed a model of roadmaps for robot motion planning using connected *clothoid* curves. However, their choice of representation is based solely on the need to generate vehicle motion. For both traffic visualization and simulation, the representation must also be suitable for the generation of road surfaces, which are not necessarily clothoid curves. Additionally, clothoid curves are expensive to compute — requiring the evaluation of Fresnel integrals — whereas our method relies solely on coordinate frames, sines, and cosines.

Nieuwenhuisen et al. [Nieuwenhuisen et al., 2004] use circular arcs, as we do, to represent curves, but these arcs are used to smooth the corners of roadmaps for motion planning as in [van den Berg and Overmars, 2007]. Furthermore, neither of these techniques have been investigated for the case of extracting ribbon-like surfaces, as we do, nor is there an established technique for fitting them to multi-segment, non-planar polylines.

I have developed a representation that offers (1) an ease of extension from widely available polyline data, and (2) a low cost to compute, evaluate, and perform geometric queries on the road model.

2.3 Preliminaries

2.3.1 Simulation Requirements

The common formulations for traffic simulation are lane-based. These lanes are treated as queues of cars, represented either as discrete agents or by continuous density values. For traffic simulation, lane geometry is irrelevant as long as speed limits and distances are available. However, geometry matters for visualization and for localizing data, such as cell phone or GPS transmissions sent to inform about traffic conditions. These lanes are connected in various ways to form a road network, and cars traverse these connected lanes by crossing intersections and merging between adjacent lanes.

The principal requirement for simulation is the creation of this network of lanes. This includes the division of roads into lanes, but also the creation of transient 'virtual' lanes within intersections: these virtual lanes exist only during specific states of a traffic signal. The creation of the network of lanes also entails determining the topological relationships between lanes (so that vehicles can change lanes and take on- and off-ramps) and making geometric modifications to the road network to allow the construction of 2D or 3D road geometry.

To efficiently support traffic simulation, there are a number of queries the network needs to be able to answer in a computationally efficient manner. The nature of these queries depends on the simulation technique, (i.e. whether the technique is continuum-based or discrete). Additionally, it is desirable that the road network representation abstract away the details the queries on the road network to maintain clear separation and software modularity between the traffic simulation and the road network.

2.3.1.1 Discrete Simulation

A discrete formulation, commonly called *microscopic* simulation (e.g. agent-based simulations), focuses on the interactions between individual cars, typically by using a *leader-follower* formula to calculate each cars' acceleration. For example

$$a_c = f(v_l, a_l, |c - l|)$$

calculates the acceleration for the car c based on the acceleration and velocity of the leading car l as well as the distance between c and l. Therefore, one requirement is that the road network representation be able to facilitate this leader-follower query. The specific formulas for this type of equation vary, but they typically require the state of the leading car and the distance along the road to that car, which I respectively call *get_leader*(c) and *get_free_dist*(c).

get_leader(c) is defined as a mapping from a car c to a car l. Let R_c be the *route* of c, where *route* is defined as an ordered set of roads such that for $r_i, r_{i+1} \in R_c$, the last vertex of r_i is the first vertex of r_{i+1}. Note that this does not require the simulator to use routing: the route can be defined as the current free path through the network ahead of the car c. It must be the case that for c and $l = \textit{get\_leader}(c)$, no cars exist between c and l along R_c. When there are no cars along R_c (or when there are no cars on R_c up to some specified distance), *get_leader*(c) must return a *virtual car.* The state of this virtual car can be defined on a per simulation basis: some reasonable definitions would be 1) a stationary car at a position sufficiently far ahead of c as to have a minimal impact on its calculations, i.e. the free distance is expected to dominate the leader-follower calculation, and 2) a car moving at the speed limit of some road in R_c at a sufficient distance ahead of c. For boundaries, such as the end of lanes and temporary stops at intersections, a virtual car of type (1) should be returned such that it's position is at the end of the lane.

get_free_dist(c) is defined as the distance from c to $l = \textit{get\_leader}(c)$ along R_c. This operation is dependent on the geometric structure used and motivates our method of *arc roads*, which have a closed form for length calculation.

2.3.2 Continuum simulation

For continuum formulations, commonly called *macroscopic* methods, the lanes are divided into cells where traffic state data are stored. As with the microscopic formulation, this requires that distances along the lanes can be computed.

Both formulations require that the network have the capability to efficiently cycle through the cars in all the lanes, in order to update their states (or update the continuum quantities of all the lane elements). Additionally, cars must be easily moved between lanes to allow for merging behavior and intersection traffic. Finally, for both visualization and for accurate representation of roads, the road network must use a visually smooth (C^1) geometric representation for lanes.

In summary, our method constructs a representation capable of efficient simulation by fulfilling specific requirements for traffic simulation, such as

- **A network of lanes:** I construct a graph with formal properties, then process the graph to construct a network of lanes with the correct topological relationships, including temporal connections at intersections and intervals that allow merging.
- **Intersections with connections and states:** I use a geometric method to truncate roads at intersections and create internal lanes for the intersection to allow through traffic. Our method can ensure that no turn is made that would violate a car's kinematic constraint on turning radius.
- **Fast calculations for $get\_leader(c)$ and $get\_free\_dist(c)$:** our method uses a geometric representation with a closed form length formula and a well-defined network of lanes and intersections for easy graph traversal.
- **Simple interface between simulation and the road representation:** our system allows for a high-level language interface. The road network representation is independent from any single simulation methodology.

2.3.3 System Overview

Our system takes a road network representation from a GIS source as input. This representation is assumed to contain polyline roads along with metadata consisting primarily of road classifications. From these road classifications, I estimate data such as the number of lanes on the road and the speed limit.

There are two phases for our system and two resulting outputs. First, there is a topological phase, in which the semantics of the network are encoded in a graph. And second, there is a geometric phase, in which the lanes and intersections are described by visually smooth, ribbon-like geometry.

In the topological phase, I first enforce constraints on the network. Primarily, as will be discussed below, I enforce a formal definition of a road as a polyline with two boundary vertices of degree not equal to two and all internal vertices having degree two. GIS data often requires filtering, including removing duplicate nodes, ensuring the vertices in a road follow the logical order of the road, ensuring one way roads are defined in the correct direction, *etc.*. I discuss filtering below.

This phase also ensures that all the interfaces between the lanes are well-defined: normal intersections have states and internal lanes; neighboring lanes have merging zones defined and the functionality for a simulator to use the zones; and ramps flow into highway merging lanes, even if the final geometry of the ramps is not yet defined.

In the geometric phase, every lane is assigned boundary curves that are calculated using the underlying polyline road representation, the offset of the lane from the road's center line, and a geometric representation introduced in the Appendix. This representation both captures the curves of the physical roads and allows fast distance calculations needed for the simulation formulation.

2.3.4 GIS Data Filtering

I filter the GIS data I use to remove the most commonly occurring errors. These changes are not meant to change the underlying geometry or topology of the network, only to correct sloppy data creation. The first filter removes points that are $\epsilon-$coincident, where ϵ is a distance argument that is kept on the order of feet. This is done prior to the splitting and joining algorithms discussed in Section 2.4.1, while the remaining filters are applied afterwards. The second filter removes collinear points within roads. The third filter ensures that no point added to a road causes it to turn too sharply or double back on itself. This filter calculates the offset, as in Figure 2.3, that would be required for a circle of minimum turning radius to be inscribed within the polyline segments. If this offset is greater than half the length of either segment, the node is not added. This ensures that when a point is added to the road, the road still satisfies the kinematic constraints of a typical car. Further filtering includes ensuring that one way roads are defined in the correct direction and that roads have been assigned the correct classification.

2.4 A Lane-Centric Graph Representation

This section discusses the transformation of GIS map data into a road network representation suitable for use in traffic simulation.

For the purposes of formal communication, I present aspects of this process using matrix notation. The road network can be represented as a directed graph, consisting of vertices, V, and edges, E. Every edge $e \in E$ has a starting vertex, e_s, and an ending vertex, e_e. I assume the vertices are sampled along the center lines of the physical roads of the network. I can describe the connectivity between the edges and vertices using a graph represented by an incidence matrix, M,

$$M_{|V|,|E|} = \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,|E|} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,|E|} \\ \vdots & \vdots & \ddots & \vdots \\ m_{|V|,1} & m_{|V|,2} & \cdots & m_{|V|,|E|} \end{pmatrix}.$$

Each element of the matrix at row i and column j is defined as

$$m_{i,j} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{if } v_i \notin e_j \end{cases}$$

Every vertex has the operator *degree* defined as the number of coincident edges, $\mathit{degree}(v_i) = \hat{e}_i^T \cdot M = \sum_{j=0}^{n} m_{i,j}$.

2.4.1 Roads

I introduce the data structure of *road* and define it as an ordered set of vertices, R, with a starting vertex r_s and an ending vertex r_e such that for all $r_i \in R$, $degree(r_i) = 2$ if and only if $r_i \notin \{r_s, r_e\}$ and $edge(r_i, r_{i+1}) \in M$. This implies that a *road* ends at a higher degree node or a node with degree one, *i.e.* an intersection or a dead end.

While GIS data sets have roads defined, it is likely that the data contains errors or does not strictly adhere to the rules I want to assume. To ensure the above definition holds on our data set, I perform two operations, *road split* and *road join*. These operations are performed on sets of vertices derived from GIS polylines.

2.4.1.1 Road Split

Let the internal vertices, $internal(R)$, be all $r_i \in R$ such that $r_i \notin \{r_s, r_e\}$. To satisfy the *road* definition given above, $\forall r_i \in internal(R)$, $degree(r_i) = 2$. Intuitively, this differs from the colloquial use of road in that *road*s do not go through intersections: they start and stop at dead ends or intersections.

The split operation is defined as a mapping from a set of vertices $p \in P$, where $edge(p_i, p_{i+1}) \in M$, to a set of sets $S = \{S_0, S_1, ...\}$ such that for all $S_i \in S$, for all $s \in internal(S_i)$, $degree(s) = 2$ and $\bigcup S_i = P$. This is achieved by Algorithm 2.1.

2.4.1.2 Road Join

A set S_i described above differs from a *road* only in that it lacks sufficient constraints on its starting and ending vertices. This condition, $degree(v_s) \neq 2$ and $degree(v_e) \neq 2$, is satisfied by Algorithm 2.2, which iterates over each vertex and joins neighbors S_i and S_j if their coincident vertex has $degree(v_c) = 2$. This algorithm uses $roads(v)$, which maps a vertex to the set of *roads* coincident with that vertex: $roads(v) = \{R | v \in R\}$.

Algorithm 2.1 Algorithm for Road Splitting.

```
Require: A set of vertices $V'$ such that $edge(v'_i, v'_{i+1}) \in M$.
Ensure: For all output $S_j \in S$, for all $v \in internal(S_j)$, $degree(v) = 2$.
  $S = \{\}$
  $S_j = \{V'_s\}$
  for all $v \in internal(V')$ do
    $S_j \leftarrow v$
    if $degree(v) > 2$ then
      $S \leftarrow S_j$
      $S_j = \{v\}$
    end if
  end for
  $S_j \leftarrow V'_e$
  $S \leftarrow S_j$
  return $S$
```

Of final note in Algorithm 2.2, the *join* operation adds every vertex of its second argument to its first argument in order and removes the vertices from the second argument, updating $roads(v)$.

2.4.1.3 Proof of Road Creation

Before proving that the above creates *roads*, I define a *degenerate road* D as a *road* in all ways except for $degree(d_s) = degree(d_e) = 2$ and $roads(d_s) = roads(d_e) = D$. In other words, a degenerate road is a loop, disconnected from the rest of the network.

Theorem 1. *Given a road network M and disjoint sets of vertices $S_i \in S$, the result of applying Algorithm 2.1 to each set S_i and applying Algorithm 2.2 is a set of roads $R_i \in R$ and a set of degenerate roads.*

Proof. Suppose on the contrary there exists a set of vertices R' produced by the above methods that is not a *road* or *degenerate road*.

R' then either has a vertex $v \in internal(R')$ with $degree(v) \neq 2$ or a vertex $u \in \{r_s, r_e\}$ with $degree(u) = |roads(u)| = 2$.

For any vertex $v \in V$, Algorithm 2.2 will ensure that v cannot have $degree(v) = 2$ and $|roads(v)| = 2$. Every vertex is processed. For any vertex with $degree(v) = 2$ and

Algorithm 2.2 Algorithm for Road Joining.

Require: The set of all vertices V in M, a set of *road* R

Ensure: For all $v \in V$, $degree(v) = 2 \Rightarrow |roads(v)| = 1$.

```
$toDelete \leftarrow \{\}$
for all $v \in V$ do
  if $degree(v) = 2$ then
    if $|Roads(v)| = 2$ then
      $a, b \leftarrow Roads(v)$
      if $a_e = b_s = v$ then
        $swap(a, b)$
      end if
      if $a_s = b_e = v$ then
        $join(a, b)$
        $toDelete \leftarrow b$
      end if
      if $a_e = b_e = v$ then
        $a \leftarrow reverse(a)$
        $join(a, b)$
        $toDelete \leftarrow b$
      end if
      if $a_s = b_s = v$ then
        $b \leftarrow reverse(b)$
        $join(a, b)$
        $toDelete \leftarrow b$
      end if
    end if
  end if
end for
return $R \setminus toDelete$
```

$|roads(v)| = 2$, one of the exhaustive joining cases will be executed resulting in $|roads(v)| = 1$. As no vertex exists with $degree(v) = 2$ and $|roads(v)| = 2$, the road R' cannot begin or end at such a vertex. Therefore, R' must either begin and end at vertices with $degree(v) \neq 2$, or R' must be a degenerate road that begins and ends and the same vertex.

Therefore, R' must have a vertex $v \in internal(R')$ with $degree(v) \neq 2$. However, as R' is a result of Algorithm 2.1, and as Algorithm 2.1 splits the set at every vertex with $degree(v) > 2$, no vertex in $internal(R')$ can have $degree(v) > 2$. Further, no vertex in $internal(R')$ can have $degree(v) < 2$, as that would contradict v being an internal vertex.

Therefore, every vertex $v \in internal(R')$ has $degree(v) = 2$ and neither v_e nor v_s have $degree(v) = 2$ and $|roads(v)| = 2$. R' is either a *road* or *degenerate road*, which contradicts our assumption. $\square$

2.4.2 Lanes

The commonly used simulation formulations are lane-based. Therefore, lanes must exist to hold cars, and they must have a relation to the roads. I assume that every road has a known number of lanes, and that these lanes belong fully to their associated roads. Each lane has the following data: an offset value, which defines how far its center line is displaced from the road center line; adjacency intervals, which define which lanes are adjacent to the lane and where they are adjacent (to allow for merging); a road membership, and a lane width value. The adjacency intervals of a lane are defined as $\{A_1, A_2, ..., A_n\}$, where $A_i = \{s_i, e_i, os_i, oe_i, l_i\}$ and $s_i \in [0, 1]$ is the parametric starting point on the lane of the adjacency interval, e_i is the intervals parametric ending point, and os_1 and oe_1 are the parametric bounds for the adjacent lane. l_i is a reference to the lane which is adjacent in the i^{th} interval. The road membership is simply one interval $\{s, e\}$ where $s, e \in [0, 1]$ are the parametric bounds that determine where on the road the lane starts and where it stops.

2.4.3 Intersections

Our road network contains polyline roads that terminate at dead ends or at intersections. In a realistic road network, intersections have their own geometries. For physical roads that meet at intersections, I can say that the roads are 2-manifolds with boundaries. As simulation systems require 1D lane structures, it is not sufficient to only create the geometry of these intersections; lanes also need to be created to define how traffic can move through the intersection at time t.

In this work, I consider two classes of intersections, signaled intersections and highway ramps. Other classes of intersections, such as n-way stops or traffic circles, have similar geometric construction as the intersection classes described here, but they require different handling at the simulation level. Signaled intersections feature a traffic light that determines the state of the intersection. This state defines which incoming lanes can send traffic into the intersection and to which outgoing lanes that traffic can flow. In our representation, this corresponds to a state defining which internal lanes exist at a certain time. For ramp class intersections, one road becomes an additional lane for a second road for some spatial interval. This allows cars on the first road to merge onto or off of the second. Our method uses a rule-based classifier[1] to determine the intersection type, but the classifier is separate from the intersection construction, and a more advanced classifier could be used with no modification to our method. For example, a classifier using a machine learning technique on satellite image data could be used to determine intersection class.

2.4.3.1 Signalized Intersections

Let $s \subset V$ be the vertices classified as signalized intersections. As shown in Fig. 2.3, I calculate an offset o for each road that is dependent on the desired minimum turning

[1]Our system classifies based on the road type information provided in the GIS metadata. Intersections on the highway type roads are treated as ramps, and all other intersections are considered signalized.

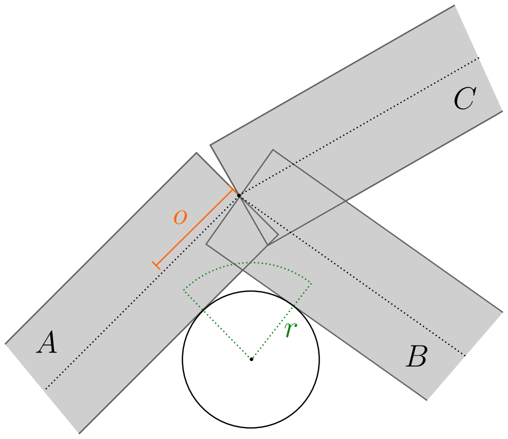


Figure 2.3: A simple intersection with three roads. For each road, I calculate an offset, o, based on each of its neighbors. Here, I see the calculation of the offset for the road A with respect to B. To calculate the offset, first the position of a circle tangent to A and B is calculated with a radius such that a car turning from A to B will have a turning radius of r. The offset is then the length on A from the intersection to the projection of the center of the circle onto A.

radius of the intersection, which is a user specified value that can be intersection specific and parametrized by speed limit, road type, or other road safety requirements, for example.

To calculate this offset, the roads are sorted by the angle each forms with the x-axis to yield a clockwise ordering. For each road R_j, I calculate the offset needed for a circle of the specified radius to be tangent to both the boundary of R_j and its clockwise and counterclockwise neighbors. The final offset assigned to R_j is the maximum offset found for either neighbor, which guarantees that no radius smaller than the specified is needed to make a turn from the end of R_j to either of its neighbors.

For some roads, the offset calculated to satisfy the minimum turning radius will be longer than the roads themselves. This is typically the case for small roads and roads that make very acute angles. If an offset for a road R is longer than the length of R, I propose collapsing the vertices v_e and v_s, the starting and ending vertices of R, combining the intersections those vertices form. The road R is then deleted from the network. As the vertices were collapsed, the topology of the network is preserved, if not the geometry.

States. Timer-based signalized intersections have an ordered set of states S in which each state $s \in S$ is defined as $s = \{P, h\}$, where $P = \{\{I_1, O_1\}, \{I_2, O_2\}, ..., \{I_m, O_m\}\}$ and $\{I_j, O_j\}$ is a pairing of an input lane and an output lane, and h is the duration for the state. The actual states for an intersection are unknown from the GIS data alone. Therefore, I assume that every pair of roads in $roads(v)$ are joined in a state, and each state is of equal duration. Further data on the actual states or more advanced methods of estimating the states could trivially be integrated with our approach.

2.4.3.2 Ramp Intersections

For vertices classified as ramp intersections, I will call one road the *ramp* and one road the *highway*, as this is where this class of intersection commonly occurs. Our end goal is to have the ramp end alongside the highway and to have a merging lane added to the highway for an interval before or after the ramp, depending on whether the ramp is an *onramp* or *offramp*[2]. The steps needed to perform this transformation are 1) joining the highway roads that connect at the intersection, 2) transforming the geometry of the ramp so that the ramp becomes tangent to the highway, and 3) adding a merging lane to the highway.

1) To accomplish this, I remove R_m from $roads(v_t)$ and decrease $degree(v_t)$ by one. I then execute Algorithm 2.2 on the intersection point to merge the highway roads that contain it.

2) The ramp needs to be tangent to the highway so cars do not appear to vanish from one road and appear on another or undergo a sudden change in orientation. To do this, I create a new vertex v_r to serve as the ramp's intersection point. I locate the closest point p on the highway's geometric representation at an offset equal to the $(n+1)^{th}$ lane, where n is the number of lanes of the highway. The intersection point of the ramp is set to be p. Additionally, a vector $\mathbf{u}$ tangent to the highway at p is calculated in the opposite direction

[2]The ramps are defined in the direction of the flow of traffic. If last vertex in the ramp is the intersection point, the ramp is an onramp. Else it is an offramp.

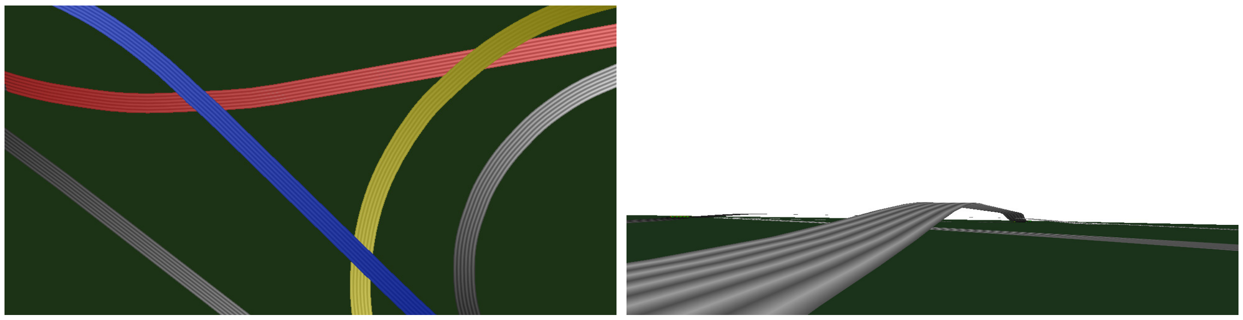


Figure 2.4: A highway interchange with overpasses generated by our method. Top: An overhead view of a highway interchange with overpasses. The highways are colored to show the arrangement computed by our method. Bottom: A view of the blue highway, which is given the greatest height.

of the ramp. A vertex is added to the ramp equal to $\mathbf{u} * \epsilon + p$ to ensure that the ramp approaches tangent to the highway.

3) Finally, the cars entering or leaving the highway need an interval in which they can merge onto or off of the highway. I consider these merging lanes to be part of the highway. Therefore, I add a merging lane to the highway that starts at the parameter value of p and continues for a user-defined distance. In our demonstrations, I used a value of 60m.

2.5 Overpasses and Underpasses

Another feature of real road networks is the presence of overpasses and underpasses, including the complex weaving of roads at multi-highway interchanges. While it is only necessary to capture the topological relationships between these roads for current traffic simulation formulations (as traffic simulations only take place in 1D), a reasonable method for estimating road heights is needed for 3D modeling. These height calculations can be done following the procedures described above, resulting in overpasses as shown in Figure 2.4.

I assume that underpasses and overpasses are represented in the GIS data by roads that contain line segments that intersect but have no shared nodes. In other words, the overpasses are represented implicitly, and the point of intersection must be calculated from the data.

2.5.1 Intersection Points

The problem of calculating intersection points among line segments lacking any structure is well-studied in computational geometry, with optimal algorithms being $\mathcal{O}(N \log N + K)$ where N is the number of line segments and K is the number of intersecting points [Balaban, 1995]. These algorithms typically consider segments with shared endpoints to be colliding, which is not a desired assumption for our application. To handle this, every road segment of every road can be truncated by ϵ to avoid these intersections being found, or a post-processing step can be done to discard all intersections for which the only intersecting segments are neighbors.

2.5.2 Road Height Levels

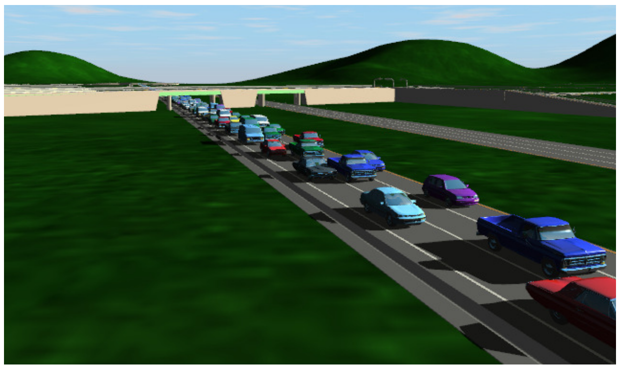
Once the intersection points have been either given or found, the road heights at those points need to be determined. In simple cases, one road will be above and one below. The specific order might be given in the metadata. In general, however, there could be multiple levels of overpasses. Our method divides the overpasses into levels with the goal of having the smallest number of roads elevated, as would be the most cost-efficient approach for a real road system.

I consider the k intersection points and their corresponding *roads*. As a pre-processing step, I first divide every *road* into multiple sets such that (1) no road intersects itself and (2) no two intersections on a road are separated by a distance greater than δ. The reason for (1) is so that I can consider every road as being on a single level, and the reason for (2) is that intersections that are separated by a large distance should be considered independently.

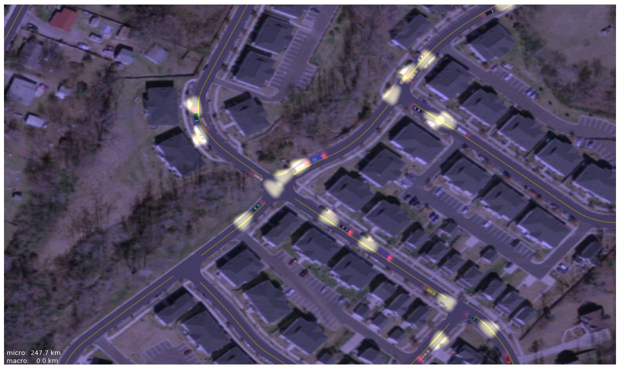
Given the above, our proposed method is to generate the set of *conflicting roads*, C, for each road with an intersection. The roads are then sorted by $|C|$ in ascending order[3] and placed in a priority queue Q_1. For the i^{th} step of the algorithm, I build level i: until Q_i is

[3] I can also add meta data into the sorting procedure. For example, if I wanted highways to be underpasses wherever possible, I could add road type as a tie-breaker in the sort.

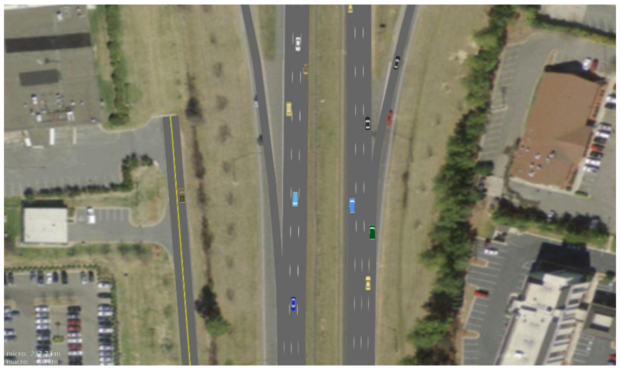
(a) A large scene featuring two towns and a highway.



