UNIVERSITY OF AUCKLAND

DOCTORAL THESIS

Improving the prediction of bus arrival using real-time network state

---

Tom Mitchell Elliott

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Statistics, University of Auckland

April 2020

UNIVERSITY OF AUCKLAND

Abstract

Faculty of Science

Department of Statistics

Doctor of Philosophy

Improving the prediction of bus arrival using real-time network state

by Tom Mitchell Elliott

The real-time prediction of bus arrival time has been a central focus of real-time transit information research over the past few decades. Much of this research has shown that the most important predictors of bus arrival time are travel time between and dwell time at bus stops. Despite this, estimated times of arrival available in Auckland, New Zealand, make no account of real-time traffic state information. As road networks are dynamic and congestion can change quickly, we present a generalised prediction procedure that uses buses to estimate traffic conditions, which are in turn used in the prediction of arrival times for all other buses travelling along the same roads, irrespective of the route they are servicing. We construct a road network from data in the General Transit Feed Specification format, allowing us to estimate real-time traffic conditions along physical roads. We use a particle filter to estimate vehicle states and road speeds, and a Kalman filter to update the road network state, together allowing us to predict bus arrival times that account for real-time traffic conditions. We use a simplified, discrete arrival time cumulative density function to make point and interval estimates, as well as estimate the probabilities of events pertinent to journey planning. Throughout, we assess the real-time feasibility of the application and show that our method, despite being computationally complex, can provide arrival time estimates for all active vehicles in 6–10 seconds.

In loving memory of Nana

Ruvae Jane Tupaea Cottrell

17/2/1932 – 3/3/2020

Acknowledgements

These last five years would not have been possible without the help of many. To all who have helped in any way, you have my gratitude.

First off, I thank my supervisor, Thomas Lumley, for proposing the entire context of this thesis, and for five years of being available (most of the time; there were one or two overseas conferences) for my usually unannounced drop-ins. I also thank Brendon Brewer, my co-supervisor, for early help getting started with C++, and for offering his assistance should ever I need it.

To the entire statistics department at the University of Auckland: thank you for being so welcoming and supportive for the past five years (and more), in particular to Chris Wild for your incredible support of both my current work and future career opportunities. Thank you to the university itself for allowing me to pursue doctoral study and providing the much needed financial support of the Doctoral Scholarship, and to the Center for e-Research for supplying the necessary computing resources for my research.

To my friends both within and without the department, thank you for providing humour, social escapes, and those essential reminders that "we all feel the same way, it's normal!". And the wine. Thank you for the wine.

To my delightful partner Mark: thank you for supporting a broke student as he frantically condenses 4 years' work into 200 pages, for your emotional support over recent months, and the occasional nudge to get things done.

And lastly, to the most loving and supportive parents. These past two years have been amongst some of the hardest. Yet you were there for me, as always, whenever I needed a ride home, a roof over my head, or a home-cooked meal. My success is a testament to you both. Thank you.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN artificial neural network.

API application programming interface.

AVL automatic vehicle location.

CDF cumulative density function.

DMS digital message sign.

ETA estimated time of arrival.

GPS global positioning system.

GTFS General Transit Feed Specification.

MAE mean absolute error.

MAPE mean absolute percentage error.

PICP prediction interval coverage probability.

RBM recursive Bayesian model.

RMSE root mean square error.

RTI real-time information.

SVM support vector machine.

List of Symbols

Vehicle state model		
t_k	time of the k^{th} observation	Unix timestamp
Δ_k	time since the last observation, $t_k - t_{k-1}$	seconds
$\boldsymbol{y}_k$	vehicle observation (at time t_k)	
ϕ_k	vehicle latitude	degrees
λ_k	vehicle longitude	degrees
A_m, D_m	observed arrival, departure time at stop m	Unix timestamp
$\tilde{D}_m$	observed dwell time at stop m	seconds
$\boldsymbol{x}_k$	vehicle state (at time t_k)	
x_k	vehicle distance into trip	m
$\dot{x}_k$	vehicle speed	m/s
N	number of particles per vehicle	
N_{eff}	effective sample size	
$w_k^{(i)}$	weight of particle i given data up to time t_k	
δ	Dirac delta function (refer to appendix B.1)	
$\tilde{\boldsymbol{y}}_k^{(i)}$	Computed coordinates of particle i at time t_k	
f	vehicle transition function	
h	vehicle measurement function	
σ^2	vehicle system noise	m/s[1]
ε^2	measurement or GPS error	meters
ξ^2	Trip update arrival/departure error	seconds
$d(a, b)$	distance between points a and b	meters
$\mathcal{P}$	shape path of the vehicle	

[1] Except the acceleration models (A3, B3) which has units m/s^2.

M	number of stops in the vehicle's active trip	
α_m	arrival time at stop m	Unix timestamp
τ_m	mean dwell time at stop m	seconds
ω_m^2	variance of dwell time at stop m	seconds
π_m	probability of stopping at stop m	
γ	minimum dwell time	seconds
d_m	total dwell time at stop m	seconds
L	number of segments in the vehicle's active trip	
$\mathcal{D}_\ell$	distance into trip of start of segment ℓ	meters
$\mathcal{L}_\ell$	length of segment ℓ	meters
T_ℓ, T'_ℓ	start and end times along road segment ℓ	Unix timestamp
B_ℓ	vehicle travel time along segment ℓ	seconds
ν_ℓ	mean wait time at intersection ℓ	seconds
ρ_ℓ	probability of stopping at intersection ℓ	
w_ℓ	total wait time at intersection ℓ	seconds

Network state model

$\beta_{\ell,c}$	road segment ℓ state at time t_c	m/s
$\mathbb{V}_\ell$	maximum speed along road segment ℓ	m/s
$\mathbf{F}_c$	road segment transition function at time t_c	
Δ_c	time since last road state update, $t_c - t_{c-1}$	seconds
q_ℓ	road segment ℓ system noise	m/s^2
$\boldsymbol{\psi}_c$	between-vehicle variance, segment ℓ, time t_c	m/s
$e_{\ell,c}^m$	measurement error for bus m, segment ℓ, time t_c	m/s
$\hat{\beta}_{\ell,c\|c}$, $\mathbf{P}_{\ell,c\|c}$	estimated state (mean, variance), segment ℓ, time t_c, given observations up to time t_{c-1}	m/s

Arrival time prediction

$\mathcal{T}_k$	trip state at time t_k	
$\hat{p}_k$	progress along trip's current segment, time t_k	
S_m	scheduled arrival time at stop m	Unix timestamp
$\hat{\alpha}_m$	predicted arrival time at stop m	seconds

$\check{\alpha}_m$	predicted arrival time at stop m	minutes
$\hat{A}_q$	$100q$% quantile of arrival time	minutes

Statistical distributions

Bernoulli (π)	Bernoulli distribution with probability of success π
$\mathcal{E}(\lambda)$	Exponential distribution with rate λ
$\mathcal{G}(r, \theta)$	Gamma distribution with shape r and scale θ
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\mathcal{N}_T(\mu, \sigma^2, a, b)$	Truncated Normal distribution with mean μ and variance σ^2 truncated to the interval (a, b)
$\mathcal{U}(a, b)$	Uniform distribution on the interval (a, b)

Chapter 1

Introduction

Over the last three decades, technological advances have changed the way people use public transport. Before real-time information (RTI), travellers would time their arrival at bus stops based exclusively on the bus's scheduled arrival time. There was no way of knowing if the bus was on-time, running late, or indeed if it was on its way at all until it finally appeared around the corner. These days, however, travellers can check the location of a bus from their phones before leaving home. Despite the accessibility of such RTI, unforeseeable traffic situations and various bus behaviours make estimating arrival times from vehicle locations a challenging task.

Around the world, public transport providers have invested in systems to provide RTI to travellers. Research has found that travellers perceive shorter wait times compared to their actual wait times in the presence of an arrival time countdown (Schweiger, 2003). Cats and Loutos (2015) and Lu et al. (2018) further reported that passengers who use RTI have shorter actual wait times on average than those who do not. However, as we shall see, estimated times of arrival (ETAs) may be unreliable, which is likely to frustrate passengers rather than appease them. For this reason, improving the *reliability* of RTI is crucial for increasing *ridership*, the number of passengers using public transport.

One limitation of RTI is the infrastructure required to keep track of vehicles and relay information to commuters (Schweiger and Shammout, 2003). As fleet sizes increase (over 1000 vehicles in Auckland), so to do the demands of advanced traveller information systems, which has led some providers, notably Auckland Transport, to use the simplest of arrival time prediction frameworks. This simplification consequently results in less reliable systems, as I will discuss in section 1.2.

The history of RTI for public transport spans several decades, which we will now examine, focussing on the key concepts essential for reliable ETAs. Afterwards, the current state of arrival time prediction in Auckland, New Zealand, is examined to establish why—despite decades of research—there is room for significant improvement. Finally, I present the aims of this research and summarize the process of estimating the arrival time of buses.

1.1 A brief history of real-time information

For a long time, public transport services such as buses and trains had only static timetables that passengers would use to plan their journey. There was no way for passengers to know if their bus was on-time, late, or early, nor could operators track how their services were running. In the 1960s, the first vehicle tracking systems were trialled in Germany and the United States of America (Okunieff, 1997). These early systems used *beacons* (on signposts) placed along the route that the bus could detect, allowing it to provide its location to the service provider in real-time. Similar vehicle tracking technologies continued to develop over the subsequent decades; however, the focus remained on service monitoring and operational control.

The earliest uses of automatic vehicle location (AVL) technology for passenger information were digital message signs at bus stops displaying a countdown to the next bus's arrival (Schweiger, 2003). One early example of this was Transport for London's *Countdown* system (Balogh and Smith, 1993) which used the beacon (or "signpost") AVL technology to determine vehicle locations and predict arrival times at bus stops.

Other vehicle tracking methods have since emerged using a range of technologies, the most notable being the widely used global positioning system (GPS) (Zhao, 1997). One significant advantage of the GPS is that it does not require fixed infrastructure (as does the signpost technology), making it easy for transit providers to add new routes or reroute existing ones without breaking the AVL system. A disadvantage of the GPS is that, rather than receiving *route-specific* positions ("Signpost 8", for example) the observations are *map coordinates* which require an additional step to match them

to the route.[1]

Cathey and Dailey (2003) proposed a general prescription for making real-time arrival time estimates from GPS vehicle location data consisting of three components. First, a *tracker* matched observations to scheduled *trips* (time-specific instances of a route), which involved mapping GPS observations to the *route path* to calculate the bus's *trip-distance-travelled* (the distance the bus has driven along the route). Second, a *filter* estimated the underlying vehicle state (including speed) that could be updated in real-time as new observations were received. Cathey and Dailey used a Kalman filter for this step due to its superior real-time performance and previous use modelling the state of transit vehicles (Dailey et al., 2001; Wall and Dailey, 1999). In the final *prediction* component of their prescription, Cathey and Dailey used the estimated vehicle state in conjunction with travel time forecasts (estimated from historical data) to predict *time until arrival*.

Some AVL systems report the vehicle's arrival and departure times from *time points* (which are usually a subset of specific stops), providing observations of travel time between them. This type of reporting led to a range of new methods of arrival time prediction. For example, artificial neural network (ANN) and support vector machine (SVM) models were implemented to predict travel time (Cats and Loutos, 2015, 2016; Jeong and Rilett, 2005; Shalaby and Farhan, 2004; Yin et al., 2017; Yu, Lam, and Tam, 2011). Many of these were in-depth research projects with access to high-quality data, such as high-frequency polling (10 seconds or less) or video footage, providing highly accurate estimates of arrival and departure times. Despite their differences, all researchers emphasized the importance of two components to arrival time prediction: *travel time* between stops (or time points), and *dwell time* (time spent at a stop while passengers board and disembark).

The concept of travel time between time points forms the foundation of arrival time prediction, so the estimation or forecasting of travel times has been the focus of much research. Returning to Transport for London's *Countdown* system, Reinhoudt and Velastin (1997) stored real-time travel times between time points in a database. They then used a Kalman filter to update link travel times, significantly improving the accuracy of arrival time prediction. Others used historical data to estimate *time*

[1] Route matching and the issues associated with it are discussed in depth in later chapters.

until arrival given a vehicle's trip-distance-travelled and speed (Cathey and Dailey, 2003; Dailey et al., 2001; Wall and Dailey, 1999). Further improvements to prediction accuracy were obtained when real-time travel times along links were updated using a Kalman filter, which was capable of reacting to changes in traffic conditions (Shalaby and Farhan, 2004).

ANN and SVM models have also been used to predict arrival time. Yu, Yang, and Yao (2006) demonstrated the feasibility and applicability of an SVM to predict travel time based on the bus's travel time along the current segment and the travel time of the most recent bus along the next segment. The limitation was that the model was trained for only one single route and only provided an arrival time for the next time point. Yu et al. (2010) proposed an improved hybrid model using an SVM to model baseline travel time and a Kalman filter to incorporate real-time data. Once more, this was limited to single-interval prediction (one stop ahead) along a single route, but demonstrated the importance of using real-time data for arrival time prediction. The next significant result was from Yu, Lam, and Tam (2011) when they demonstrated the use of data across multiple routes to predict arrival time. In their study, Yu, Lam, and Tam used the travel times of recent buses between an automatic toll reader and a specific bus stop. While not generalised, they demonstrated the advantage of combining data across routes.

Yin et al. (2017) presented a model to predict travel time between stops using the travel times of previous buses from multiple routes, in which they chose two overlapping routes and manually identified common stops. They were able to predict travel time reasonably well, though both their SVM and ANN models failed to capture the peak period congestion. An alternative to machine learning models was proposed by Chen and Rakha (2014), who used a *particle filter*[2] to predict car travel time along roads from a database of historical data. Another unique approach by Julio, Giesen, and Lizana (2016) used the concept of *traffic shock waves* to predict travel times of buses along links based on the travel time along adjacent links. They used GPS observations to calculate vehicle trajectories over time and developed an ANN to learn patterns: as congestion builds along a segment, so too does congestion along

[2]While their particle filter is unrelated to ours, the general methodology is the same and described in section 2.4.

the segment before it, for example.

Most of the methods described above only provide predictions for travel time along the next link, so arrival times are not available for stops further down the route. Chang et al. (2010) developed a *k*-nearest neighbour algorithm trained on historical vehicle trajectories to predict travel time along multiple upcoming links for a single route at a speed of 4000 predictions per minute. However, there has been an otherwise noticeable gap in the transit literature for making long-term predictions of travel time, as noted by Moreira-Matias et al. (2015).

Since the travel time of previous buses had been shown to improve arrival time predictions, other sources of information were explored to further improve prediction accuracy. This additional information could be significant along roads with low-frequency trips or subject to fast-changing traffic conditions. Xinghao et al. (2013) and Ma et al. (2019) incorporated real-time taxi data to model traffic state. While this showed improvements, it is limited to locations with open access to taxi data.

Another source of uncertainty in arrival time comes from *dwell time*, the time taken for passengers to board and disembark at stops. Shalaby and Farhan (2004) showed that dwell times could have a substantial influence on arrival times. Along with a Kalman filter for link travel times (each link consisted of 2–8 stops), Shalaby and Farhan also implemented a Kalman filter on stop dwell times, allowing them to respond to real-time demand fluctuations which they measured using automatic passenger counters. Other work has also demonstrated the necessity of incorporating dwell times into arrival time predictions (Cats and Loutos, 2015, 2016; Jeong and Rilett, 2005).

When it comes to modelling transit vehicles and predicting arrival times, there is a lot of uncertainty in vehicle trajectories, particularly when observations are sparse. Additionally, much of this uncertainty is non-Gaussian (particularly multi-modal). Hans et al. (2015) presented a particle filter for modelling transit vehicles incorporating bus stop dwell times and traffic light phasing using data from GPS devices and automatic passenger counters. The primary advantage of the particle filter over other methods (such as the Kalman filter) is its ability to sample a wide range of plausible trajectories, which is of particular importance as uncertainty increases for later stops. The distribution can also be strongly multi-modal due to buses not necessarily

stopping at all stops.[3]

Since uncertainty is an unavoidable component of arrival time prediction, some attempts at conveying this to travellers have been explored. Mazloumi et al. (2011) assessed the use of *prediction intervals* obtained from an ANN model, focussing on the accuracy (coverage) of these intervals. A unique approach was demonstrated by Fernandes et al. (2018) that involved displaying *uncertainty graphs*—including quantile dot plots—and found that their test subjects made better decisions when they had uncertainty information.

The final component of RTI that we are interested in is *journey planning*, which involves the selection of an optimal route to get to a destination, often under one or more constraints. Horn (2004) developed a real-time routing method that could incorporate *walking* and *waiting costs* (each of these should be minimised where possible), with a focus on *demand-responsive* services. More recently, Häme and Hakula (2013a,b) proposed a *Markov decision process* which could maximise the probability of arriving at the destination on-time, accounting for walking legs and distributions for the arrival time of buses at stops. Zheng, Zhang, and Li (2016) incorporated travel time uncertainty into their method while also demonstrating the complexity of the problem: it could take 10^3 seconds to find an optimal route; however, by storing paths, they were able to reduce computations to less than a second. As an alternative to incorporating travel time uncertainty, Bérczi et al. (2017) present a dynamic routing strategy that uses any probabilistic model of arrival and departure times to maximise the probability of arriving at the destination on time.

1.2 The state of Auckland's RTI

Our case study is the public transport system of Auckland, New Zealand, operated by Auckland Transport. In the following pages, I describe some scenarios that occur within Auckland,[4] although they likely occur in other cities that use the same AVL system (described in detail in section 2.1). Additionally, many locations in Auckland lack transit infrastructure, such as bus priority lanes, which can lead to long delays during heavy congestion.

[3] I will demonstrate this in more detail in the coming chapters.
[4] Some of which I have personally witnessed.

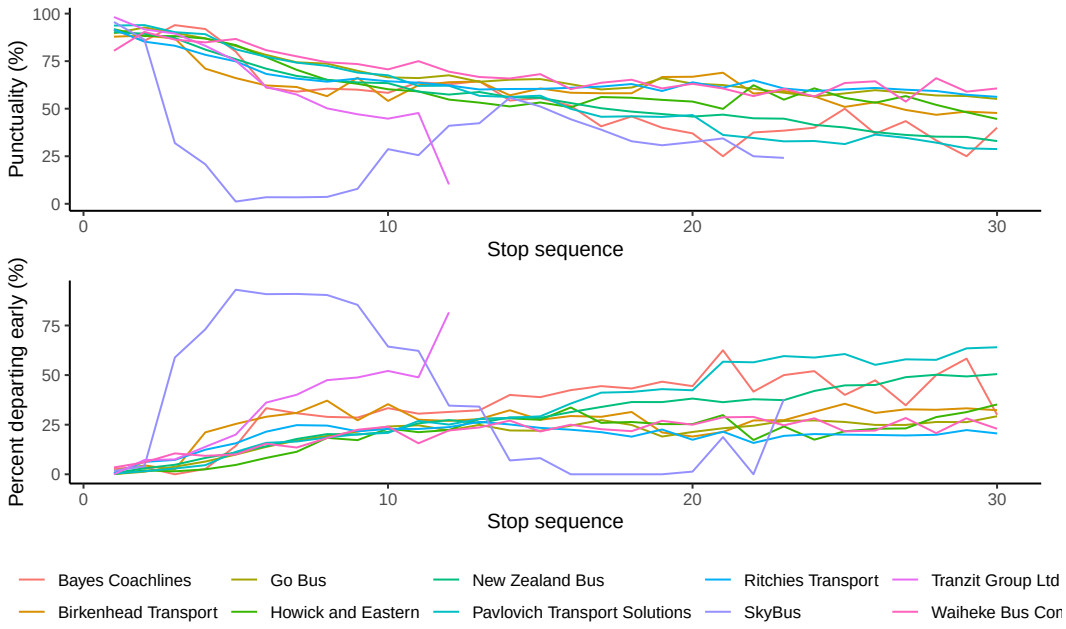


FIGURE 1.1: Punctuality (top) and percentage of services arriving early (bottom) for Auckland's transport service providers by stop sequence. Punctuality is measured by the percentage of services departing between 1 minute early and 5 minutes late.