(b) An overpass created with our method and procedural modeling.



(c) A suburban neighborhood road network created by our method and aligned with a satellite image.



(d) A divided highway with two ramps created by our method.

Figure 2.5: A collection of road networks, all generated by our method. The traffic in these figures was simulated using the technique of [Sewall et al., 2010a]

empty, I remove the road j with the smallest $|C_j|$. This road is set to level i. For every road c in C_j, I remove c from Q_i, and add it to Q_{i+1}. When Q_i is empty, the loop repeats for Q_{i+1} until no roads remain.

The result of this loop is that all the roads that don't conflict with one another are placed on the same level, and the levels are ordered such that those roads that conflict with the greatest number of other roads are placed at higher levels: this is a greedy approach to create levels that have fewer roads the higher the level.

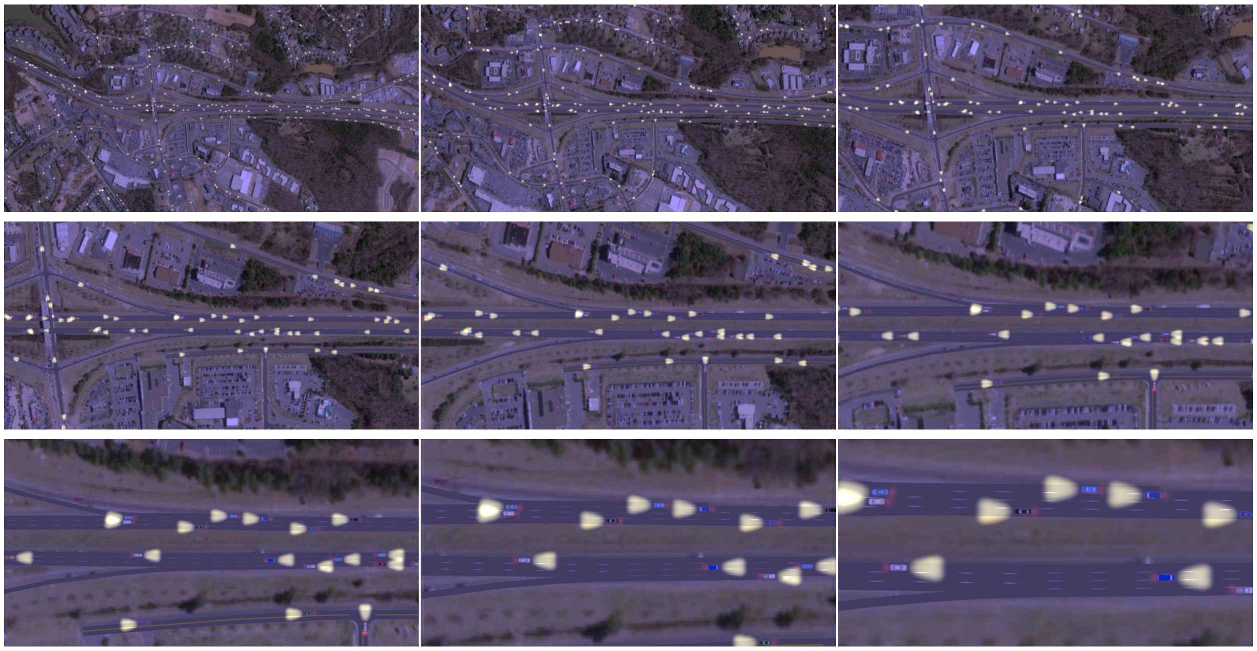


Figure 2.6: A series of images showing ramps smoothly connected to a highway, at a variety of scales

2.6 Results

I have implemented our method described in this paper. In Figure 2.5, I show some example road networks created by our technique used in real-time traffic simulation. The models illustrated here were created using GIS data from the Open Street Map website. The first image, Figure 2.5(a), shows a large road network created within only a few seconds by our method. The network features two small towns in North Carolina, Biscoe and Star, connected by a highway with multiple overpasses and ramps. Figure 2.5(b) shows a highway overpass that was created using our GIS processing and 3D procedural modeling. Figure 2.5(c) shows a road network created by our method from GIS data overlaid on top of a satellite image. Notice the degree to which the roads align, despite the fact that the original data is C^0 polylines. Figure 2.5(d) shows another road representation created by our method and overlaid on a satellite image. This map is from another set of highways with an on-ramp and an off-ramp. Note that the ramps C^1 smoothly connect to the highway, enabling simulated cars to merge onto and off of the highway seamlessly, given the input polyline data not suitable for traffic simulation. This is also shown in the sequence of images in Figure 2.6, which show an on-ramp and off-ramp to a highway from above at a decreasing height.

As shown in Figure 2.5(a), 2.5(c) and 2.5(d), the seamless alignment between the large-scale road networks in the satellite images and the models created directly from GIS data using our method provides a convincing visual validation of our technique. The functional validity of the road networks created by our algorithm is further demonstrated by the successful real-time traffic simulations on the road networks generated from real-world GIS data, with the resulting visual simulations mapped back to the original satellite images showing no noticeable artifacts in the simulated traffic flows. The video clips for the real-time traffic simulations using the road network models created by our method, as shown in Figure 2.5(a) to Figure 2.5(d), can be found at:

`http://gamma.cs.unc.edu/RoadNetwork`.

2.7 Summary

I have presented a method for transforming GIS data into a topological and geometric representation suitable for use in traffic simulation. The geometric representation of roads created by my method is visually smooth, including at ramps and intersections. The method preserves the topological relationships of the GIS road network. I have shown examples of GIS data that have been processed by our method and composed with satellite images. These figures illustrate features of the generated road networks, such as intersection handling and highway ramps, as well as the extensive scale of models that my method can process within a matter of few seconds.

CHAPTER 3: STATE ESTIMATION

3.1 Introduction

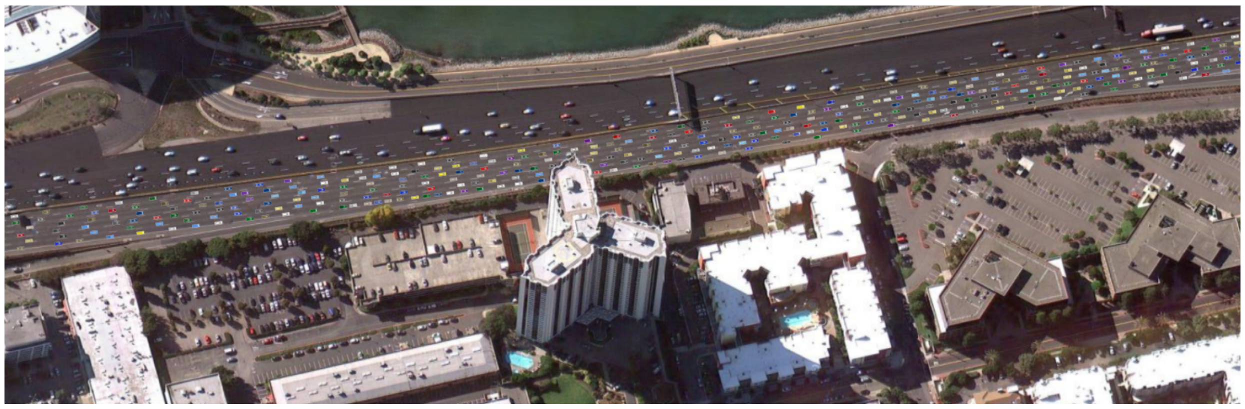


Figure 3.1: The divided highway is populated with traffic using my flow reconstruction method. Individual virtual cars are animated, and the dynamic state corresponds to the real-world traffic conditions observed by sensors.

A perfect model of a road network coupled with a perfect traffic simulator and perfect routing algorithm would all be useless for many real-world applications without an approach to estimating the current traffic conditions. Today, the state of the traffic system at any time is largely unknown, and can only be estimated using historical and real-time sensor data. The most common and accurate form of sensor data for traffic is from loop detectors, which are typically located on highways. These devices record the number of vehicles that pass and their velocities, but are sparsely located. In this Chapter, I propose an approach to using loop detector measurements, which are discrete in time and space, to reconstruct both the continuus traffic and a consistent microscopic visualization. With this reconstructed state, I enable real-world applications and analysis that depend upon having a detailed estimate of traffic conditions.

Another use of my reconstruction technique is the population of digital worlds with realistic traffic. Numerous efforts have been devoted to acquiring and visualizing "digital urbanscapes". Over the last decade, there has been considerable progress on multiple fronts, includig acquisition of imagery and 3D models using improved sensing technologies, real-time rendering, and procedural modeling. For example, aerial imagery of most cities is used in Google Earth and Microsoft Virtual Earth. The problem of reconstructing 3D geometric models from videos and scanners has been an active area of research. Similarly, many efficient techniques have been proposed to stream the imagery and geometric data over Internet and display them in real time on high-end workstations or handheld devices. However, all these efforts are limited to capturing, displaying, or modeling predominantly static models of urbanscapes and do not include dynamic elements, such as traffic. The realism of a virtual urbanscape in a digital globe system can be considerably enhanced by introducing such intrinsic dynamic elements of an urban landscape.

As VR applications in flight and driving simulators [Cremer et al., 1997, Donikian et al., 1999, MITSIM, 2011, SUMO, 2009, Pausch et al., 1992, Thomas and Donikian, 2000, Wang et al., 2005] for training have evolved from its earlier single-user VR system into online virtual globe systems and distributed, networked gaming, the demand to recreate large-scale traffic flows, possibly driven by traffic sensor data from the real-world observations, has emerged. The concept of 'virtualized traffic' was first introduced in [van den Berg et al., 2009] to create dynamic vehicle flows based on real-world traffic sensor data to enhance the sense of immersion for a virtual urbanscape.

I propose an efficient technique for reconstructing the traffic state from continuous streams of traffic data *either* from in-road sensor measurements *or* procedurally generated control/user input. This approach consists of (1) a data analysis stage, in which the traffic state is estimated via an ensemble Kalman smoothing process and a macroscopic model for representing aggregate traffic flow, and (2) a visualization stage, in which a detailed reconstruction of the traffic flow is rendered and displayed. The final rendering is a 3D visualization of vehicle traffic as it might appear to a camera, but reconstructed wholly from sparse, discrete traffic data. This technique could be used by someone planning a trip, enabling them to do a flyover or see the traffic from a driver's perspective; it could be used by city planners and traffic controllers to see an integrated visualization of traffic over a large metropolitan area; and it could be used to populate virtual worlds to create a more immersive experience and make vehicle flows more realistically reflecting the real-world traffic. Unlike video recordings from individual cameras, our method uses existing infrastructure, requires very little bandwidth for communication, and allows for large-scale integrated views in any VR application.

Traffic itself is an aggregate of individual vehicles, each moving at a velocity dependent on the surrounding vehicles, the road on which it rides, and conditions too numerous to fully list (or perhaps know), such as the time of day, weather, signage, road work, et cetera. I consider an estimate of traffic flow over a span of roadway to be the combination of a density field and a velocity field. These fields model the macroscopic conditions of traffic flow at a particular instant.

Sensors come in several forms. The most widespread are called loop detectors. These sensors are placed on roadways and record attributes about every vehicle that passes. Another form of sensor is within the vehicle itself: cell phones and GPS devices can monitor the speed of the vehicle and report it along with their position. Finally, video cameras can also be used to monitor traffic, providing full trajectories for all observed vehicles. In this Chapter, I focus on in-road sensing or trajectory recording, such as loop detectors and video cameras. These sensors can detect when a vehicle has passed and can provide such information as the speed the vehicle is traveling, the length of the vehicle, whether the vehicle is a truck or car, *et cetera*. Typically, these sensors batch their transmissions and send aggregate data for all vehicles within a time span. Even if the transmissions are instantaneous, they are spatially and temporally discrete readings. Although our method assumes the available traffic sensor data from loop detectors and video cameras in our implementation, the overall computational framework can be easily generalizable and applicable to other forms of sensor measurements.

Main Result. I present an efficient technique that enables the creation of a detailed 3D traffic reconstruction from sparse traffic data. My method features a traffic state estimate phase, in which an ensemble Kalman smoother and a continuum traffic simulator are used to create an estimate of velocity and density fields over the entire road network. My method uses this estimate as a control for an agent-based traffic simulator to create the 3D animation of individual vehicle motion. The agent-based system is controlled by automatically assigned boundary conditions, the estimated velocity field, and a simplified leader-follower relationship. Finally, the output is a 3D visual display of traffic flow consistent with the original traffic pattern measured by the sensors. It is important to note that my method is not a traffic simulation, but a reconstruction of vehicle flows that corresponds to measured traffic conditions as observed in the real world. Furthermore, the resulting 3D traffic animation can be interacted with and respond to user control and manipulation in a virtual environment.

The rest of the Chapter is organized as follows. Section 3.2 provides related research; state estimation methods and traffic simulation are covered, as well as other approaches to traffic reconstruction. Section 3.3 describes my approach, including both the state estimation phase and data reconstruction phase. Section 3.4 provides statistical and visual results from my approach using real traffic data sets. Section 3.5 concludes the Chapter with discussion and future work.

3.2 Related Work

Research on traffic estimation has a long history, dating back to the 1970s (see an early review by Cremer [Cremer, 1991]). Recent projects have similar goals and approaches: they want a high-level estimation of macroscopic traffic quantities, and they use a filtering algorithm to estimate the state based on incoming sensor data and a model of traffic dynamics. Hegiy et al. [Hegyi et al., 2007] discusses parallelization of a Particle Filter approach to traffic state estimation. Jacquet et al. [Jacquet et al., 2005] present an approach to handle the challenge of shock waves in traffic state estimation. Jacquet et al. [Jacquet et al., 2006] describe an approach that allows gradient descent to be used on traffic simulation models, with applications to traffic state estimation. Sau et al. [Sau et al., 2007] discuss real-time state and parameter estimation using particle filters. Wang and Papageorgiou [Wang and Papageorgiou, 2005] present a detailed investigation of real-time traffic state estimation using the extended Kalman filter. Finally, Work et al. [Work et al., 2010] develop a velocity based traffic model in order to estimate traffic using cell phone signals alone.

The majority of these traffic estimation approaches rely on a Kalman filtering approach. One particular class of filters of note is the *Ensemble Kalman Filter* (EnKF). This is the filter of choice for estimation of systems involving (very) high dimensional states. It has been used for weather prediction [Houtekamer and Mitchell, 2001], ocean current estimation, as well as traffic estimation [Work et al., 2008],[Work et al., 2010]. Similar to a particle filter, the EnKF propagates a set of samples of states that model a probability distribution. For the EnKF these samples form the *ensemble* for the system. Unlike a particle filter, the EnKF is correct in the limit (infinite number of samples) only for linear systems with Gaussian noise. However, the fact that its running time is only *linear* in the dimension of the state, rather than *cubic* as for other filters, makes its use advantageous for non-linear systems as well, if it can be reasonably assumed that the distribution of the state has a single mode. While these works provide an estimation of the entire traffic flow on a single highway, they offer little detail to visualize the actual vehicle traffic condition.

An important component in state estimation is a model of traffic dynamics. A frequently used model is that of Lighthill and Whitham [Lighthill and Whitham, 1955] and Richards [Richards, 1956], called the LWR model, which was a seminal macroscopic traffic dynamics model. Traffic simulation is well studied, and the amount of literature is too vast to summarize here. A detailed survey can be found in [Helbing, 2001]. Some notable methods include the ARZ model, by Aw and Rascle [Aw and Rascle, 2000] and Zhang [Zhang, 2002], which is what this work and earlier continuum simulation [Sewall et al., 2010b] are based on, and the model of Papageorgiou *et al.* [Papageorgiou et al., 1990], which was used to model the traffic dynamics in the traffic state estimation work in [Wang and Papageorgiou, 2005].

This work builds on recent advances in predictive traffic simulation. Wilkie et al. [Wilkie et al., 2012] propose a method for extrapolating GIS road networks into a C^1 smooth road network representation suitable for traffic simulation and animation. I use this method to construct the entire road network for running our traffic simulation on. I also adapt the visualization technique by Sewall et al. [Sewall et al., 2010b] on continuum traffic simulation for a large-scale vehicle traffic visualization using animated, individual vehicle representations instead of particle tracers or vector fields commonly performed in flow visualization [Laramee et al., 2004]. However, [Sewall et al., 2010b] is only for forward simulation, not for data-driven animation or traffic reconstruction. Our 3D traffic visualization take into account of vehicle kinematics and dynamics for lane changing and visualization of individual cars. Our method differs from [van den Berg et al., 2009] and [Sewall et al., 2011a], which reconstruct plausible trajectories for individual cars based on microscopic boundary conditions using priority-based motion planning techniques. Such techniques can quickly become intractable as the resolution of discretization in search space increases [Sewall et al., 2011a]. In contrast, I use discrete agent-based simulations with controlled temporal and spatial constraints that ensure the cross flows among multiple lanes are consistent with the high-level state estimation of the overall traffic flow. I also assume more realistic sensor models in our implementation.

Most recently [Sewall et al., 2011b] proposed a hybrid simulation method that use both macroscopic and microscopic simulators simultaneously operating on distinct regions of a road network. Although both continuum and discrete simulation methods are used in this work, our method differs by first applying the macroscopic method for overall flow estimation and then the particle system simulation to reconstruct the detailed vehicle flow; it then refines the global state estimation that accounts for the difference between the estimated and simulated states of the traffic system due to cross flows among multiple lanes. Our selective, sequential use of each type of simulation methods best exploits the individual method's strength in our iterative estimation-reconstruction-refinement framework and does not apply both methods at the same time to different locations as in [Sewall et al., 2011b].

One simple and straight-forward method to visualize the current traffic condition is direct playback of the captured video data from the road networks. But, the collective loads on the transmission bandwidth and the number of camera installations would make such an obvious approach impractical; furthermore, it cannot offer other computational benefits for driver assistance, navigational aid, and important decision-making analysis critical to other applications. To the best of our knowledge, this work is the first method that offers real-time visualization of vehicle traffic reconstructed directly from temporal-spatial data readily available from existing in-road sensors on the road networks. It does not require additional identification of individual vehicles, such as [Sewall et al., 2011a, van den Berg et al., 2009], to visualize the current vehicle traffic condition based on sensor measurements.

3.3 Approach

Our approach estimates traffic conditions based on a stream of traffic data. The estimate is created by a smoothing process involving an Ensemble Kalman Smoother and a macroscopic traffic simulator. I use the estimate to drive an agent-based simulation to produce a visualization of the traffic conditions. An overview of the approach can be seen in Fig 3.2.

3.3.1 Preliminaries

Sensors: Our model assumes that the traffic conditions are monitored by sensors. I assume these sensors are discrete spatially and provide measurements at discrete time intervals. These measurements are assumed to be an estimate of the density and velocity, or flow, of the traffic over the time interval.

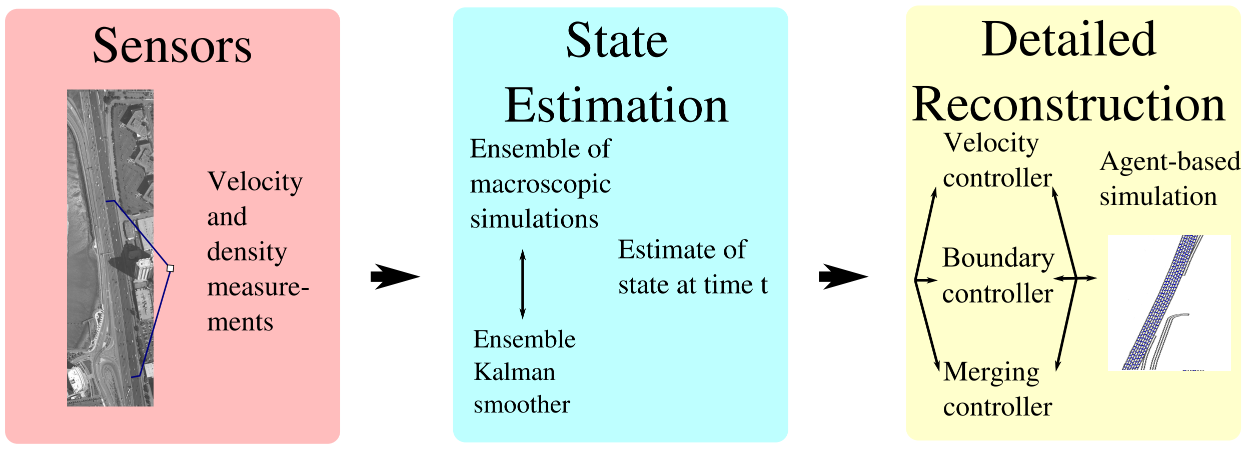


Figure 3.2: A schematic view of the approach.

These sensors can come in multiple forms as long as certain assumptions are satisfied. Specifically, I assume that I know the location of the sensor, and that the position of the sensor is fixed. I assume that the data the sensors provide can be transformed into a density and velocity estimate.

A prominent form of sensor of this type is the loop detector. This form of sensor provides volume and velocity information, from which the density of traffic can be calculated. Another form of sensor that could satisfy these requirements are cameras, from which the density and velocity of traffic at a point on the roadway could be estimated.

Road Networks: The domain of a traffic simulation is a road network. These networks can vary in complexity from being a single lane to being a vast network of roads with multiple lanes, intersections, highway ramps, overpasses, sources, sinks, and other features. Our method assumes a multi-lane highway as its domain.

I follow [Wilkie et al., 2012] in creating the road representation from available GIS data. The created geometry is C^1 smooth and ensures that car movements will satisfy the *simple car* kinematic constraints [van den Berg et al., 2009, Sewall et al., 2011a]. Each lane has relations defined to allow merging both between neighboring lanes as well as between the highway and ramps.

The geometry of each lane is discretized for the purposes of macroscopic simulation as well as state representation. The lanes have associated lengths, and cars moving along the lanes have positions in a local 1D coordinate frame, where the beginning of the lane is at position 0 and the end is at position *length*. The cell containing a particular position p will be referred to as *cell(p)*.

3.3.2 State Estimation

3.3.2.1 Overview

Our approach assumes traffic data are spatially and temporally sparse and therefore describe only a small fraction of the overall traffic conditions. Under such assumptions, it is necessary to estimate the full traffic state given the sensor measurements. I achieve this estimation via a state estimation process, which makes use of all available sensory information, as well as a macroscopic model of traffic dynamics. An example of the ground truth and resulting estimate can be seen in the Appendix.

The full traffic conditions, as I described previously in Section 3.1, involve the states of all the individual vehicles. To abstract these conditions to the form of a *state* of traffic, I must make an assumption about the dynamics of traffic, as different dynamics models will require different state formulations in order to fully specify how the traffic will evolve. I assume a macroscopic model described in [Sewall et al., 2010b], based on the equations of Aw and Rascle [Aw and Rascle, 2000] and Zhang [Zhang, 2002]. Following Lebacque [Lebacque et al., 2007], I refer to this as the Aw-Rascle-Zhang (ARZ) model. This traffic model describes the evolution of aggregate traffic statistics, density and velocity, along lanes and makes use of a parameter for determining the speedlimit, $v_{\max}$, and a parameter γ to define a relationship between the velocity and the density. Following this, the state of traffic will be a vector,

$$\mathbf{x} = (\rho_1, y_1, \rho_2, y_2, ..., \rho_N, y_N, \gamma, v_{\max}) \tag{3.1}$$

where ρ_i is the density and y_i is the relative velocity, defined below, of the i^{th} *cell* of the lane.

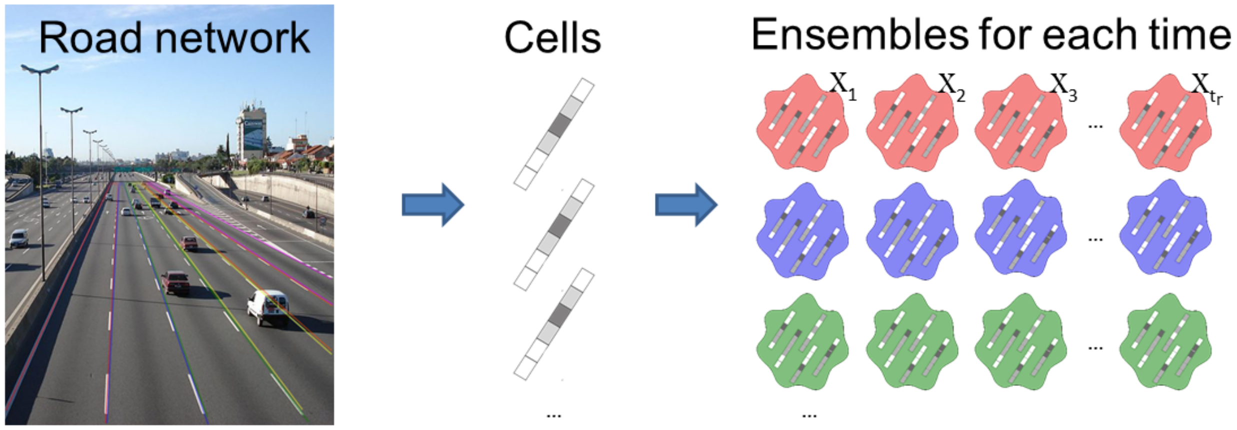


Figure 3.3: The initialization of the smoothing process.

3.3.2.2 Flow Reconstruction

To create a state estimate, I do a *smoothed reconstruction* of the traffic using sensor measurements and a macroscopic traffic simulator. This reconstruction uses the Ensemble Kalman Smoothing (EnKS) method [Evensen, 2003]. The estimation process maintains an *ensemble* of states for each timestep. In 3.3, we can see that each lane is discretized into cells, which hold continuum quantities. For each timestep, I create an ensemble representation of the traffic state for each lane. Then each ensemble member is updated sequentially when a new sensor measurement is received. As seen in 3.4, the update involves advancing the ensemble member via simulation and applying corrections based on the difference between the received measurements and the current state, using a calculated Kalman Gain matrix. Finally, I use the traffic the mean of the ensemble of states for each timestep as the estimate of the traffic state.

The EnKS has two principal components, a *motion model* and an *observation model*. The motion model,

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{m}_t), \quad \mathbf{m}_t \sim \mathcal{N}(0, I), \tag{3.2}$$

describes the evolution of the dynamic system. It propagates the state $\mathbf{x}_{t-1}$ to the state $\mathbf{x}_t$ given noise $\mathbf{m}_t$. This particular formulation assumes that the state depends only on

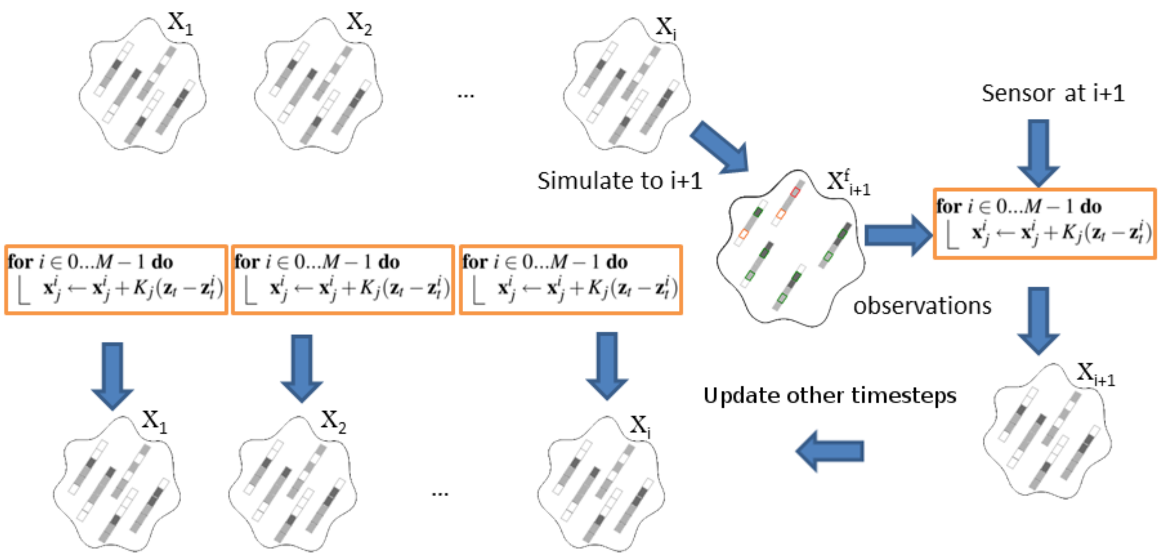


Figure 3.4: An ensemble member is simulated forward in time, then corrected given the sensor measurement. Finally, previous ensembles are updated to account for the new sensor reading.

the previous state and noise, not control inputs. The motion model used is based on the nonlinear hyperbolic system of partial differential equations proposed by Aw and Rascle [Aw and Rascle, 2000] and Zhang [Zhang, 2002] (ARZ); this system describes the motion of vehicles along a lane in terms of vehicle density ρ, velocity v, and the quantity y, the so-called *relative velocity*. The system is:

$$\mathbf{q}_t + \mathbf{f}\left(\mathbf{q}\right)_x = 0,\ \mathbf{q} = \begin{bmatrix}\rho \\ y\end{bmatrix},\ \mathbf{f}\left(\mathbf{q}\right) = \begin{bmatrix}\rho v \\ y v\end{bmatrix} \tag{3.3}$$

To obtain numerical solutions to Eq. (3.5), I use the Finite Volume Method with Riemann solver specific to the equations.

The observation model,

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), \quad \mathbf{n}_t \sim \mathcal{N}(0, I), \tag{3.4}$$

relates the simulated dynamic states, $\mathbf{x}_t$ to real world measurements, given measurement noise $\mathbf{n}_t$. Our observation model creates a measurement z^i_t of the simulated state x^i_t in order to compare the simulation with the sensor measurements: for each real-world sensor location p, I return the ρ, v pair for $cell(p)$. These measurements are perturbed with Gaussian noise.

These models are used to propagate an ensemble of states, $\mathcal{X}_t = \{\mathbf{x}_t^0, \ldots, \mathbf{x}_t^{M-1}\}$ that represents the distribution of the unknown true state of the system. At each time interval, both models are calculated, and the result is used to update the ensemble. The algorithm for this process is shown in 3.1, where the $\mathbf{m}_t^i$ is drawn randomly and independently from $\mathcal{N}(0, I)$, $\mathbf{n}_t^i$ is drawn randomly and independently from $\mathcal{N}(0, I)$, and $\mathbf{z}_t$ is the obtained measurement. Note that $\hat{\mathbf{z}}_t = E(\mathbf{z}_t)$, $\hat{\mathbf{x}}_t = E(\mathbf{x}_t)$, $\Sigma_t = Var(\mathbf{z}_t)$, and $\Gamma_t = Cov(\mathbf{x}_t, \mathbf{z}_t)$.

Algorithm 3.1 EnKS for Traffic State Estimation

```
Input: Traffic sensor measurements $z_1 ... z_{t_n}$, ARZ simulator in $f$, observation model in $h$,
        initialized ensemble $\mathbf{x}_0^0 ... \mathbf{x}_0^{m-1}$, noise vectors $\mathbf{m}_t^0 ... \mathbf{m}_t^{m-1}$ and $\mathbf{n}_t^0 ... \mathbf{n}_t^{m-1}, \forall t \in 1 ... t_n$
Output: Traffic state estimates $\hat{\mathbf{x}}_t, \forall t \in 1 ... t_n$
for $t \in 1 ... t_n$ do
    for $i \in 0 ... M-1$ do
        // Motion model
        $\mathbf{x}_t^i \leftarrow f(\mathbf{x}_{t-1}^i, \mathbf{m}_t^i)$ // Observation model
        $\mathbf{z}_t^i \leftarrow h(\mathbf{x}_t^i, \mathbf{n}_t^i)$
    // Analysis
    $\hat{\mathbf{z}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{z}_t^i$ $\Sigma_t \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$
    for $j \in 1 ... t$ do
        $\hat{\mathbf{x}}_j \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_j^i$ $\Gamma_j \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{x}_j^i - \hat{\mathbf{x}}_j)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$
        $K_j \leftarrow \Gamma_j \Sigma_t^{-1}$
        for $i \in 0 ... M-1$ do
            $\mathbf{x}_j^i \leftarrow \mathbf{x}_j^i + K_j(\mathbf{z}_t - \mathbf{z}_t^i)$,
// Final output
for $t \in 1 ... t_n$ do
    $\hat{\mathbf{x}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_t^i$
```

where the $\mathbf{m}_t^i$ is drawn randomly and independently from $\mathcal{N}(0, I)$, $\mathbf{n}_t^i$ is drawn randomly and independently from $\mathcal{N}(0, I)$, and $\mathbf{z}_t$ is the obtained measurement. Note that $\hat{\mathbf{z}}_t = E(\mathbf{z}_t)$, $\hat{\mathbf{x}}_t = E(\mathbf{x}_t)$, $\Sigma_t = Var(\mathbf{z}_t)$, and $\Gamma_t = Cov(\mathbf{x}_t, \mathbf{z}_t)$.

In the sequel, I discuss the details of this process, including ensemble initialization, the motion and observation model formulations, and issues concerning noise and tuning.

3.3.2.3 Ensemble Initialization

I initialize each ensemble member, $\mathbf{x}_0^i$, with random values for density, ρ, relative velocity, y, and the speedlimit, $v_{\max}$, and gamma, γ. $v_{\max}$ is drawn uniformly from $[10, 60]$ meters per second; γ is drawn uniformly from $[0.05, 0.9]$. The density for the first cell is drawn uniformly from $[0, 0.9]$. The velocity, v, for the first cell is drawn uniformly from $[0, v_{\max}]$, from which a relative velocity is calculated.

The density and velocity for subsequent cells is the result of a dynamic random walk with bounded $\dot{\rho}$, $\ddot{\rho}$, $\dot{v}$, and $\ddot{v}$. The created density and velocity fields are C^2 when they are within the imposed bounds on ρ and v. While I discuss the creation of the state in terms of the velocity, it is the subsequent relative velocity that is stored in the state.

3.3.2.4 Motion Model

As discussed in the previous section, the motion model propagates each ensemble member forward to time t given the state at time $t-1$. The state, as described above, is defined to fully specify the evolution of traffic using the ARZ simulation formulation. For each iteration, each ensemble state x_{t-1}^i is evolved forward using the motion model. Each lane is simulated independently, with the density and velocity values for the 0 and N cells being held constant to implement the boundary conditions. The duration of the simulation is defined by Δt_{filter}, i.e. the frequency with which traffic data are received. After each simulation run, Gaussian noise is added to each member of the state, as described below.

ARZ system of equations The macroscopic simulation component of our method is based on the nonlinear hyperbolic system of partial differential equations proposed by Aw and Rascle [Aw and Rascle, 2000] and Zhang [Zhang, 2002]; this system describes the motion of vehicles along a lane in terms of vehicle density ρ, velocity v, and the derived quantity y, the so-called *relative velocity.* The system is:

$$\mathbf{q}_t + \mathbf{f}\left(\mathbf{q}\right)_x = 0, \ \mathbf{q} = \begin{bmatrix} \rho \\ y \end{bmatrix}, \ \mathbf{f}\left(\mathbf{q}\right) = \begin{bmatrix} \rho v \\ y v \end{bmatrix} \tag{3.5}$$

In the parlance of hyperbolic PDEs, Eq. (3.5) is in *conservation form*; the column vector $\mathbf{q} = \left[\rho, y\right]^{\mathsf{T}}$ describes quantities whose totals in the whole system change only due to what flows in and out of boundaries. Conservation forms are convenient for discovering important properties of and finding solutions to hyperbolic systems, and this motivates the introduction of the relative velocity y.

Relative velocity It makes intuitive sense that the density of vehicles ρ must be conserved (its sum over the system reflects the total quantity of vehicles therein), but I should not expect that the velocity v should directly be conserved — were this so, whenever a vehicle slowed down, another would have to accelerate!

Thus, the concept of relative velocity y is introduced, which roughly describes how fast vehicles are traveling compared to the 'optimal velocity' for a given density. Specifically, y is related to ρ and v as follows:

$$y\left(\rho, v\right) = \rho\left(v - v_{\text{eq}}\left(\rho\right)\right) \tag{3.6}$$

The quantity $v_{\text{eq}}\left(\rho\right)$ is the *equilibrium velocity* for ρ — the largest velocity that may be achieved at a given ρ. This is closely related to an important concept in traffic flow, the *fundamental diagram*, which is a curve that plots the *flux* of traffic (i.e. the throughput of vehicles ρv) as a function of increasing density. Fundamental diagrams differ from flow to flow, but invariably theoretical models and empirical observations show that they are convex functions of ρ with a single, unique maximum; this fits intuition — where $\rho = 0$, there is clearly no flux, and as ρ approaches 1 (bumper-to-bumper traffic), velocity approaches 0 and flux falls back to zero.

In the ARZ model, $v_{\text{eq}}(\rho)$ is defined as follows:

$$v_{\text{eq}}(\rho) = v_{\max}(1 - \rho^{\gamma}) \tag{3.7}$$

where the $v_{\max}$ is the speedlimit of the lane and $\gamma \in (0, 1)$ is a parameter that characterizes the fundamental diagram.

Numerical macroscopic solutions To obtain numerical solutions to Eq. (3.5), I use the Finite Volume Method with Riemann solver specific to the equations.

A lanes is partitioned into N discrete cells of length Δx. The discrete quantity $\mathbf{Q}_i = [\rho_i, v_i]^{\mathsf{T}}$, $i \in \mathbb{Z}[0, N)$ represents the average value of $\mathbf{q}$ over cell i.

To advance the solution from time t_n to t_{n+1}, I use forward Euler integration:

$$\mathbf{Q}_i^{n+1} = \mathbf{Q}_i^n - \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-}^n + \mathbf{F}_{i+}^n\right) \tag{3.8}$$

where $\mathbf{Q}_i^n$ is the average value of $\mathbf{q}$ in cell i at time n, $\Delta t = t_{n+1} - t_n$ and the fluxes $\mathbf{F}_{i-}^n$ and $\mathbf{F}_{i+}^n$ are the average value of $\mathbf{f}(\mathbf{q})$ from t_n to t_{n+1} at the interface between i and the neighboring cells $i-1$ and $i+1$, respectively (i.e. $\mathbf{F}_{i-}^n = \mathbf{F}_{i-1^+}^n$ and $\mathbf{F}_{i+}^n = \mathbf{F}_{i+1^-}^n$).

In practice, for nontrivial $\mathbf{f}$ such as that found in Eq. (3.5), exact computation of $\mathbf{F}_{i-}^n$ and $\mathbf{F}_{i+}^n$ is not possible and I must settle for an approximation; the most robust such approximations are found by solving the so-called *Riemann problem* at each interface between adjacent cells to determine how the solution is evolving there. The computation of these Riemann problems is the most computationally expensive portion of the macroscopic solution procedure, but is critical to obtain correct, stable solutions in the presence of nonlinear phenomena such as shocks, rarefaction, and “vacuum” states.

A detailed description of the Riemann problem for the ARZ system of equations and the appropriate accompanying Riemann solver can be found in Appendix B of [Sewall, 2010].

3.3.2.5 Observation Model

The observation model creates a measurement z_t^i of the simulation state x_t^i in order to compare the simulation with the sensor measurements. I create a vector containing density and velocity measurements of the simulation. For each real-world sensor location p, I return the ρ, v pair for $cell(p)$. These measurements are perturbed with Gaussian noise, as described below.

3.3.2.6 Tuning and Noise

After each ensemble member has been propagated by the motion model, zero-mean Gaussian noise is added to each cell. For each cell, the noise is scaled to account for the differences between different parts of the states. Specifically, the γ and v_{max} elements of the state receive more noise than the density and relative velocity elements, and the boundary elements receive more noise than the interior elements. Additionally, as I have topological information about the simulation domain, I can enhance the model by transforming the noise at points according to the road network. For example, I allow a wider range of noise at on ramps and off ramps (respectively additive noise and subtractive noise) to allow the system to account for cars entering or leaving the lane. Values for these noise weights can be found in 3.4.4.

3.3.3 Detailed Reconstruction

3.3.3.1 Overview

In the state estimation phase, I generate estimates for the average state of traffic over a time interval, defined as the time between two subsequent sensor measurements. These state estimates are created with a macroscopic simulation, implying no individual cars exist to be visualized. Additionally, the state estimation is done independently for each lane of the road network.

In this section, I present our approach to creating an agent-based simulation that matches the given macroscopic state estimate. This stage simulates traffic on all the lanes of the roadway simultaneously and uses controlled merging between them. The end result of this process is an animation that provides real-time visualization of the traffic flow, where the output is a collection of states for individual cars. The simulation is controlled via boundary conditions, parameter fitting, and merging.

3.3.3.2 Vehicle Instantiation

I must provide an initial population of vehicles along each lane. To do this, I use a straight-forward approach of creating cars as evenly spaced as possible to satisfy the density specified in the state estimate. The result of this step may appear too uniform to be mistaken for real traffic, but our primary concern here is to show that the statistical properties of the original traffic flow are captured. As such, I wish to introduce as little additional noise as possible.

To achieve this for lane j, I take its state estimate at the first considered timestep, $\mathbf{x}_0$. Let ρ_0 be the vector of densities in $\mathbf{x}_0$, and $\mathbf{v}_0$ be the velocities derived from the relative velocities, $v_{\max}$, and γ of $\mathbf{x}_0$. Let p be a position in the local 1D coordinate frame of the lane, as described in Section 3.3.1. I initialize p to 0, the beginning of the lane. Then, iteratively, I calculate the *separation distance* that is implied by the *cell(p)*th member of ρ_0, i.e. the density estimate for that part of the lane. The separation distance is defined as $\frac{1}{\rho_p} * \text{car length} - \text{car length}$. The separation distance is added to p, and a car is added to the roadway at p. The cars initial velocity is set to the *cell(p)*th member of v_0.

3.3.3.3 Particle Advection

I use an agent-based simulation to visualize the individual cars that make up the traffic flow. Each car is advected using the velocity field estimated previously. The velocity field estimates are lane-specific and constant during a filter interval. The field is discretized into cells, each corresponding to a cell of the macroscopic simulation (as discussed in 3.3.1). To increase the heterogeneity of the simulation, each car can optionally be given a velocity scaling factor, drawn from a bounded normal distribution, to bias its preferred speed.

To allow for accurate integration over large timesteps, I calculate a velocity for each car that is the weighted combination of the cell velocities through which the car will pass during the timestep. Each velocity is weighted by the amount of time the car will spend within the corresponding cell.

As the velocity field is constant over a time interval, it is possible for cars to come into collision. To prevent this, I can restrict the velocity applied to a car in *cell(p)* as $v = \min(v_p, \Delta x - s_1 + v_l)$, where v is the velocity applied to the car, v_p is the velocity of the estimated velocity field, Δx is the distance between the two cars, s_1 is a minimum separation distance (including the car length), and v_l is the velocity of the leader car. Thus a car cannot take a velocity that would result in it being a distance less than $s1$ from the leading car.

3.3.3.4 Boundary Conditions

As the agent-based simulation advances, cars will leave the network and new cars need to be added to visualize the traffic flow. The inflow requirement is defined a flow rate for each lane as $r_l = \rho_l^s * v_l^s$, where ρ_l^s and v_l^s are the density and velocity of the first cell in the state estimate for the lane. I create cars uniformly to satisfy this rate as follows. From the density component, ρ_l^s, I can derive the separation distance s_l, as described in 3.3.3.2. During each timestep, the last car in a lane will be at position p. I create cars such that the i^{th} created car is at position $p - s_l * i$. The cars are created until there is no longer any space in the lane. The velocity component of the flow rate is satisfied implicitly: the time required for sufficient free space to be created is dependent on the velocity that the created cars are moving.

Creating cars in this manner can match the flow rate as well as allow large time steps if desired, however the traffic created can appear too uniform to be realistic. To prevent this, I can add bounded zero-mean Guassian noise, n_s, to the separation distances used for car creation. The noise is bounded such that $n_s > s_m + s_{mT} * v_l^s$, where s_m is a minimum distance including the car length and some separation distance, s_{mT} is a minimal stopping time that determines, with v_l^s, a dynamic minimal distance. This constraint is derived from the Intelligent Driver Model [Treiber et al., 2000], which is a microscopic traffic model based on a leader-follower equation.

3.3.4 Merging

Modeling traffic merging is an area of ongoing research that involves driver decision making as well as a complicated interplay of dynamics, kinematics, and multi-agent reactions. For visually realistic and detailed animations, a merge must be considered as having a time duration larger than a simulation timestep. This implies cars must be capable of being in the state of *merging*, and other cars need to react appropriately to merging cars. Further, for controlled traffic animation, I need to constrain and direct merging to bring the animation in line with the target state, while allowing users to specify a level of merging activity to achieve their desired animation goal.

Our approach to controlled merging takes into account the accumulated error in each lane's density field. Starting from the most downstream point of the highway, density error is aggregated for each lane. If the absolute error is greater than one car, the corresponding cell of the road is marked as a merge point. Cars at these merge points can switch between lanes to account for the density errors, as long as the agent-based kinematic and dynamic conditions are satisfied.

I use additional decision criteria to determine if a car should merge based on the local dynamics of the system. A car will not merge if there is not enough space for it, if it would cause too great a deceleration in its target lane, if, by merging, it would decelerate itself, or if there is already a car merging into the lane in the neighborhood upstream. These rules ensure that merging does not detract from the natural flow of traffic or create unrealistic looking situations.

I implement these criteria using rules defined in terms of leader, follower, target lane, and the dynamic and geometric characteristics of the cars.

Definition 1. target lane. For a car a, an adjacent lane is one reachable by a right or left merge. A target lane is an adjacent lane that is a is considering merging into.

Definition 2. leader. For a car a and a lane l that either contains a or is adjacent to a lane that contains a, I define the leading cars as $LC = \{c|c.p > a.p_L\}$, and the leader to be $b \in LC$, s.t. $b.p <= c.p$ forall $c \in LC$.

Definition 3. follower. For a car a and a lane l that either contains a or is adjacent to a lane that contains a, I define the following cars as $FC = \{c|c.p < a.p_L\}$, and the follower to be $b \in FC$, s.t. $b.p >= c.p$ forall $c \in LC$.

Definition 4. f_{LF}. A leader-follower function that returns an acceleration value given the positions and velocities of a leader car and a follower car.

The four criteria based on these terms are defined as follows.

Criterion 1. For a car a, let b and c be the leader and follower in the target lane respectively. A merge is forbidden if $((b.p - a.p) < d_{carspace}) or ((a.p - c.p) < d_{carspace}) or (b.p - c.p < d_{clear})$, where $d_{carspace}$ is the length of a car and a small buffer and d_{clear} is the *gap clearance* required for a safe merge.

Criterion 2. For a car a, let b be the leader in the target lane. A merge is forbidden in $f_{LF}(a.p, b.p, a.v, b.v) < 0$.

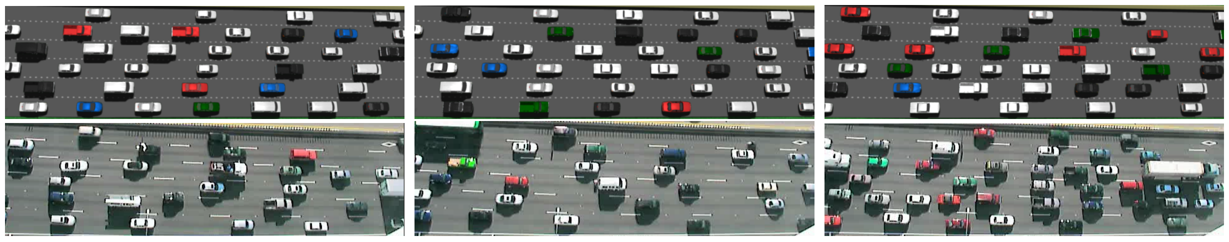


Figure 3.5: In this figure, there are three comparisons between the reconstruction and video footage from 5:17, 5:21, 5:25 (Minutes two, six, and ten for the 5:15 data set). In the first image, we see our method over estimates the density in at least two of the lanes in this area. In the next two images, congested traffic is both reconstructed and found in the source data. We can see lane-specific features of the traffic are reconstructed: in all cases, the HOV lane shows a much lower density (and has a higher velocity) than the other lanes. (It should be noted that the camera placement for the reconstruction images is only approximate.)

Criterion 3. For a car a, let b be the follower in the target lane. A merge is forbidden if $f_{LF}(a.p, b.p, a.v, b.v) < a_{disrupt}$, where $a_{disrupt}$ is an acceleration limit.

Criterion 4. For a car a, let b be the follower in the target lane. A merge is forbidden if b is in the state of *merging* to the target lane.

Once a car a decides to merge, the actual merge operation must take place over a simulation time interval. This involves moving the car a from its current lane to its target lane in a manner that respects both kinematic and dynamic constraints. In regard to the kinematic constraints, merges are often geometrically modeled as a trajectory resulting from a constant rate of change in the steering angle, i.e. turning the steering wheel at a constant rate into the turn and then at a constant rate out of the turn. The resulting path is a Clothoid (or Euler or Cornu) curve, for which no analytical expression exists. As a substitute, I use a polynomial approximation of the curve. In regard to the dynamic context, once a begins a merge, it can be said to belong to both its current lane and its target lane. However, a conservative approach to reconciling these two control influences leads to unrealistic traffic flow and congestion. I therefore allow a merging car to safely ignore its leader once it has passed the midway point of its merging curve.

3.4 Results

I have implemented our techniques presented here and tested them on traffic data from publicly available online sources.

3.4.1 Data

To demonstrate our method, I reconstruct virtualized traffic on a segment of highway. The sensor data used for these experiments is from the Next Generation Simulation (NGSIM) program [NGSIM, 2013]. The roadway on which the data are measured is a stretch of I-80 in Northern California, with an on-ramp and an off-ramp. The highway has six lanes, one of which is a 'car-pool lane'. The data are broken up into three segments, each covering a fifteen minute period.

The data consists of detailed trajectory data for every car on the highway during the periods of observation. The trajectories are ultimately from multiple cameras installed along the highway. The advantage of this data set is that it provides a detailed 'ground truth' for the traffic, which is normally unknown.

From these trajectories, I extract data to use as the *sensor measurements* in our system, to represent loop detector measurements. These measurements are a subset of the information in the full trajectory data set. To create the measurements, I calculate density and velocity fields over each lane at each time step. The sensors measure these fields in the neighborhood around their fixed positions. For our experiments, I use only two sensors, each only viewing one *cell*-length (See Section 3.3.1.) of the flow field. The approximate locations of the sensors can be seen in Fig. 3.2. Ultimately the sensors each provide an average density and velocity reading for a time span, which corresponds to the kind of measurements actual loop detectors provide [Jia et al., 2001].

3.4.2 Metric

I present the results of our experiments using *lane-mean* values, defined for density as

$$\rho_l = \frac{\sum_{j=0}^{N-1} \rho_j}{N}. \tag{3.9}$$

This measure can be defined for the ground truth data (the hidden, full car trajectory data), the macroscopic state estimation, and the agent-based simulation. The velocity version of this calculation follows directly. This measure was used so as to demonstrate the global tracking ability of the system.

3.4.3 Experiments

The results presented in Fig. 3.5 are for the time interval of 5:15 pm to 5:30 pm, when the highway traffic experiences congestion and traffic jams. In Figure 3.6, I show the lane-mean densities over time for a lane of reconstructed highway traffic. (Please see the Appendix in the supplementary document for the lane-mean density and velocity on all six lanes). This scenario is fairly challenging. The traffic is not only in free flow, but rather breaks down into traffic jams.

3.4.3.1 Error in Mean Flow

In these plots (shown in Fig. 3.6), I can see the state estimates of density and velocity for traffic along a highway lane (red) and the agent-based detailed reconstruction (blue) closely tracking the ground truth (green). The maximum error is on the order of 0.1 cars per 4.5 m (the car length) for the density tracking and 3 m/s for velocity tracking.

The root mean square (RMS) of the error for all lanes of the data set can be found in the Appendix. The density error ranges from 0.03 to 0.1 cars per car length (or proportion filled with cars), and the velocity error ranges from 1 to 2.5 meters per second.

3.4.4 Implementation Details

For our clovel leaf demo, I used an ensemble size of 100. For our highway reconstruction, I increased the ensemble size to 250 to create a high quality reconstruction. The cell length, Δx, in each case was 9.4 meters. The timestep used for the ARZ simulator during the reconstruction was $\Delta t_1 = 0.3$. The timestep used for the detailed reconstruction was $\Delta t_2 = 0.05$ seconds. Sensor measurements occur at $\Delta t_3 = 15$ second intervals.

The motion model's $\mathbf{m}$ and observation model's $\mathbf{n}$ noise vectors were weighted with hand-tuned parameters. The parameter values were as follows. For the observation model, 0.15 was used for the density measurement, and 4.5 was used for the velocity measurement. For the motion model, 0.005 was used for the state density and relative velocity; 6 was used for the speed limit, 0.1 was used for gamma, 0.2 was used for ramp density and relative velocity, and 0.3 was used for density and relative velocity at the lane boundaries.

3.4.5 Performance Analysis

Assuming the number of ensemble members dominates the dimension of the observation vector, the complexity of estimating the traffic state for one lane for one time interval is $O(t_i * M * |\mathbf{x}| * |\mathbf{z}| + t_i * f(\mathbf{x}, \mathbf{m}))$, where t_i is the number of sensor measurments, M is the number of ensemble members, $|\mathbf{x}|$ is the state size, $|\mathbf{z}|$ is the size of the observation vector, $f(\mathbf{x}, \mathbf{m})$ is the cost of the motion model. The complexity of a detailed reconstruction step for one lane is $O(|\mathbf{x}| * c)$, where c is the number of cars.