Other issues are related to driver behaviour and *schedule adherence*. Auckland Transport itself does not actively monitor schedule adherence beyond the first stop: "Punctuality is measured by the percentage of total scheduled services leaving their origin stop no more than one minute early or five minutes late" (Auckland Transport, 2019, p. 13). As demonstrated in figure 1.1,[5] buses tend to run early, particularly towards the end of the route, which I would argue is more frustrating than running late for anyone wanting to catch it. At other times, travellers may need to catch multiple buses to get somewhere, in which case a *transfer* between services is necessary. However, if the first bus runs late, the connecting bus will usually depart on time (before the first bus gets there) leaving the passenger stranded until the next trip. In some cases, this may be 30 minutes or more, which inevitably leads to frustrated passengers.[6]

Given that many of the issues with Auckland's transport system relate to infrastructure, the simplest solution is improved, reliable RTI. Auckland Transport uses

[5] Sadly in this scenario there is no evidence to suggest 'Bayes' Coachlines perform any better than the more 'frequently' observed alternatives.

[6] I have experienced this more than once, where the second bus for my trip home departed as the one I was on pulled into the station.

General Transit Feed Specification (GTFS), a specification for how transit data should be organised (Google Developers, 2006). Part of GTFS provides a simple method for predicting arrival times, which is what Auckland Transport and other agencies use. In this method, *trip updates* are reported when a bus arrives at or departs from a stop, and includes the vehicle's *delay* (the difference between scheduled and actual arrival times). Arrival time at upcoming stops is then estimated by adding this delay to the scheduled arrival times. This method quickly runs into problems if the scheduled travel times are not accurate and drivers do not actively adhere to them (which is the case in Auckland as we saw in figure 1.1).

Besides being inaccurate and unreliable, arrival times based on scheduled arrival times and current delay are prone to sudden changes in the latter. For example, if the bus encounters heavy traffic between stops, it will get later, but this is not reported to passengers until the bus finally arrives at the next stop. When it does, passengers waiting at stops farther along see the ETA suddenly increase. Another common occurrence is when the bus is delayed on a previous trip: the default arrival time estimate is the scheduled arrival time, so until the bus starts the trip, the ETA is as scheduled. When the bus is delayed, it may commence the route ten or more minutes late, and when it finally does, the ETAs at stops suddenly increase by ten (or more) minutes. If the ETA reaches zero before the bus begins the trip, the bus disappears from the digital message sign completely. These scenarios all contribute to traveller frustration and lack of trust in the system.

Conversely, GTFS provides an opportunity for a standard, globally available framework: "The standardised format means that innovative tools and products that utilize GTFS can easily be applied across transit agencies." (Viggiano et al., 2019, p. 26). It is therefore desirable to have a framework that is capable of using solely GTFS data to model transit vehicles, estimate traffic conditions throughout the network, and predict arrival times. Location-specific features, such as taxis or intersection locations, should not be required, but could be added as desired provided the framework is structured to allow it.

The components described in the literature as essential to arrival time prediction are travel time and dwell time. To the best of our knowledge, there is no system based solely on GTFS data that provides a means of combining data across routes to

estimate traffic conditions and use them to improve arrival time prediction.

1.3 Research outline

Public transport is essential wherever people need to get to and from work, school, or other activities dependably. However, in this age of cellphones and RTI, travellers have come to expect, at the very least, a reliable countdown on the digital message sign at their stop. In Auckland, this is not the case. The unreliability of ETAs makes them unusable for real-time journey planning, causing difficulties for commuters who rely on public transport and potentially driving them to consider alternative—often private—transport options.

The central focus of this thesis is to develop an arrival time prediction application that accounts for real-time traffic information obtained from both static and real-time data provided by GTFS. This means that the application can be applied to any city using GTFS, and not solely Auckland, as many of the problems discussed will be present elsewhere. We explore how traffic state can be estimated independently of routes so that those using the same roads can share travel time or speed information. Our goal is to make arrival time predictions which, by accounting for real-time traffic conditions, are more reliable than those currently available in Auckland. However, it is clear from both the literature and personal experience that substantial levels of uncertainty are inescapable in transit prediction. For this reason, we focus on capturing the uncertainty in the model and estimating the *arrival time distribution*, which can be summarized using, for example, prediction intervals.

We also consider journey planning, notably the use of arrival time distributions to make decisions, since Auckland (and other) transit providers offer no such service. Auckland Transport's "journey planning" is based solely on the scheduled timetables, as do transit directions provided by Google Maps.[7] Bérczi et al. (2017) demonstrate the use of probabilistic arrival time distributions in a real-time dynamic routing problem. It is not our intention to solve the routing problem (selecting candidate routes from origin to destination), but instead to demonstrate the reliability (or usefulness) of our arrival time distribution in choosing between alternative routes. This could then be

[7] https://maps.google.com

incorporated into a dynamic routing framework that uses the probabilities of events to make decisions.

Before approaching this problem, I first provide an overview of the relevant background information in chapter 2. This includes an overview of GTFS and a discussion of the characteristics of real-time transit data. We also examine how a *network* can be constructed from GTFS data to allow the sharing of information across independent routes. Lastly, I give a brief introduction to *recursive Bayesian filtering*, an approach to modelling data in real-time that is used extensively throughout this thesis.

Having laid some foundations, we begin the process of real-time prediction by first modelling the state of a transit vehicle, which is updated whenever new data is received. One important aspect of chapter 3 involves removing or modelling as much uncertainty as possible. Cathey and Dailey (2003) (and others) used *map-matching* to calculate the vehicle's trip-distance-travelled from the reported GPS coordinates. As part of the vehicle model, I propose a method of removing this step and incorporating GPS observations directly into the likelihood function to overcome several of the issues identified in chapter 2. The main goal of chapter 3 is to use real-time transit vehicle observations to estimate vehicle speed along roads. This is similar to the work of Čelan and Lep (2017, 2018) who used historical data to estimate average vehicle speed along roads in a *network*.

After observing the speeds of transit vehicles travelling around Auckland, we can model the real-time *traffic state* of the road network. Similar to the work of Shalaby and Farhan (2004) we develop a Kalman filter that allows real-time estimation of network state in chapter 4. A large part of this consists of estimating the necessary model parameters from historical data and assessing the real-time accuracy of our model. Predicting future travel times (which depend on road state) is difficult (He et al., 2020) and requires a more complex model than the Kalman filter. At the end of chapter 4, I discuss a possible approach to forecasting that could later be incorporated into the arrival time prediction component of our application.

Finally, we come to arrival time estimation itself, which combines vehicle and road states to obtain an arrival time distribution. In chapter 5, I demonstrate how the distribution itself is computed and compare our approach to several others. Then, in

chapter 6, I derive a simplified arrival time cumulative density function and present several forms of summary statistics that could be displayed to commuters (point and interval estimates). We will also explore the use of the cumulative density function to answer journey planning questions concerning, for example, journey duration or the probability of on-time arrival. These results focus on the reliability and potential usefulness of our method's predictions compared to those currently available in Auckland.

Since this is a real-time application, we assess its computational performance throughout. Therefore, at the end of each chapter, I describe the programmatic considerations of the application and discuss the relative components of the R package 'transitr' developed as part of this research. The target for real-time predictions—from the time the data is first requested from the server until ETAs are available for travellers—is 30 seconds.

Finally, in chapter 7, we review our methods and findings and comment on what could be done to improve our application in the future. The role of RTI in public transport is only going to become more critical as cities—particularly Auckland—grow. Improving the reliability of the information is the first step to increasing ridership.

Chapter 2

Real-time transit data

Auckland Transport is the primary organisation responsible for the fleet of buses, trains, and ferries that transport passengers around Auckland, New Zealand's most populous urban region (Statistics New Zealand, 2019), from Wellsford in the north to Pukekohe in the south. Serving the region are 1040 routes,[1] each providing a service from an *origin* to a *destination* via a specified sequence of *stops* at which passengers may board or alight.

For each route, one or more *trips* are scheduled to operate, typically departing from the origin stop at a fixed time, with approximate arrival times for the intermediate stops along the way. In some cases, services wait at specific stops until the scheduled departure time, known as a *layover*; in others, drivers may adjust their speed to adhere to the schedule. In most cases, however, drivers do not heed the schedule and drive along the route at whatever speed is reasonable.[2] Not all trips run every day; typically trips are scheduled as 'weekday', 'Saturday', or 'Sunday and public holidays'. Altogether there are (at the time of writing) 24,919 trips provided by Auckland Transport, 14,022 of which run every weekday.

The assignment of routes, trips, and stops is a common occurrence in transport systems, to the extent that it was formalised by Google's *General Transit Feed Specification (GTFS)* (Google Developers, 2006). GTFS provides detailed guidelines for the organisation of transit data, making it significantly easier for external systems to access, with the primary example being Google Maps, although a raft of other applications exist that make use of it.[3] GTFS has now been adopted by 1,234 providers

[1] As of 12 August 2019, sourced from `https://at.govt.nz/`.

[2] This is what I have observed over 10 years of catching public transport, but also refer back to figure 1.1.

[3] Including the one we are presenting.

FIGURE 2.1: There are (at least) 672 GTFS feed providers for locations all around the world. Figure from http://transitfeeds.com/, accessed on 12 Feb 2020.

across 672 locations globally which are shown in figure 2.1. Section 2.1 provides further information on GTFS.

The vehicles servicing the trips are observed as they travel along the route, either as intermittent GPS locations (*vehicle positions*), or on arrival at or departure from bus stops along the way (*trip updates*). The structure of this data is specified by 'GTFS-realtime',[4] and is described in detail in section 2.2, along with a discussion of the issues encountered using the Auckland Transport data.

One of the main ideas we discussed in section 1.1 was the importance of combining data from multiple routes when predicting arrival times. As it stands, there is no direct method for determining if two routes overlap—that is, share a common path—from GTFS data. The most straightforward approach is to compare subsequences of stops, with routes servicing the same stops most likely following the same path between them. There are, however, several situations where this is not the case. We formalise this idea, as well as expand on it, in section 2.3.

The real-time nature of the application demands that predictions are made as soon after observing the data as possible. For this reason, recursive Bayesian models (RBMs) are a logical choice for estimating states in real-time and have been used extensively in the literature. This family of models are commonly used in vehicle tracking applications (Carpenter, Clifford, and Fearnhead, 1999; Mutambara and Durrant-Whyte, 2000; Wall and Dailey, 1999; Zhao, 1997), allowing the vehicle's state

[4] https://developers.google.com/transit/gtfs-realtime

Term	Definition
route	a collection of *trips* that are displayed to comuters as a single service
trip	a journey servicing two or more stops at a specific time
stop	a location where passengers are picked up or dropped off
stop time	the (scheduled) times at which vehicles will arrive at stops for each trip
shape	the GPS track a vehicle will take for a specific route

TABLE 2.1: Definitions of the relevant GTFS terms from https://developers.google.com/transit/gtfs/reference/

(such as speed) to be estimated from a real-time sequence of noisy measurements of its location. The final section of this chapter provides an introduction to the RBMs used throughout the remainder of this thesis.

2.1 GTFS: General Transit Feed Specification

The adoption of GTFS (Google Developers, 2006) by public transport agencies around the world has made it possible for applications such as Google Maps to access and display public transport data to users, regardless of their location. The main goal of GTFS is to specify, in detail, how transit data should be organised so that it is consistent across agencies around the world. As a result, it does not matter whether you are in Auckland, Paris, or Córdoba; Google Maps can show you public transport directions by accessing local GTFS data.

GTFS consists of two components, *static* and *real-time*.[5] The static component specifies how information about the schedule, fares, and route geographies are organised, while the *real-time* component specifies the format for real-time data: vehicle locations, stop updates and ETAs, and service advisories. Each of the static and real-time components is implemented by the transit provider; for example, Auckland Transport's GTFS service is hosted at https://dev-portal.at.govt.nz.

2.1.1 Static GTFS

There are several components of GTFS that are of particular interest to us: routes, trips, stops, stop times, and shapes. The definitions of these terms are given in table 2.1 and appendix A. Extensive documentation can be found on the GTFS website.[6]

[5] The official name is 'GTFS-realtime'.

[6] https://developers.google.com/transit/gtfs/

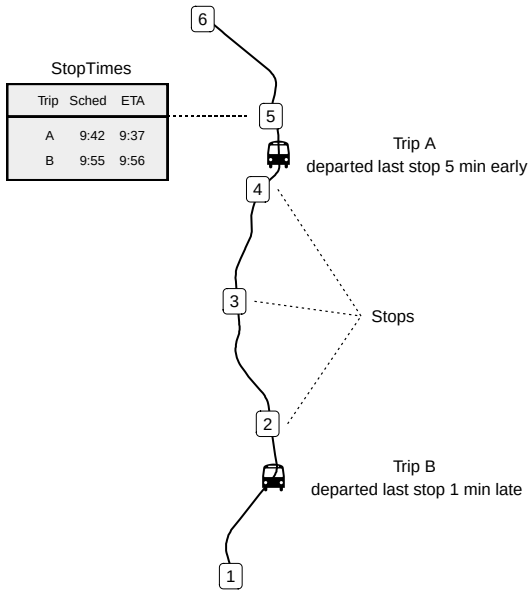


FIGURE 2.2: The main components of a GTFS feed for a single *route* with two *trips* (A and B) travelling along a *shape* path. The numbered squares represent *stops*, and the scheduled arrival times of each trip at stop 5 are displayed in the *stop times* box.

Figure 2.2 demonstrates a single *route* along which there are two active *trips* (A and B). The route's *shape* is represented by the line connecting the six *stops* numbered 1–6. The real-time arrivals board or DMS is shown for stop 5, displaying the scheduled *stop time* (arrival time) for each trip at that stop. The additional information displayed is described in the next section.

Transport providers typically distribute static GTFS data in plain text files, one for each of the components (such as `routes.csv` and `trips.csv`). Often these are available to download as a single ZIP archive, so the data is easily loaded into a *relational database*, which is described further in appendix A. The R package developed as part of this work provides the `create_gtfs()` function to do this automatically:

```
library(transitr)
nw <- create_gtfs("at_gtfs.zip", db = "at_gtfs.sqlite")
```

Within these separate files or tables, the data is stored as per GTFS. Importantly, shapes are stored as sequences of coordinates that draw a path on a map, while stops are represented as a single coordinate marking the location of the bus stop. Stop times are stored in trip-stop pairs with one row for every stop of each trip, along with

the scheduled arrival and departure times.[7]

On its own, the static GTFS information can be used like a printed timetable, allowing simple journey planning to take place. As such, it provides a "fallback" state in situations where no real-time information is available for a given trip: scheduled stop times can be thought of as *prior information*, a core component of the Bayesian paradigm.

2.1.2 Real-time GTFS

The real-time component of GTFS is responsible for handling vehicle positions, trip updates (arrivals at and departures from stops), and service alerts (cancellations and stop closures, for example). Data is processed by a central server and then stored appropriately to enable quick access via an application programming interface (API). There is, therefore, the additional need of a server that can handle vast numbers of API requests. As such, only a subset of transport providers using GTFS have also implemented the real-time component. Below we give a summary of these components, but further information is available on the GTFS website.[8]

Vehicle positions

There are several key components to a vehicle position (in the context of GTFS). The *vehicle descriptor* includes information about the physical vehicle, while a *trip descriptor* holds information about the trip being serviced. A *timestamp* specifies exactly when the observation was made, and a *position* contains the actual data, such as the GPS observation, as is demonstrated by the bus images in figure 2.2.

The specification also allows for additional measurements, such as *speed* or an *odometer* reading. However, these were not available from Auckland Transport at the time this work was carried out, so they are not included in our application. It is well worth noting, however, that they could be integrated with minimal effort if they become available.

[7] There are 732,184 rows in the stop times table for Auckland Transport!
[8] See https://developers.google.com/transit/gtfs-realtime/.

Trip updates

As vehicles equipped with AVL technology arrive at and depart from stops, information about their times of arrival and, most importantly, *schedule adherence* is stored in trip updates. These also contain a *trip descriptor*, as well as one or more *stop time updates*. GTFS-realtime allows storing of stop time updates for all previously visited stops, as well as predictions for upcoming stops. However, Auckland Transport only stores the most recent update.

Each stop time update reports either the arrival or departure time and, where schedule information is available, the schedule adherence by way of an *arrival* or *departure delay* (in seconds). Figure 2.2 displays this information offset to the right of each bus location. The on-board AVL device is responsible for detecting these events, and they are not necessarily linked to the GPS position of the vehicle. In Auckland, each trip update also triggers a vehicle location update, but the coordinates are those of the *stop*, not of the vehicle. This leads to some of the problems addressed in section 2.2.

In Auckland, the current delay is added to the scheduled arrival time, as shown on the DMS in figure 2.2. We will certainly be discussing the issues associated with this method shortly, and indeed be making comparisons to it throughout the thesis.

Service alerts

Less important for the current work, but essential for reliable real-time information, *service alerts* enable transit operators to modify the static GTFS in real-time. So, when a trip is cancelled, they can send out a service alert announcing the cancellation,[9] which is then displayed to passengers as a "C" on the DMS. It is also possible to add trips, for example during special events, or to reroute trips around stop closures, but this is beyond the scope of our work.

2.1.3 Accessing real-time data (API)

Distributing vehicle locations, trip updates, and service alerts to passengers quickly and usefully requires more than just a DMS at bus stops. Personal mobile devices have

[9] Although they often don't.

revolutionised the way we live our lives, and developers are continually creating new applications to assist with everyday activities. Amongst these are transit applications, which are capable of relaying real-time GTFS data to passengers.

The most common method of distributing real-time data is via an API. This is, in simple terms, a fixed web address from which developers can request a data file—either JSON or, in the case of some GTFS systems, protobuf[10]—which can then be parsed and displayed to users. Usually, developers need to register for an *API key* which helps to control server demand by limiting the number of requests a user can make, or control access to specific data. The 'transitr' package includes the ability to connect to a GTFS-based API easily:

```
url <- "https://api.at.govt.nz/v2/public/realtime/vehiclelocations"
nw <- load_gtfs("at_gtfs.sqlite") %>%
    realtime_feed(url) %>%
    with_headers(
        # this varies by provider
        "Ocp-Apim-Subscription-Key" = Sys.getenv('APIKEY')
    )
```

2.2 Charactersitics of real-time transit data

A range of AVL technologies exist which allow transit vehicles to report their locations in real-time. The most common of these is now the GPS, which provides the longitude and latitude of the vehicle. In the simplest of deployments, each vehicle reports its location to a central server at a fixed time interval. The server then collates the reports from all buses in the fleet and makes them available via the API.

Another type of real-time data available to us is arrival and departure information at bus stops. When a vehicle arrives at or departs from a stop,[11] it reports to the server which stop it is at along with its time of arrival or departure. The server then computes the delay (the time between actual and scheduled arrival or departure), collates the data from multiple vehicles, and also makes these available through the API.

[10] See https://developers.google.com/protocol-buffers

[11] Or thinks it does, see section 2.2.1.