Our functional implementation serves more as a proof-of-concept than a benchmark; nonetheless, the efficiency of this technique is promising. We have tested a prototype implementation using the EnKF (single threaded, unvectorized code) on an Intel© Core i7 980X processor. We visualize the detailed reconstructed traffic on a six-lane portion of I-80 in Emeryville, California.

The run time is dominated by the state estimation, which takes under 3 seconds per step for a lane of 1km with 2 sensors. This computation is almost completely dominated by a large ensemble of macroscopic traffic simulation runs, wherein each of the motion ensemble is simulated forward in time. In this test, there were 80 motion ensembles per lane in the base reconstruction that needed to be simulated independently at each state estimation step.

The performance of this technique can be substantially accelerated through an architecturally-aware (such as GPU-based or many-core) implementation. In particular, there are multiple opportunities to exploit thread-level and data-level parallelism in the ensemble of many macroscopic simulations for state estimation. Using the increasing computational power will allow our method to effectively handle even larger road networks more efficiently.

3.4.6 Comparison with Related Work

This is a similar problem to [van den Berg et al., 2009, Sewall et al., 2011a], however our approach is radically different and has multiple advantages. Unlike VT, which required individual car data and identification, I make realistic assumptions about sensor data. Furthermore, our method is real-time and scalable, whereas VT uses an exponentially complex per-car motion planning algorithm in a discretized acceleration space. Our method creates data-driven traffic animation from reconstructed flows and traffic models, allowing user-interaction and flexibility. Our work features a novel estimation approach using a state of the art simulation formulation, novel extensions to microscopic simulation, and efficient lane changing formulations, in addition to the generality of the overall approach. Our approach can lead to future work in user-controlled animation, traffic analysis and diagnosis, and numerous other extensions. Further details on specific differences follow.

- **Sensors:** Our work assumes a form of traffic sensor measurement that exists currently and is publicly available. I assume measurements of volume and velocity, which are available for in-road loop detectors [Rice and Van Zwet, 2004] and can be derived from camera data. The earlier works [van den Berg et al., 2009, Sewall et al., 2011a] assumes readings for individual cars at road boundaries, which is not readily available. Furthermore, our work allows for sensors to be located arbitrarily along a lane.

- **Macroscopic:** The related work reconstruct trajectories for individual cars. Our approach focuses on a macroscopic reconstruction first, and then a visualization consisting of individual cars. This has two benefits. First, it protects the privacy of drivers by making it impossible to trace individual cars. Second, it drastically decreases the amount of information needed to perform a reconstruction. In our work, the size of the sensor data is constant, while in the previous work the size scales with the number of cars.

- **Simulation:** Our work creates a simulation-based reconstruction with which a user could interact. A user could reconstruct and visualize traffic, then drive a virtual car in that traffic. The reconstructed cars would react to the user, for example allowing the user to create a traffic jam when one did not occur in the observed data. This capability is not possible with the earlier approach.

In terms of quantitative accuracy, it is difficult to make a meaningful comparison between the two approaches due to their differing formulations. However, in [Sewall et al., 2011a], the authors determine that their method was able to find feasible trajectories for 82% of the vehicles. This statistics may be viewed roughly as a measure of density accuracy. That is, their method was able to route 82% of the observed vehicles, and therefore its density accuracy is bounded by at most 82%. However, this metric is overly simplistic for a few reasons. First, it ignores the variation between lanes. Some lanes may be congested while others are free flowing. Second, it ignores inner lane density variations, i.e. a traffic jam on the first part of the highway and free flowing traffic farther downstream. However, ignoring these complications, I can state that our method can reconstruct the motion of all vehicles and it should have at least comparable density accuracy, if not higher at several places

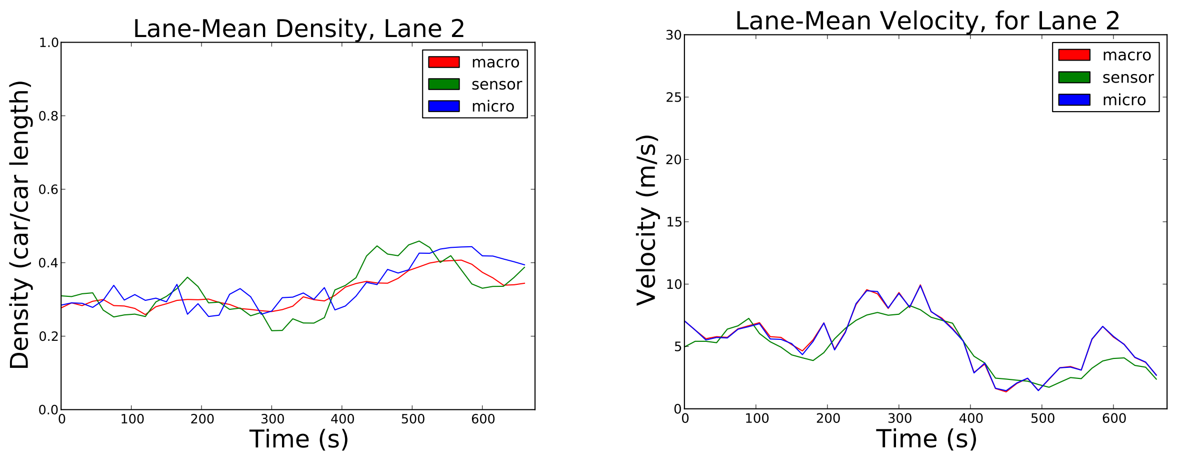


Figure 3.6: The lane-mean density and velocity for a lane of highway traffic. The **green** line is the ground truth, the **red** line is the state estimate, and the **blue** line is the agent-based simulation.

(see Appendix for details). It is not possible to compare the accuracy in the velocity field reconstructions with earlier methods, as such information was not available.

Our work compares favorably with the prior work in terms of computational cost. where [Sewall et al., 2011a] reconstructed 15 minutes of real-world traffic in 6.64 minutes of computation. Using the same dataset, our method can reconstruct 15 minutes of traffic in a matter of seconds, as discussed above in Sec. 3.4.5. This performance improvement can be easily explained by the fact that our method does not require an expensive, priority-based, multi-agent route-planning algorithm.

3.5 Summary

I have presented a real-time traffic reconstruction and visualization algorithm using traffic sensor data obtained from in-road loop detectors. My approach seamlessly integrates an efficient state estimation method using Ensemble Kalman Filter and continuum traffic simulation with a fast traffic reconstruction using an agent-based traffic simulation system that produces realistic motion for individual vehicles whose global states (e.g. density and velocity fields) are consistent with the estimated traffic flow in the real world and individual vehicle's kinematic and dynamic constraints. The computational framework presented here has been implemented, tested, and validated on real-world traffic data.

3.5.1 Limitations

My formulation builds on related work in traffic state estimation, but these approaches currently do not consider complete networks of roads, including arterial roads and intersections. My current implementation also does not consider additional sources and sinks of cars other than at the boundaries. Furthermore, while my method demonstrates good tracking for the lane-mean, the method cannot capture all the fluctuations of traffic within the lane and within the filter interval, especially if the sensor period is longer than assumed here. While loop detectors are common on highways, they are less frequently used on arterial and residential streets.

3.5.2 Future Work

I would like to extend this work to handle large, complete networks of highways and the road networks surrounding them. Another interesting extension of this work would be to add visualization of traffic predictions along with the current state estimate. The work could also use both historical data, as well as driver specific data and cell-phone sensing to improve the filtering performance. Finally, my system could be combined with routing algorithms to provide both efficient plans and visualizations of those routes on virtual globe systems.

CHAPTER 4: SELF-AWARE ROUTE PLANNING

4.1 Introduction

Once we can model the traffic system's domain, simulate its dynamics, and observe and estimate the state of the traffic system, we can attempt to optimize the traffic system with a model-based approach. My approach to this is novel: I propose using a *participatory* vehicle route planner that predicts the impact of the routes it plans on the future traffic system states. The system then uses its projected future states to route subsequent vehicles, better distributing the vehicle load throughout the road network.

One of the most ubiquitous applications of AI is vehicle route planning. State-of-the-art route planners consider possible delays due to traffic congestion based on current traffic conditions and/or historical traffic data. Live traffic data can be collected by loop-detectors, cameras, toll port data, and cell phone localization. These systems provide the traffic velocity at certain locations at a fixed frequency [Brakatsoulas et al., 2005], which can then be used for vehicles to plan around congested areas.

Live traffic data alone does not enable optimal routing as the future traffic conditions can change, and they can even change in response to the routes planned. If a route is planned to let a car arrive at a certain road in half an hour, the current conditions may no longer be an accurate estimate for that road 30 minutes later. This problem can be partially addressed by predicting the future traffic state from historical data [Horvitz et al., 2005], [Nikolova et al., 2006a], [Min et al., 2007]. But this does not solve the problem: a route planner will affect future traffic conditions if it controls a large portion of the vehicles, thus making estimations based only on current or historical data incorrect.

Thus, the route planner must also take into account its own previous actions. For example, if a route planner controlled *every* car in the system, historically congested areas would be unduly avoided, causing congestion to appear at the routes that the planning system has provided. This is clearly the worst-case scenario, but the underlying problem is that the historical prediction assumes that cars tend to act the same way as they have historically, which may no longer be the case if a route planner is controlling the trajectories of all vehicles. I propose a novel *participatory* traffic route planner that uses the planned routes of vehicles in the system far to more accurately predict future traffic conditions for vehicles whose routes are subsequently planned (see Fig. 4.1). As a result, my approach overcomes the oscillation issue in case of large-scale adoption of traffic route planners.

My approach accounts for the fact that a route planned for a car will cause a marginal increase in traffic density on the roads it plans to traverse. I use the predicted paths of the route planner itself in addition to historical data to estimate future traffic conditions. Assuming that a large percentage of the cars use the route planning system, the collection of all their planned routes can be used to accurately estimate the future traffic conditions. Every car that queries the route planner can then use this information to plan a route for itself. Its planned route is then used to update the estimate of future traffic conditions for vehicles come in later in the road network. My experimental results suggest that my route planner can more accurately predict future traffic conditions, resulting in a reduction of the travel time for those vehicles that use my algorithm.

The rest of this Chapter is organized as follows. In Section 2, I discuss background work related to our approach. In Section 3, I detail the method used to update the historical probabilistic prediction and the overall planning system. In Section 4, I discuss the implementation and validation of our method. In Section 5, I provide some concluding remarks.

4.2 Related Work

Our work is perhaps most similar to that of [Nikolova et al., 2006b] and [Lim et al., 2009]. In fact, our work directly extends these methods to perform 'self-aware' routing. In [Nikolova et al., 2006b], the authors propose a method to optimally route cars given uncertain information about travel times within a network. [Lim et al., 2009] provides an optimization to the procedure that allows fewer paths to be explored, while optimizing for a specific arrival deadline, and additional extensions were carried out in [Nikolova, 2010] and [Hua and Pei, 2010].

Another area of similar work is the study of Dynamic Traffic Assignment (DTA) done primarily in Civil Engineering. This problem involves flows of traffic from known origins to destinations (OD flows). The solution approaches attempt to optimally route all the flows in order to maximize aggregate or individual statistics. A summary of approaches can be found in [Peeta and Ziliaskopoulos, 2001]. The most relevant of these approaches are the simulation methods, such as [Florian et al., 2008]. In these approaches, cars are iteratively routed and simulated. The simulation provides the estimate of the network state that is used for the next iteration of routing. Over a number of iterations, the routes settle into an equilibrium.

Our work is inspired by [Lim et al., 2009], which presents an planning algorithm using graphs with stochastic (time-invariant) edge costs. Their planner assumes a cost function that invalidates the "optimal substructure" property of paths, which prevents using a straightforward A* algorithm, and present an efficient approach to compute optimal paths for such cost functions. In contrast, our algorithm uses a simpler cost function that still makes it possible to use an A* search algorithm, but assumes *time-varying* stochastic edge costs. I use insights from [Chabini and Lan, 2010] regarding the first-in-first-out property of traffic that allows us to use A* even if the edge-costs are time-dependent (in general, time-dependent edge costs prohibit the use of A*). Typical traffic related planning approaches assume the edge cost to be given as travel times (potentially time-varying and stochastic) [Lim et al., 2009, Chabini and Lan, 2010].

Our approach assumes the input data to be traffic *densities* of the road segments in the network, and I use the *fundamental diagram* of traffic to translate these densities to velocities and travel times. Maintaining densities allows us to update the data with the routes that our system generate to create a self-aware routing system.

The observation that the flow (and velocity) of traffic was dependent on the traffic density was made in early traffic studies [Greenshields et al., 1935]. Since then, the concept has been used as a basis for continuum traffic simulation formulations [Siebel and Mauser, 2005] as well as in schemes to estimate the state of traffic given sparse data, such as cell phone localization signals[Work et al., 2010].

4.3 Approach

I assume the road network to be given as a directed graph $\mathcal{G} = (V, E)$ consisting of a set of vertices V that model road intersections and edges $E \subset V \times V$ that model road segments between intersections. Associated with each edge e is the capacity C_e, maximum speed $v_e^{\max}$, and length ℓ_e of the corresponding road segment. In addition, a stochastic function $\rho_e(t) \sim \mathcal{N}(\bar{\rho}_e(t), \tilde{\rho}_e(t))$ is maintained for all road segments e that gives a normal distribution with mean $\bar{\rho}_e(t)$ and variance $\tilde{\rho}_e(t)$ of the traffic density of e at time t. I assume this distribution is independent from the density distributions at other road segments or at other times (similar assumptions were made in [Lim et al., 2009]). Further, I assume that the time-axis is cyclical (e.g. with a daily or weekly period) and discretized into small steps, such that only a finite amount of data is stored with each edge e. The stochasticity of the density function models the uncertainty about future traffic conditions as well as the variability of conditions from day to day.

Our approach can be summarized as follows. I assume that over time, different queries for optimal routes come in from cars that use our self-aware planning system. If a query

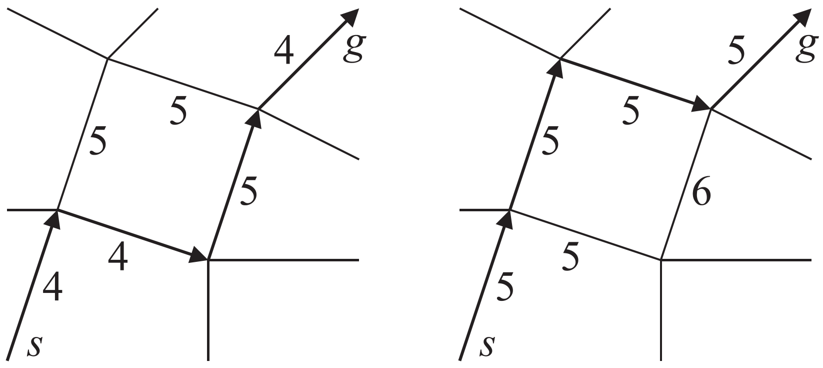


Figure 4.1: A schematic picture illustrating the idea of our approach. A road network is shown with edge costs. (a) If a route from s to g is requested, the optimal path is computed (shown with thick arrows). If the car follows this route, the densities and hence the edge costs along its path increases (in this case with 1). In (b), the network with the updated edge costs are shown. If a same query (s, g) comes in from a subsequent car, it takes a different route (shown with thick arrows) to avoid the increased traffic densities. Note that this schematic picture does not illustrate the stochastic and time-varying aspects of our approach.

comes in from a car i at a given time t_0, I plan a route for car i between its start node s and goal node g that optimizes a cost function based on its expected travel time given the current density functions $\rho_e(t)$ for each edge e. Subsequently, assuming this car will actually follow the route it was given, I update the density functions $\rho_e(t)$ along its route such that its presence is accounted for when a route is planned for a subsequent car $i + 1$. This cycle continues indefinitely with each query coming in for an optimal route computation. As such, the planner is aware of the routes it has suggested earlier, in order to optimally estimate future traffic conditions.

I will first describe how an optimal route is planned for a car given the stochastic density functions $\rho_e(t)$. Next, I will discuss how this plan is used to update the stochastic density functions such that the presence of the car is accounted for in future plans for other cars. Finally, I will discuss how the problem of "double-counting" cars can be avoided, which would occur when a car being routed also appears in the historical data.

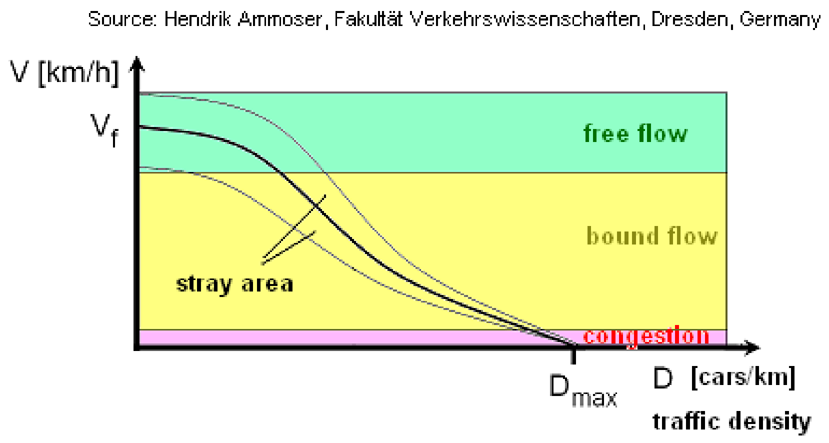


Figure 4.2: The fundamental diagram relating traffic density to travel speed.

4.3.1 Route Planning

4.3.1.1 Density and Travel Time

Given a query (s, g, t_0) for a car between a start node $s \in V$ and a goal node $g \in V$ leaving s at time t_0, I want to compute a route that minimizes the travel time to g given the stochastic density functions $\rho_e(t)$. To relate density to travel time, I use the *fundamental diagram*, which is a well-known empirical concept in traffic simulation research [Greenshields et al., 1935, Siebel and Mauser, 2005, Work et al., 2010] that gives a mapping from the traffic density ρ to the average speed v on a road segment e, given the maximum speed $v_e^{\max}$ and capacity C_e of the road segment e. Let the function described by the fundamental diagram be given by $v = f_e(\rho)$ (see Fig. 4.2).

Now, if a car arrives at the beginning of a road segment e at time t, I assume the speed with which it can traverse the road segment is given by $f_e(\rho_e(t))$. The travel time $\tau_e(t)$ to traverse e starting at time t is then given by:

$$\tau_e(t) = \frac{\ell_e}{f_e(\rho_e(t))}. \tag{4.1}$$

Since, the density function $\rho_e(t)$ is stochastic, the travel time is stochastic too. I approximate it with a normal distribution as $\mathcal{N}(\bar{\tau}_e(t), \tilde{\tau}_e(t))$, with mean $\bar{\tau}_e(t)$ and variance $\tilde{\tau}_e(t)$ given by the first-order Taylor expansion of $\tau_e(t)$:

$$\bar{\tau}_e(t) = \frac{\ell_e}{f_e(\bar{\rho}_e(t))} \tag{4.2}$$

$$\tilde{\tau}_e(t) = \big(\frac{d\tau_e(t)}{d\rho}[\bar{\rho}_e(t)]\big)^2 \tilde{\rho}_e(t). \tag{4.3}$$

For a path $\pi = \{e_1, \ldots, e_n\}$ consisting of a series of road segments when travel is commenced at time t_0, the total travel time $\tau_\pi(t_0)$ is given recursively by:

$$\tau_{\{e_1\}}(t_0) = \tau_{e_1}(t_0) \tag{4.4}$$

$$\tau_{\{e_1,\ldots,e_k\}}(t_0) = \tau_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \tau_{e_k}(t_0 + \tau_{\{e_1,\ldots,e_{k-1}\}}(t_0)) \tag{4.5}$$

Its mean $\bar{\tau}_\pi(t_0)$ and variance $\tilde{\tau}_\pi(t_0)$ are hence given by:

$$\bar{\tau}_{\{e_1\}}(t_0) = \bar{\tau}_{e_1}(t_0) \tag{4.6}$$

$$\tilde{\tau}_{\{e_1\}}(t_0) = \tilde{\tau}_{e_1}(t_0) \tag{4.7}$$

$$\bar{\tau}_{\{e_1,\ldots,e_k\}}(t_0) = \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \bar{\tau}_{e_k}(t_0 + \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0)) \tag{4.8}$$

$$\tilde{\tau}_{\{e_1,\ldots,e_k\}}(t_0) = \tilde{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \tilde{\tau}_{e_k}(t_0 + \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0)). \tag{4.9}$$

4.3.1.2 Cost Function

Our objective is to find a route π that minimizes the expectation $\mathbb{E}[c(\tau_\pi(t_0))]$, given a cost function $c(\tau)$ on the travel time τ. I consider two cases here:

- *Linear cost:* The cost increases linearly with the travel time: $c(\tau) = \tau$. Let $pdf_A(t)$ denote the probability density function of normal distribution A. Then, the expected cost is given by

$$\mathbb{E}[c(\tau_\pi(t_0))] = \int_{-\infty}^{\infty} pdf_{\tau_\pi(t_0)}(t) \cdot t \, dt = \bar{\tau}_\pi(t_0),$$

  which is the mean of the travel time of route π when travel is commenced at time t_0.

- *Exponential cost:* The cost increases exponentially with the travel time to more heavily penalize late arrivals: $c(\tau) = \exp(2w\tau)$ for some weight parameter w. The expected cost in this case is given by

$$\mathbb{E}[c(\tau_\pi(t_0))] = \int_{-\infty}^{\infty} pdf_{\tau_\pi(t_0)}(t) \exp(2wt) \, dt$$

$$= \exp(\bar{\tau}_\pi(t_0) + w\tilde{\tau}_\pi(t_0)).$$

  The result of minimizing for $\mathbb{E}[c(\tau)]$ in equivalent to minimizing for $\log \mathbb{E}[c(\tau)]$. Following this, the cost then becomes $\bar{\tau}_\pi(t_0) + w\tilde{\tau}_\pi(t_0)$, and is hence a linear combination of the mean and the variance of the travel time [Lim et al., 2009].

Our approach works for either of these cost functions. In our implementation I used the exponential cost function, for it attempts to minimize both the mean and the variance of the travel time.

4.3.1.3 Planning Algorithm

To find a path in the graph $\mathcal{G}$ between start node s and goal node g, I are confronted with a shortest path problem in a graph with time-varying and stochastic edge costs. In general, such problems are hard [Lim et al., 2009, Chabini and Lan, 2010], but in our case I can exploit properties of the cost function that allow us to a standard A* algorithm, which I will slightly adapt to handle the stochastic travel times.

Firstly, both of the cost functions as defined above are *additive* given the way the mean and variance of the travel time are computed (see Equations (4.8) and (4.9)). That is,

$$\mathbb{E}[c(\tau_{\{e_1,\ldots,e_k\}}(t_0))] = \mathbb{E}[c(\tau_{\{e_1,\ldots,e_{k-1}\}}(t_0))] + x, \tag{4.10}$$

where x is a linear combination of the second terms of Equations (4.8) and (4.9). Second, traffic observes the so-called *first-in-first-out* property [Chabini and Lan, 2010]. This means that arriving earlier at a node u in the graph will never produce a costlier route than a route that arrives later at u. Note that this is not the case for graphs with general time-dependent edge costs.

These two properties allow us to use the standard A* algorithm, which is adapted to handle the stochastic travel times along a route. The algorithm is given in Fig. 5.4. Instead of maintaining a single cost value of each node u in the graph as in standard A*, I maintain both the mean $\bar{\tau}_u$ and variance $\tilde{\tau}_u$ of the travel time of the current-best route from s to u. Initially, these are infinity for all nodes u, except the start node s, for which they are zero. The heuristic value $\bar{h}(u)$ provides a lower-bound estimate of the mean travel time to the goal g from a given node u, for which I use the Euclidean distance between u and g divided by the largest maximum speed in the road network. The heursitic value $\tilde{h}(u)$ provides a lower-bound estimate of the variance of the travel time between u and g, for which I use $\tilde{h}(u) = 0$. The functions $\bar{\tau}_e(t)$ and $\tilde{\tau}_e(t)$ which I refer to in lines 10 and 11 are given by Equations (4.2) and (4.3).

4.3.2 Maintaining Density Functions

Once a route has been planned for a car, I wish to take its presence into account when subsequent routes are planned for other cars. Based on the route that has been suggested, I can assume the car will follow it and add to the traffic densities at the road segments along its route at the times it is expected to traverse these road segments. At the same time, not

```
TrafficA*$(s, g, t_0)$
1: $\forall u \in V : \bar{\tau}_u \leftarrow \infty, \tilde{\tau}_u \leftarrow \infty$; $\bar{\tau}_s \leftarrow 0$; $\tilde{\tau}_s \leftarrow 0$
2: $OPEN \leftarrow \{s\}$
3: while $OPEN \neq \emptyset$ do
4:    $u \leftarrow \arg\max_{u \in OPEN} \{\bar{\tau}_u + \bar{h}(u) + w(\tilde{\tau}_u + \tilde{h}(u))\}$
5:    $OPEN \leftarrow OPEN \setminus \{u\}$
6:    if $u = g$ then
7:       return
8:    end if
9:    for each edge $e = (u, v)$ in $\mathcal{G}$ do
10:      $\mu \leftarrow \bar{\tau}_e(t_0 + \bar{\tau}_u)$
11:      $\sigma \leftarrow \tilde{\tau}_e(t_0 + \bar{\tau}_u)$
12:      if $\bar{\tau}_u + \mu + w(\tilde{\tau}_u + \sigma) < \bar{\tau}_v + w\tilde{\tau}_v$ then
13:         $\bar{\tau}_v = \bar{\tau}_u + \mu$
14:         $\tilde{\tau}_v = \tilde{\tau}_u + \sigma$
15:         pred$(v) \leftarrow u$
16:         $OPEN \leftarrow OPEN \cup \{v\}$
17:      end if
18:   end for
19: end while
```

Figure 4.3: The modified A* algorithm to compute an optimal route with respect to the exponential cost metric between start node s and goal node g when traffic is commenced at time t_0. When planning has finished, the optimal route is inferred by following the backpointers from the goal g.

all cars on the road use our self-aware planning system, and the system is not aware of the future plans of the cars that do use our system but have not entered the road network (yet). So, previously planned routes alone do not provide an accurate estimate of future traffic data.

4.3.2.1 Blending Historical and System Data

In order to predict future traffic conditions, I let the density functions $\rho_e(t)$ used in the above algorithm be a combination of historical traffic density data $\rho_e^{\text{hist}}(t)$, and density data $\rho_e^{\text{syst}}(t)$ generated by route plans provided by our self-aware system. However, care needs to be taken that the historical data is partly *phased out* when actual data of planned routes is included in the densities, as to avoid cars being double counted. I proceed as follows. Let

$\alpha \in [0,1]$ be the proportion of cars that use our system to compute their routes. Further, let there be a function $\beta(\Delta t) \in [0,1]$ that provides the proportion of cars that will be on the road at Δt time into the future which are already on the road currently. I assume $\beta(\Delta t)$ is time-independent, and can be inferred from historical traffic data on average travel times.

The traffic density $\rho_e(t)$ as used in our algorithm for a car starting travel at time t_0 is then computed as follows:

$$\rho_e(t) = (1 - \alpha\beta(t - t_0))\rho_e^{\text{hist}}(t) + \rho_e^{\text{syst}}(t). \tag{4.11}$$

This can be explained by the fact that a fraction $\alpha\beta(t - t_0)$ of all cars that will be on the road at time t have already been accounted for at time t_0 in the densities generated by our system.

4.3.2.2 Updating Traffic Densities

When a route has been planned for a car by our algorithm, I wish to take its presence into account when subsequent routes are planned for other cars. To this end, I update the density data $\rho_e^{\text{syst}}(t)$ that only counts cars for which routes have been planned using our self-aware system. I update these traffic densities as follows.

The algorithm above will give us a route $\pi = (e_1, \ldots, e_n)$ and distributions $\tau_u \sim \mathcal{N}(\bar{\tau}_u, \tilde{\tau}_u)$ of the travel times from the start node s to each node u along path π. Let edge $e = (u, v)$ be part of π. The car for which a path is planned will be on $e = (u, v)$ at time t with probability:

$$q_{(u,v)}(t) = \int_{-\infty}^{t-t_0} pdf_{\tau_u}(t')\, dt' \cdot \int_{t-t_0}^{\infty} pdf_{\tau_v}(t')\, dt', \tag{4.12}$$

where t_0 is the time at which the car commences its route π.

The above equation computes the probability that the car both arrives at e before time t and leaves e after time t. The density on e at time t is defined by the number of cars on e at time t divided by the length ℓ_e of e. Hence, the distribution of the density $\rho_e^{\text{syst}}(t)$ at time

t is updated as follows:

$$\bar{\rho}_e^{\text{syst}}(t) \leftarrow \bar{\rho}_e^{\text{syst}}(t) + q_e(t)/\ell_e \tag{4.13}$$

$$\tilde{\rho}_e^{\text{syst}}(t) \leftarrow \tilde{\rho}_e^{\text{syst}}(t) + q_e(t)(1 - q_e(t))/\ell_e^2, \tag{4.14}$$

which follows from treating the distribution $\rho_e(t)$ for each edge e and for each time t as an independent Poisson binomial distribution consisting of a number of cars each with a different probability of contributing to the density. When the number of cars gets large, the Poisson binomial distribution is well approximated by a normal distribution $\mathcal{N}(\bar{\rho}_e(t), \tilde{\rho}_e(t))$. This justifies the assumption in the planning algorithm of Fig. 5.4 that $\rho_e(t)$ is a normal distribution.

As the time axis is discrete, I only need to update a finite set of density distributions along the route planned for the car. I use the same discretization to compute the integrals in Equation (4.12). I use the updated mean and variances for the densities to route subsequent cars. This cycle of routing cars and updating densities continues indefinitely.

4.4 Empirical Results

In this section, I present our empirical study of the performance of our approach. Our hypothesis is that our system plans routes that have, on average, lower travel times than routes planned using the shortest path metric or stochastic-historical prediction method, increasingly so when the proportion of users of our system increases. This reinforces the essential claim of our paper: by taking into account the routes planned by the system itself, a planning system can find routes with substantially shorter travel time.

I have validated our approach by calculating plans for a fixed population of cars and queries using varying route planning methods. The validation was done in four parts. First, I compare our method with using a single path in a network. Next, i compare the performance of our self-aware algorithm to using shortest path A* searching. Next, I compare our

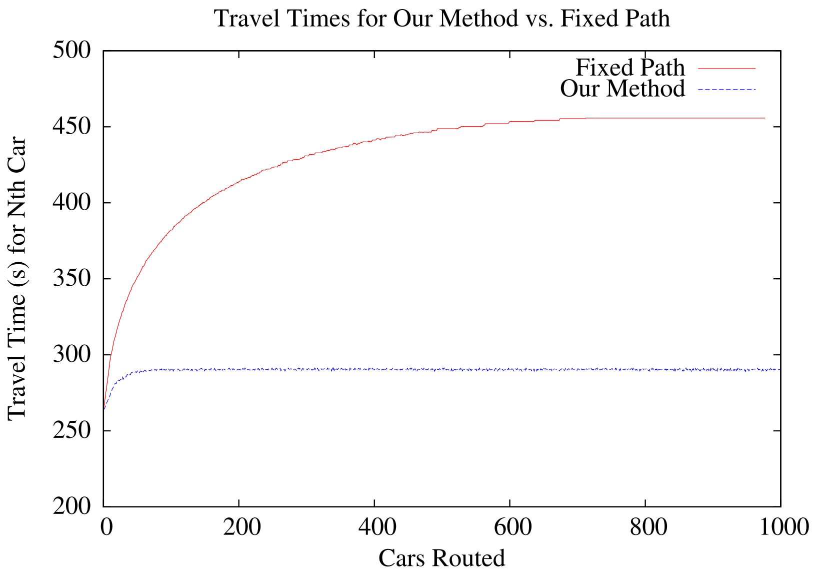


Figure 4.4: We highlight the performance of our algorithm compared with routing cars along a single path. The flow of cars quickly leads to congestion and long travel times for the single path. Our approach distributes the cars and settles to a constant travel time.

algorithm to using stochastic-historical prediction. Finally, I investigate the performance of our algorithm as the percentage of total number of cars that are controlled varies.

4.4.1 Traffic Simulation

To simulate the travel times of the cars in these experiments, I use the same derivation as above. I calculate the estimated travel times using the fundamental diagram and the time-varying density data. This density data is then updated for every car that travels the network. For these experiments, I have added a *cutoff-capacity* to our road network edges. This is the maximum density value the edge will be assigned, regardless of how many cars are routed on it. This ensures that the planners I compare against do not plan routes with infinite time duration.

4.4.2 Avoiding Congestion on a Single Path

I designed this benchmark to showcase the basic premise of our approach. The road network is a rectangular grid of 5 $\times$ 5 intersections, connected by a road segments with equal maximum speeds, 22.35 m/s, capacities, 0.09 cars per meter, and cut-off capacities, 0.085 cars per meter. Each road segment is approximately 1000m. I assume the road network is initially empty (i.e. there is no traffic). Then, I begin routing a set S of cars, each defined by a tuple (s, g, t) of a starting vertex s, an ending vertex g, and a starting time t, enter the road network and traverse the route given by a route planner. I assign the starting vertex for each car to be the bottom left corner of the grid and the goal to be the top right corner. Each car is given a starting time $t = 2 * i$ seconds, where i is the car index in S.

I ran this experiment for both planners, the route planner that simply returns the shortest path between the start and goal vertex, independent of traffic conditions, and our self-aware system.

The result of this can be seen in Figure 4.4. Obviously, since all cars have the same start and goal vertices, they are all assigned the same route by the shortest-path planner, quickly causing growing congestion on this route. The incoming flow is enough to cause significant congestion, but not to cause a complete traffic jam. Our method distributes the cars along multiple paths to the goal based on densities predicted by earlier planned routes. By doing so, it can handle the flow of vehicles at a relatively constant travel time.

4.4.3 Comparison with Shortest Path Planner

In the second benchmark, I compare the behavior of our algorithm to the shortest path planner while routing a set of cars S with random initial and goal intersections. This scenario takes place on a grid road network with 15 $\times$ 15 intersections and a road length of 100m. The road network is initially empty. As above, I perform the experiment for both planners: first, I route each car in S using a shortest path planner and calculate the resulting travel times. Second, I do the same assuming all cars are routed using our self-aware route planner.

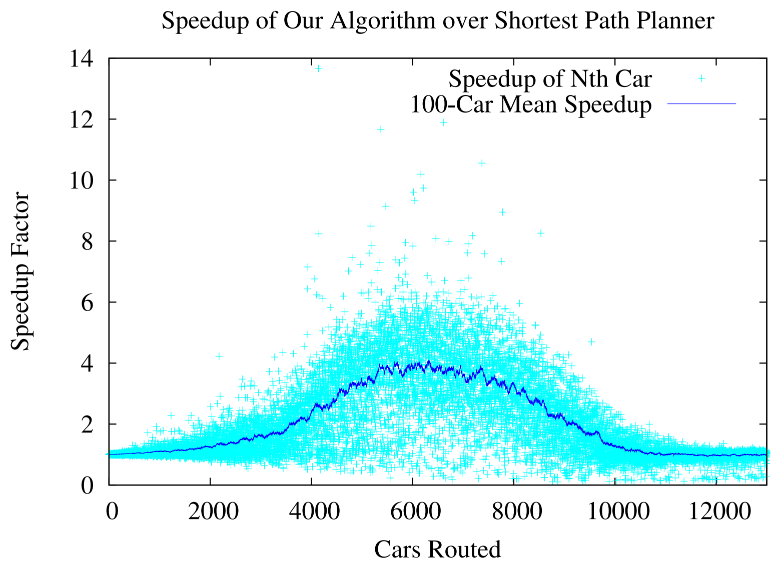


Figure 4.5: This figure shows the speedup factor that our method achieves over a shortest path planner for a series of cars. Each car has a random start and goal position.

To best illustrate the effect of network load in this scenario, each car has a starting time of zero. The parameters used for the experiment were capacity = 0.09 cars per meter, maximum velocity = 22.35 m/s, and cut-off capacity = 0.085. For storing the density information for each edge, a time discretization of 15s was used.

Figure 4.5 shows the result of this experiment. The result is given in terms of the speedup of our method over the shortest path planner, i.e. the ratio of the travel time planned by the shortest path planner over the travel time planned by our method. The speedup for every car routed is displayed as the cyan scatter plot, and the average speedup factor for each cohort of 100 cars is displayed as the blue line. As can be seen, initially the speedup factor is negligible, since the road network has low traffic densities that do not significantly slow down traffic. Hence, in these cases the shortest path is indeed also the time-optimal path. However, as more an more cars have entered the road network, the average speedup in travel time by using our self-aware planner rather than a shortest path planner increases, peaking at a factor of approximately 4 after 6,000 cars have been planned. Eventually, the average

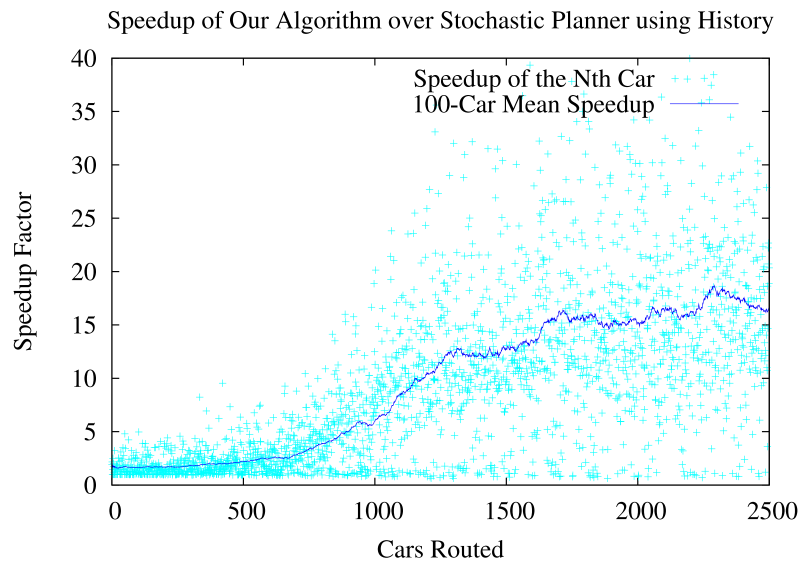


Figure 4.6: This figure shows the speedup factor (up to 20 for the 100-car mean) that our method achieves over a stochastic planner using only historical data for a series of cars, indexed from 0 to 2500. Each car has a random start and goal position.

speedup decreases again, since the road network has become so congested that alternative routes do not provide any benefit in terms of travel time.

4.4.4 Comparison with Stochastic Planner Using Historical Data

In this experiment, I compare the behavior of our algorithm to a stochastic planner using historical data (SPUHD) when routing a population of cars S. This experiment takes place on the same grid road network as above, with 15×15 intersections. I generate *historic* traffic data by defining a set of cars S with random start and goal intersections, routing each car in S using a shortest path planner, and calculating the resulting densities. The cars are created in 40 batches of 200, with each batch having a starting time of 5 seconds later than the preceding batch. This creates a maximum average network density, using the shortest path routing, of 0.054 cars per meter, and areas of full congestion, i.e. 0.085 cars per meter. These time-varying, edge specific stochastic densities are the *history* for the scenario. I assume that S represents the typical traffic flow. I compare our method and the SPUHD by planning for all of S, given the calculated stochastic *history*. In our self-aware routing algorithm, the planned routes are used to update the traffic densities for plans of future cars, but our planner is unaware of the *history*. In the SPUHD, each car in S is planned for assuming the *history* is a valid prediction, and these predicted densities are not updated based on the SPUHD's planned routes.

The results are shown in Figure 4.6. I can see that only using the stochastic *history* is not an effective strategy when all cars are being navigated by the planner. This is an extreme example, but it illustrates a basic motivating problem with using stochastic prediction. If the planner were routing one car in S, then the *history* would be almost perfect; as the planner is routing all of S, the predictions based on the *history* are not valid. As the SPUHD *believes* a certain congestion pattern will occur, due to the *history*, it routes cars around that congestion pattern. However, as the SPUHD is controlling all the cars, the predicted congestion pattern does not occur, but instead the planner creates congestion in other areas. The SPUHD assigns cars sub-optimal routes due to its belief that the typical, historical traffic flows will remain constant. Our method, on the other hand, distributes car routes and achieves a 100-car mean speedup of up to a factor of 20.

The high speedup factors here illustrate how unsuitable pure historical prediction is when the entire set of cars is being routed. In attempting to avoid predicted densities, the stochastic planner using historical data irrationally prefers domains of the road network, which then causes congestion and traffic jams in those areas. Even at low network densities, approximately 0.014 cars per meter, the SPUHD causes traffic jams, with some roads being saturated to their cutoff capacity, 0.085 cars per meter.

4.4.5 Effect of Adoption Rate

In our final benchmark, I analyze the effect of the adoption rate of our system in scenarios in which part of the cars use our self-aware system, and the other part of the cars use a shortest path planner. I let a set S of cars with random start and goal intersections enter

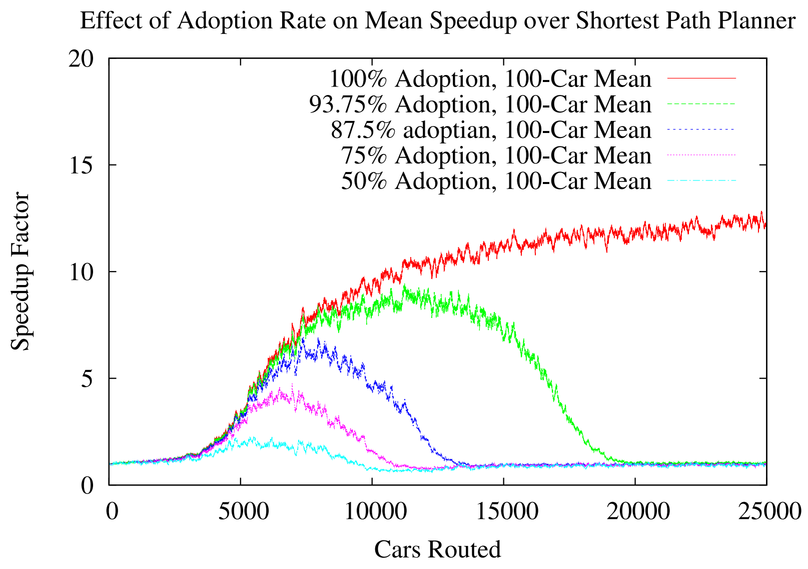


Figure 4.7: The 100-car mean speedup of our method over the simple planner for varying adoption percentages.

the (initially empty) road network at a rate such that over time heavy congestion is likely to be created on the road network. I use the same road network as above. A percentage α of the cars (randomly sampled from S) use our self-aware traffic route planner to plan their routes, whereas $1 - \alpha$ of the cars use a shortest path planner. For the sake of the simplicity, our self-aware route planner ignores the portion of cars it does not control; in reality this data can be estimated from historical data (see Section **Blending Historical and System Data**). I repeat this experiment for various values of α. The proportion of cars, α, that are routed by our method are chosen using a consistent random number seed: this implies that a car routed for a low α will also be routed for a high α, preserving features of the graphs for each value of α. The cars enter the road network at a rate of 50 per second.

Figure 4.7 shows the results, a graph depicting the 100-car mean speedup for various adoption values from 50% to 100%. I see a peek speedup of over 10 when 100% of the cars are controlled: our system avoids the creation of heavy congestion and large scale traffic jams. However, the maximum speedup and the integral of the speedup curve decrease rapidly with α, showing a strong sensitivity to uncontrolled cars creating traffic jams. At the 50%, a maximum speedup of 2 is observed, and for smaller α values, a similarly small speedup is observed. These results clearly show that the benefit of our system increases with the adoption rate.

4.5 Summary

State of the art approaches can handle stochastic planning and can make use of traffic predictions, but they ignore a useful source of information, i.e. the previously planned routes. As routing systems become more pervasive, the routes they plan will begin to significantly influence the future state of traffic. Prior plans then become relevant to future plans. My approach addresses this issue. I provide a method to update stochastic traffic predictions with previously planned routes. Given stochastic predictions of future traffic states, I plan for cars within this space. For each path planned, I update the stochastic predictions based on the routes planned for each car. The density of each edge is updated according to the estimated arrival and departure times for the car. The velocity of each edge is then updated according to the fundamental diagram, an empirical relationship between density and velocity. In my experiments, my routing algorithm results in better utilization of the road network, reduces congestion and the travel time for each car.

CHAPTER 5: PARTICIPATORY ROUTING: REAL-WORLD ROAD NETWORKS AND PROTOTYPE SYSTEM

5.1 Introduction

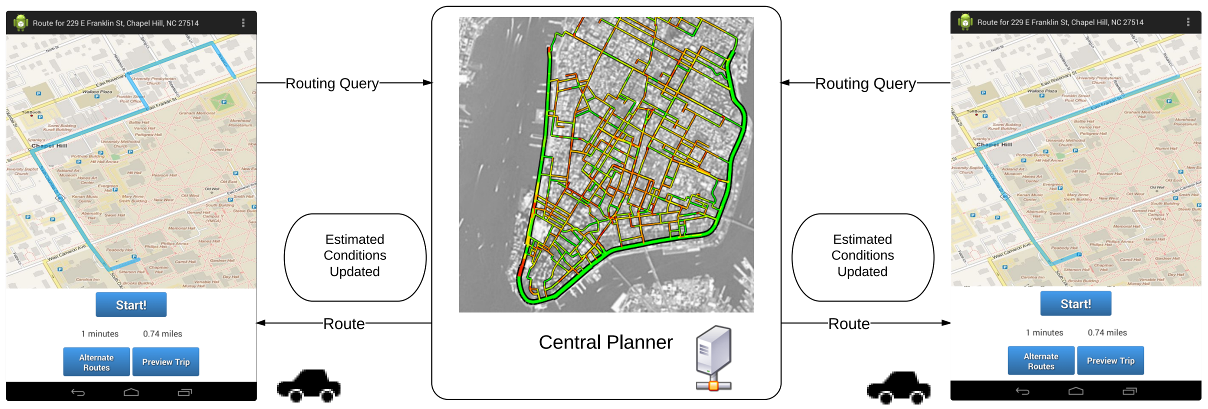

Route for 229 E Franklin St, Chapel Hill, NC 27514
Routing Query
Routing Query
Estimated Conditions Updated
Estimated Conditions Updated
Route
Route
Central Planner
Start!
1 minutes
0.74 miles
Alternate Routes
Preview Trip

Figure 5.1: **A System Architecture of Participatory Route Planning:** The mobile clients send route queries to the central planner, which takes the updated traffic and routing information from participants and live traffic sensor information to plan a new route for each participating client in the network.

In the previous chapter, I discussed and demonstrated the theory behind my participatory routing approach. Here, I further develop the method, expanding it to handle real-world road networks, demonstrate its effectiveness via large-scale, state-of-the-art simulations, and present a prototype implementation. A system such as this could be a step toward addressing traffic system inefficiencies and provide a next-generation solution to traffic congestion management.

Traffic congestion management is a global challenge that has increasingly important economic, societal, and environmental impacts. It is unlikely that traditional physically-centered mitigation strategies by themselves will be successful or sustainable. Numerous strategies have been proposed to construct Intelligent Transportation Systems (ITS) by incorporating sensing, information, and communication technologies in transportation infrastructure and vehicles. Through networks of sensors, recent cutting-edge efforts can provide real-time traffic monitoring for subsets of the road network, they have not yet offered system-level relief to the traffic congestion problem.

When planning a route, state-of-the-art planners use traffic predictions derived largely from recent and/or historical traffic data. Live traffic data can be collected by loop-detectors, cameras, toll port data, and cell phone localization. These systems provide the traffic velocity at certain locations at a fixed frequency [Brakatsoulas et al., 2005], which can then be used by vehicles to plan around congested areas. However, live data alone does not enable predicting future traffic conditions. For example, if a route is planned in which a car will arrive at a certain road in half an hour, the current conditions may not be an accurate estimate for the conditions of that road 30 minutes later.

This problem can be addressed by using a prediction scheme for the future traffic conditions based on the current and historical probabilistic data of traffic conditions at similar times of the day under the similar weather [Horvitz et al., 2005], [Nikolova et al., 2006a], [Min et al., 2007]. However, these predictions are only valid if the vehicles in the system continue to travel by the same routes, i.e. if only a small portion of the vehicles take the historical knowledge into account. For example, if there exists a centralized route planner controlling *every* car in the system, historically congested areas would be unduly avoided, causing congestion to appear in new areas: the historically predicted pattern would not arise.

Extending and exploiting the idea of participatory sensing [Burke et al., 2006], I propose the novel concept of *participatory route planning*, which uses the routes of vehicles in the current networked system to coordinate with each other by sharing their planned routes with a central router via mobile communication. My adaptive approach accounts for the fact that when a route is planned for a car, that car will then cause a small increase to the traffic density on the roads it later traverses. I can thus use the predicted paths from the route planner itself as an information source, *in addition to* historical data and/or current traffic condition. This allows my participatory route planning system to plan routes more effectively for a large percentage of the cars as the impact of their routing choices on future traffic conditions is taken into account. Every car that queries the route planner can then use this information to plan a route for itself and implicitly coordinates the route with the previously routed vehicles. Its planned route is then used to update the estimate of future traffic conditions for other vehicles participating in this system.

I present large-scale experiments using a state-of-the-art microscopic traffic simulator to demonstrate the functionality of my system in scenarios ranging from simple convoys of a couple hundred cars to city-scale rush hour scenarios with thousands of cars. My experimental results suggest that such participatory route planning can better coordinate and distribute vehicles, resulting in an overall reduction of the travel time. Further, I provide a prototype system demonstrating how a system such as mine could be feasibly implemented for city-scale demand.

The rest of the Chapter is organized as follows. In Section 2, I discuss related work to my approach. In Section 3, I give an overview of my prototype mobile system. Section 4 presents an overview of the mathematical framework I build upon and my new algorithms for participatory route planning. Finally, I discuss the implementation, experimentation and validation of this novel approach in Section 5. I conclude by discussing future research directions.

5.2 Related Work

A related body of work includes [Dresner and Stone, 2004], in which a multi-agent reservation-based system was proposed to replace existing signalized intersections. Our route planning algorithm also takes into account the effect of other users in the system, but, in contrast, our algorithm uses a global, macroscopic approach to alleviating overall traffic congestion, rather than local considerations of intersection throughput. To create the system proposed by [Dresner and Stone, 2004] in the real world would also require highly accurate sensors at every intersection in the system, whereas our system requires no additional sensing infrastructure.

Another body of related work is the study of Dynamic Traffic Assignment, the problem of distributing flows of traffic from known origins to destinations. A summary of approaches can be found in [Peeta and Ziliaskopoulos, 2001]. Most relevant to our work are the solutions involving simulation, such as [Florian et al., 2008]. In these approaches, vehicles are iteratively routed and simulated until an equilibrium is reached in which no further reduction in travel time can be obtained.

I have created a mobile system that features our participatory route planning algorithm. Our mobile system is similar to CarTel ([Hull et al., 2006]), a mobile sensor computing system that utilizes connectivity to the internet and GPS to collect and process data from sensors located on mobile systems. In this respect, our work also resembles that of the Mobile Millennium project, which attempts to use GPS in cellular phones to gather traffic information in real time ([Jariyasunant et al., 2010], [Amin et al., 2008]). However, in contrast to these projects, which sense live traffic data, our system not only considers current traffic conditions, but also utilizes the plans of the users to coordinate those users by predicting their effect on the road conditions.

With respect to our destination prediction model, our algorithm is inspired by the work of [Ashbrook and Starner, 2003] and [Liao et al., 2007]. In both of these approaches, the authors proposed a method that utilized GPS data to construct a Markov model, which was then used to generate accurate inferences about the user's daily movements. Our prediction model is closest to the work done by [Ashbrook and Starner, 2003], where a Markov model was created for each distinct location, with transitions to every other location. Since our app may be not running at all times, GPS data is not present continually. Hence, I extend the proposed method by leveraging information about the user's location and his or her past routing queries to forecast the user's destination.

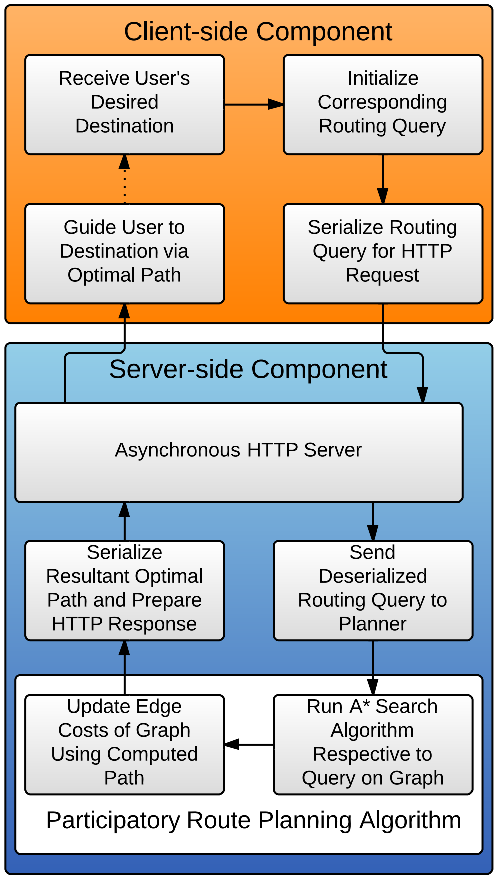

Client-side Component
Receive User's Desired Destination
Initialize Corresponding Routing Query
Guide User to Destination via Optimal Path
Serialize Routing Query for HTTP Request
Server-side Component
Asynchronous HTTP Server
Serialize Resultant Optimal Path and Prepare HTTP Response
Send Deserialized Routing Query to Planner
Update Edge Costs of Graph Using Computed Path
Run A* Search Algorithm Respective to Query on Graph
Participatory Route Planning Algorithm

Figure 5.2: **System overview of our client- and server-side model:** The client-side component is portrayed in the diagram (with the orange background) and illustrates our mobile app system; the routing query is received from the client, serialized, and sent to the server, which handles the request. The server-side component (shaded in blue background) shows the handling of the routing query and our participatory route planning process. The routing algorithm is ran with respect to the destination request, the stochastic road map is updated, and the optimal route in term of travel time is sent back to the client.