Real-time information can be displayed in one of two ways. GPS positions are displayed on a map accessed through a mobile app, allowing commuters to see where their bus was when it last reported its location. For trip updates, the *current delay* is added to the scheduled arrival time and displayed to commuters, usually as a *time until arrival*, in minutes.[12] The ETA can be displayed either on a mobile app or, more commonly, on a DMS at the stop.

What we have just described is, in fact, the entirety of real-time information in Auckland and some other transit locations around the world. While at first it seems an adequate solution, discussion with just about any regular public transport user suggests otherwise. The reasons for this become apparent with a little scrutiny, which we will now uncover.

2.2.1 Vehicle positions

Every measurement of a data point comes with some associated error. In the case of GPS devices, this error is usually small with precision depending on the quality of the device. However, any device can succumb to several factors which may place the bus far from its expected path, the primary reason being buildings or other obstacles resulting in a poor signal (Mutambara and Durrant-Whyte, 2000; Xinghao et al., 2013; Zhao, 1997). Surprisingly, however, this is not the main issue with vehicle position data.

Object tracking has been well studied, and many algorithms exist for tracking an object through space using GPS observations. However, these usually take high-frequency observations (Gustafsson et al. (2002) updated the car's location twice per second) which can generate an almost exact real-time estimate of the object's actual position.

Many examples of real-time object tracking exist, but the most relatable to most readers will be in their pocket. When getting directions from your phone, the maps application requests the user's phone's location continuously, providing the exact location with a second or less of delay. However, have you ever been driving along, following directions, and accidentally missed the turn-off? Often, the maps application will show you as *on course* for several seconds until it realises that you have

[12] That is "scheduled arrival time + delay - current time".

well and truly gone off-track and reroutes you. This is an example of a real-time position tracking algorithm that is attempting to follow the device's location while simultaneously accounting for inherent noise in the measurements. When the driver first goes off track, the algorithm assumes this is due to measurement error and assumes the vehicle is on course. Eventually, the error becomes large and consistent enough that the model stops assuming the driver is following the planned route.

With real-time transit data, the frequency of observations is vastly reduced, with observations obtained with anything from ten seconds to a minute (or more) between them. This makes it very difficult to estimate the vehicle's exact location. On receiving a vehicle position that seems incorrect, we must wait until we receive the next observation to determine if it was the result of a bad measurement or a real event.

Another major complication with the Auckland Transport vehicle data is that vehicles often report their location when arriving at or departing from a bus stop. However, instead of reporting their GPS location as measured from the GPS device, they report the location of the bus stop itself, which is known exactly. So, what happens when the bus approaches a bus stop at speed, only to come to a halt at an intersection 100 m before hand? The vehicle's AVL system continues predicting the trajectory and places the vehicle at the bus stop before it actually arrives. This triggers a trip update (section 2.2.2), which itself produces a vehicle position update *at the bus stop*. However, a passenger standing at the stop will see the bus sitting at the lights even though the DMS displays the bus as having arrived. For the passenger waiting at this stop, this is of no concern, as they can see the bus and have no need for real-time data. For passengers farther down the line, however, this can have some frustrating implications.

Another related phenomenon exists in which the bus may appear to go backwards according to the sequence of GPS coordinates, discussed in detail in section 3.3.3. The main consequence of this problem is that within the data processing component of our application, we check for vehicle position updates associated with trip updates and remove them (that is, we use the trip update and not the vehicle position).[13]

[13]This seems straightforward, but was frustrating until identified.

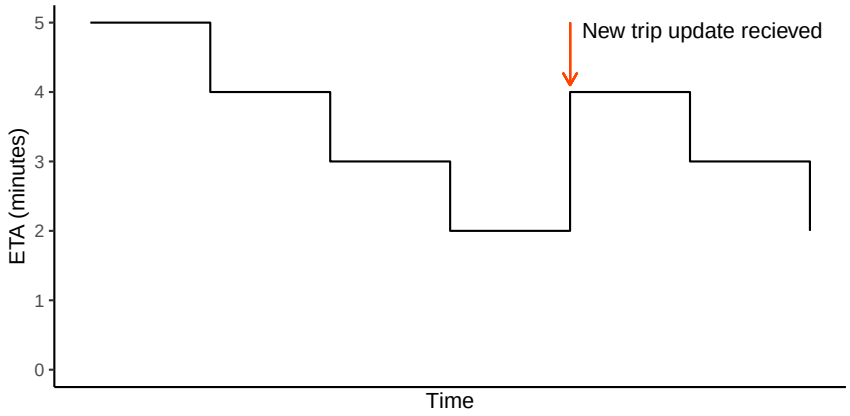


FIGURE 2.3: The current system displays ETAs to passengers as integer minutes which decrease over time. When new data is recieved (marked by an arrow) the ETA is updated, which may result in a sudden "jump", as demonstrated.

2.2.2 Trip updates

As alluded to earlier, trip updates are prone to measurement error. Without human intervention, it is challenging for the GPS tracking system on the bus to determine exactly when the bus arrives at or leaves a bus stop. In situations like the one described above, the arrival time may be reported before the bus arrives, resulting in a premature arrival time and, more importantly, *a reduced delay*. Traffic lights may hold up a bus for a minute (or more), so the bus may, for argument sake, appear to arrive precisely on time with a delay of zero minutes. The result of this is the propagation of the current delay to all future stops, which then display an ETA that matches the scheduled arrival time. However, two minutes later, after the bus has finally arrived at the stop, dropped off and picked up passengers, it departs, triggering another trip update. The delay, now two minutes, is propagated to upcoming stops. Passengers waiting at these stops see the ETA jump suddenly by two minutes, as demonstrated in figure 2.3, leading inevitably to much frustration for passengers.

The reverse can also occur, for example, if for some reason the bus reports its arrival or departure late. However, more prevalent is that the update is skipped altogether, so a bus that was five minutes behind schedule has made up several minutes and arrives at the bus stop while the DMS still shows it as being two minutes away. At first, this may seem like a good thing: passengers at the stop have a shorter wait; other passengers making their timely way to the bus, however, may have to

sprint or miss the bus altogether, which is not ideal.

The main repercussion of the described issues is that arrival and departure time data become very noisy and difficult to trust. They do, however, provide a lot of information without which it would be challenging to infer the trajectory of a bus between stops—which is the primary aim of this part of our work. We discuss this further in section 3.1.2 when I present the likelihood function.

2.3 Constructing a network from GTFS data

In section 1.1, I reported how arrival time prediction is greatly improved when data from multiple routes is combined to estimate traffic conditions (Yu, Lam, and Tam, 2011). However, many of these applications were specific to a certain set of routes, or used external information (such as automatic toll readers and taxis) to get real-time traffic information. We wanted to develop a *simpler*, *more generalised* approach using solely GTFS data (stops and shapes) to construct a network of non-overlapping[14] road segments. The network should consist of *nodes* (which could be intersections or bus stops) connected by *edges*, or road segments, which is similar to the structure used by Čelan and Lep (2017) and Vuurstaek et al. (2018).

The primary purpose of network construction is to detect where any two routes overlap, even partially. There are several possible approaches to this, some of which use GPS traces (Xie et al., 2016; Zhang et al., 2017), but the simplest is to look at common subsequences of bus stops. Of course, stops do not make optimal node locations because routes do not diverge *physically* at stops—that would require locating *intersections*. However, due to difficulties in finding intersections from either the GTFS data (using algorithms such as the one proposed by Xie et al. (2016)) or outside information, we opted to use bus stops for this work. Figure 2.4 shows a simple route diagram with several overlapping routes. We see that each link between stops is common across routes, but routes that merge at intersections do not merge in the network until they reach a common stop.

One other issue with Auckland Transport's GTFS data is that all object IDs are *versioned*. That is, instead of a route having an ID of `02702`, the date and version

[14] Except of course reverse directions.

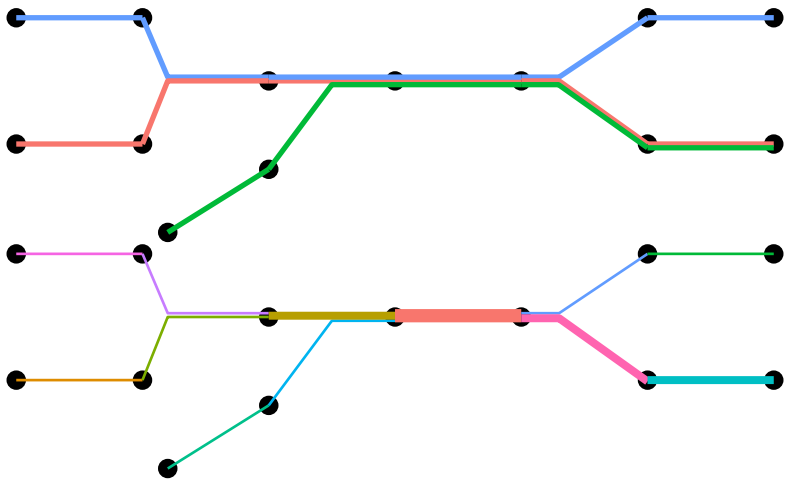


FIGURE 2.4: A simplified transit network with road segments (edges) connecting stops (nodes). Top: three routes with common subsequences of stops. Bottom: the widths of road segments represents the number of routes using them. There remains some segment overlap since routes merge at stops, not intersections.

number are appended to it: `02702-20190806160740_v82.21`. This is the same for the IDs of trips, shapes, and stops, making it challenging to transfer existing segments based on stops from previous versions of the API and requiring an additional step to remove the version information from the raw data.

The algorithm we implemented uses only bus stops but can extend to intersections, if available. Each stop along a route is assessed to see if any nodes already exist in that location to avoid the versioning issue mentioned above. If so, the route is assigned the existing node; otherwise, a new node is created and assigned instead. Road segments are defined by unique one-way connections between two stops: there can only be one segment going from node A to node B (the reverse is considered a different segment). This network construction is implemented in 'transitr':

```
nw <- load_gtfs("at_gtfs.sqlite") %>% construct()
transitr:::load_road_segments(nw) %>% head(4)
```

```
##   road_segment_id node_from node_to    length
## 1               1      5414     537  485.6548
## 2               2       537    3987  196.5567
## 3               3      3987    7265  567.0883
## 4               4      7265    1255 1049.0181
```

2.4 Recursive Bayesian models

The most challenging problem we face in this application is its real-time nature. Vehicles report their current location, from which we want to estimate road speeds to predict arrival times, all within no more than 30 seconds. Fortunately, a certain class of models suit themselves well to real-time applications, and are indeed used in many vehicle tracking and robotics applications (Anderson and Moore, 1979; Gustafsson et al., 2002; Mutambara and Durrant-Whyte, 2000; Ulmke and Koch, 2006): RBMs, sometimes referred to a *sequential Bayesian estimation*.

In a typical analysis, data $\mathbf{Y}$ would be stored in an $n \times k$ matrix corresponding to k measurements of n variables over time. Then Bayes' rule is used to estimate the posterior distribution of an $m \times k$ matrix of parameters $\mathbf{X}$ all at once,

$$p(\mathbf{X} \mid \mathbf{Y}) = \frac{p(\mathbf{X})p(\mathbf{Y} \mid \mathbf{X})}{p(\mathbf{Y})}, \tag{2.1}$$

using, for example, a Markov chain Monte Carlo algorithm. This is usually a computationally intensive procedure and takes much longer than 30 seconds to converge.[15]

In a real-time application, the columns of $\mathbf{Y}$ are observed sequentially, so that at time t_k we have observed

$$\mathbf{Y} = \boldsymbol{y}_{1:k} = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_k], \tag{2.2}$$

to estimate

$$\mathbf{X} = \boldsymbol{x}_{0:k} = [\boldsymbol{x}_0, \cdots, \boldsymbol{x}_k], \tag{2.3}$$

where $\boldsymbol{x}_0$ is the *initial state* of the object. Rather than refitting the full model, as would be necessary when using Markov chain Monte Carlo, recursive Bayesian estimation allows us to combine the previous *posterior estimate* of the state with the new information.

RBMs make two general assumptions. The first is that $\boldsymbol{x}$ follows a Markov process such that the state at time t_k depends only upon the state at time t_{k-1}:

$$p(\boldsymbol{x}_k \mid \boldsymbol{x}_{0:k-1}) = p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}). \tag{2.4}$$

[15] That's only for one bus, too!

The joint distribution of the underlying state parameters is therefore

$$\begin{aligned} p(\boldsymbol{x}_{0:k}) &= p(\boldsymbol{x}_0)p(\boldsymbol{x}_1 \mid \boldsymbol{x}_0)p(\boldsymbol{x}_2 \mid \boldsymbol{x}_{0:1}) \cdots p(\boldsymbol{x}_k \mid \boldsymbol{x}_{0:k-1}) \\ &= p(\boldsymbol{x}_0)p(\boldsymbol{x}_1 \mid \boldsymbol{x}_0)p(\boldsymbol{x}_2 \mid \boldsymbol{x}_1) \cdots p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) \\ &= p(\boldsymbol{x}_0) \prod_{i=1}^{k} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{i-1}). \end{aligned} \tag{2.5}$$

The second assumption is that the observations $\boldsymbol{y}_k$ of the state depend only on the current state and are independent of one another:

$$p(\boldsymbol{y}_k \mid \boldsymbol{x}_{0:k}, \boldsymbol{y}_{0:k-1}) = p(\boldsymbol{y}_k \mid \boldsymbol{x}_k). \tag{2.6}$$

Thus, the data has joint likelihood

$$\begin{aligned} p(\boldsymbol{y}_{1:k} \mid \boldsymbol{x}_{0:k}) &= p(\boldsymbol{y}_1 \mid \boldsymbol{x}_{0:1}) \cdots p(\boldsymbol{y}_k \mid \boldsymbol{x}_{0:k}) \\ &= p(\boldsymbol{y}_1 \mid \boldsymbol{x}_1) \cdots p(\boldsymbol{y}_k \mid \boldsymbol{x}_k) \\ &= \prod_{i=1}^{k} p(\boldsymbol{y}_i \mid \boldsymbol{x}_i) \end{aligned} \tag{2.7}$$

and marginal distribution

$$p(\boldsymbol{y}_{1:k}) = p(\boldsymbol{y}_1) \cdots p(\boldsymbol{y}_k) = \prod_{i=1}^{k} p(\boldsymbol{y}_i). \tag{2.8}$$

We now express the posterior distribution $p(\boldsymbol{x}_{0:k} \mid \boldsymbol{y}_{1:k})$ using equation (2.1) along with equations (2.5), (2.7) and (2.8) as

$$\begin{aligned} p(\boldsymbol{x}_{0:k} \mid \boldsymbol{y}_{1:k}) &= \frac{p(\boldsymbol{x}_{0:k})p(\boldsymbol{y}_{1:k} \mid \boldsymbol{x}_{0:k})}{p(\boldsymbol{y}_{1:k})} \\ &= \frac{p(\boldsymbol{x}_0) \prod_{i=1}^{k} p(\boldsymbol{x}_i) \cdot \prod_{i=1}^{k} p(\boldsymbol{y}_i \mid \boldsymbol{x}_i)}{\prod_{i=1}^{k} p(\boldsymbol{y}_i)} \\ &= p(\boldsymbol{x}_0) \prod_{i=1}^{k} \frac{p(\boldsymbol{x}_i)p(\boldsymbol{y}_i \mid \boldsymbol{x}_i)}{p(\boldsymbol{y}_i)}. \end{aligned} \tag{2.9}$$

However, we can substitute the first $k-1$ terms in equation (2.9) to obtain the following recursive representation:

$$p(\boldsymbol{x}_{0:k} \mid \boldsymbol{y}_{1:k}) = p(\boldsymbol{x}_{0:k-1} \mid \boldsymbol{y}_{1:k-1}) \frac{p(\boldsymbol{x}_k)p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)}{p(\boldsymbol{y}_k)}. \tag{2.10}$$

Here, the posterior distribution from the previous time step is used as a *prior* for the next, and so it *recursively* (or *sequentially*) updates the state estimate as new data are observed, making RBMs ideal for real-time applications.

There are several types of recursive Bayesian estimation which consist of two main steps: *predict* and *update*. In the prediction step, the algorithm uses a *transition function* f (or matrix $\mathbf{F}$) to predict the new state based only upon the current state (equation (2.4)), while the *update* step incorporates the data to adjust the estimate using a *measurement function* h (or matrix $\mathbf{H}$) describing the relationship between x and y (equation (2.6)). These can be expressed by the following model equations:

$$\begin{aligned} \boldsymbol{x}_k &= f(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k), \\ \boldsymbol{y}_k &= h(\boldsymbol{x}_k) + \boldsymbol{v}_k. \end{aligned} \tag{2.11}$$

The additional parameters $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ represent *system noise* and *measurement error*, respectively. The choice of f, h, and the distributions for the error terms depends on the choice of model. We now discuss two types of recursive Bayesian estimation used throughout this thesis: the Kalman filter and the particle filter.

2.4.1 Kalman filter

Commonly used in vehicle tracking applications, the Kalman filter is a very fast, simple estimation method (Anderson and Moore, 1979) that assumes Gaussian errors and represents the state using a multivariate Normal random variable with length m mean vector and $m \times x$ covariance matrix

$$\hat{\boldsymbol{x}}_{k|k} = \mathbb{E}\left[\boldsymbol{x}_k | \boldsymbol{y}_{1:k}\right] \quad \text{and} \quad \mathbf{P}_{k|k} = \operatorname{Var}\left[\boldsymbol{x}_k | \boldsymbol{y}_{1:k}\right], \tag{2.12}$$

respectively. A Kalman filter implementation requires an $m \times m$ *transition matrix*, $\mathbf{F}_k$, which defines the relationship between states at time t_{k-1} and t_k, and an $n \times m$ *measurement matrix*, $\mathbf{H}_k$, defining the relationship between $\boldsymbol{y}$, a length n vector, and $\boldsymbol{x}$, a length m vector.

The Kalman filter version of the model presented in equation (2.11) is

$$\begin{aligned} \boldsymbol{x}_k &= \mathbf{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k, \\ \boldsymbol{y}_k &= \mathbf{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k, \end{aligned} \tag{2.13}$$

where $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are multivariate Normal random variables with mean zero and covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$, respectively. The $m \times m$ covariance matrix $\mathbf{Q}_k$ represents the system noise at time t_k, which is defined as "the rate of change of the variance of process noise"' (Cathey and Dailey, 2003, p. 253). Measurement error is expressed by the $n \times n$ covariance matrix $\mathbf{R}_k$.

The Kalman filter is implemented through two sets of equations. First is the prediction step, in which the current state (represented by the mean vector and covariance matrix) is predicted based solely on the previous state:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= \mathbb{E}\left[\boldsymbol{x}_k | \boldsymbol{x}_{k-1}\right] = \mathbf{F}_k \hat{\boldsymbol{x}}_{k-1|k-1}, \\ \mathbf{P}_{k|k-1} &= \mathrm{Var}\left[\boldsymbol{x}_k | \boldsymbol{x}_{k-1}\right] = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k. \end{aligned} \tag{2.14}$$

Next, the update step uses the observation $\boldsymbol{y}_k$ to adjust the predicted state using the following set of equations, which are effectively taking a ratio of the state and measurement uncertainties:

$$\begin{aligned} \boldsymbol{z}_k &= \boldsymbol{y}_k - \mathbf{H}_k \hat{\boldsymbol{x}}_{k|k-1}, \\ \mathbf{S}_k &= \mathbf{R}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top, \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}, \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k \boldsymbol{z}_k, \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^\top + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^\top. \end{aligned} \tag{2.15}$$

The simplicity of the Kalman filter means that, so long as the number of parameters m is small, states can be estimated quickly with minimal processing demand. This simplicity, together with the fact that it provides an exact solution for the multivariate Normal state space model (Anderson and Moore, 1979), makes the Kalman filter a strong contender for real-time applications. However, it also makes several strong assumptions about the shape of the parameter distributions and is limited to linear

relationships between states and measurements since these must be expressed in matrix form.