5.3 Prototype System

Participatory routing has performance requirements distinct from typical routing systems due to the heavy volume of queries that are expected. I have therefore created a prototype system to show that such a system can be created and perform in real time.

Our system implements a client and server architecture (Figure 5.2). For our client implementation, I have created a mobile application that is capable of asynchronously sending routing queries to the server and obtaining routes to the desired location. Once a user has

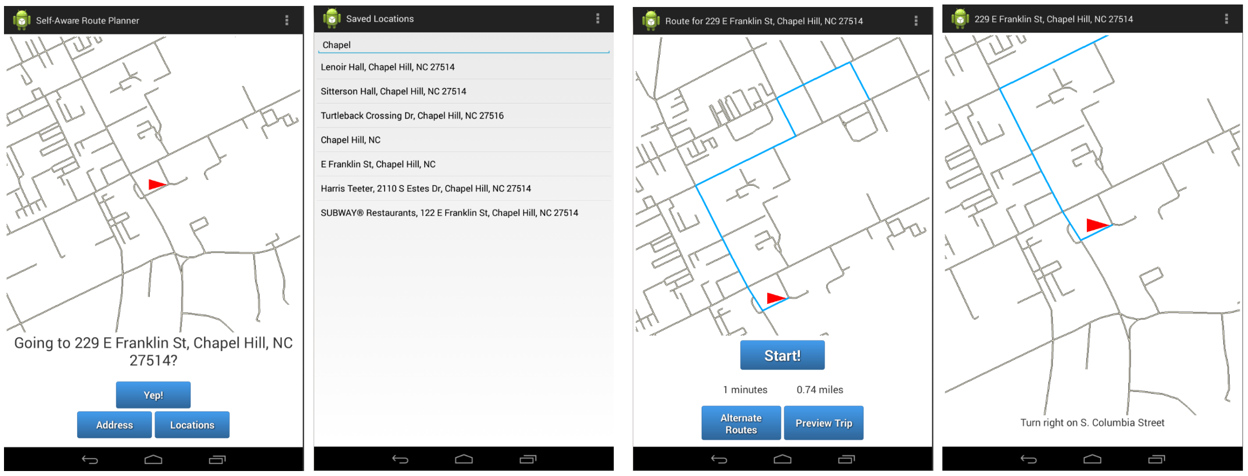

Self-Aware Route Planner
Going to 229 E Franklin St, Chapel Hill, NC 27514?
Yep!
Address
Locations
Saved Locations
Chapel
Lenoir Hall, Chapel Hill, NC 27514
Sitterson Hall, Chapel Hill, NC 27514
Turtleback Crossing Dr, Chapel Hill, NC 27516
Chapel Hill, NC
E Franklin St, Chapel Hill, NC
Harris Teeter, 2110 S Estes Dr, Chapel Hill, NC 27514
SUBWAY® Restaurants, 122 E Franklin St, Chapel Hill, NC 27514
Route for 229 E Franklin St, Chapel Hill, NC 27514
Start!
1 minutes
0.74 miles
Alternate Routes
Preview Trip
229 E Franklin St, Chapel Hill, NC 27514
Turn right on S. Columbia Street

Figure 5.3: **Different screens featured by our Android App** (from left to right): (a) the Main Screen, hosting the interactive map display and destination prediction output; (b) the Locations Screen, showing previous destination queries; (c) the Route Confirmation Screen, detailing the route to destination; (d) the En Route to Destination Screen, portraying turn-by-turn directions for the user to follow and the interactive map display.

chosen a destination on the client, an asynchronous routing query is sent to the server with the most recent information regarding the user's latitude, longitude, and orientation.

In order to handle requests from numerous clients, our server employs an asynchronous and multi-threaded approach. One thread is reserved for routing and update computations, as each route is technically dependent on the previous routes' updates. The remaining threads take queries from clients, process them, and return computed routes, which are composed of a vector of road segments along with their respective geometries.

Our client's Graphical User Interface (GUI) is based on common mobile and GPS routing programs on both Android devices and iPhones. The screens I have implemented can be seen in Figure 5.3. However, a participatory system has some distinct requirements for a GUI as well. As mentioned before, this system is envisioned to involve a large portion of the traveling vehicles: a typical use case would be someone commuting to work in the morning. As such, the interface for such a system needs to be as unobtrusive and intuitive as possible. One approach to achieving this design goal is having the client work automatically, or as automatically as possible. To this end, our client application will estimate the destination for the user based on their routing history. This can be seen in 5.3(a) where an address is suggested on the main screen. The user then needs to only confirm. One could also imagine a completely automatic client built into a car's GPS system.

Our method for estimating these destinations is an adaption of the algorithms proposed by [Ashbrook and Starner, 2003] and [Liao et al., 2007]. These methods use a Markov model to estimate the transition probabilities between locations. At the time a query is requested, the client checks to see what the most likely destination is from the user's current location, based on the frequency of previous destinations. This model could also be extended to take the time of departure into account.

5.4 Algorithms

In this section, we present the key mathematical foundation and the new algorithms for participatory route planning.

5.4.1 Mathematical Framework

This system builds upon the theoretical framework introduced in the preivous Chapter. This work suggests a theoretical approach to routing vehicles that considers the system's own planned routes as a new source of information. The method routes cars individually, and then uses the planned routes as estimates for the cars' trajectories in the near future. These trajectories, in aggregate, form an estimate of the future traffic pattern. This method is composed of (a) a *route planner* that computes paths for cars through a time-dependent density field defined on the road network and (b) an *update* that modifies the density field according to the calculated route.

The *route planner* makes use of an stochastic A*-search algorithm through a time-dependent density field. This field is composed of Gaussian distributions, discrete in time and space, defined over the road network graph, $\mathcal{G}$. The algorithm is shown in Figure 5.4. For each road explored, the cost of traversing the road is the estimated travel time

```
TrafficA*$(s, g, t_0)$
1: $\forall u \in V : \bar{\tau}_u \leftarrow \infty, \tilde{\tau}_u \leftarrow \infty$; $\bar{\tau}_s \leftarrow 0$; $\tilde{\tau}_s \leftarrow 0$
2: $OPEN \leftarrow \{s\}$
3: while $OPEN \neq \emptyset$ do
4:     $u \leftarrow \arg\max_{u \in OPEN}\{\bar{\tau}_u + \bar{h}(u) + w(\tilde{\tau}_u + \tilde{h}(u))\}$
5:     $OPEN \leftarrow OPEN \setminus \{u\}$
6:     if $u = g$ then
7:         return
8:     end if
9:     for each edge $e = (u, v)$ in $\mathcal{G}$ do
10:        $\mu \leftarrow \bar{\tau}_e(t_0 + \bar{\tau}_u)$
11:        $\sigma \leftarrow \tilde{\tau}_e(t_0 + \bar{\tau}_u)$
12:        if $\bar{\tau}_u + \mu + w(\tilde{\tau}_u + \sigma) < \bar{\tau}_v + w\tilde{\tau}_v$ then
13:            $\bar{\tau}_v = \bar{\tau}_u + \mu$
14:            $\tilde{\tau}_v = \tilde{\tau}_u + \sigma$
15:            pred$(v) \leftarrow u$
16:            $OPEN \leftarrow OPEN \cup \{v\}$
17:        end if
18:    end for
19: end while
```

Figure 5.4: The A* algorithm to compute an optimal route with respect to the cost metric between start node s and goal node g at time t_0. τ values are travel time estimates/costs, and h is the heuristic function.

$$\tau_e(t) = \frac{\ell_e}{f_e(\rho_e(t))},$$

which is the length of the road divided by the estimated velocity, $f_e(\rho_e(t))$, which is a function of the estimated density. The function uses values for the maximum density and maximum velocity to determine the current velocity, which can be estimated using a number of models, as discussed in [Greenshields et al., 1935, Work et al., 2010]. In my work, I use a version of the function for the equilibrium velocity presented in the *Aw-Rascle-Zhang* traffic model,

$$f_e(\rho_e(t)) = (v_{max} - v_{min}) * (1 - \frac{\rho_e(t)}{\rho_{max}})^\gamma + v_{min}$$

Once a route has been planned, that route is considered an estimate for where that car will go in the future. For each car routed, the method adds a marginal amount of density to the road network along the planned path. For each edge of the route, the travel time estimates are used to calculate the probability that the car is on that edge at a certain time,

$$q_{(u,v)}(t) = \int_{-\infty}^{t-t_0} pdf_{\tau_u}(t')\, dt' \cdot \int_{t-t_0}^{\infty} pdf_{\tau_v}(t')\, dt',$$

where $q_{(u,v)}(t)$ is the probability that a particular car is on edge $e = (u, v)$ at time t. This car is then added to the density field for that road segment, taking into account the length of the road.

$$\rho_e(t) = \rho_e(t) + q_{e=(u,v)}(t)/\ell_e$$

This approach, though promising, has several shortcomings. First, the method ignores the effect of traffic intersections, which can be a significant cause of congestion and delay in urban scenarios. Second, the system ignores the potential growth and propagation of traffic jams. Lastly, it does not make use of 'participatory sensing' to compute and update the routes planned.

5.4.2 Participatory Route Planning

Compared to prior work, this algorithm can handle traffic jams and takes into account delays caused by signalized intersections.

5.4.2.1 Traffic Jams

Traffic jams are a common occurrence both on highways and on urban roads. One obvious effect of traffic jams is a reduced velocity for vehicles on the affected road. But traffic jams have the added effect of limiting the number of cars that can use a roadway. The result is that traffic jams can grow upstream, expanding to adjacent roads.

I model this effect by including a capacity for each road. In the original formulation, the travel time estimate for a road is calculated as

$$\tau_{travel} = \frac{\ell}{f(\rho(t))},$$

where $\rho(t)$ is the time-dependent density estimate, ℓ is the road length, and $f(\rho)$ is a function that relates density to speed.

Our formulation adds an additional term for each road,

$$T_{clear} = \{t | \rho(t) < \rho_{max}, t \geq t_A\}$$

$$\tau_{delay} = \min(T_{clear})$$

where $\rho(t)$ is the time-dependent density, ρ_{max} is the maximum density, and t_A is the estimated arrival time of the car at this road. The new estimated travel time is thus

$$\tau_{est} = \tau_{travel} + \tau_{delay}$$

For most roads, this delay will be zero as the roads will have less than the maximum density, but for congested roads, this can cause an increase in the estimated travel time.

This delay allows traffic congestion to grow back through the road network. I modify the *update* algorithm such that the delay time for entering a road is propagated backward as long as there are congested roads to sustain it. In other words, if a car is attempting to enter a congested road segment, it needs to spend time waiting in its current road segment. If the current road segment is also congested, the delay propagates back farther. This waiting time adds density upstream of the congestion, growing the traffic jam – similar to how traffic shockwaves travel.

5.4.2.2 Intersection Delay

In the earlier work on traffic-predictive routing, intersections were only considered for their topological properties: they only created the connections of the network. In real world scenarios, especially urban scenarios, intersections cause delays that need to be modeled.

To model intersection delay, I build a *queuing model* over the continuum traffic representation. For this, I first calculate the expected number of cars on the road

$$c = \rho(t) * \ell$$

where c is the number of cars, $\rho(t)$ is the time-dependent density, and l is the road length. I then calculate the *outflow*, o, or the number of cars per time interval that can depart, using a Riemann solver for the *Aw-Rascle-Zhang (ARZ)* system [Sewall et al., 2010c], as my routing system models traffic in the same manner as ARZ[1]. The Riemann solver finds the flux between two road segments, given their densities, velocities, and maximum velocities. However, it is not known where cars currently on the road will go, and thus what road(s) to use to calculate the outflow. These cars could have been previously routed, sensed, or estimated from historical conditions. I make a *worst-case* assumption that the preceding cars on the road will choose the downstream road with lowest outflow.

Alternatively, the model could choose a best-case outflow or an average outflow. However, I have observed that the worst-case assumption works best in practice.

The time required for the existing cars to leave the road is $\tau_{drain} = \frac{c}{o}$. This cost is only taken into consideration if it is greater than the estimated velocity,

$$\tau_{est}* = max(\tau_{est}, \tau_{drain})$$

as a car either freely traverses the road and departs or it reaches the existing queue and waits for it to empty, but not both.

[1]Though my system has zero relative flow.

This model, however, assumes an uninterrupted flow out of the road. If the road has a signalized intersection, this flow will be broken by the traffic signal's cycles. Therefore, for signalized intersections, I modify the τ_{drain} equation to incorporate the cycles of the traffic signal. The cars can only leave the road during the green light portion of the cycle, which is τ_g seconds long. Due to reaction time and acceleration constraints, only a portion of this will be usable, τ_g^* [Church and ReVelle, 1978]. To empty the queue of cars will require a number of cycles

$$c_g = \frac{\tau_{drain}}{\tau_g *}$$

where c_g is the number of effective green lights needed for the outflow to finish.

Each of these cycles takes

$$\tau_g * i = \tau_f$$

where i is the number of independent green cycles. The final time estimate then is

$$\tau_{light} = c_g * \tau_f$$

and thus the overall model is

$$\tau_{final} = \max(\tau_{est}, \tau_{light}) + \tau_c$$

where τ_c is a constant cost for the intersection regardless of the current queue length, $\tau_c = \tau_f(1 - \frac{1}{i})\frac{1}{2}$.

5.5 Experiments

As a proof-of-concept, I conduct city-scale experiments to demonstrate the functionality of my system. These experiments start off by illustrating simple behaviors and then build up to more complex scenarios.

To perform these experiments, I use the Simulation of Urban Mobility (SUMO) simulator [Krajzewicz et al., 2012], a state-of-the-art microscopic traffic simulator. My experiments consist of comparing the performance of my participatory routing system against baseline routing systems in various scenarios. For each experiment, I create a population of vehicles, each with an origin, a destination, and a departure time. The cars are populated using a Poisson instantiation process at a specified rate, creating a relatively uniform distribution of cars over time. I create routes for these vehicles using both my method and baseline methods. These sets of routes are independently simulated, and I compare the resulting statistics, particularly the travel time, for each vehicle. The road networks used for these experiments are from Open Street Maps. The speed limit parameter for each road segment was assigned by SUMO, the maximum density for each road was calculated from the road and car lengths, and the average length of the green-light cycles for intersections are assigned by SUMO. I consider the exact cycle timings for the intersections to be unknown: i.e. for a road at a given time, I do not know if the light is green or red.

5.5.1 Travel Time Prediction

One of the most significant advantages of my approach is that I can more accurately predict the travel time of cars than [Wilkie et al., 2011] (as shown in Figure 5.5), and thus I can use the planned routes themselves to better predict future traffic patterns. This capability enables my method to be used in conjunction with real-time sensing and planning. An example of this can be seen in Figure 5.10, in which my method can predict patterns of congestion, while the Self-Aware framework predicts fast travel and little to no congestion.

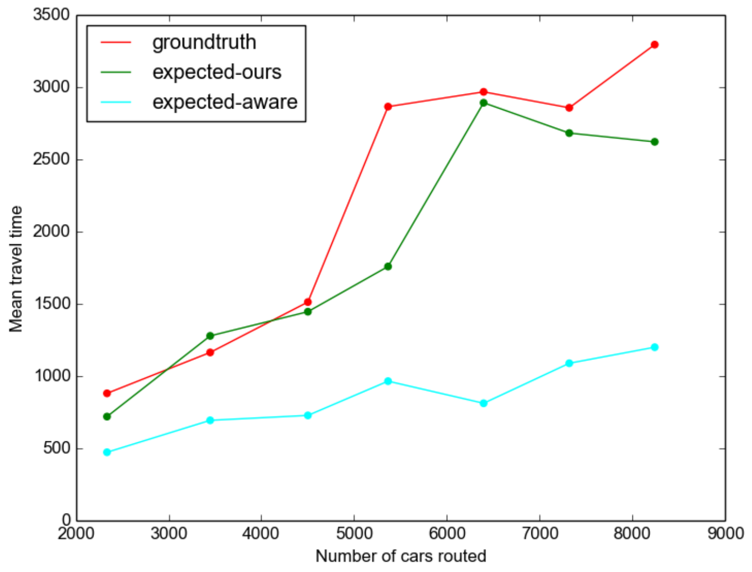


Figure 5.5: The mean travel time for the simulated ground truth (red), my method (green), and the self-aware routing algorithm (blue) for the MSMD scenario, described below. My method can closely track the ground-truth travel time, while the self-aware algorithm systematically underestimates the actual travel time due to its inability to account for intersection delay.

5.5.2 Baselines

The following simulations compare my method with multiple baseline systems.

5.5.2.1 Shortest-Path Router (SP)

The simplest baseline system I use is a shortest-path router. The router I use for this is *Duarouter*, part of the SUMO package. I provide this router with the start and goal roads for each query, and it returns a route that I then provide to the simulator.

5.5.2.2 Sensor-Data Router (SD)

This baseline system models existing, state-of-the-art commercial systems. These systems can receive sensor readings from highway loop-detectors, mobile devices, and other sources to create velocity estimates for the road network. These velocity estimates can then be used to find the fastest path to the goal. Similarly, this baseline router receives the mean velocity for every road in the network every 60 seconds, which represents the aggregation period used with real sensors. For each vehicle, this baseline plans the fastest route given the current velocities and road geometries. To implement this, I simulate all cars up to the end of the current time window, t_i, and then export the mean velocities of that time window, $\delta t = t_i - t_{i-1}$, back to the router. I then route vehicles that depart from t_i to t_{i+1} using the reported velocities, simulate all cars up to t_{i+1}, and so on.

It should be noted that real world systems would have noisy sensor measurements and would likely only have measurements on highways and major roads. In these regards, this **SD** baseline system is more powerful than existing navigation systems and routing alternatives.

5.5.3 Scenarios

In this section, I discuss my experimental set-up for various scenarios. These scenarios use two road networks, one of the lower portion of Manhattan and one of the city of Sioux Falls.

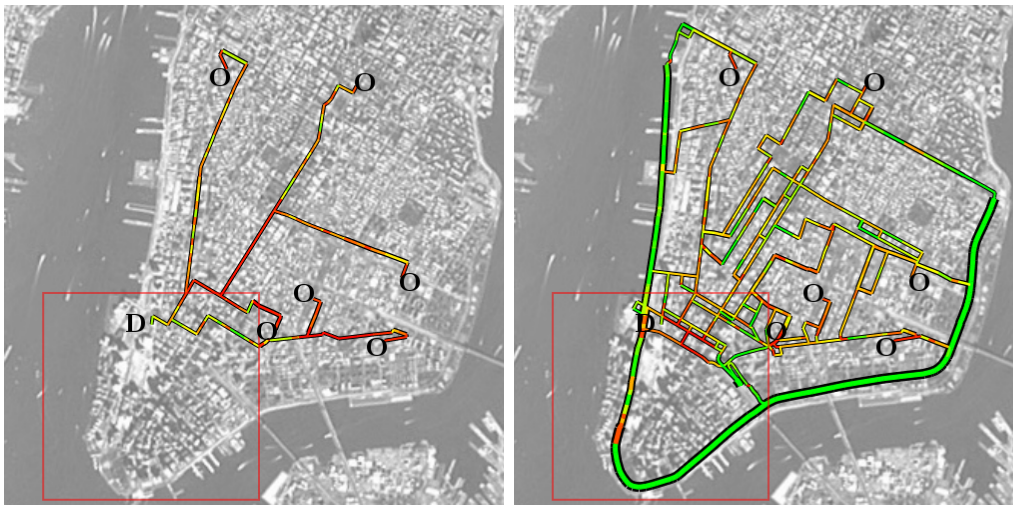


Figure 5.6: **An example of the *MSSD* scenario:** in which vehicles are spawned at various origin points and drive to a single destination. On the left is the mean velocity field of the fastest-path baseline router, and on the right is the mean velocity field resulting from my method. The destination point was chosen randomly inside the destination region, designated by the red rectangle. The origin points were chosen randomly outside this region with the additional constraint that they be at least 1km away from the destination. The mean velocity is shown as a color ranging from red, 0 m/s, to green, 13 m/s, the speed limit for most of the roads.

5.5.3.1 Manhattan

These experiments show the behaviors and characteristics of the routing system in a realistic urban setting. They feature several traffic flows intended to illustrate the performance of the system in different scenarios. The map used for these experiments is of lower Manhattan from Open Street Maps.

Single-Source, Single-Destination (SSSD). The simplest routing test is between a single origin and destination (OD). Car queries were generated between a randomly chosen origin and a randomly chosen destination over a 20 minute period at a fixed rate using Poisson instantiation.

Multiple Source, Single Destination (MSSD). This scenarios are modeled on a typical commuting pattern, in which vehicles from a surrounding area travel to a downtown area. The downtown area is defined by a bounding box, seen in Figure 5.6. The destination must be within this bounding box, while the origins must be outside and at least 1000m away. For the scenario, there are 5 origin locations generated and one destination. Traffic queries were generated using Poisson instantiation over a 20 minute period.

Multiple Sources, Multiple Destinations (MSMD). This scenario simulates a larger, more realistic traffic flow throughout a city. The traffic is generated by combining 10 **MSSD** scenarios, each of which has 5 origins and 1 destination. I again use a bounding box to restrict the destinations. This scenario is motivated by traffic patterns during the morning rush hour, when residents leave their homes and commute to workplaces. Traffic queries were generated using Poisson instantiation over a 20 minute period.

Using real-time sensing with my system. In this experiment, I demonstrate how real-time sensor data could be combined with my system. Currently, sensor data is only available on a limited portion of the road network, primarily on highways where loop detectors are present. This data is also not typically available in real time. As my system assumes a large number of vehicles are participating, the vehicles themselves can report conditions in real time by providing velocity and position updates. These updates can be filtered to create an estimate of the velocity for each road, as was done in [Work et al., 2010].

To simulate this, I iteratively route batches of vehicles. Each batch consists of vehicles spawned during a time window of 60 seconds. For each batch, the mean velocity of each road in the simulation during the preceding batch is used as an input by my participatory routing system. This is done in a direct way by setting the speed-limit, v_{max}, for each road in my system to the corresponding simulated velocity. Historical data could be incorporated into my system in a similar manner.

5.5.3.2 Sioux Falls

These experiments are intended to show the behavior of the system in as realistic a scenario as I can create. For these experiments, I use a set of trips, presented in [Chakirov and Fourie, 2014], that match real-world conditions as defined by U.S. census

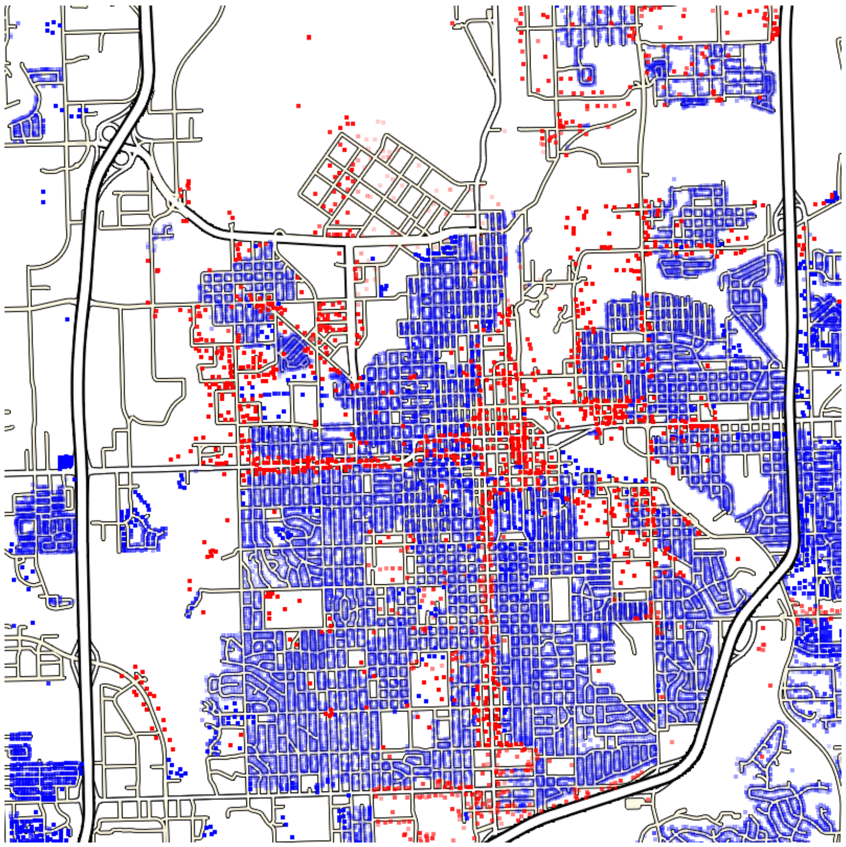


Figure 5.7: This figure shows the origins (blue) and the destinations (red) for the Sioux Falls morning rush hour commute.

data for the city of Sioux Falls. The origins and destinations for the morning rush hour portion of these trips can be seen in Figure 5.7. Versions of this scenario have been used by many traffic engineering studies, including [LeBlanc et al., 1975, LeBlanc, 1975, Suwansirikul et al., 1987]. One difference between my experiments and previous work is that I use a realistic street map from Open Street Maps data, whereas prior studies used a more abstract road network graph, representing only the major arterial roads and highways.

In this experiment, I compare my routing approach against two baselines: (a) the basic shortest-path router and (b) the sensor-data router. The set of vehicle queries used is a portion of the Sioux Falls morning rush hour traffic, from 7 am to 9 am, and limited to specified numbers of cars in order to show the system performance at different load levels. Each data point represents a separate experiment with the specified number of total vehicles

	Speedup					minutes
SP Baseline	mean	std. dev.	max	min	# cars	saved
SSSD	1.58	0.34	2.81	0.85	194	10.14
MSSD	1.99	1.14	7.64	0.35	599	14.51
MSMD-sparse	1.04	0.21	2.04	0.50	269	0.21
MSMD-medium	1.48	0.77	6.01	0.32	1231	5.21
MSMD-dense	2.02	1.76	18.35	0.10	2375	13.97
SD Baseline						
SSSD	1.13	0.33	2.00	0.63	194	1.98
MSSD	1.38	0.78	4.80	0.20	599	2.85
MSMD	1.16	0.53	5.51	0.37	1231	1.19
ours+sensors over ours	1.23	0.178	1.612	0.824	2400	NA

Table 5.1: Performance speedup of my approach over the Shortest Path (**SP**) and Sensor-Data (**SD**) baseline route planning. my method achieves upto a maximum speedup of 18.35 over SP baseline in congested traffic for the MSMD benchmark.

routed. In Figure 5.9, I can see the mean travel time of my method and of the two baselines for various numbers of vehicles.

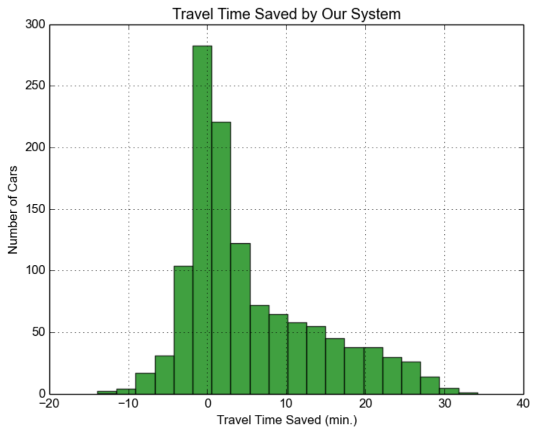
5.5.4 Results

This section details and analyzes the results of the experiments described above. First the results of the Manhattan experiments are discussed, followed by the results of the Sioux Falls experiment.

5.5.4.1 Manhattan

A summary of the experimental results for the Manhattan scenarios can be seen in Table 5.1. The first table shows the speed up statistics and the average time saved over the shortest-path baseline (**SP**). The second table shows the same results for the sensor-data baseline (**SD**).

- **SSSD.** my planner is able to get a mean speedup of 1.58 over **SP** by utilizing the spare capacity of the road network, saving an average of 10 minutes per car. One interesting



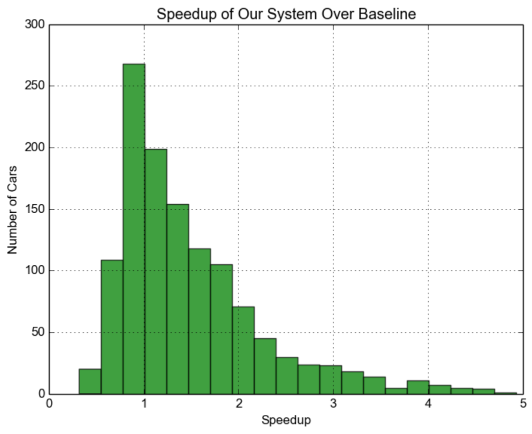


Figure 5.8: **MSMD Scenario:** These histograms show the performance of my system relative to the **SP** baseline router for a city-traffic scenario. On the top is shown the time saved by each vehicle by using my system, and on the bottom is shown the speedup of each car achieved by my system. For this example, the mean speedup of my system over the baseline was 1.48, and the mean time saved was 5.21 minutes.

aspect of this simple experiment is that my system was able to achieve a speedup even though the total number of cars was less than 200! For the **SD** baseline, using the same car population, I achieve a more modest speedup of 1.13.

- **MSSD.** For this experiment, the mean speedup of my system over the **SP** baseline was 1.99, and the mean time saved was 14.51 minutes. One reason this scenario has a higher speedup than **SSSD** is that the routes chosen must converge on a single destination, creating areas where traffic flows together and creating congestion. For the **SD** baseline, I have a speedup factor of 1.38.

- **MSMD.** In the case of the highest traffic demand, my planner was able to achieve an average speedup of 2.02 over **SP** saving over 13 minutes of travel time. The sparse demand scenario shows the behavior when there is insufficient flow to cause congestion – there is no speedup and traffic lights cause random variations in travel time. Finally, in the mid-range scenario, my system achieved a speedup of 1.48, with 5 minutes of travel time saved. my method achieved a small speedup of 1.16 over the **SD** baseline itn this case. (For the dense case, the **SD** baseline created gridlock and its cars could not execute their routes.)

- **Real-time Sensing.** I performed 20 randomized trials to investigate this system. For these trials, an average of 2400 cars were routed over a 40 minute period through the Manhattan road network using the same City-Traffic scenario set up described above. In Table 5.1 , I present the speedup of my system with this real-time sensor data over my system without real-time sensor data, *Mine no sensor*. I can see that on average using this real-time sensing data provides an additional speedup. I believe this could be even greater if the vehicles in the system were able to transmit position updates, which is a functionality that cannot be simulated using SUMO.

5.5.4.2 Sioux Falls

The mean travel times for various flow levels can be seen in Figure 5.9. I can see that my method out performs both baselines, achieving a mean time of 1189.99 seconds for twenty-five thousand cars, while **SP** had a time of 1996.89 and **SD** had a time of 1573.06, the speedups for which are 1.68 and 1.32. I can also see that at low flows, negligible speedup is present as there is less possible congestion.

Routing as traffic prediction. my approach allows the routes of the vehicles to be used to estimate the future traffic pattern. This is possible because my routing system functions like a mesoscopic traffic simulator, i.e. a simulator in which individual vehicles are propagated while a macroscopic density field is maintained. Being able to predict the traffic pattern that results from the vehicles following their routes is important: this enables the approach to be enhanced using real-time sensor data. If the system has a good prediction for the traffic state at a time, it can then fuse that prediction with incoming sensor data.

However, using routing to predict the traffic state is also difficult. Traffic is a complex dynamical system, with continuous and discrete aspects. Only the average behavior of the

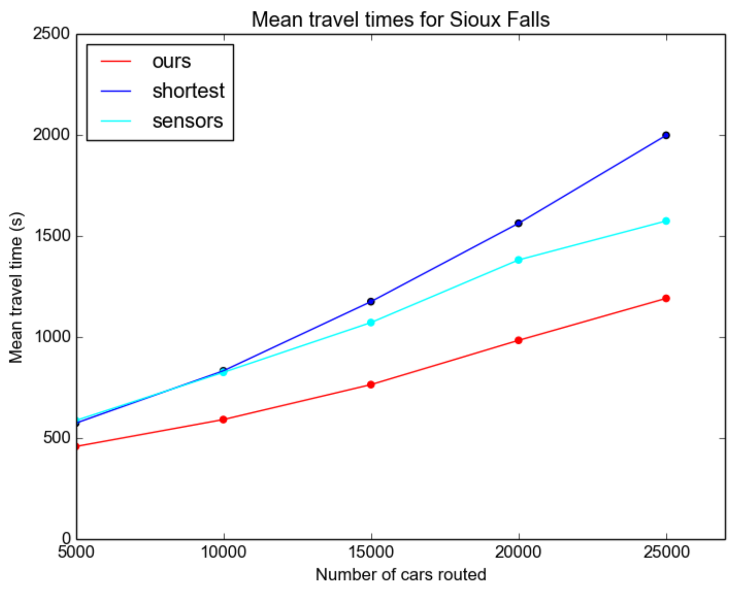


Figure 5.9: This figure shows the mean travel times for varying numbers of vehicles using my method, a shortest-path baseline (**SP**), and a commercial-like system using sensor data (**SD**). I observe that my method (red) outperformed both the shortest-path baseline (navy) and the commercial-like system with sensor data (sky blue).

traffic lights is known. A single point of congestion will grow outward through the network and cause large deviations.

In Figure 5.10, I can see my system's prediction for the evolution of the traffic state based on its routes. Two velocity fields are shown for my system's prediction (bottom) and for the simulated ground truth (top). Each figure shows the mean velocity for each road over a 10 minute period. I can see that the system predicts many areas of congestion correctly. However, the ground truth has some congested jams of greater severity than predicted. These jams are caused by intersections with stop signs, which can create an arbitrarily large delay for vehicles, causing congestion that grows through the network. This level of delay is difficult to predict, and so the travel cost for these intersections can be easily underestimated.

An Appendix with additional data and a video demonstration can be found at `bit.ly/1qwTfVy`.

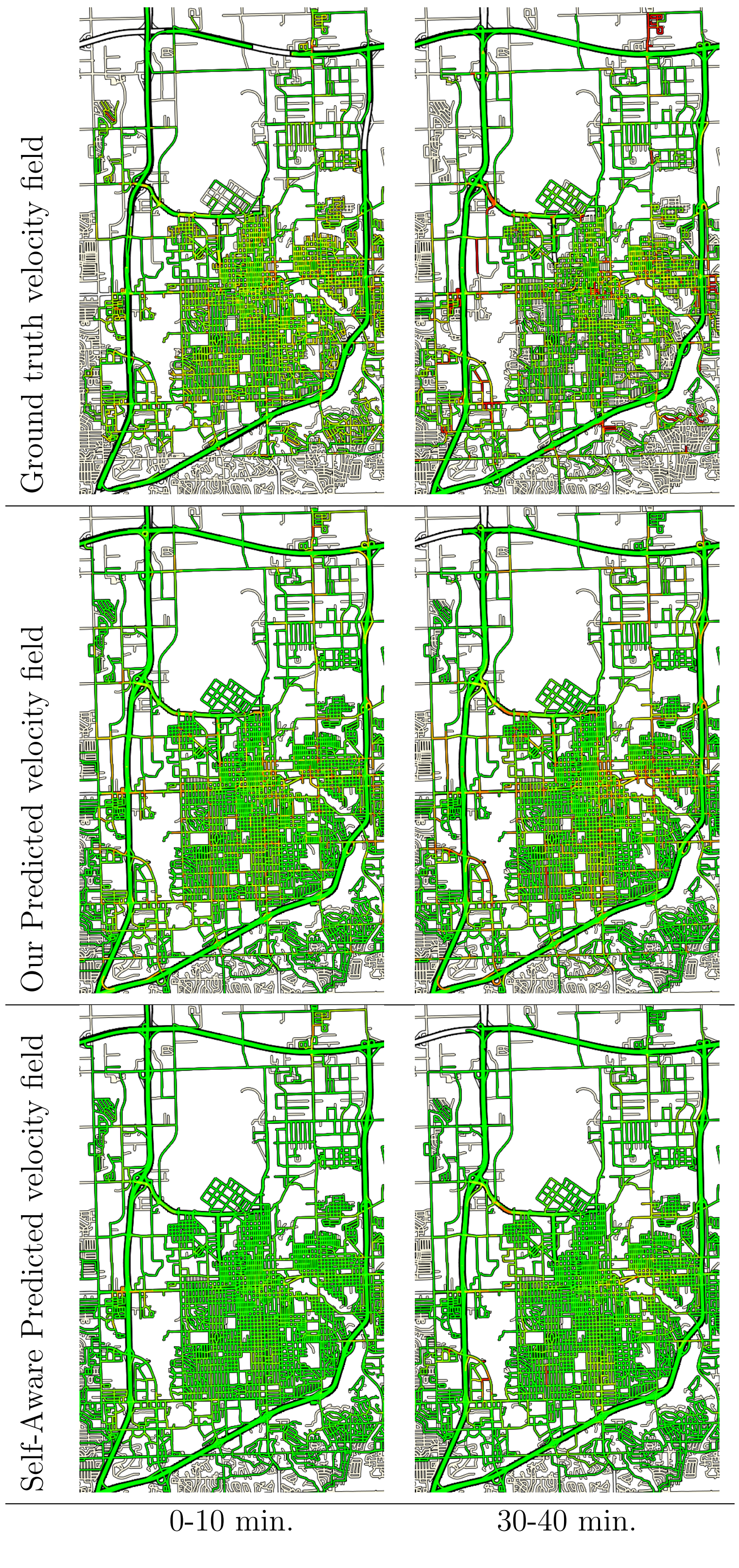

Ground truth velocity field
Our Predicted velocity field
Self-Aware Predicted velocity field
0-10 min.
30-40 min.

Figure 5.10: Each plot shows the mean velocity field over a ten minute time window, with a color ranging from red, 0 m/s, to green, 13 m/s. The top row is the ground truth; the middle row is the velocity field predicted solely by aggregating the routing requests of participating cars; the bottom row is the velocity field from using [Wilkie et al., 2011].

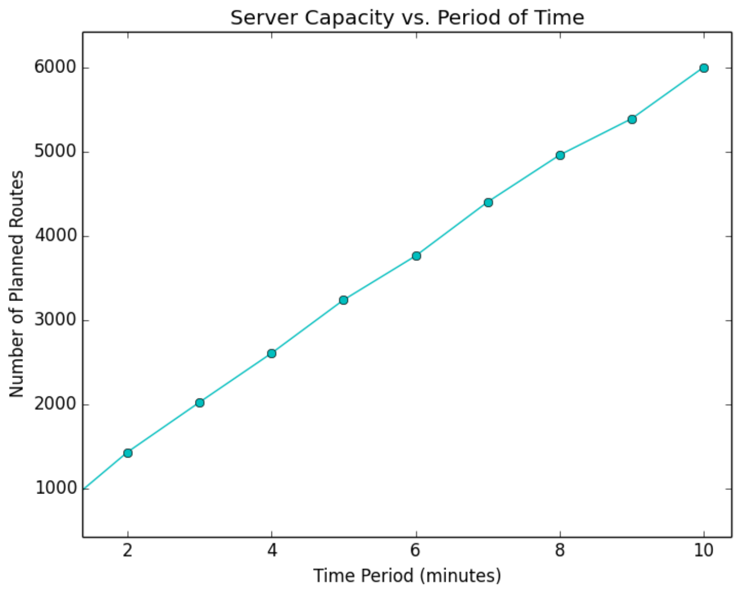


Figure 5.11: This figure shows the number of routes the server can generate in a period of time.

5.5.5 System Performance

In these experiments I demonstrate the performance and responsiveness of my server. In this experiment, the server was run on a 4-core, 3.33 GHz machine with 6 GB of memory. The simulated clients were run on a separate machine with the same specifications. The experiment consists of simulated clients generating queries, sending them to the server, and receiving responses. The queries were generated by uniformly sampling random origin and destination positions.

In Figure 5.11 I can see the number of requests that the server can handle within varying periods of time. In this scenario, a single simulated client continuously sent queries to the server. I can see in Figure 5.11 that the server was capable of planning for 6,000 routing requests within a period of ten minutes, which is a rate of 0.1 second per request.

5.5.6 Discussion

The experiments show a mean speedup for all the scenarios, though a few vehicles do have a slower travel times. One factor that leads to this is the random effects of traffic lights: the actual timings of traffic signals are unknown. An identical route will perform better or worse than its equivalent depending on how many red lights a vehicle needs to stop at. Another factor is the traffic jam growth. A vehicle that is stopped by an unexpected traffic jam will have a much larger travel time than a vehicle traveling an otherwise identical route.

An interesting question is why the sensor-aware routing baseline does not do perform better, as knowing the current velocity instantaneously would seem to give a great advantage to routing. However, planning a route given the *current* conditions can actually be worse than using no traffic information at all. If the conditions for some road change before a vehicle reaches it, then its plan was based on faulty knowledge. For example, consider planning a trip from California to New York: certainly a traffic jam that exists in Ohio at the time of the query should not influence the planned route.

The experiments I performed have some limitations. First, the noisiness of real-world traffic and sensing are unknown and therefore not accurately accounted for in my simulation. Third, the traffic simulator, SUMO, has difficulty simulating highly congested urban traffic. In these cases, the simulator can experience gridlock, which prevents the simulation from terminating. This limited the density of cars I could generate in some experiments. I believe that my system would perform even better at higher demand levels.

5.6 Summary

I have presented an approach to coordinating vehicles using participatory route planning. With this method, planned vehicle routes are used as an additional source of information for estimating future traffic conditions, enabling the system to plan routes for a large portion of vehicles, or even for every vehicle, and achieve a speedup in travel time over planners that only use static conditions or only use historically-based predictions of traffic conditions. I have presented novel algorithmic contributions that enable the system to work in real-world conditions. I have also implemented and demonstrated a prototype mobile client-server system. Furthermore, I have presented experiments that validate the performance and effectiveness of our system in terms of improved travel time and reduced traffic congestion.

CHAPTER 6: CONCLUSION

Traffic systems have changed little over the years. The traffic light is over a hundred years old but is still the most common control mechanism in the traffic system. Computer and information technologies are starting to reshape the traffic system, but their integration requires overcoming significant challenges.

To integrate computational techniques with traffic systems, it is necessary to model these systems and determine their current conditions (or state), linking the physical world into the digital, as well as to use these models for control and communication, linking the digital world back to the physical. My thesis statement addresses these challenges by proposing approaches to several challenges that stand in the way of creating a *cyber-physical* traffic system.

In this dissertation, I have presented techniques for integrating computational techniques with traffic systems that overcome several key challenges. These techniques combine new sensor data and open databases with digital models and simulations of traffic systems to achieve new means of communication, analysis, and optimization. I have shown the efficiency and efficacy of these methods via experiments and demonstrations.

6.1 Summary of Results

I have proposed a method to create metropolitan-scale road networks suitable for simulation and animation from publicly available GIS data. The resulting network is suitable for real-time, 3D simulation and animation. The method includes approaches to formalizing the road network, filtering common noise and errors, creating intersection geometry, and assigning heights and an ordering to highways, overpasses, and underpasses.

I have proposed a method to create data-driven traffic reconstructions. The method estimates the time-varying state of traffic from sensor data and creates a corresponding simulation. Unlike previous work, my method uses the state-of-the-art, second-order ARZ traffic model [Aw and Rascle, 2000, Zhang, 2002]. The resulting simulation recreates traffic that is qualitatively and quantitatively similar to the sensed real-world conditions.

I have proposed a method that can efficiently route thousands of vehicles in a manner that minimizes congestion and travel time. In my method, for the first time, the traffic routing system takes into account the effect of its own routes have on the future traffic conditions. In realistic metropolitan-scale simulations, my method achieves a speedup of over 2x over shortest path routing.

Finally I have proposed an architecture, design, and prototype implementation of a system that implements my routing proposal as a client-server system. The prototype server can handle 1000 near simultaneous routing requests in under 15 seconds, with an average response time of under a second. In the system, each vehicle plans its own route; the routes are then shared and used to help estimate the future traffic conditions. The mobile client features a novel use of destination prediction for ease-of-use and a UI similar to existing navigation systems, with a local map of the road network, highlighted route, and turn-by-turn directions.

6.2 Limitations

For the chapters above, I have discussed limitations specific to each piece of work individually. Here, I will address some limitations to my approach as a whole. My work focuses on creating simulation models that can be used for effective communication, analysis, and optimization of the traffic system for congestion control. To achieve this objective, the models need to accurately model real-world dynamics and conditions, which means much more needs to be known about the traffic system and its state. But data on this scale is generally not widely available. We may have GIS descriptions of road networks, but we don't know how many lanes a road has, how fast traffic is moving on it at any given time, what kind of traffic light is controlling the in and out flows from the road, what the light cycle is, etc. We have sensors on some major highways and mobile devices with sensing are becoming more common, but the data is noisy and we still lack the broader picture. Successful cyber-physical traffic systems in the future will likely need a better quality and greater quantity of data, both in the form of survey data and real-time sensing.

An important aspect in modeling is validation – comparing the simulation with reality. However, for traffic systems, we rarely have a comprehensive ground truth. For example, how can a simulation of a particular city be verified? The daily number of trips, or their origins and destinations, is unknown, as are the resulting flows and velocities throughout the road network. Applications such as routing are even more challenging: validation requires having actual vehicles follow specified routes: for large numbers of vehicles, this becomes quite impractical. All we can do is validate the work with the ground truth data that is available or that can be estimated and with simulated scenarios in place of real-world experiments.

In regard to optimization techniques, this dissertation only considered routing. Another important control in the system is the traffic light behavior, which could be networked for coordinated control and driven by local sensing and estimates of the overall traffic conditions. Work along these lines could be integrated with my approach to routing and traffic management.

6.3 Future Work

Certainly, this work is only a step toward creating a cyber-physical traffic system, the creation of which will be a vast endeavor. I have suggested areas of future work in the previous chapters, but here I would like discuss broader areas of future work that apply to my research as a whole.

While some of my work has dealt with sensing data from mobile devices, specifically the routing work, this is an area that deserves more attention. Unfortunately, these data sources are not publicly available generally. They are either owned by taxi fleets that generated them or by large companies, who have harvested the data via their mobile applications. Mobile data has the potential to be used to refine and describe our road network models, from geometric and topological information to the cycles of traffic light systems, and to determine the regular patterns of traffic flow and to detect when anomalies have arisen.

Another area of research that would complement my work would be to create a detailed ground truth dataset for traffic throughout a city. Such a dataset could be used to validate numerous other systems, such as state estimation from sensing, traffic prediction from conditions, and optimization potential from routing and traffic management. Two potential avenues exist to create such a data set. One, which was used in Chapter 5 to validate my routing work, is to fuse population and employment survey data with other data and models in order to estimate what trips are made during a typical day. Such a methodology is possible today, but the result may not be as accurate or as detailed as desired. A second potential methodology would be to use persistent sensing of the entire area, such as high altitude cameras, to track every vehicle and record trips. However, this methodology has technical hurdles, is expensive, and has serious privacy concerns.

Additionally, while the technology is still emerging, autonomous vehicles (AVs) have the potential to revolutionize modern traffic systems. For this possibility to happen, however, significant computational infrastructure would need to exist. For example, AVs would require digital maps of the road network in order to both navigate locally and route themselves to their destinations, maps similar as those that I proposed in Chapter 2, but with less margin for error. AVs could also contribute enormously to overall traffic system management by reporting their locations and the surrounding traffic conditions, allowing for much more accurate traffic state estimation. These vehicles would also be easier to manage: robot drivers would more predictably follow their assigned plans, improving the efficacy of coordinated routing such as I discussed in Chapter 5.

Finally, for those of us drivers who do not turn over the wheel to a robot, there are additional possibilities for improving the driving experience and the traffic systems efficiency by modeling and sensing the driving habits and patterns of individuals. A traffic simulation of a city would be far more accurate, for example, if it considered what percentage of the drivers were aggressive and likely to speed and what percentage might go below the speed limit or readily allow other vehicles to merge onto the roadway. This would have potential benefits for routing as well: a personalized driving model could more accurately predict travel time for a user. However, the trade-off that such a system would demand could be decreased privacy for the users.

Bringing the benefits of information technology to traffic systems will be a long and difficult problem, but one that will continue paying dividends into the future, as it is very likely that traffic systems, in one form or another, will be a part of human societies as long as people need to get from point A to point B. The challenges of traffic systems require novel algorithms to handle real-world dynamics, noisy sensing, human actors, and difficult trade offs between privacy and efficiency – all of which promise exciting research opportunities.

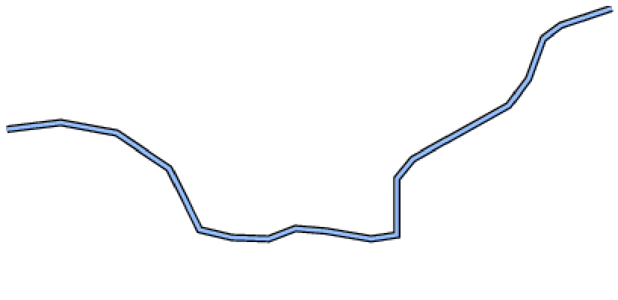
APPENDIX

A.1 Road Network Geometry

Most digital maps use *polylines* — C^0 series of line segments — to represent road shapes. However, real-world roads are curved, and these polylines visibly deviate from the real shape of the road and give the road an angular shape. Furthermore, linear segments generally lead to visible artifacts in the motion of vehicles along these roads: the sudden change in direction between segments produces large instantaneous changes in vehicle orientation, in violation of car kinematics. Figure A.1(a) shows a polyline-based road imported from GIS data.

Given the true curve of the underlying road, one could simply refine the approximation of the underlying curve by adding more straight line segments. This can give acceptable results in certain cases, although it leads to a proliferation of data points. However, barring extreme levels of refinement, the C^0 nature of the representation is still observable, and it should also be noted that there is no clear method by which to use polylines in three dimensions to describe vehicle motion with consistent orientation. However, I have only the polylines in the input GIS data set to work with and lack information needed for such refinement.

Numerous techniques have been proposed for fitting curves to polyline road data and for the refinement and smoothing of these curves [McMaster, 1987, Ware et al., 2003, Burghardt, 2005, Harrower and Bloch, 2006]. These methods are useful for visual description, but can be computational obstacles for simulation. For example, splines are frequently costly to compute with. B-splines have no general closed form for arc-length and require numerical quadrature to evaluate. Traffic simulation techniques frequently represent vehicles' positions along roads parametrically. To advance the position of vehicles along roads, information about velocity is integrated into position and translated into parametric space; to display vehicles' locations, this parametric information is translated into a point x in $\mathbb{R}^3$.


(a) Polyline road geometry.

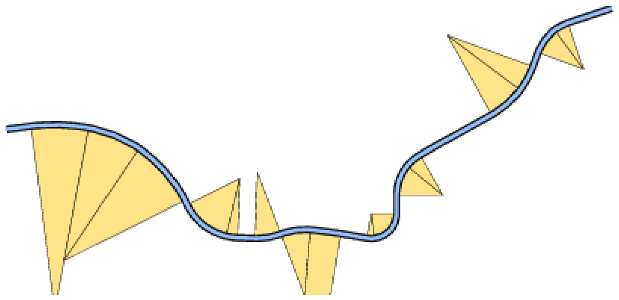

(b) An *arc road* derived from the above polyline. The orange arcs show the center and radius of each arc used to give the road its smooth appearance.

Figure A.1: A polyline and derived arc road

This process can occur multiple times per vehicle per simulation step, and when it requires quadrature, its expense can dominate runtimes.

As an alternative, we can use *arc roads*, which consist of alternating straight line segments and circular arcs. This representation has numerous advantages over polylines:

- The curve is C^1.
- It is well-defined in three dimensions and suitable for traffic simulation — a consistent Frenet frame defining 'forward', 'up', and 'right' is available at each point.
- It is straightforward to derive from existing polyline data.
- It admits a simple and inexpensive parametrization.
- It allows efficient computation of vehicle position and orientation.
- It enables much smoother animations of vehicle motion.

Figure A.1(b) shows an exemplary arc road derived from the polyline shown in Figure A.1(a). Arc roads are particularly suitable for describing vehicle motion; the turning behavior it prescribes matches the kinematic model of the simple car [LaValle, 2006].

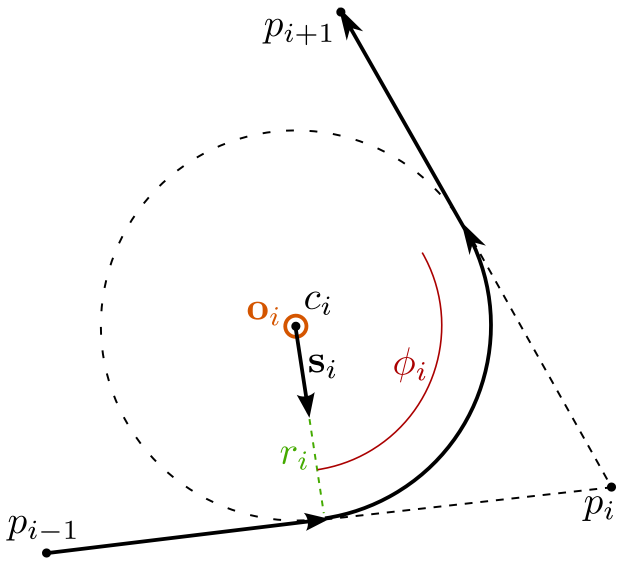


Figure A.2: The quantities defining an arc i corresponding to interior point p_i. The orientation vector $\mathbf{o}_i$ is coming out of the page.