2.4.2 Particle filter

Another framework we use is the particle filter, a more generalised, numerical approach to recursive Bayesian estimation (Gordon, Salmond, and Smith, 1993). The state is approximated by a sample of N particles, each of which is an independent point estimate of the state, $\boldsymbol{x}_k^{(i)}$, with a weight $w^{(i)} \geq 0$ and $\sum_i w^{(i)} = 1$. The state estimate is expressed using the Dirac delta measure δ (see appendix B.1 for details), such that

$$p(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1}) \approx \sum_{i=1}^{N} w_{k-1}^{(i)} \delta_{\boldsymbol{x}_{k-1}^{(i)}}(\boldsymbol{x}_{k-1}) . \tag{2.16}$$

One advantage of the particle filter is that we are no longer constrained in the choice of transition function, f, or the error distribution. Instead of working with the mean and variance of the state (as is the case with the Kalman filter) we work with *independent point estimates* which each represent a plausible state. In chapter 3, I use a Normal random variable for system noise, but this could be any appropriate distribution (for example a Cauchy if heavier tails are necessary). The state prediction step is carried out independently on each particle,

$$\boldsymbol{x}_k^{(i)} = f\left(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{v}_k^{(i)}\right), \quad \boldsymbol{v}_k^{(i)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{Q}_k) . \tag{2.17}$$

The prior-predictive distribution of the state, using the Dirac delta measure, is given by

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) \approx \sum_{i=1}^{N} w_{k-1}^{(i)} \delta_{\boldsymbol{x}_k^{(i)}}(\boldsymbol{x}_k) . \tag{2.18}$$

Next, the state is updated to account for the observation using the likelihood function directly. The particles are reweighted and normalised,

$$w_k^{(i)} = \frac{w_{k-1}^{(i)} p(\boldsymbol{y}_k|\boldsymbol{x}_k^{(i)})}{\sum_{j=1}^{N} w_{k-1}^{(j)} p(\boldsymbol{y}_k|\boldsymbol{x}_k^{(j)})}, \tag{2.19}$$

which yields the final posterior distribution of the state,

$$p(\boldsymbol{x}_k|\boldsymbol{y}_{1:k}) \approx \sum_{i=1}^{N} w_k^{(i)} \delta_{\boldsymbol{x}_k^{(i)}}(\boldsymbol{x}_k)\,. \tag{2.20}$$

Over time, many of the particle weights will go to zero as they disperse due to the system noise (Doucet, Godsill, and Andrieu, 2000). This leaves only a few particles contributing to the state, which, if none end near the actual vehicle location, will result in all vehicles having likelihoods close to zero. To prevent this situation of particle filter *degeneration*, we perform *importance resampling* (a weighted bootstrap) in which particles are sampled, with replacement, according to their weight. Afterwards, particle weights are reset to N^{-1}.

Doucet, Godsill, and Andrieu (2000) describe the use of the *effective sample size*,

$$N_{\text{eff}} \approx \frac{1}{\sum_{i=1}^{N}(w_k^{(i)})^2}, \tag{2.21}$$

which provides an estimate of how varied the particles are: if too few particles contain most of the weight, resampling should occur. This is similar to the Metropolis-Hasting's proposal and acceptance steps: we propose a sample of state estimates and reject those that are not plausible given the data. If the system noise (proposal distribution) is small, then more particles will be "accepted", but we may not propose sufficient trajectories. Conversely, if the noise is too large, then there will be many "rejections", reducing the variety of particles (small N_{eff}). Indeed, an alternative name for the particle filter is *sequential Monte Carlo* (Davidson, Collin, and Takala, 2011).

Another advantage of the particle filter is its generality. As we discuss in section 3.1, we can easily construct a transition function to describe a complex system, as well as define a likelihood function that accurately represents the relationship between observed and unobserved states.

The main disadvantage is, of course, that the computational demand is high, as each particle is transitioned independently, and resampling can be slow for large N, having order $\mathcal{O}(N \log N)$ in the C++ algorithm we use. However, as we discuss in the next chapter, the advantages far outweigh the additional computational demands of the particle filter, which can be minimised if implemented carefully.

2.5 Real-time implementation in 'Rcpp'

In section 2.1, I introduced the R package 'transitr', which loads a GTFS database and connects to a real-time API. The advantage of R (R Core Team, 2018) is its superior ability for processing data of various forms through additional packages, and providing an easy-to-use interface for users. However, when it comes to computational efficiency, C++ is the better choice. Fortunately, R offers an interface to C++ through the 'Rcpp' package developed by Eddelbuettel and François (2011). This gives us the speed and memory management capabilities of C++ from within R.

The general structure of our program has two parts. The first handles data collection from a transit provider and creation of the transit network (section 2.3), all stored in an SQLite database.[16] Users can connect to a GTFS-realtime API, as well as pass additional arguments to the generated `gtfs` object, which are then forwarded onto the second part which runs purely within C++.

Within the C++ component, there are two main phases: set-up and modelling. During the set-up phase, the GTFS database is loaded and parameter values are established. A vector of `Vehicle` objects is initialised to contain the real-time vehicle states. The modelling phase consists of a single `while` loop which runs until the operating system sends the program a kill signal. It is inside this loop that all of the real-time modelling discussed throughout the remainder of this thesis occurs.

Within the program, we use object oriented programming to represent objects—both static (routes and trips) and real-time (vehicles, road segments). These objects contain a lot of *interdependence*; for example, a trip belongs to a route and vehicles services trips. Pointers are used in C++ to allow easy access to these relationships without any duplication of information. For example, the route number for a vehicle can be obtained as follows:

```
vehicle.trip ()->route ()->route_short_name ();
```

As pointers are not fixed, the above example works even after the vehicle changes to another trip. In addition to retrieving information, pointers can be used to pass information between objects. Later in this thesis, vehicles are used to estimate traffic conditions along individual roads, and then forward these observations on to the appropriate road segment. First, note that a route follows a sequence of road segments,

[16] https://www.sqlite.org/index.html

which are stored in C++ using a `std::vector` containing an ordered sequence of `RouteSegment` objects, each pointing to the appropriate `Segment`. Thus, once a vehicle completes travel along the segment with index `si`, the observed average speed can be passed directly to the `Segment`:

```
vehicle.trip ()->route ()
    ->segments ().at (si)->segment ()
    ->push_data (speed, uncertainty);
```

This data is then handled by the `Segment` object (described later in chapter 4).

The main issue with pointers is that, if an object is deleted or moved, a pointer may no longer point to the appropriate object, resulting in a *segmentation fault* and crashing the program at runtime. However, with care, and by checking a pointer is valid before using it, it is possible to avoid such problems. For example, the above would be better written as follows:

```
Trip* trip = vehicle.trip ();
if (trip == nullptr) return();
Route* route = trip->route ();
if (route == nullptr) return();
if (route->segments.size () <= index) return ();
Segment* segment = route->segments.at (index)->segment ();
if (segment == nullptr) return ();
segment->push_data (speed, uncertainty);
```

The last topic for this section is *multithreading*, which is the process of using more than one CPU core to run the program with the assistance of OpenMP (Dagum and Menon, 1998). This requires that the program is *threadsafe*, such that two independent cores will not adversely interact (for example by both trying to modify the same object). The simplest way around this is to perform read-only operations on common resources.

Sometimes, however, write operations are unavoidable, such as when passing data to road segments. *Mutex locking* provides a simple way of ensuring that only one process can perform a specified task at a time. Continuing with the same example, if two vehicles traverse a road at the same time (which happens frequently), they will both want to push their observed speed observations to this

same `Segment` object at the same time. To prevent errors, we create a lock at the beginning of the `Segment::push_data ()` method. Now when the first `Vehicle` calls `push_data ()`, the `Segment` is "locked" until the method is complete (the data is processed and stored). If a second `Vehicle` calls the method before the first has finished, it will find the `Segment` locked and the process will wait until the lock has been released before it can continue. This lets us parallelise the processing of vehicles to multiple cores, greatly speeding up iteration timings.

2.6 Chapter contributions

- Generalised construction of a transit road network from (solely) GTFS data. Other research has used similar approachs, but I have not seen any explicit construction of a "network" in this way.
- Implementation of automatic network construction within the R package 'transitr'.

Chapter 3

Modelling transit vehicles

As vehicles travel along their respective routes, they report—in real-time—their geographical positions as GPS coordinates. While these observations of location can be useful on their own, they also allow the inference of *vehicle speed* which we cannot observe directly.[1] These speeds can then be mapped to physical road segments and used (in chapter 4) to update the network state.

Transit vehicles come in a variety of forms such as trains, trams, and buses. The last of these is the most notable for us since they are most affected by external factors, such as traffic congestion, and are therefore harder to predict. It is for this reason that we have used buses to develop the model presented in this chapter, which explores the following behaviours:

- general travel along a known path (including acceleration and deceleration);
- bus stops, at which the bus may (or may not) stop;
- intersections (both controlled and uncontrolled) which may (or may not) temporarily halt a bus; and
- driver behaviour (this mostly comes under vehicle speed but is more significant in modelling buses versus trains).

It is evident from the behaviours listed above (described in further detail in section 3.1) that there are many factors involved in determining a bus's trajectory. Many of these factors, such as speed, are unmeasurable (but can be estimated), while others are often completely unknowable. An example of the latter is whether the bus

[1] At least, not without being on the bus or standing on the roadside with a speed radar.

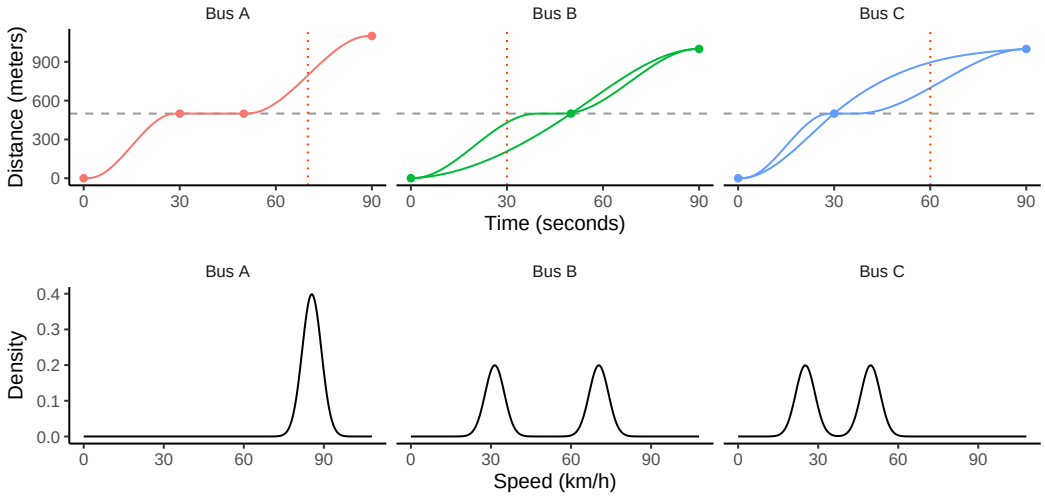


FIGURE 3.1: Uncertainty in whether or not a bus stops can lead to multimodality in its speed dsitribution. Top: observations (points) with possible trajectories which fit the data. The horizontal dashed line represents the location of the stop, and the vertical dotted line indicates the time point at which vehicle speed is estimated. Bottom: estimated speed distribution at the time indicated in the top graph.

stops at a bus stop. Figure 3.1 shows a series of *distance* observations for three buses passing a stop:

- bus A stops and reports both its arrival and departure times;
- bus B stops, but only reports its departure time;[2]
- bus C does not stop and reports a departure time as it passes the stop.

Overlaid are possible trajectories for each vehicle: for bus A, there is only one since we observe both the vehicle's arrival and departure. However, B and C each have two potential paths: one in which the vehicle stops, and another in which it does not. The main point is that *we cannot know which is true*, leading to *multimodality* in the vehicle's state. In the lower half of figure 3.1 are the possible distributions of the vehicles' speeds at the time marked with a dotted vertical line in the top figure. For bus A, the distribution is *unimodal*, which satisfies the assumptions of many estimation methods (such as the Kalman filter, section 2.4.1). For buses B and C, however, the speed state is *bimodal*, so the Kalman filter would no longer be an appropriate choice for these data. This multimodality was one of the main reasons we chose to use a particle filter

[2]This can happen if the arrival observation is "lost" or overwritten by the departure time within the GTFS update interval.

to implement the vehicle model (section 3.1), as it does not encounter these same issues and is ideal for sampling a wide range of disparate trajectories (Hans et al., 2015; Ulmke and Koch, 2006).

Once the vehicle's state has been estimated, we may infer its average speed along a given road segment, which we cover in section 3.2. Included is a simulation comparing the models introduced in section 3.1, and showing the effectiveness of our approach with simulated data. I conclude this chapter with a discussion of the real-time implementation of the model (section 3.3), including the process of parameter estimation for the various model parameters, and details of some difficulties experienced while modelling these data.

3.1 Real-time estimation of vehicle state

Real-time vehicle tracking has been the centre of much research, particularly for robotics applications and self-driving cars (Daum, 2005; Gustafsson et al., 2002). In many of these, data is sampled at a very high frequency, often multiple times per second, leading to high-accuracy estimates of the object's *state*. Alas, with transit data this is seldom the case; instead, it is common for observations to be reported once every 10–30 seconds or, in some cases, even longer.

Low-frequency observations lead to high uncertainties of state parameters and difficulty capturing all possible trajectories, which can lead to degradation of the model and loss of information. In figure 3.1, we show two possible trajectories for buses B and C; there are, in fact, countless possibilities: perhaps the bus waited at the stop a little longer? In this case, there is uncertainty about the bus's dwell time and speed; however, the first step of a recursive Bayesian model is to *predict* the upcoming state *before observing it* (section 2.4). If the model does not cover the vehicle's actual trajectory, we effectively "lose" the vehicle, and the filter is said to *degenerate* (Chen and Rakha, 2014). In that case, we typically need to reinitialise the vehicle's state, resulting in a loss of any information we might have gained in the last time step.

When deciding which type of model implementation to use, historically the most common for this type of data is the Kalman filter (Cathey and Dailey, 2003; Dailey et al., 2001; Wall and Dailey, 1999). However, it assumes a Gaussian state distribution,

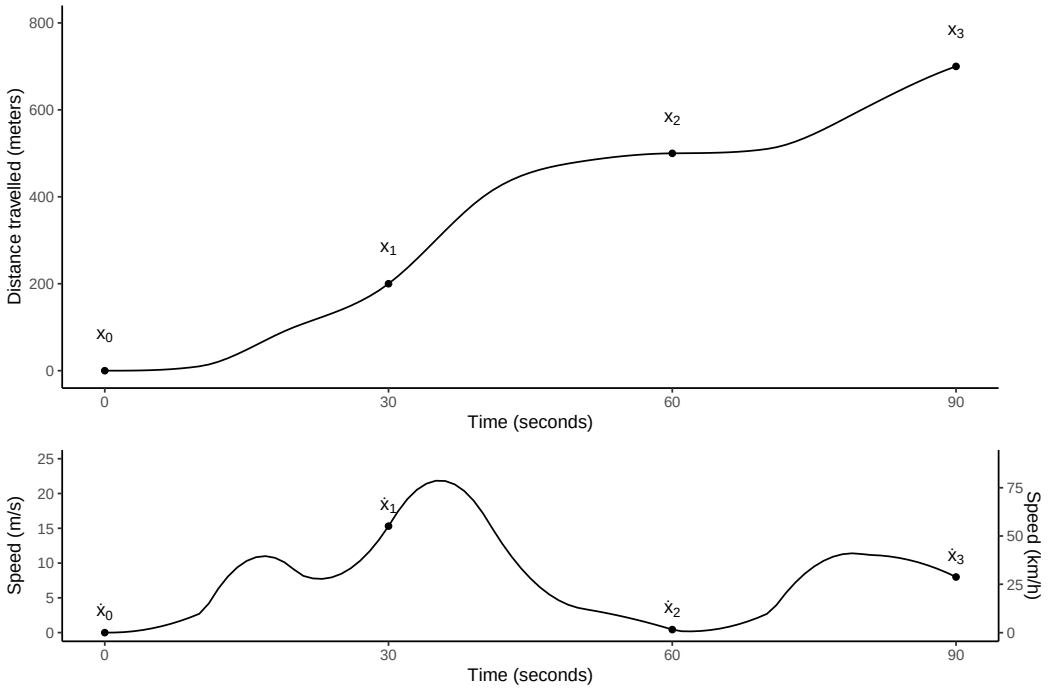


FIGURE 3.2: Visualisation of vehicle state showing *distance travelled* and *speed*. The distance travelled, x, over time includes points of observed states $\boldsymbol{x}_k, k = 0, \ldots, 3$. The gradient of distance over time is speed, $\dot{x}$: steeper sections of the top graph correspond to higher speeds, and vice versa. The speed graph also includes a second right-hand axis with speed in kilometers per hour (km/h) for reference.

which I have demonstrated is not the case. Instead, we use a *particle filter* due to its high flexibility and ability to explore a broad set of plausible trajectories (Hans et al., 2015) and excels in situations where there are multiple distinct possiblities, as in figure 3.1 (Ulmke and Koch, 2006). This makes it well suited for transit modelling, and has been used in many other vehicle tracking applications (Davidson, Collin, and Takala, 2011; Gustafsson, 2010; Gustafsson et al., 2002; Ulmke and Koch, 2006).

The vehicle model itself involves describing bus behaviour algorithmically to infer the vehicle's *unknown state* $\boldsymbol{x}_k$ from a sequence of GPS observations. Since buses service routes with a known path, we represent their position using *distance travelled* along the route's path, which we denote x_k at time t_k. Additionally, we are interested in the *speed* of the vehicle, $\dot{x}_k$. Therefore, the underlying, unknown, and *unobservable* state of a vehicle at time t_k is denoted

$$\boldsymbol{x}_k = [x_k, \dot{x}_k]^\top . \tag{3.1}$$

Figure 3.2 graphs a series of vehicle states with time along the x-axis and distance travelled (in meters) along the y-axis. The derivative (gradient) of the curve represents the vehicle's speed. So, for example, the vehicle state at time $t_1 = 30$ seconds is $\boldsymbol{x}_1 = [x_1, \dot{x}_1]^\top = [200\text{ m}, 15\text{ m/s}]^\top$. The remainder of this section presents a model, along with a particle filter implementation of it, to estimate vehicle state.

The particle filter represents the vehicle's state using a sample of *particles*, each with an individual state, $\boldsymbol{x}_k^{(i)}$. We can think of each particle as an "imaginary bus" which represents one possible state (and associated trajectory) of the true bus. Together, the particles approximate the posterior distribution of the vehicle's state at time t_k given all observations up to time t_k, denoted $\boldsymbol{y}_{1:k}$. The posterior distribution is expressed using the Dirac delta measure, δ, (appendix B.1):

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) \approx \sum_{i=1}^{N} w_{k-1}^{(i)} \delta_{\boldsymbol{x}_k^{(i)}}(\boldsymbol{x}_k)\,. \tag{3.2}$$

Before implementing a recursive Bayesian model for these data, we need a *transition function*, a *measurement function*, and a *likelihood*. The transition function, f, describes how a vehicle behaves between two consecutive observations: that is, the shape of the curves in figure 3.2. This allows us to *predict* the future state of the vehicle *given its last known state*. Additionally, we include the *system noise* parameter σ m/s^2, which represents the rate of change of speed. The measurement function, h, describes the relationship between the vehicle's state, $\boldsymbol{x}_k$, and the observation of this state, $\boldsymbol{y}_k$, which is a GPS location with an associated *measurement error*, ε. We represent this using two simple equations,

$$\begin{aligned} \boldsymbol{x}_k &= f(\boldsymbol{x}_{k-1}, \sigma^2), \\ \boldsymbol{y}_k &= h(\boldsymbol{x}_k) + \varepsilon^2. \end{aligned} \tag{3.3}$$

The *likelihood* function quantifies the plausibility of an observation $\boldsymbol{y}_k$ given an underlying state $\boldsymbol{x}_k$.