A.2 Arc Formulation

We have an ordered sequence P of n points:

$$P := (p_0, p_1, \ldots, p_{n-2}, p_{n-1}) \tag{A.1}$$

These points define a polyline, which is not necessarily planar, with $n-1$ segments such as that in Figure A.1(a). Assume, without loss of generality, that there are no two points adjacent in the sequence that are equal, and that there are no three adjacent points that are collinear; we wish to smooth this polyline to what is shown in Figure A.1(b), which we shall refer to as P_S. We construct P_S by replacing the region around each interior point p_i of P with a circular arc and retaining the exterior points p_0 and p_{n-1}. Each of these circular arcs can be characterized by a center c_i, radius r_i, orientation $\mathbf{o}_i$, start radius direction $\mathbf{s}_i$, and angle ϕ_i; see Figure A.2. Each arc i corresponds to an interior point p_i, and we require it to be tangent to $\overline{p_{i-1}p_i}$ and $\overline{p_ip_{i+1}}$.

To help describe each arc i, we introduce the following quantities derived from the polyline P:

$$\mathbf{v}_i = p_{i+1} - p_i \tag{A.2}$$

$$L_i = |\mathbf{v}_i| \tag{A.3}$$

$$\mathbf{n}_i = \frac{\mathbf{v}_i}{L_i} = \frac{\mathbf{v}_i}{|\mathbf{v}_i|} \tag{A.4}$$

These are: vectors from point p_{i+1} to point p_i (Eq. (A.2)), the vector lengths (Eq. (A.3)), and associated unit vectors (Eq. (A.4)), respectively. We will also refer to $-\mathbf{n}_{i-1} = \frac{p_{i-1}-p_i}{|p_{i-1}-p_i|}$ and to the normal of the plane containing the circle,

$$\mathbf{o}_i = -\mathbf{n}_{i-1} \times \mathbf{n}_i \tag{A.5}$$

At certain times, it is useful to construct a matrix $\mathbf{F}_i$ that describes the axes defined by $\mathbf{n}_i$, $\mathbf{s}_i$, and $\mathbf{o}_i$:

$$\mathbf{F}_i = \begin{bmatrix} \mathbf{n}_i & \mathbf{s}_i & \mathbf{o}_i \end{bmatrix} \tag{A.6}$$

Parametrization Arc roads admit straightforward parametrizations $P_s(t) = \mathbf{x}$ because their lengths are simple to compute. They are simply the length of the original polyline P adjusted by the difference between each arc and the 'corner' of the polyline it replaces. Just as with a polyline, parametrization operations can be accelerated by storing the cumulative length of each segment and arc. Binary search can then be used to find the relevant length for a given t. This is considerably less compute-intensive than performing quadrature to determine lengths, as is necessary for many spline representation.

Speed limit estimation Where input GIS data lacks detail about speed limits, arc roads can be used to estimate speed limits:

$$v_{\max} = \sqrt{g\mu_s r} \tag{A.7}$$

Here g is acceleration due to gravity, μ_s the coefficient of static friction, and r the radius (of a given arc) — $v_{\max}$ is the highest velocity achievable on the arc without slipping; safe speed limits should be proportional to this value to meet the road safety requirements.

A.3 Fitting arc roads to polylines

Given an arbitrary polyline P, it is desirable to automatically select the r_i to complete the definition of a C^1 smoothed polyline P_S. A reasonable goal is to pick the r_i such that the quantity

$$\min_{i\in[1,n-2]} r_i \tag{A.8}$$

is *maximal* over all valid configurations of r_i; this helps minimize the 'sharpness' of each corner.

We have developed a recursive algorithm for selecting the r_i for each arc given a polyline P that satisfies Equation (A.8). Briefly, we iterate over all of the segments $\overline{p_i p_{i+1}}$, $i \in [0, n-2]$ and consider how *large* a radius it is possible to assign to the arcs i, $i+1$ at either end of the segment i; we take the *smallest* such radius over all considered segments and assign it to the associated arcs. This process is repeated recursively until each interior point has been assigned a radius value.

In the interest of space, we have omitted the details of the algorithm; a complete description and proof of its optimality in satisfying Equation A.8 is given in [Sewall, 2010].

A.4 Offset polylines

It is natural to consider roads as 2-dimensional surfaces in 3-dimensional space. The arc road formulation above describes 1-dimensional curves — we can define such a 2-dimensional surface by 'offsetting' the curve in a given direction. That is, we consider a new curve P'_S is 'offset' from P_S to one side by a signed distance d; see Fig. A.3. P'_S that has the property that at every point, the nearest point on P_S is exactly distance d away. We have used the convention that $d > 0$ refers to a 'right' offset (the lower blue line in Fig. A.3) and $d < 0$ to a 'left' offset (the upper blue line in the same figure). The new arcs i corresponding to P'_S

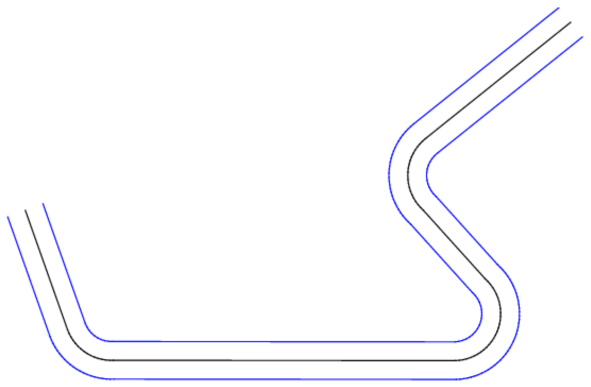


Figure A.3: A 'fattened' arc road; the original arc road P_S as computed above is drawn in black. The blue lines represent the same shape offset to either side by an equal distance.

(with signed offset d) can be derived from P_S by replacing each r_i with $r_i + d$.

New endpoints p'_0 and p'_{n-1} must be established for this line. A reasonable definition is use the plane of the first and last arcs to choose perpendiculars suitable for placing these offset endpoints, resulting in,

$$p'_0 = p_0 + d(\mathbf{n}_0 \times \mathbf{o}_1) \tag{A.9}$$

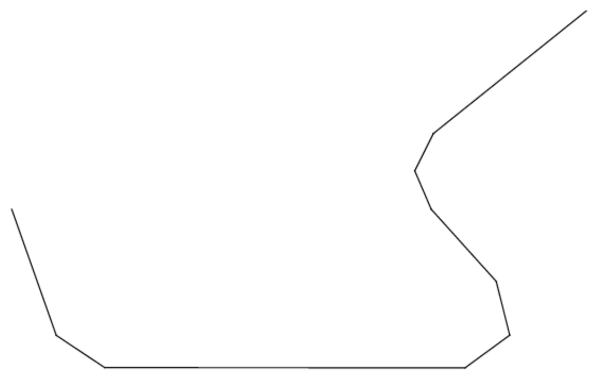
$$p'_{n-1} = p_{n-1} + d(\mathbf{n}_{n-2} \times \mathbf{o}_{n-1} \tag{A.10}$$

A.5 Discrete approximations of arc roads

To visually depict an arc road, I may wish to compute a discrete representation.

A.5.1 Polylines

One way to do this is by approximating the shape by a series of segments, a new polyline P^*. See Figure A.4(a). Each arc i must be approximated with a sequence Γ_i of $q_i \in \mathbb{Z}_{>1}$


(a) P^*: A polyline approximation of an arc road P_S

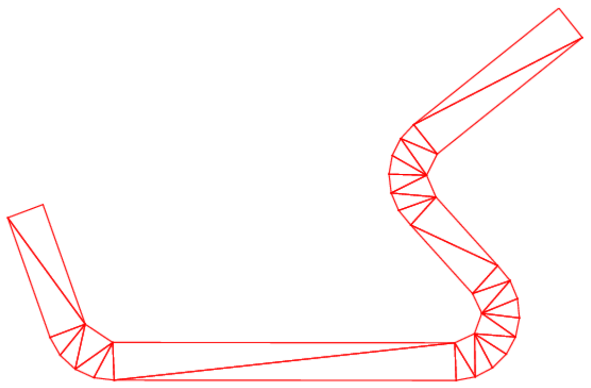

(b) A triangle mesh approximation of a 'fattened' arc road P_S

Figure A.4: Discrete approximations of arc roads

connected points. Then the sequence of points in P^* is simply:

$$P_S^* = \left(p_0, \Gamma_1^0, \ldots, \Gamma_1^{q_1-1}, \ldots, \Gamma_{n-2}^0, \ldots, \Gamma_{n-2}^{q_{n-2}-1}, p_{n-1}\right) \tag{A.11}$$

Each Γ_i is generated by rotating and scaling the frame $\mathbf{F}_i$ (from Equation (A.6)) of each arc incrementally and translating by the center c_i,

$$\Gamma_i^j = c_i + r_i \mathbf{F}_i \left[\cos t^j, \sin t^j, 0\right]^{\mathrm{T}}, \; j \in \mathbb{Z}[0, q_i - 1] \tag{A.12}$$

Here the t^j are the elements of a sequence $[0, \phi_i/(q_i - 1), 2\phi_i/(q_i - 1), \ldots, \phi_i]$ of length q_i.

A.5.2 Triangle meshes

A surface representation of an arc road can be easily computed from a pair of offset arc roads (computed as in Sec. A.4). Given an arc road P_S and two polylines offset from P_S, order the polylines by offset so that I have a 'left' polyline P_S^l and a 'right' polyline P_S^r. Now I can use any constrained triangulation technique to compute a planar triangle mesh with P_S^l and P_S^r as the boundaries; see Figure A.4(b).

A.6 Road mesh geometry

In Sec. A.5, I show how to construct a mesh for a pair of given offsets. To create geometry for roads, I need only compute the width of each road — this is easily done by examining the position of each member lane of a given road: the position of the left edge of leftmost member lane forms the left offset, and the position of the right edge of the rightmost member lane forms the right offset.

Our data structure allows for the member lanes of a road to vary along its length; in the case, I simply extract a mesh for each parametric interval along the road that has a constant arrangement of member lanes.

BIBLIOGRAPHY

[Amin et al., 2008] Amin, S., Andrews, S., Apte, S., Arnold, J., Ban, J., Benko, M., Bayen, R. M., Chiou, B., Claudel, C., Claudel, C., Dodson, T., Elhamshary, O., Flens-batina, C., Gruteser, M., carlos Herrera, J., Herring, R., Hoh, B., Jacobson, Q., Iwuchukwu, T., Lew, J., Litrico, X., Luddington, L., Margulici, J., Mortazavi, A., Pan, X., Rabbani, T., Racine, T., Sherlock-thomas, E., Sutter, D., and Tinka, A. (2008). Mobile century using gps mobile phones as traffic sensors: A field experiment. 87

[Ashbrook and Starner, 2003] Ashbrook, D. and Starner, T. (2003). Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286. 87, 90

[Aw and Rascle, 2000] Aw, A. and Rascle, M. (2000). Resurrection of" second order" models of traffic flow. *SIAM journal on applied mathematics*, pages 916–938. 10, 42, 45, 47, 50, 111

[Balaban, 1995] Balaban, I. (1995). An optimal algorithm for finding segments intersections. In *Proceedings of the eleventh annual symposium on Computational geometry*, page 219. ACM. 33

[Brakatsoulas et al., 2005] Brakatsoulas, S., Pfoser, D., Salas, R., and Wenk, C. (2005). On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864. VLDB Endowment. 65, 85

[Burghardt, 2005] Burghardt, D. (2005). Controlled line smoothing by snakes. *GeoInformatica*, 9(3):237–252. 115

[Burke et al., 2006] Burke, J. A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. (2006). Participatory sensing. *Proc. of ACM WSW at SenSys 2006*. 85

[Chabini and Lan, 2010] Chabini, I. and Lan, S. (2010). Adaptations of the A* Algorithm for the Computation of Fastest Paths in Deterministic Discrete-Time Dynamic Networks. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):60–74. 67, 68, 72, 73

[Chakirov and Fourie, 2014] Chakirov, A. and Fourie, P. (2014). Enriched sioux falls scenario with dynamic and disaggregate demand. Working paper, Future Cities Laboratory, Singapore - ETH Centre (SEC), Singapore. 100

[Chen et al., 2008] Chen, G., Esch, G., Wonka, P., Mueller, P., and Zhang, E. (2008). Interactive procedural street modeling. In *SIGGRAPH 2008*, New York, NY, USA. ACM. 17

[Church and ReVelle, 1978] Church, R. and ReVelle, C. (1978). Modelling an oversaturated intersection. *Transportation Research*, 12(3):185–189. 95

[Cohen et al., 2001] Cohen, E., Riesenfeld, R., and Elber, G. (2001). *Geometric modeling with splines: an introduction*. AK Peters Ltd. 18

[Cremer et al., 1997] Cremer, J., Kearney, J., and Willemsen, P. (1997). Directable behavior models for virtual driving scenarios. *Trans. Soc. Comput. Simul. Int.*, 14(2):87–96. 38

[Cremer, 1991] Cremer, M. (1991). Flow variables: estimation. *Concise encyclopedia of traffic and transportation systems*, pages 143–148. 40

[Donikian et al., 1999] Donikian, S., Moreau, G., and Thomas, G. (1999). Multimodal driving simulation in realistic urban environments. *Progress in System and Robot Analysis and Control Design (LNCIS)*, 243:321–332. 38

[Dresner and Stone, 2004] Dresner, K. and Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, pages 530–537, Washington, DC, USA. IEEE Computer Society. 86, 87

[Evensen, 2003] Evensen, G. (2003). The ensemble kalman filter: theoretical formulation and practical implementation. 46

[Farin, 1996] Farin, G. (1996). *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Academic Press, Inc. Orlando, FL, USA. 18

[Florian et al., 2008] Florian, M., Mahut, M., and Tremblay, N. (2008). Application of a simulation-based dynamic traffic assignment model. *European Journal of Operational Research*, 189(3):1381–1392. 67, 87

[Fortier et al., 1999] Fortier, A., Ziou, D., Armenakis, C., and Wang, S. (1999). Survey of work on road extraction in aerial and satellite images. *Center for Topographic Information Geomatics, Ontario, Canada. Technical Report*, 241. 17

[Galin et al., 2010] Galin, E., Peytavie, A., Maréchal, N., and Guérin, E. (2010). Procedural Generation of Roads. In *Computer Graphics Forum*, volume 29, pages 429–438. Wiley Online Library. (Proceedings of Eurographics 2010). 17

[Greenshields et al., 1935] Greenshields, B. et al. (1935). A study of traffic capacity. In *Highway Research Board Proceedings*, volume 14, pages 448–477. 68, 70, 91

[Harrower and Bloch, 2006] Harrower, M. and Bloch, M. (2006). MapShaper. org: A map generalization web service. *Computer Graphics and Applications, IEEE*, 26(4):22–27. 115

[Hegyi et al., 2007] Hegyi, A., Mihaylova, L., Boel, R., and Lendek, Z. (2007). Parallelized particle filtering for freeway traffic state tracking. In *European Control Conference*. 40

[Helbing, 2001] Helbing, D. (2001). Traffic and related self-driven many-particle systems. *Reviews of modern physics*, 73(4):1067. 41

[Horvitz et al., 2005] Horvitz, E., Apacible, J., Sarin, R., and Liao, L. (2005). Prediction, Expectation, and Surprise: Methods, Designs, and Study of a Deployed Traffic Forecasting Service. *Conf. on Uncertainty in Artificial Intelligence*. 65, 85

[Houtekamer and Mitchell, 2001] Houtekamer, P. and Mitchell, H. (2001). A sequential ensemble kalman filter for atmospheric data assimilation. *Monthly Weather Review*, 129(2):123–137. 41

[Hua and Pei, 2010] Hua, M. and Pei, J. (2010). Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 347–358. ACM. 67

[Hull et al., 2006] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S. (2006). Cartel: A distributed mobile sensor computing system. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 125–138. ACM. 87

[Jacquet et al., 2005] Jacquet, D., Canudas De Wit, C., and Koenig, D. (2005). Traffic control and monitoring with a macroscopic model in the presence of strong congestion waves. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2164–2169. IEEE. 41

[Jacquet et al., 2006] Jacquet, D., Krstic, M., and de Wit, C. (2006). Optimal control of scalar one-dimensional conservation laws. In *American Control Conference, 2006*, pages 6–pp. IEEE. 41

[Jariyasunant et al., 2010] Jariyasunant, W., Kerkez, S., and Glaser, B. (2010). Mobile transit trip planning with real-time data. In *Transportation Research Board 89th Annual Meeting*. 87

[Jia et al., 2001] Jia, Z., Chen, C., Coifman, B., and Varaiya, P. (2001). The pems algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 536–541. IEEE. 58

[Krajzewicz et al., 2012] Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138. 96

[Laramee et al., 2004] Laramee, R. S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F. H., and Weiskopf, D. (2004). The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203221. 42

[LaValle, 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press. 116

[Lebacque et al., 2007] Lebacque, J.-P., Mammar, S., and Haj-Salem, H. (2007). The aw-rascle and zhang's model: Vacuum problems, existence and regularity of the solutions of the riemann problem. *Transportation Research Part B*, (41):710–721. 45

[LeBlanc, 1975] LeBlanc, L. J. (1975). An algorithm for the discrete network design problem. *Transportation Science*, 9(3):183–199. 101

[LeBlanc et al., 1975] LeBlanc, L. J., Morlok, E. K., and Pierskalla, W. P. (1975). An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, 9(5):309–318. 101

[Liao et al., 2007] Liao, L., Patterson, D. J., Fox, D., and Kautz, H. (2007). Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311 – 331. 87, 90

[Lighthill and Whitham, 1955] Lighthill, M. and Whitham, G. (1955). On kinematic waves. ii. a theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 229(1178):317–345. 41

[Lim et al., 2009] Lim, S., Balakrishnan, H., Gifford, D., Madden, S., and Rus, D. (2009). Stochastic Motion Planning and Applications to Traffic. *Algorithmic Foundation of Robotics VIII*, pages 483–500. 67, 68, 72

[McMaster, 1987] McMaster, R. (1987). Automated line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 24(2):74–111. 115

[Mena, 2003] Mena, J. (2003). State of the art on automatic road extraction for gis update: a novel classification. *Pattern Recognition Letters*, 24:3037–3058. 17

[Min et al., 2007] Min, W., Wynter, L., and Amemiya, Y. (2007). Road traffic prediction with spatio-temporal correlations. In *Proceedings of the Sixth Triennial Symposium on Transportation Analysis, Phuket Island, Thailand (June 2007)*. 65, 85

[MITSIM, 2011] MITSIM (2011). MITSIM. MIT Intelligent Transportation Systems. 38

[NAVTEQ, 2010] NAVTEQ (2010). NAVTEQ. 17

[NGSIM, 2013] NGSIM (2013). Next Generation SIMulation. 58

[Nieuwenhuisen et al., 2004] Nieuwenhuisen, D., Kamphuis, A., Mooijekind, M., and Overmars, M. (2004). Automatic construction of roadmaps for path planning in games. In *Proceedings of the International Conferefence on Computer Games, Artificial Intelligence, Design and Education*, pages 285–292. 19

[Nikolova, 2010] Nikolova, E. (2010). High-Performance Heuristics for Optimization in Stochastic Traffic Engineering Problems. *Large-Scale Scientific Computing*, pages 352–360. 67

[Nikolova et al., 2006a] Nikolova, E., Brand, M., and Karger, D. (2006a). Optimal route planning under uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling*. 65, 85

[Nikolova et al., 2006b] Nikolova, E., Kelner, J., Brand, M., and Mitzenmacher, M. (2006b). Stochastic shortest paths via quasi-convex maximization. *Algorithms–ESA 2006*, pages 552–563. 67

[OpenStreetMap community, 2010] OpenStreetMap community (2010). OpenStreetMap. 17

[Papageorgiou et al., 1990] Papageorgiou, M., Blosseville, J., and Hadj-Salem, H. (1990). Modelling and real-time control of traffic flow on the southern part of boulevard périphérique in paris: Part i: Modelling. *Transportation Research Part A: General*, 24(5):345–359. 42

[Park et al., 2002] Park, J., Saleh, R., and Yeu, Y. (2002). Comprehensive survey of extraction techniques of linear features from remote sensing imagery for updating road spatial databases. In *ASPRS-ACSM Annual Conference and FIG XXII Congress*. 17

[Pausch et al., 1992] Pausch, R., Crea, T., and Conway, M. (1992). A literature survey for virtual environments - military flight simulator visual systems and simulator sickness. *Presence: Teleoperators and Virtual Environments*, 1(3):344–363. 38

[Peeta and Ziliaskopoulos, 2001] Peeta, S. and Ziliaskopoulos, A. (2001). Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and Spatial Economics*, 1(3):233–265. 67, 87

[Piegl and Tiller, 1997] Piegl, L. and Tiller, W. (1997). *The NURBS book*. Springer Verlag. 18

[Prodecural Inc., 2011] Prodecural Inc. (2011). CityEngine Manual. xv, 18

[Rice and Van Zwet, 2004] Rice, J. and Van Zwet, E. (2004). A simple and effective method for predicting travel times on freeways. *Intelligent Transportation Systems, IEEE Transactions on*, 5(3):200–207. 61

[Richards, 1956] Richards, P. (1956). Shock waves on the highway. *Operations research*, pages 42–51. 41

[Sau et al., 2007] Sau, J., El Faouzi, N., Aissa, A., and de Mouzon, O. (2007). Particle filter-based real-time estimation and prediction of traffic conditions. *Recent advances in stochastic modeling and data analysis: Chania, Greece, 29 May-1 June 2007*, page 406. 41

[Schrank et al., 2012] Schrank, D., Eisele, B., and Lomax, T. (2012). Ttis 2012 urban mobility report. 1

[Sewall, 2010] Sewall, J. (2010). *Efficient, Scalable Traffic and Compressible Fluid Simulations Using Hyperbolic Models*. PhD thesis, University of North Carolina at Chapel Hill. 52, 119

[Sewall et al., 2011a] Sewall, J., van den Berg, J., Lin, M., and Manocha, D. (2011a). Virtualized traffic: Reconstructing traffic flows from discrete spatiotemporal data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(1):26–37. 42, 43, 44, 61, 62, 63

[Sewall et al., 2011b] Sewall, J., Wilkie, D., and Lin, M. C. (2011b). Interactive hybrid simulation of large-scale traffic. *ACM Transaction on Graphics (Proceedings of SIGGRAPH Asia)*, 30(6). 42, 43

[Sewall et al., 2010a] Sewall, J., Wilkie, D., Merrell, P., and Lin, M. (2010a). Continuum Traffic Simulation. In *Computer Graphics Forum*, volume 29, pages 439–448. Blackwell Publishing. (Proceedings of Eurographics 2010). xv, 15, 34

[Sewall et al., 2010b] Sewall, J., Wilkie, D., Merrell, P., and Lin, M. (2010b). Continuum traffic simulation. In *Computer Graphics Forum*, volume 29, pages 439–448. Wiley Online Library. 42, 45

[Sewall et al., 2010c] Sewall, J., Wilkie, D., Merrell, P., and Lin, M. C. (2010c). Continuum traffic simulation. In *Computer Graphics Forum*, volume 29, pages 439–448. Wiley Online Library. 94

[Siebel and Mauser, 2005] Siebel, F. and Mauser, W. (2005). On the fundamental diagram of traffic flow. *Arxiv preprint cond-mat/0503290*. 68, 70

[SUMO, 2009] SUMO (2009). SUMO — Simulation of Urban MObility. 38

[Suwansirikul et al., 1987] Suwansirikul, C., Friesz, T. L., and Tobin, R. L. (1987). Equilibrium decomposed optimization: a heuristic for the continuous equilibrium network design problem. *Transportation science*, 21(4):254–263. 101

[Thomas and Donikian, 2000] Thomas, G. and Donikian, S. (2000). Modelling virtual cities dedicated to behavioural animation. *CGF*, 19(3). 38

[Treiber et al., 2000] Treiber, M., Hennecke, A., and Helbing, D. (2000). Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824. 15, 55

[U.S. Census Bureau, 2010] U.S. Census Bureau (2010). TIGER/Line®. 17

[van den Berg and Overmars, 2007] van den Berg, J. and Overmars, M. (2007). Kinodynamic motion planning on roadmaps in dynamic environments. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4253–4258, San Diego, CA. 19

[van den Berg et al., 2009] van den Berg, J., Sewall, J., Lin, M., and Manocha, D. (2009). Virtualized traffic: Reconstructing traffic flows from discrete spatio-temporal data. In *Virtual Reality Conference, 2009. VR 2009. IEEE*, pages 183–190. IEEE. 38, 42, 43, 44, 61

[Wang et al., 2002] Wang, H., Kearney, J., and Atkinson, K. (2002). Arc-length parameterized spline curves for real-time simulation. In *Proc. 5th International Conference on Curves and Surfaces*, pages 387–396. Citeseer. 19

[Wang et al., 2005] Wang, H., Kearney, J., Cremer, J., and Willemsen, P. (2005). Steering behaviors for autonomous vehicles in virtual environments. In *Proc. IEEE Virtual Reality Conf.*, pages 155–162. 38

[Wang and Papageorgiou, 2005] Wang, Y. and Papageorgiou, M. (2005). Real-time freeway traffic state estimation based on extended kalman filter: a general approach. *Transportation Research Part B: Methodological*, 39(2):141–167. 41, 42

[Ware et al., 2003] Ware, J., Jones, C., and Thomas, N. (2003). Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science*, 17(8):743–769. 115

[Wilkie et al., 2012] Wilkie, D., Sewall, J., and Lin, M. (2012). Transforming gis data into functional road models for large-scale traffic simulation. *Visualization and Computer Graphics, IEEE Transactions on*, 18(6). 42, 44

[Wilkie et al., 2011] Wilkie, D., van den Berg, J. P., Lin, M. C., and Manocha, D. (2011). Self-aware traffic route planning. In *AAAI*. xviii, 96, 106

[Willemsen et al., 2006] Willemsen, P., Kearney, J., and Wang, H. (2006). Ribbon networks for modeling navigable paths of autonomous agents in virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, pages 331–342. 18

[Work et al., 2010] Work, D., Blandin, S., Tossavainen, O., Piccoli, B., and Bayen, A. (2010). A traffic model for velocity data assimilation. *Applied Mathematics Research eXpress*. 41, 68, 70, 91, 100

[Work et al., 2008] Work, D., Tossavainen, O., Blandin, S., Bayen, A., Iwuchukwu, T., and Tracton, K. (2008). An ensemble kalman filtering approach to highway traffic estimation using gps enabled mobile devices. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 5062–5068. IEEE. 41

[Zhang, 2002] Zhang, H. (2002). A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B: Methodological*, 36(3):275–290. 10, 42, 45, 47, 50, 111