We now discuss the details of the two components of the model and their implementations. The *prediction step* (section 3.1.1) supplies an estimate of the bus's state *before observing it* by using a model of bus behaviour. Then the *update state* (section 3.1.2) combines the prediction and observation, using the measurement function

and likelihood, to obtain a posterior estimate of the vehicle's state.

3.1.1 Predicting vehicle state: the transition function

The aim of the prediction step is to obtain a *prior predictive distribution* of the vehicle's state at time t_k based on its previous state at time t_{k-1}, $p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$. Using the model definition in equation (3.3) along with the particle filter approximation of state in equation (3.2), we can write the prior prediction of the vehicle's state as

$$p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) \approx \sum_{i=1}^{N} w_{k-1}^{(i)} \delta_{\boldsymbol{x}_k^{(i)}}(\boldsymbol{x}_k), \tag{3.4}$$

where $\boldsymbol{x}_k^{(i)} = f(\boldsymbol{x}_{k-1}^{(i)}, \Delta_k, \sigma^2)$ and $\Delta_k = t_k - t_{k-1}$.

The transition function f is where we define the vehicle behaviours mentioned earlier. Kalman filter applications are limited to linear transformations of the state which can be expressed in a *transition matrix*. However, equation (3.4) shows that, in the case of the particle filter, the transition function is applied to each particle independently, allowing for a lot more flexibility in f. We now describe the various model components of f, implemented as an algorithm in the 'transitr' package (see section 3.3 for details). The core components are vehicle motion (speed and acceleration along a known path), stopping behaviour at known locations (bus stops and intersections), as well as the ability to handle several other scenarios to avoid degeneration.

Component A: vehicle motion

The first behaviour to consider is the vehicle's motion along a known path: the *speed* at which a vehicle is travelling will affect where it ends up. Since speed, $\dot{x}$, is the derivative of distance travelled over time, if $x = d(t)$, then $\dot{x} = d'(t)$, which is shown visually back in figure 3.2. It follows that we can predict a vehicle's future state, given its current state (distance travelled and speed) and system noise, σ, which is interpreted as *the average change in speed per second*:

$$\boldsymbol{x}_{k|k-1} = f_{A1}(\boldsymbol{x}_{k-1|k-1}, \Delta_k, \sigma) = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \Delta_k \dot{x}_k \\ \dot{x}_{k-1} + v_k \end{bmatrix} \tag{3.5}$$

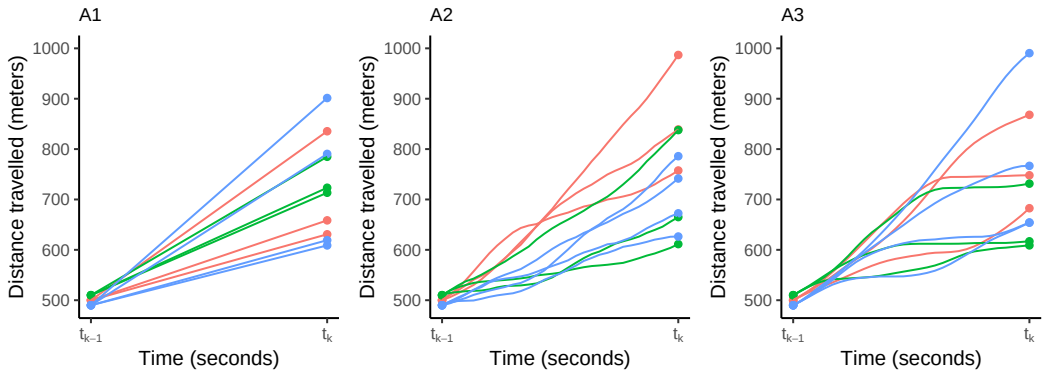


FIGURE 3.3: Simulated particle trajectories using three transition functions, f_{A1}, f_{A2}, and f_{A3} from left to right. Points have been coloured by their parent (after resampling) to demonstrate the affect of system noise.

where

$$v_k \mid \boldsymbol{x}_{k-1}, \sigma^2 \sim \mathcal{N}_T\left(0,\, \sigma^2,\, -\dot{x}_{k-1},\, 30 - \dot{x}_{k-1}\right). \tag{3.6}$$

The noise term is truncated to ensure the new speed is both positive and under 30 m/s, which is 108 km/h (the maximum road speed in Auckland is 100 km/h). The resulting transition function, f_{A1}, is easily implemented by simulating, for each particle, a new speed before calculating the final state using equation (3.5). A similar model was used by Cathey and Dailey (2003) and Dailey et al. (2001).

Figure 3.3 demonstrates transition model f_{A1} (left) using a sample of $N = 10$ particles which, at time t_{k-1}, take one of three unique states (due to resampling in the previous iteration, section 2.4.2). The points have been coloured by their initial state to demonstrate the effect of adding system noise *before* transitioning to allow the particles to disperse. Not doing so would, in this instance, yield only three unique states in the final predictive distribution.

The main issue with this model is that it assumes constant speed between observations, which—given Auckland traffic—is unlikely to be the case. In figure 3.2, we showed speed varying over time, even between observations. To model this, we update the vehicle's state by reusing f_{A1} iteratively[3] Δ_k times, setting $\Delta_k = 1$ in each iteration. Thus,

$$\boldsymbol{x}_{k|k-1} = f_{A2}\left(\boldsymbol{x}_{k-1|k-1}, \Delta_k, \sigma\right) \tag{3.7}$$

[3]This is one advantage of the particle filter—we can easily perform iterative updates!

is implemented by initialising a temporary value $z_0 = x_{k-1|k-1}$ and computing, for $s = 1, \ldots, \Delta_k$,

$$z_s = \begin{bmatrix} z_s \\ \dot{z}_s \end{bmatrix} = \begin{bmatrix} z_{s-1} + \dot{z}_s \\ \dot{z}_{s-1} + v_s \end{bmatrix}, \tag{3.8}$$

leading to the final value of $x_{k|k-1} = z_{\Delta_k}$. The system noise v_s is sampled in each iteration from equation (3.6). The second graph in figure 3.3 shows the effect this has on the particles' trajectories.

Of course, if speed is the first derivative of the vehicle's trajectory function, then *acceleration* is the second, $\ddot{x} = d''(t)$. In this case, we add a third component to the vehicle's state and consider speed similarly to distance. The transition function

$$x_{k|k-1} = f_{A3}\left(x_{k-1|k-1}, \Delta_k, \sigma\right) \tag{3.9}$$

is also iterative. As before, we initialize $z_0 = x_{k-1|k-1}$ and iterate the following for $s = 1, \ldots, \Delta_k$ to obtain $x_{k|k-1} = z_{\Delta_k}$:

$$z_s = \begin{bmatrix} z_s \\ \dot{z}_s \\ \ddot{z}_s \end{bmatrix} = \begin{bmatrix} z_{s-1} + \dot{z}_s \\ \dot{z}_{s-1} + \ddot{z}_s \\ \ddot{z}_{s-1} + v_s \end{bmatrix}. \tag{3.10}$$

System noise (interpreted as *the average change in acceleration per second*) is applied to the acceleration term and truncated to ensure the speed remains positive and less than 30 m/s,

$$v_s \,|\, z_{s-1}, \sigma^2 \sim \mathcal{N}_T\left(0,\, \sigma^2,\, -\dot{z}_{s-1} - \ddot{z}_{s-1},\, 30 - \dot{z}_{s-1} - \ddot{z}_{s-1}\right). \tag{3.11}$$

These trajectories are shown in the third graph of figure 3.3. The main issue with f_{A3} is that it is difficult to parameterize and constrain the acceleration to ensure speed remains within the interval $(0, 30)$ while preserving the function's ability to generate plausible trajectories.

Component B: stopping at nodes

In section 2.3, we discussed using a road network consisting of nodes—either bus stops or intersections—and the roads connecting them. This *segmentisation* allows each route to be expressed as a *sequence of nodes* and connecting *road segments*. Component A of the transition model deals with vehicle behaviour along the road segments; we now examine bus behaviour at nodes.

There are two types of nodes in our framework: *bus stops* where passengers can board and disembark, and *intersections* where buses (and other road users) wait depending on congestion levels and traffic lights. In each case, the bus's behaviour involves:

1. if the bus stops:
   (a) deceleration to zero,
   (b) wait, and
   (c) acceleration to traffic speed;
2. otherwise, continue as though there is no node.

Only step 1b depends on the type of node.

Bus stops A bus's wait time at a bus stop is referred to as *dwell time* and consists of three phases (Hans et al., 2015; Meng and Qu, 2013; Robinson, 2013; Wang et al., 2016):

i. doors open;

ii. passengers board and alight; and

iii. doors close, and vehicle waits for a gap in the traffic.[4]

Phases i and iii can be combined with 1a and 1c from above (deceleration and acceleration, respectively) into a single parameter, γ, which is assumed constant across all buses, stops, and routes.

[4] In New Zealand, buses do not get right-of-way, so it is common for them to be stuck for a while in a bus bay.

Phase ii includes the time taken for passengers to get on and off the bus, which varies significantly between stops, routes, and time of day. There are many ways to model service time: an exponential (Hans et al., 2015), regression models (Shen and Li, 2013), or a real-time Kalman filter (Shalaby and Farhan, 2004). Here, I present a truncated Normal distribution (denoted $\mathcal{N}_T$) using the mean and variance of historical dwell times, truncated at zero. The service time at stop m along a route ($m = 1, ..., M-1$; dwell time does not apply to the last stop) is denoted $\tilde{d}_m$, modelled by its mean and variance, τ_m and ω_m^2, respectively, as estimated from historical data:

$$\tilde{d}_m \mid \tau_m, \omega_m^2 \sim \mathcal{N}_T\left(\tau_m, \omega_m^2, 0, \infty\right). \tag{3.12}$$

For each bus stop, we also have a *stopping probability* $\pi_m \in (0, 1)$. For the first and last stops, $m \in \{1, M\}$, this is unity; for the remainder, we have no data with which we can reliably estimate π_m, so, for now, we assume all stops have the same values specified by a global π parameter (see section 3.3.4 for details on this). The outcome of whether a bus stops at a bus stop, p_m, is a Bernoulli trial,

$$p_m \mid \pi_m \sim \text{Bernoulli}\left(\pi_m\right). \tag{3.13}$$

The total dwell time of a bus at stop m can be expressed as

$$d_m = p_m(\gamma + \tilde{d}_m), \tag{3.14}$$

which, under the particle filtering framework, is straightforward to implement. Figure 3.4 displays the dwell time distribution at a stop.

One other scenario for bus stops is a *layover*, which is when a bus arrives early at a major stop[5] and waits until the scheduled departure time. In the GTFS stop times table, layovers are denoted by an arrival time earlier than the corresponding departure time, as shown in table 3.1. Again, under the particle filtering framework, this is simple enough to implement: if the particle arrives early, it waits until the scheduled departure time before leaving. However, it is not always the case that buses will wait for the layover, so we add a probability of adherence, similar to the

[5]This is more common on longer routes in an attempt to keep them to schedule.

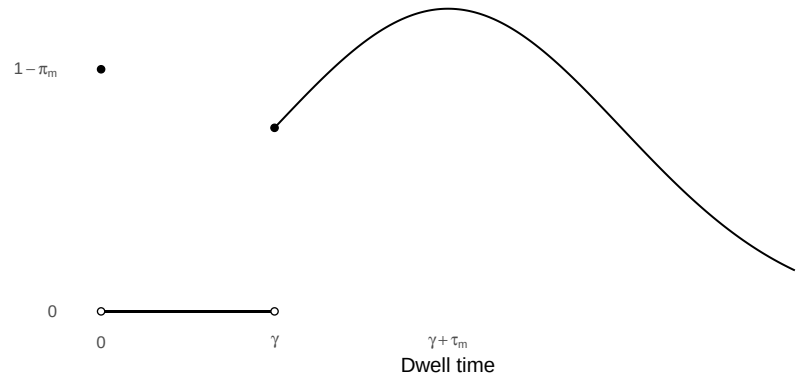


FIGURE 3.4: Probability density function of dwell time at bus stop m.

TABLE 3.1: Arrival times for a trip with a layover stop (marked with an asterix, $\star$). This is indicated in the GTFS data by a departure time later than the arrival (in this case the difference is one second).

Stop	Arrives	Departs
1	19:30:00	19:30:00
2	19:30:36	19:30:36
3	19:31:33	19:31:33
4	19:32:04	19:32:04
5	19:32:59	19:33:00 *
6	19:33:42	19:33:42
7	19:34:14	19:34:14

stopping probability in equation (3.13). However, since this has greater implication for arrival time prediction, we defer the details for chapter 5.

Intersections Modelling intersections is much the same as bus stops, with a few differences:

i. the bus doors do not open, passengers do not get on or off, so there is no longer a γ parameter;

ii. the bus does not necessarily stop at the node location, since there may be a queue;

iii. depending on the length of the queue and the type of intersection, the bus may require more than one "light phase" to pass through the intersection (for traffic lights), OR the bus may creep forward slowly (at uncontrolled (give-way) intersections and roundabouts).

However, we had no reliable intersection location information available, so were unable to explore this component of the model thoroughly. Instead, I describe here a single behaviour but note that it could easily be modified in the future.

We model the wait time w at intersection ℓ with an exponential distribution with rate ν_ℓ^{-1}:

$$\tilde{w}_\ell \,|\, \nu_\ell \sim \mathcal{E}\left(\nu_\ell^{-1}\right). \tag{3.15}$$

As at bus stops, the stopping outcome r_ℓ is a Bernoulli trial with probability ρ_ℓ,

$$r_\ell \,|\, \rho_\ell \sim \text{Bernoulli}\left(\rho_\ell\right), \tag{3.16}$$

leading to an overall wait time of

$$w_\ell = r_\ell \tilde{w}_\ell. \tag{3.17}$$

However, from point iii above, once the wait time is over, we conditionally allow the bus to move forward with the possibility of stopping again. In practice, this would be proportional to the distance of the vehicle from the intersection and other factors. As mentioned in section 2.2.1, this is not as easy as it seems since observations may be linked to a waypoint at the intersection[6] instead of where the bus is, so we cannot determine the bus's distance from the intersection reliably.

3.1.2 Likelihood

The second component of a recursive Bayesian model is the update step, which incorporates the likelihood of the data into the predicted state, $p(\boldsymbol{y}_k \,|\, \boldsymbol{x}_k)$. In the particle filter, as discussed in section 2.4.2, updating is performed by reweighting each of the particles based on their respective likelihoods, $p(\boldsymbol{y}_k \,|\, \boldsymbol{x}_k^{(i)})$. That is,

$$p(\boldsymbol{x}_k \,|\, \boldsymbol{y}_{1:k}) \approx \sum_{i=1}^{N} w_k^{(i)} \delta_{\boldsymbol{x}_k^{(i)}}(\boldsymbol{x}_k) \tag{3.18}$$

where

$$w_k^{(i)} = \frac{w_{k-1}^{(i)} p(\boldsymbol{y}_k \,|\, \boldsymbol{x}_k^{(i)})}{\sum_{j=1}^{N} w_{k-1}^{(j)} p(\boldsymbol{y}_k \,|\, \boldsymbol{x}_k^{(j)})}. \tag{3.19}$$

If necessary, the particles are resampled with replacement (discussed further in section 3.3.1).

[6] Actually, in the middle of it.

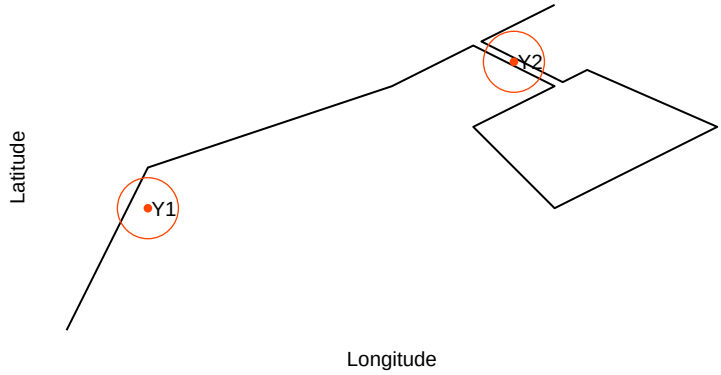


FIGURE 3.5: Observations of a vehicle travelling along a path. The red points indicate the reported GPS positions, with circles indicating the GPS error associated with each observation. Observation Y1 is easy to map to the route, while Y2 is more complicated: it could be *entering* or *exiting* the loop.

The likelihood function is where the particle filter is superior in this application. Were we to model the vehicle's state using a Kalman filter, we would need to somehow compare a distribution in one dimension (distance travelled) with an observation in two dimensions (GPS coordinate). Cathey and Dailey (2003) used an optimisation technique to obtain an estimated observation of distance travelled based on the observed location, which they then used as data for a Kalman filter. However, as demonstrated in figure 3.5, there are situations where the "maximised" location may be wrong, in which case the resulting state will be erroneous.

The particle filter effectively checks to see how plausible the observation is assuming each particle is the truth, allowing us to weight each particle by its plausibility. Since there are two types of data, two likelihood functions are required: one for GPS observations, and a second for trip updates. There is no need to match the observations to distance: instead, each particle's state can be transformed into a GPS coordinate, making it directly comparable to the vehicle's reported location (see appendix B.2).

GPS vehicle locations

For the GPS location update, the inherent GPS error, ε, needs to be filtered out to get better estimates of the vehicle's state. Examples of positions are shown in figure 3.5. A particle's likelihood should represent the *geographical closeness* of the estimated position to the vehicle's reported position. Therefore, we want the likelihood to

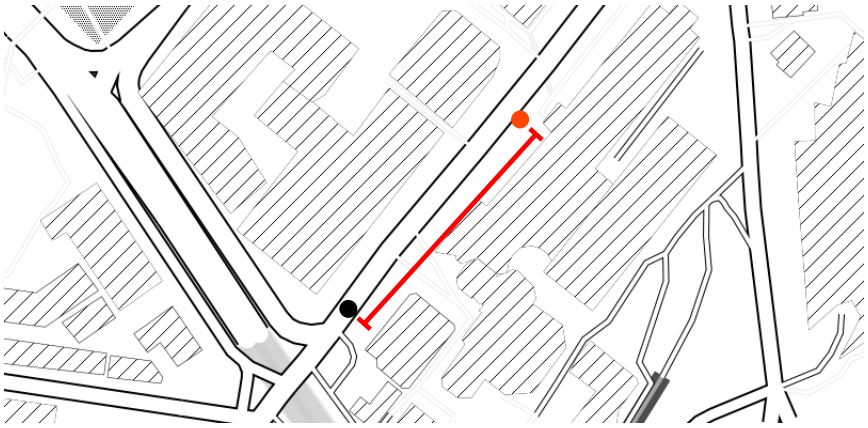


FIGURE 3.6: GPS distance between two points. The vehicle's reported location is shown in orange; one particle estimate of its location is in black. The desired distance is denoted by the red line.

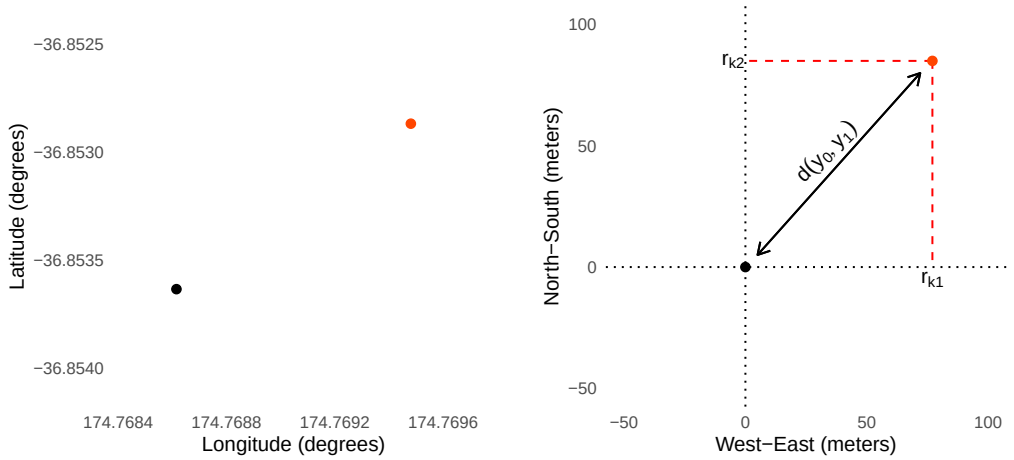
depend on the *distance between the observed and predicted* vehicle locations. The first step to computing the likelihood is therefore to calculate the GPS position of the *particle*, $\tilde{\boldsymbol{y}}_k^{(i)}$, by way of the *measurement function*,

$$\tilde{\boldsymbol{y}}_k^{(i)} = h(\boldsymbol{x}_k^{(i)}, \mathcal{P}), \tag{3.20}$$

which is simply a deterministic function given the route's path, $\mathcal{P}$, a sequence of latitude-longitude pairs and the cumulative distance along the line. Details are given in appendix B.2. Once the geographical position of the particle is obtained, it can be compared to the observed vehicle location, as shown in figure 3.6.

Computing the distance between two GPS coordinates can be achieved using several formulae, each with varying levels of accuracy. Since all the distances are going to be (very) small, the *equirectangular projection* is sufficiently accurate for computing geographical distances (Snyder, 1998). This projection transforms the point $\boldsymbol{y}_1 = [\lambda_1, \phi_1]^\top$, where latitude ϕ and longitude λ are in radians (the width of one longitudinal radian depends on latitude), onto a surface with meters on both axes, centred on the point $\boldsymbol{y}_0 = [\lambda_0, \phi_0]^\top$ and using the Earth's radius $R = 6.371 \times 10^6$ meters:

$$g\left(\boldsymbol{y}_1 \,|\, \boldsymbol{y}_0\right) = \begin{bmatrix} x \\ y \end{bmatrix} = R \begin{bmatrix} (\lambda_1 - \lambda_0)\cos\phi_0 \\ (\phi_1 - \phi_0) \end{bmatrix}. \tag{3.21}$$



(A) GPS coordinates (B) Projected points

FIGURE 3.7: Equirectangular projection of GPS coordinates (A) onto a flat surface (B), which allows the easy calculation of distance between a particle (black) and the vehicle's observed location (orange).

Now the distance between the points is easily computed using the *Euclidean distance* between their transformed coordinates,

$$d\left(\boldsymbol{y}_0, \boldsymbol{y}_1\right) = \sqrt{x^2 + y^2}, \tag{3.22}$$

as shown visually in figure 3.7. Note that conversion from degrees to radians is achieved by multiplying degrees by $\frac{\pi}{180}$.

Now that spherical observations can be compared on a flat surface, we assume that GPS observations are distributed as a multivariate Normal random variable around the true position of the vehicle on the ground, with a GPS error of ε, and that the variation does not depend on direction. That is, if the observation $\boldsymbol{y}_k$ is projected using equation (3.21) conditional on the true position $h(\boldsymbol{x}_k)$, then the projected point will be a multivariate random variable $\boldsymbol{r}_k$. More simply,

$$\boldsymbol{r}_k = g\left(\boldsymbol{y}_k \,\middle|\, h(\boldsymbol{x}_k)\right) \sim \mathcal{N}\left(\mathbf{0}, \varepsilon^2 \mathbf{I}\right), \tag{3.23}$$

as is shown graphically in figure 3.7b.

From equations (3.21) to (3.23), the distance between the true and observed locations is the magnitude of the error $\boldsymbol{r}_k$,

$$d\left(\boldsymbol{y}_k, h(\boldsymbol{x}_k)\right) = ||\boldsymbol{r}_k|| = \sqrt{r_{k1}^2 + r_{k2}^2}. \tag{3.24}$$

However, this error can also be expressed in terms of two independent, standard Normal random variables $z_1, z_2 \sim \mathcal{N}(0, 1)$, such that $r_{jk} = \varepsilon z_j$ for $j = 1, 2$, leading to

$$d\left(\boldsymbol{y}_k, h(\boldsymbol{x}_k)\right) = \sqrt{(\varepsilon z_1)^2 + (\varepsilon z_2)^2} = \varepsilon\sqrt{z_1^2 + z_2^2}. \tag{3.25}$$

Since the distribution of two squared standard Normal random variables is χ^2 distributed with 2 degrees of freedom, which is itself exponential with rate 0.5, then

$$z_1^2 + z_2^2 \sim \mathcal{E}\left(\frac{1}{2}\right). \tag{3.26}$$

Further, if $X \sim \mathcal{E}(\theta)$, then $cX \sim \mathcal{E}\left(\frac{\theta}{c}\right)$, so the squared distance is exponential with mean dependent only on the GPS error,

$$d\left(\boldsymbol{y}_k, h(\boldsymbol{x}_k)\right)^2 \,\big|\, \varepsilon \sim \mathcal{E}\left(\frac{1}{2\varepsilon^2}\right). \tag{3.27}$$

Given a particle state estimate of $\boldsymbol{x}_k^{(i)}$, the likelihood of a GPS observation using only the distance between two coordinates is now

$$p(\boldsymbol{y}_k \,|\, \boldsymbol{x}_k^{(i)}) = \frac{1}{2\varepsilon^2} \exp\left\{-\frac{d\left(\boldsymbol{y}_k, h(\boldsymbol{x}_k^{(i)})\right)^2}{2\varepsilon^2}\right\}, \tag{3.28}$$

allowing the particles to be reweighted using equation (3.19). This is shown visually in figure 3.8 (darker particles have greater weight), which demonstrates not only the particle reweighting by geographical proximity, but also the particle filter's innate ability to handle loops (as displayed) and other situations involving multimodality.

Trip updates

As well as vehicle position updates from GPS data, GTFS also provides trip updates from arrival and departure information. In many situations, it is difficult to infer

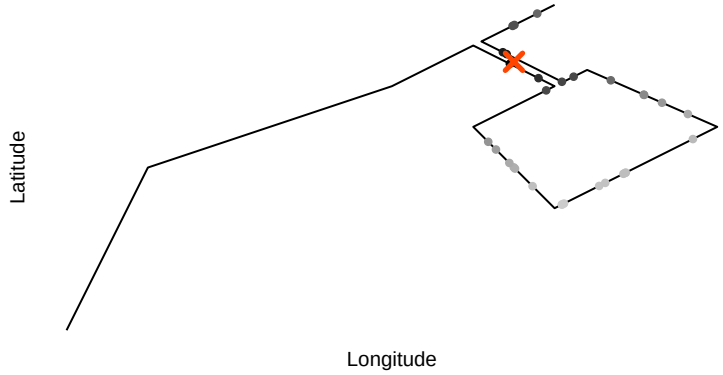


FIGURE 3.8: Particles (black points) are reweighted (darker particles have more weight) based on their geographic proximity to the vehicle's observed location (red cross). There is no need to decide if the bus is going into or coming out of the loop.

a vehicle's trajectory based solely on GPS data, so trip updates are an invaluable part of the update step. In this case, the particle filter prediction step goes ahead as presented in section 3.1.1, but instead of comparing the coordinates, we use the arrival or departure times to compute the likelihood of the particles.

The trip update observations differ from the GPS observations in that there are now three situations which may occur. The observation can be of arrival time at stop m, A_m, or it can be of departure time, D_m. In the latter case, treatment of the observations depends on whether A_m was observed. This gives us three likelihood functions to derive,

- $p(A_m \mid \boldsymbol{x}_k)$, the arrival time likelihood function,
- $p(D_m \mid \boldsymbol{x}_k, \text{arrival missing})$, the departure time likelihood conditional on not having observed arrival time, and
- $p(D_m \mid \boldsymbol{x}_k, \text{arrival observed})$, the departure time likelihood conditional on having observed arrival time.

To compute these likelihoods, we recall the dwell time model described by equations (3.12) to (3.14). Two additional parameters are needed: the actual arrival time of the bus at stop m, α_m, and the measurement error of arrival time in seconds, ξ. The arrival time can be computed for each stop m directly from the model (via interpolation). The departure time is then computed by summing the arrival and dwell times.

The observed arrival and departured times, denoted A_m and D_m, respectively, are modelled as Normal random variables with mean and variance determined by the described model. For arrival time, this is

$$A_m \mid \alpha_m, \xi \sim \mathcal{N}\left(\alpha_m, \xi^2\right), \tag{3.29}$$

and for departure time, using dwell time d_m from equation (3.14), is

$$D_m \mid \alpha_{mr}, d_m, \xi \sim \mathcal{N}\left(\alpha_m + d_m, \xi^2\right). \tag{3.30}$$

In our particle filter implementation, each observation is processed individually: in situations where more than one type of observation is recieved, they are processed in chronological order and the particles reweighted between each. The likelihood for the trip update, using $\tilde{\boldsymbol{y}}_k$ to represent the most recent arrival or departure event at time t_k, is

$$p(\tilde{\boldsymbol{y}}_k \mid \boldsymbol{x}_k^{(i)}) = \begin{cases} \frac{1}{\sqrt{2\pi\xi^2}} \exp\left\{-\frac{(A_m - \alpha_m^{(i)})^2}{2\xi^2}\right\} & \text{for arrival times,} \\ \frac{1}{\sqrt{2\pi\xi^2}} \exp\left\{-\frac{\left(D_m - \left(\alpha_m^{(i)} - d_m^{(i)}\right)\right)^2}{2\xi^2}\right\} & \text{for departure times,} \end{cases} \tag{3.31}$$

allowing the particle sample to be reweighted according to the temporal difference in arrival and departure times as demonstrated in figure 3.9.

3.2 Estimating road speeds

Now that we have estimated the necessary vehicle states and their respective trajectories, we can infer each vehicle's *average speed* along road segment ℓ of its current route, B_ℓ. These estimates are later used to update the *road network state* (chapter 4) and ultimately estimate arrival times (chapter 5).

Estimation of average road speed is performed by first computing the *travel time* along each road segment as the bus traverses the network. To do so, we record the time when the vehicle starts and ends each segment, T_ℓ and T'_ℓ, respectively, and take the difference to obtain the travel time in seconds. By using a particle filter, we record these values for each particle as it is transitioned to each new state. Finally,

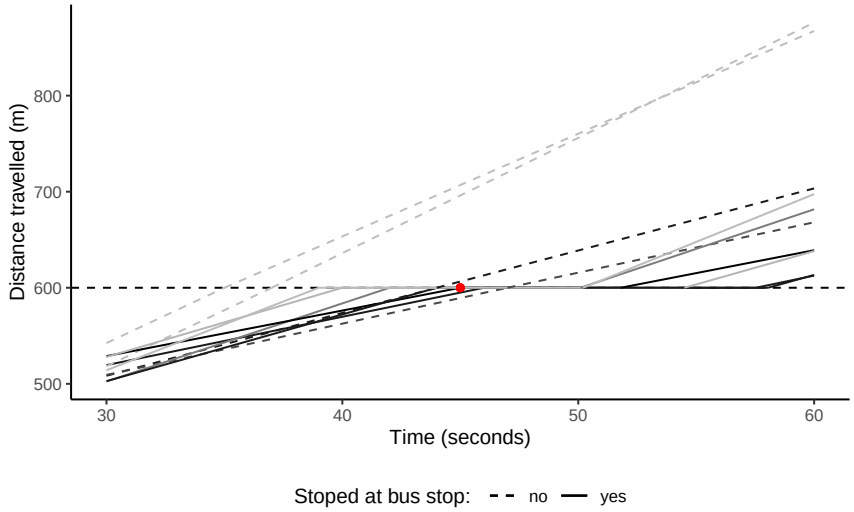


FIGURE 3.9: Particles travelling past a stop with an arrival time observation (red point). Solid lines represent particles that stopped, while dashed lines indicate particles that did not. Darker lines represent particles with bigger weights according to the arrival time likelihood.

transforming to average speed uses the length of the segment, $\mathcal{L}_\ell$, in meters, and the standard speed formula (speed $= \frac{\text{distance}}{\text{time}}$):

$$B_\ell^{(i)} = \frac{\mathcal{L}_\ell}{T'^{(i)}_\ell - T^{(i)}_\ell}. \tag{3.32}$$

Since estimating equation (3.32) is straightforward for each particle, the posterior distribution of the vehicle's average travel time along segment ℓ, given all observations up to and including time t_k, is approximated using the Dirac delta measure,

$$p(B_\ell \,|\, \boldsymbol{y}_{1:k}) \approx \sum_{i=1}^{N} w_k^{(i)} \delta_{B_\ell^{(i)}}(B_\ell)\,. \tag{3.33}$$

In situations where only some particles have completed travel along a segment, the application waits until the next iteration to re-check that all particles have completed it and, if so, the average speed is calculated.

3.2.1 Simulation study

To assess the accuracy of the models presented in section 3.1, vehicle simulations were performed with known road speeds while tracking the vehicle along the route. Three different sampling methods were used to obtain observations:

- uniform sampling with 10 second intervals;
- uniform sampling with 30 second intervals; and
- non-uniform sampling at nodes.

As mentioned in section 2.2.1, the last of these is, in fact, a common feature of the Auckland Transport data; we discuss the complications further in section 3.3.1. In each simulation, we implemented the three variations of the transition function: f_{A1}, f_{A2}, and f_{A3}.

The posterior mean travel time was used to examine and compare the estimation accuracy of the models, which is simple to calculate from the particle filter estimates of travel time using the weighted mean of the sample (appendix B.4):

$$\bar{B}_\ell = \mathbb{E}\left[B_\ell | \boldsymbol{y}_{1:k}\right] = \sum_{i=1}^{N} w_k^{(i)} B_\ell^{(i)}. \tag{3.34}$$

To evaluate and compare the estimation performance of the models, we use root mean square error (RMSE) and mean absolute error (MAE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(X_a - \hat{X}_n\right)^2}, \tag{3.35}$$

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}\left|X_a - \hat{X}_n\right|, \tag{3.36}$$

where X_a is the true travel time and $X_n : n = 1, \ldots, N$ are the model estimates.

Simulation A: general vehicle model

The simulated data, shown in figure 3.10, uses the transition model described by f_{A3} to simulate a vehicle trajectory ignoring bus stops. Observations are obtained using three sampling methods: uniform sampling with high and low frequency, and non-uniform sampling, which is more in line with how the Auckland Transport data is collected.

The goal of the simulation is to estimate the vehicle's average speed along several road segments, as well as the associated uncertainty. The simulation was performed in R (R Core Team, 2018) using $N = 2000$ particles per vehicle, and so the implementation is slightly different from the C++ one defined in section 3.3.1.

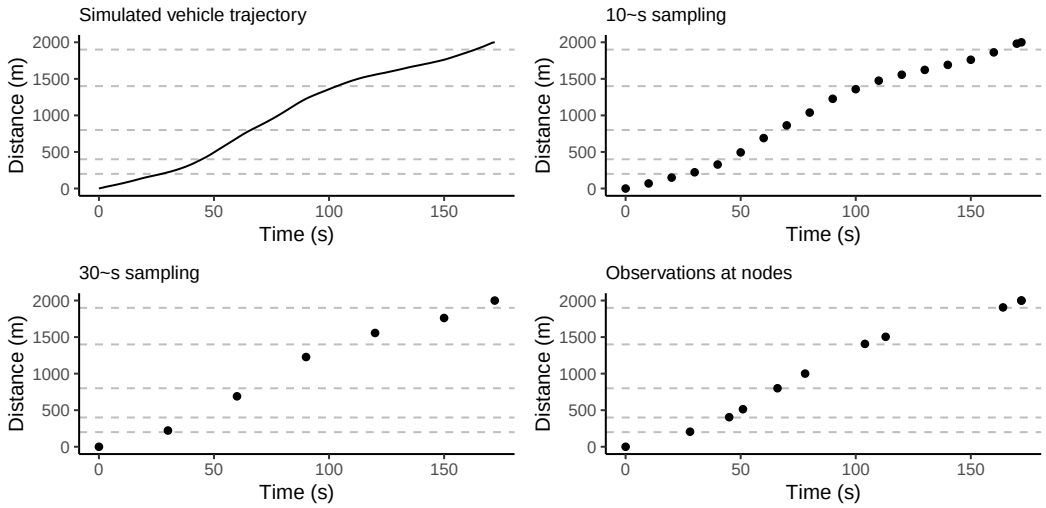


FIGURE 3.10: A simulated vehicle trajectory (top left) for simulation A along five road segments (dashed grey lines). Observations are sampled using three techniques (see text).

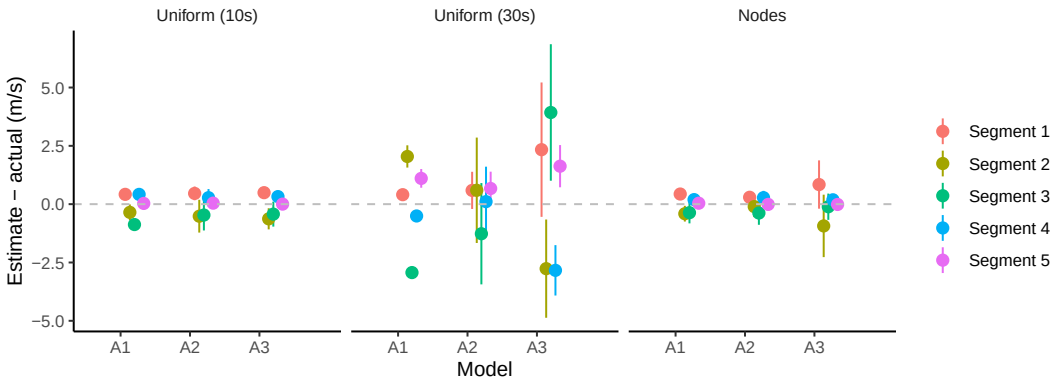


FIGURE 3.11: Simulation A results for the three models (A1, A2, A3) applied to the data from three sampling methods using $N = 2000$ particles. Shown is the travel time prediction error along with its standard deviation.

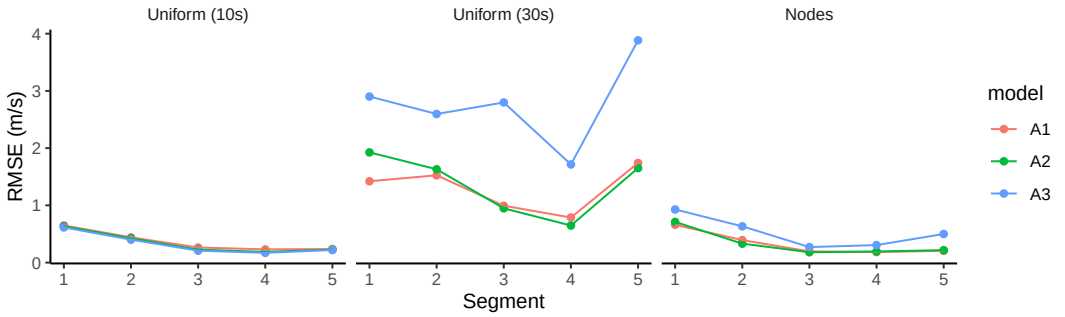


FIGURE 3.12: Speed estimation results for 100 simulations. In each the vehicle trajectory was simulated using a different seed and speed estimated by the mean of the particles, which is compared to the true speed using RMSE.

TABLE 3.2: RMSE and MAE of average speed estimation for simulation A for three models and three sampling techniques.

Sampling method	Model	RMSE (m/s)	MAE (m/s)
Uniform (10s)	A1	0.40	0.27
	A2	0.38	0.26
	A3	0.36	0.24
Uniform (30s)	A1	1.34	0.95
	A2	1.44	1.01
	A3	2.87	2.07
Nodes	A1	0.38	0.25
	A2	0.38	0.25
	A3	0.58	0.36

The results of the simulation applied to the data displayed in figure 3.10 is shown in figure 3.11. Under the high-frequency uniform sampling method, all models perform similarly with high precision (the errors are all close to zero) and accuracy (the uncertainty is small enough that the error bars are barely visible). For the low-frequency sampling, however, model A2 shows slightly better precision than A1 and A3. Finally, for sampling at nodes, the models all perform similarly.

To further examine the comparative performance of the models, we repeated the simulation 100 times using the same segments and sampling points, but varying the underlying trajectory of the vehicle, with the results displayed in figure 3.12. Models A1 and A2 have better accuracy than A3. Most obviously, however, is that the sampling rate significantly affects accuracy. An overall comparison of RMSE and MAE are displayed in table 3.2, which affirms the findings that A3 is less accurate than the other methods (except under high-frequency sampling where they all perform similarly).

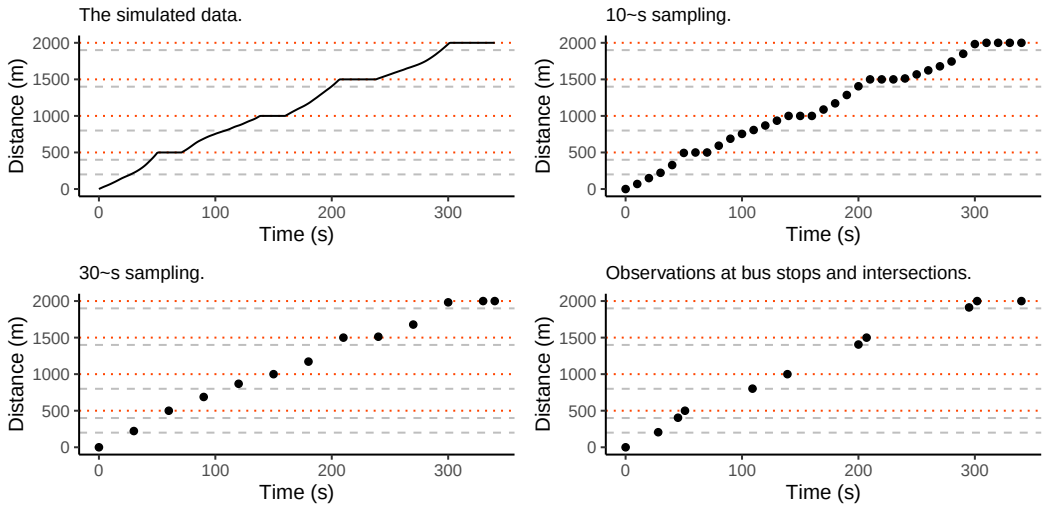


FIGURE 3.13: A simulated vehicle trajectory (top left) for simulation B along five road segments (dashed grey lines) with four stops (dotted orange lines) sampled using the three techniques (see text).

Simulation B: bus stop model

Simulation A assumed the vehicle travelled along the route without stopping. Now, we add bus stop behaviour to the model, as shown in figure 3.13. In the simulated data, the bus stops at all stops with unknown dwell time, and we use $\pi = 0.5$ for the stopping probability in the particle filter when estimating vehicle state. As before, we use 10 second and 30 second sampling rates, as well as observations at nodes (intersections and bus stops). Models B1, B2, and B3 are modified versions of A1, A2, and A3, respectively, but include bus stopping behaviour from section 3.1.1.

The results of the second simulation are shown in figure 3.14, where we see somewhat similar results as before: the models perform equally well under high-frequency uniform sampling, with lower precision and accuracy under low-frequency sampling. For sampling at nodes, the models perform similarly.

Repeating the simulation 100 times with different vehicle trajectories, we can better compare the models (figure 3.15). Uncertainties are now much higher, on average, particularly under low-frequency sampling. The models all perform similarly, though B3 has the worst accuracy overall. Table 3.3 compares the estimates numerically using RMSE and MAE, where we see that within each sampling method the errors are similar. Models B1 and B2 perform similarly, while B3 is again consistently worse except under high-frequency sampling.

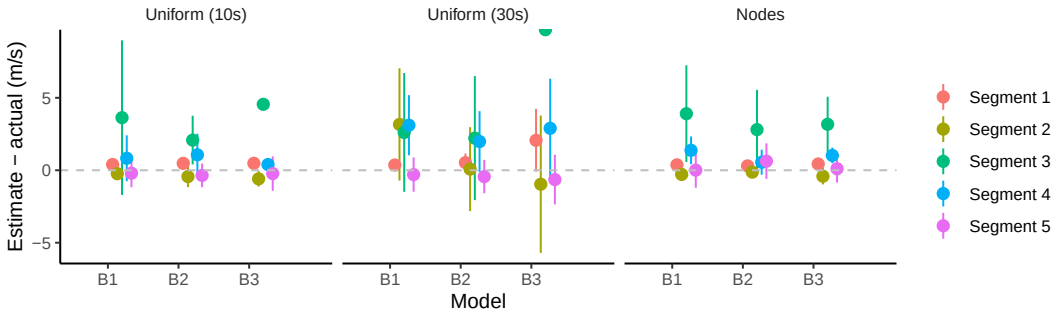


FIGURE 3.14: Simulation B results for the three models (B1, B2, B3) applied to the data from three sampling methods using $N = 2000$ particles. Shown is the travel time prediction error along with its standard deviation.

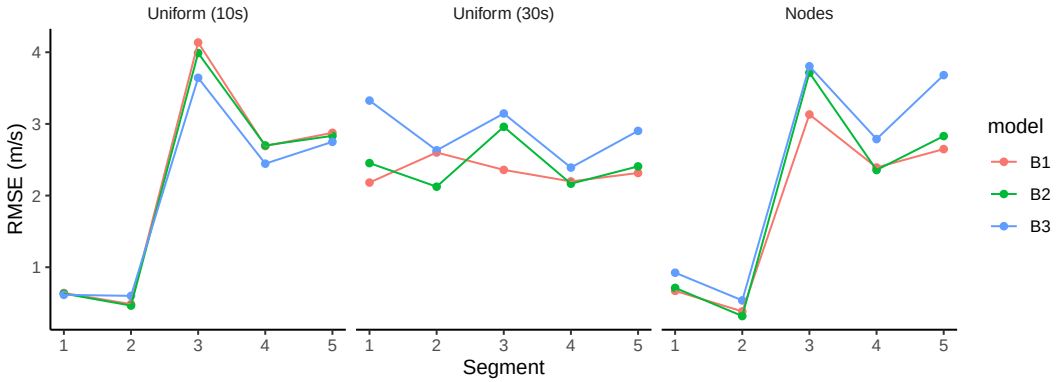


FIGURE 3.15: Speed estimation results for 100 simulations. In each the vehicle trajectory was simulated using a different seed and speed estimated by the mean of the particles, which is compared to the true speed using RMSE.

TABLE 3.3: RMSE and MAE of average speed estimation for simulation B for three models and three sampling techniques.

Sampling method	Model	RMSE (m/s)	MAE (m/s)
Uniform (10s)	B1	2.56	1.40
	B2	2.51	1.35
	B3	2.34	1.35
Uniform (30s)	B1	2.34	1.58
	B2	2.43	1.63
	B3	2.90	2.14
Nodes	B1	2.15	1.31
	B2	2.36	1.35
	B3	2.70	1.57

3.3 Implementing the real-time particle filter

The next major step in this work was to implement the model in real-time using C++. Our implementation involves a single call from the R package 'transitr' which commences an infinite loop within C++ which:

1. fetches the lastest observations;
2. attaches them to an existing vehicle, or creates a new one;
3. updates or initialises vehicle states using the particle filter model described in section 3.1; and
4. estimates average road speeds as vehicles traverse the network as described in section 3.2.

During the process of coding our implementation (section 3.3.1), we inevitably ran into issues, some of which are related to computational complexity, while others are due to imperfections with the data (section 3.3.2). Once the implementation was complete, we needed to estimate the parameters (such as system noise and measurement error) for the model.

To assess the real-time performance of the model and our particle filter implementation, we created a virtual real-time server which serves historical data. This means we can both process the data faster—we do not need to wait in real-time for new observations—and allows us to run the model on the same data with different settings and parameter values for comparison. We used (a subset of) data from Tuesday 8 October 2018 for the simulations used in sections 3.3.2 and 3.3.4, which were carried out on a virtual machine with 8 Intel Xeon 3.00GHz CPU cores and 32 GB of memory, running Ubuntu 16.04 and R 3.4.1. The results themselves were processed locally using R 3.6.0.

3.3.1 C++ particulars

The general construction of the particle filter is straightforward and involves creating `Vehicle` and `Particle` objects (as well as the GTFS objects described in section 2.1).

The `Vehicle` objects are stored within an `std::unordered_map` using their GTFS `vehicle_id` as the key.

When a vehicle is first observed, a new `Vehicle` is created and inserted into the map. Then its state is initialised by creating a vector of N `Particle` objects, each of which is assigned a speed between 0 and 30 m/s, and a distance based on the observation: for GPS observations, map matching is used to determine the approximate location, around which the particles are scattered. In situations where there are multiple candidate locations, particles are distributed uniformly between the minimum and maximum likely distances. For trip updates, the particles are placed at the appropriate stop. When new data for an existing vehicle is observed, the `Vehicle`'s relevant properties are updated (such as position and timestamp). Then each `Particle` is mutated using the transition function from section 3.1 and reweighted according to the likelihood function (section 3.1.2). If the effective sample size drops below the threshold $N_{\text{thres}} = \frac{N}{4}$, the state is resampled with replacement and particle weights reset to $\frac{1}{N}$. See appendix B.3 for details on particle resampling.

Since the vehicles are modelled independently, and they only need to read from the GTFS object (not modify it) the vehicle update step is easily parallelised to use $\tilde{M}$ cores using the OpenMP library (Dagum and Menon, 1998). This enables up to an $\tilde{M}$-fold increase in the speed of this step.

Once the update is complete, the segment index of all particles is obtained, and the *minimum* is used as the vehicle's *current segment*. If this is greater than it was at the end of the previous iteration, the average speed along all intermediate segments is computed by using `std::accumulate`,

```
double avg_speed = std::accumulate(state.begin (), state.end (), 0.0,
  [](double x, Particle& p) {
    return x + p.weight () * p.segment_speed.at (seg_index);
  });
```

along with the variance,

```
double var_speed = std::accumulate(state.begin (), state.end (), 0.0,
  [&avg_speed](double x, Particle& p) {
    return x + p.weight () *
      pow (p.segment_speed.at (seg_index) - avg_speed, 2.0);
  });
```

These observations are then passed to the relevant `Segment` object which contains a vector of new data (used in chapter 4). The `Vehicle` object contains a pointer to its `Trip`, which has a list of `Segment` objects:[7]

```
trip ()->segments ().at (seg_index)->push_data (avg_speed, var_speed);
```

At this point, we have completed the modelling of the vehicle's state and obtained estimates of any available road speed information.

3.3.2 Real-time performance of the particle filter

The two components of the model to assess are the iteration timings and the performance of the particle filter itself. That is, does the program run fast enough to be feasible in real-time, and is the model (and its particle filter implementation) capable of modelling transit vehicles in real-time?

Is the particle filter fast enough? At peak hour on a typical weekday morning, there can be in excess of 1000 buses operating in Auckland. This leads to having more than $1000N$ particles in memory, each being mutated and reweighted approximately once every 30 seconds or so. By varying N, we can control how quickly each set of observations is processed. Figure 3.16 shows the average timings of the vehicle model component of our application, as well as the average time per particle, for varying N. More particles require more processing power, though there is additional overhead during the *resampling* phase. Limiting the frequency of resampling is therefore necessary to make the program run faster, which is why we use the effective sample size, N_{eff}, described in equation (2.21) on equation (2.21).

How does the model perform? To assess how well our model performs in real-time, we repeated the simulation with a range of values of system noise σ, GPS error ε, and the number of particles N. For each simulation, we computed *proportional effective sample size*, *degeneration rate*, and *relative uncertainty*.

The *proportional effective sample size* is the effective sample size relative to N, $\tilde{N}_{\text{eff}} = \frac{N_{\text{eff}}}{N}$. The higher this value, the less often the vehicle's state needs resampling, decreasing the average iteration time. Figure 3.17 shows the effect of N, system noise,

[7] Note that we must still ensure each pointer exists before proceeding, as mentioned in section 2.5.

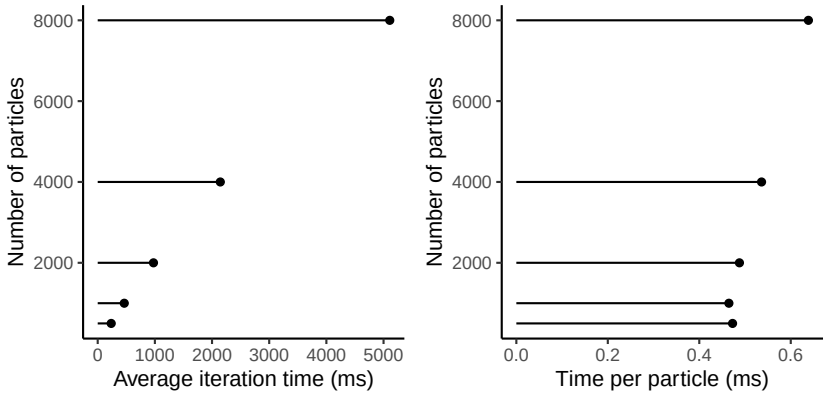


FIGURE 3.16: Timings of the particle filter implemention for varying number of particles. Left: the average iteration time (wall clock). Right: the average time per particle.

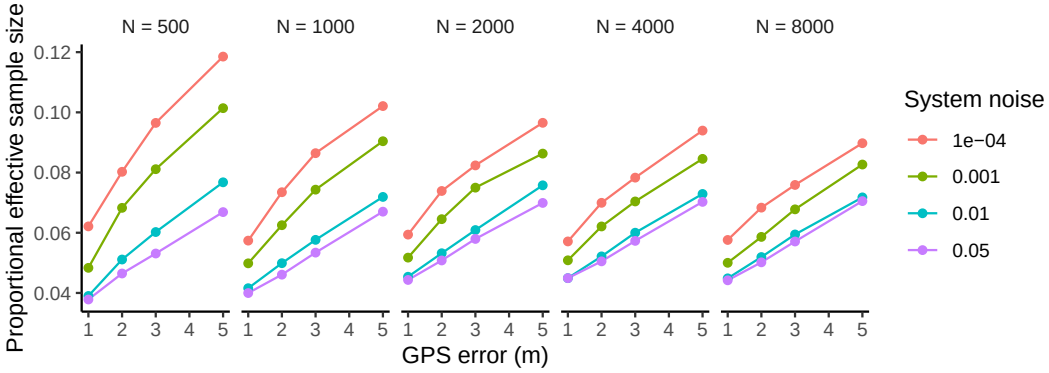


FIGURE 3.17: Proportional effective sample size for varying values of GPS error, system noise, and number of particles.

and GPS error on $\tilde{N}_{\text{eff}}$. The most striking relationship is between $\tilde{N}_{\text{eff}}$ and GPS error: for larger error, more particles retain a high likelihood, and so the total weight is more evenly distributed. Conversely, larger values of system noise result in more variation between particles, leading to fewer particles ending near the observation position and decreasing $\tilde{N}_{\text{eff}}$.

The *degeneration rate* is the proportion of samples in which no particles end near the vehicle's reported position. In this case, all the particle likelihoods tend to zero, so the weights become undefined, resulting in the need to reinitialise the vehicle's state which results in the loss of any vehicle speed information along the most recently travelled road segment(s). We see from figure 3.18 that increasing GPS error or the number of particles reduces degeneration rate while increasing system noise shows a negligible reduction in degeneration rate. Larger GPS error means that particles do not need to end as close to the observed location to have a positive likelihood while

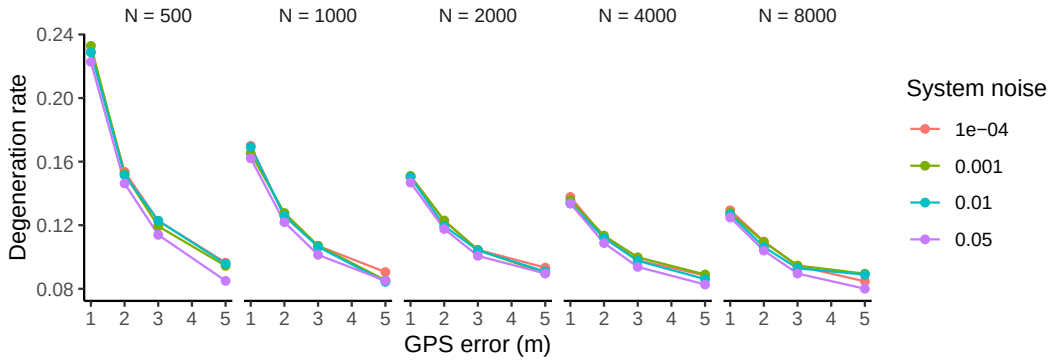


FIGURE 3.18: Degeneration rate for varying values of GPS error, system noise, and number of particles.

increasing N means more chance for a particle to end near the true bus.

Finally, we have *relative speed estimation uncertainty* along roads, which we use to compare the precision of the various models. It is the ratio of the uncertainty of speed estimates for a single simulation compared to the overall uncertainty for all simulations. Let $\boldsymbol{z}_\ell^{e,s}$ be a vector of all vehicle speeds along road segment ℓ during the simulation with GPS error and system noise equal to $e = \{1,2,3,5\}$ and $s = \{0.0001, 0.001, 0.01, 0.05\}$, respectively. The ratio of the uncertainty of speed for the single simulation compared to all simulations along one single road segment is given by

$$v_\ell^{e,s} = \frac{\mathrm{SE}(\boldsymbol{z}_\ell^{e,s})}{\mathrm{SE}(\cup_e \cup_s \boldsymbol{z}_\ell^{e,s})} = M \frac{\mathrm{SD}(\boldsymbol{z}_\ell^{e,s})}{\mathrm{SD}(\cup_e \cup_s \boldsymbol{z}_\ell^{e,s})}, \tag{3.37}$$

where M is the number of simulations, in this case 16, and SE and SD are standard error and standard deviation, respectively. That is, for each segment in each simulation, we have a value representing whether this simulation estimates speed more or less accurately. We then compute the average ratio for all L road segments,

$$\bar{v}^{e,s} = \frac{1}{L} \sum_{\ell=1}^{L} v_\ell^{e,s}. \tag{3.38}$$

A small value of $\bar{v}^{e,s}$ tells us that, on average, the simulation with GPS error e and system noise s estimates road speed *with greater precision* than the other simulations. Figure 3.19 presents these results, where we see the greatest effect on relative uncertainty caused by increasing GPS error. Increasing N produces a small decrease, and changes to system noise show no discernible effect.

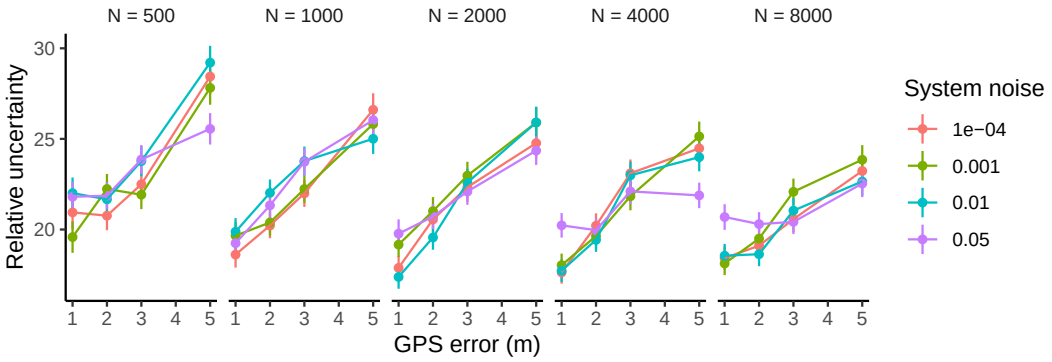


FIGURE 3.19: Relative speed uncertainty for varying values of GPS error, system noise, and number of particles.

The results displayed in figures 3.17 to 3.19 present a *trade-off* between performance and estimation. Increasing GPS error increases the effective sample size, which reduces the frequency of resampling and speeds up each iteration. We also see a reduction in the rate of degeneration, which implies the particle filter is less likely to lose the vehicle and provide the desired speed estimates. However, increasing GPS error also increases the relative uncertainty of speed estimates. Increasing the number of particles generally results in a reduction in both degeneration rate and relative uncertainty, but, from figure 3.16, this comes at the cost of increased computational demand.

3.3.3 Handling invalid data in real-time

Much of the degeneration or *vehicle loss* can be attributed to irregularities with the incoming real-time data. There are two leading causes of this, described below, which can be detected and potentially avoided with a similar solution.

Buses that appear to go backwards One of the prominent assumptions of our model is that the bus cannot go backwards along the route, which is implemented by only allowing for non-negative values of speed. However, a couple of scenarios can lead to the *appearance* of a reversing bus, both of which are related to *GPS waypoints*; that is, the automatic vehicle location systems in Auckland Transport's buses are programmed to not only report their position periodically but also to report their arrival at certain waypoints. These can include bus stops (which result in an arrival or departure time update) as well as some major intersections. The issue is that,

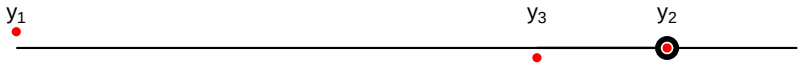


FIGURE 3.20: Three sequential observations of a vehicle approaching an intersection (hollow circle). When the bus nears the intersection, it reports its position exactly at the intersection ($\boldsymbol{y}_2$); however, there is a queue at this intersection, so the next observation ($\boldsymbol{y}_3$) is behind the previous one, demonstrating the "reversing bus" phenomenon.

rather than reporting the bus's GPS position, they report the GPS coordinates of the waypoint itself. This is often acceptable since the bus continues straight past the intersection or bus stop before reporting another position. However, if there is congestion leading into the waypoint, the bus may:

i. approach the waypoint, decide it has arrived, and report its position *at the waypoint*;

ii. get stopped in a queue for some time;

iii. decide it is time for another position update, based on its GPS position.

Figure 3.20 presents an example of this.

The effect this has on the particle filter is that, after observing $\boldsymbol{y}_2$, the vehicle's state (represented by a sample of particles) will all be around the intersection. On receiving the next observation $\boldsymbol{y}_3$, the particles are transitioned *forward* by $(t_3 - t_2)$ seconds, which places them at or beyond the intersection; none of the particles will be near observation $\boldsymbol{y}_3$ and will likely all have a likelihood of zero. In this case, the particle filter has degenerated and needs reinitialising.

A similar situation occurs when approaching a bus stop that has an intersection just before it. Here, the bus gets stopped at the intersection, but not before reporting its position at the stop since it was almost there. A subsequent observation then shows the bus at the intersection, which again appears to involve a reversing bus. The main issue we face is that *we do not know the location of intersections* since there is no easily accessible data for this.[8]

Checking for the bus-reversing scenario in real-time requires mapping of observations onto the route shape; that is, the *inverse measurement function*, allowing us to

[8]We presented an intersection model, but this was more of a demonstration of flexibility, not of functionality.

detect if the bus has gone backwards:

$$h^{-1}(\boldsymbol{y}_k) < h^{-1}(\boldsymbol{y}_{k-1}). \tag{3.39}$$

Of course, the inverse function is not exact, since the true location is unlikely to be positioned exactly on the line (roads have width, and GPS devices have error). To compute h^{-1}, we find the shortest distance along the route's shape that is closer than a threshold of $3\varepsilon^2$, which allows for situations where the route passes the observed location point more than once (such as in loops).

If the observation is determined to be going backwards, we have to decide between:

i. ignoring the current observation; or

ii. ignoring the previous observation and restoring the vehicle's previous state.

If we can determine that the first observation is a pre-emptive observation (that is, at a waypoint), then option (ii) is the logical choice, although this requires storing each vehicle's state *twice*. Otherwise, it is easier to ignore the current observation completely and use option (i). Note that, were intersection locations knowable, we could include this behaviour in the model; as it stands, however, this workaround is required to avoid unnecessary degeneration.

Buses that remain stationary Another situation (which can also occur around unknown intersections) is when the bus does not move between successive observations (or it moves very slowly). For example, the bus may need to turn onto a major road, for which there is a queue of traffic. The bus will report its position when it arrives, and may again report its position after moving a few meters. If our model does not allow for the bus to suddenly slow down in these situations, the particles will all be far ahead of the bus and degeneration will occur.

We can determine if the bus has remained more-or-less stationary by computing the distance between successive observations and comparing to a threshold:

$$d\left(\boldsymbol{y}_{k-1}, \boldsymbol{y}_k\right) < D_{\text{thres}} = \Delta_k \min_i \left(\dot{x}_k^{(i)}\right). \tag{3.40}$$

The threshold here is the expected distance travelled in Δ_k seconds by the slowest particle. Rather than slowing the particles (which could run into the opposite problem once the vehicle passes through the intersection), we sample a temporary speed,

$$\dot{x}_k^{(i)} \sim \mathcal{U}\left(0, \frac{d\left(\boldsymbol{y}_{k-1}, \boldsymbol{y}_k\right)}{\Delta_k}\right), \tag{3.41}$$

which is used for one single iteration.

Again, if we knew the locations of intersections, we could integrate this behaviour into the model. Particles could partake in "creeping" up to the intersection, before quickly accelerating back up to speed on the next road segment. Even then, however, the above behaviour would still need to be included to handle non-intersection related slowing down, for example, when reaching a congested section of a road.

3.3.4 Parameter selection

Now that we understand the issues with the data and can deal with them, we can begin determining the values of the model parameters. Some of these are fixed and constant across all vehicles, routes, and stops, for example, GPS error, ε, system noise, σ, and minimum dwell time, γ. Others inevitably vary between routes, stops, and time of day, such as stopping probability, π, and mean dwell time, τ.

GPS error

The GPS or *measurement* error used in the model has a strong effect on performance, as we saw in section 3.3.2. We can get a simple estimate of GPS error by examining the distribution of observations around the route path; that is, by computing the shortest distance between the route and each observation, which is graphed in figure 3.21. We see two modes at about 0.5 and 2.5 meters, which could be due to a multitude of reasons. One likely one, however, is *road width*, since the route shapes typically run in the middle of the road[9] and the buses drive either side of the centre line.[10] Therefore, on single-lane roads, there is a smaller average distance between the bus and the centre line, while on roads with two or more lanes, the average distance is larger.

[9] I manually inspected a sample of route shapes overlaying a road map.

[10] New Zealand roads are 2.5–3.5 m wide, https://www.nzta.govt.nz/assets/resources/road-traffic-standards/docs/rts-15.pdf.

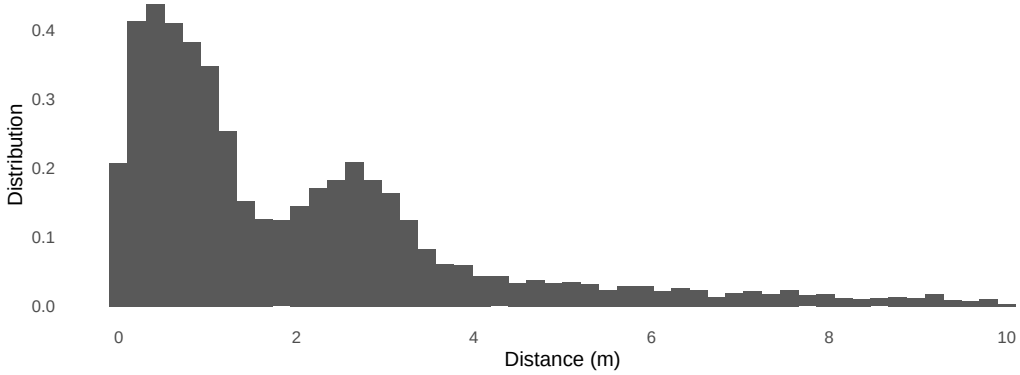


FIGURE 3.21: Distribution of distance from observation to nearest point on the route, truncated to 10 m.

Additionally, some roads have a median (either painted or raised) which further increases the distance between the bus and the "centre line". Some combination of these could result in the distribution shown in figure 3.21.

The other issue is the heavy tail in the distribution of distance to the path, which we truncated to 10 meters to more easily see the modes in figure 3.21. GPS devices usually have good accuracy, but occasionally they may be quite far off of the true location, possibly due to physical interference. Around 6% of bus observations were more than 10 meters from the shape, excluding any observations greater than 50 meters since these were most likely attributed to the wrong trip (and therefore not anywhere near the route path). In the current version of our application, we skip observations that are more than 50 meters from the shape path. Based on figure 3.21 and the results from section 3.3.2, I have used a value of $\varepsilon = 3$ for the results in the following chapters.

System noise

The definition of system noise is model-dependent; for transition models f_{A1} and f_{A2} it is *the average change in speed per second*, while for model f_{A3} it is *the average change in acceleration per second*. From the simulations in section 3.3.2, we demonstrated that system noise affected the performance of the particle filter (how often resampling is required) but neither the degeneration rate nor parameter estimation.

Unlike GPS error, it is not possible to estimate system noise directly from the data. Indeed, most of the time a vehicle's speed is constant but may change suddenly at

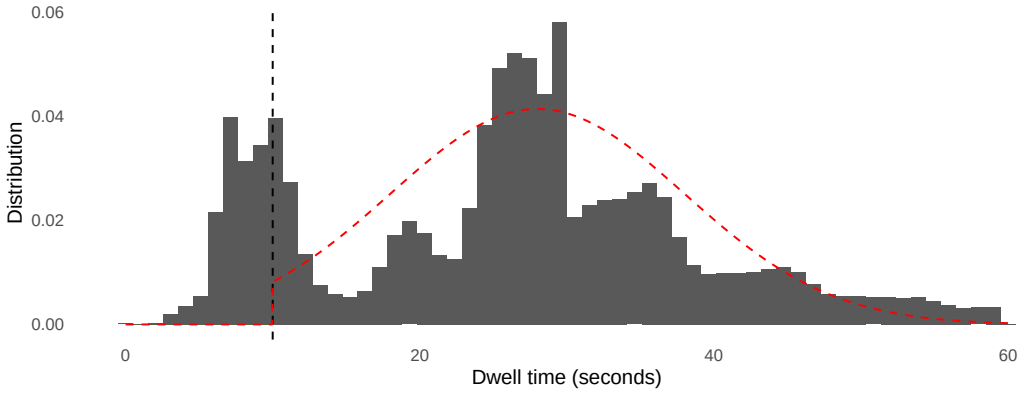


FIGURE 3.22: Distribution of dwell times observed over the course of five days, truncated at one minute. Here we see the "9-second" phenomenon.

specific locations (which are unknown), so the system noise must allow for this. We found that a smaller value of system noise under the second transition model f_{A2} gave the best results in terms of sampling possible trajectories, and as such this was the model used during the simulation. In the remaining chapters, a value of $\sigma = 0.01$ was used, as for large N particles it performs about the same (or better) than the larger value of 0.05 and represents a trade-off between performance and precision.

Dwell times

We can observe a large proportion of dwell times at stops by compiling all those for which we have observed both arrival times A_{srm} and departure times D_{srm} at stop m of trip r on day s giving us a set of dwell times

$$\tilde{D}_{srm} = D_{srm} - A_{srm}. \tag{3.42}$$

There is no precise way to measure the minimum dwell time parameter γ. Hans et al. (2015) used 14.25 seconds as the time lost due to deceleration and acceleration, and 4.1 seconds as the time needed to open and close the doors. Other studies have found values of 3–42 seconds for acceleration and deceleration (Robinson, 2013), and 2–5 seconds for doors to open and close (Meng and Qu, 2013).

The raw data from five days of observations are shown in figure 3.22. Here, we see an interesting pattern with apparent peaks every nine seconds. While we could not determine the precise cause, we assume it to be due to a systematic problem in the

arrival time recording system used to collect the data. However, this makes the first peak at 9 seconds difficult to evaluate: are there dwell times this short (contrary to the literature) or is it an artefact? For example, if a bus passes a stop without stopping and reports an arrival time followed a few seconds later by a departure time, does the 9-second anomaly affect this?

From the historical dwell time data, we were able to estimate the mean and variance of dwell time for each stop m, $\bar{\tau}_m$ and ω_m^2, respectively, which could then be used in the dwell time model. The dwell times for each stop calculated above *include* the minimum dwell time phase. To avoid having to recompute each stop's dwell time parameter whenever γ is changed, we adjusted the mean of each stop's dwell time at run time to account for the minimum dwell time. That is, we use $\tau_m = \bar{\tau}_m - \gamma$ in equation (3.12). The remaining chapters use $\gamma = 10$ seconds for the minimum dwell time parameter.

As for the probability of stopping parameter, π, this cannot be estimated from these data. Using a vague value of $\pi = 0.5$ for the particle filter is sufficient; however, this has a more dramatic effect on arrival time prediction as we will see in chapter 5.

3.4 Chapter contributions

- I developed a particle filter to estimate road speeds using only real-time bus locations and arrival/departure data, implementing vehicle behaviours for bus stops and intersections. This is particularly useful when the truth of whether or not the bus did indeed stop is often unknowable.

- The likelihood function I have presented makes it possible to compare the particles directly to the observation, rather than the other way around (which involves matching map locations to the shape path and introducing potential errors).

Chapter 4

Transit network

The collection of bus stops and intersections, and the roads which connect them, we refer to as a *transit network*. The *state* of this network changes in response to events throughout the city, some of which are predictable—peak traffic, for example—while others are not, such as accidents or weather events. In chapter 3, we measured this state by tracking individual vehicles as they moved through the network. Now we discuss the network state itself, how it changes over time, and how we use the speed observations to update it. Finally, we discuss short-term forecasting as could be used for arrival time estimation in chapter 5.

Estimation of the network's state from vehicle travel time observations requires an understanding of the relationship between them. The underlying state of the network determines how fast (or slow) vehicles travel through it, which changes over time. Since speed is unlikely to be constant over segments, we use *average speed* over the length of a segment, calculated using the length of the segment and how long vehicles take to travel along it,

$$\text{average speed (m/s)} = \frac{\text{segment length (m)}}{\text{travel time along segment (s)}}. \tag{4.1}$$

However, there are a variety of factors that may affect the average speed of individual buses, such as driver behaviour, cyclists sharing the road, variance between lanes, or pedestrians crossing the road, to name a few. Thus, even buses travelling along a road at the same time often have different (average) speeds, resulting in a *distribution* of vehicle speeds, as demonstrated in the top half of figure 4.1. To further complicate matters, the observed average vehicle speeds are made with a level of uncertainty,

which is depicted in the second half of figure 4.1.

Previous work has used *travel time* to model the network state (Cats and Loutos, 2015; Gong, Liu, and Zhang, 2013; Reinhoudt and Velastin, 1997; Shalaby and Farhan, 2004; Yu, Lam, and Tam, 2011), while others have used *vehicle speed*, as I introduced above (Čelan and Lep, 2017, 2018; Ma et al., 2019; Xinghao et al., 2013). We found that travel times tended to have long tails which the Kalman filter was less capable of modelling, and required a lot of manual work determining the appropriate parameter values. Speed, however, is independent of road length, so most of the parameter values could take a single value across all segments.

In section 4.1, we develop a model to estimate the distribution of vehicle speeds along roads throughout the transit network. In section 4.2, I describe the real-time implementation of the model, while section 4.3 considers the estimation of model parameters from historical data. The model's forecasting ability can also be improved with historical data, as we discuss in section 4.4 and revisit in chapter 5. Finally, we examine the computational aspects of the model's real-time implementation and its feasibility in section 4.5. As with the vehicle model, we must keep in mind the real-time nature of the application, so computational efficiency is an important consideration throughout.

4.1 A model of average road speed

The relationship between the underlying road speed and the value observed in chapter 3 involves two steps. The first is the relationship between the average traffic speed and the vehicle's true average speed. Second is the relationship between *actual* and *observed* (or, more specifically, estimated) average speed. This type of model is referred to as a *hierarchical (Bayesian) model*, as demonstrated graphically in figure 4.1.

Let us first consider the state of the network at time t_c, which we denote

$$\boldsymbol{\beta}_c = \left[\beta_{1,c} \ \cdots \ \beta_{L,c}\right]^\top, \tag{4.2}$$

a vector containing the current real-time average traffic speed along all L roads in the network. However, modelling the full state would increase computational demand

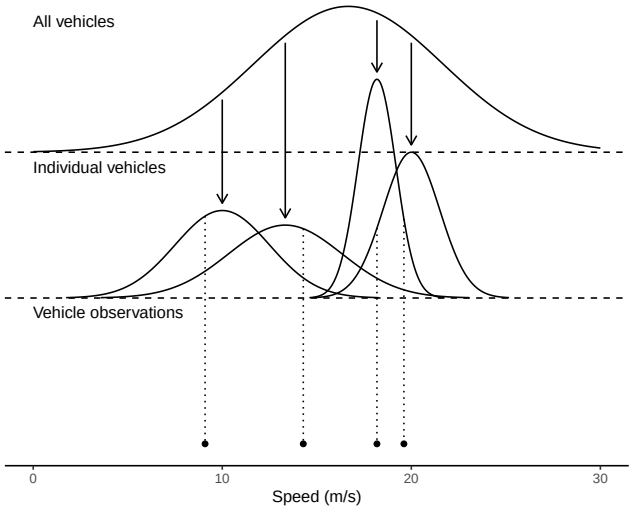


FIGURE 4.1: The hierarchy of speed uncertainty along a single road segment is composed of *between-vehicle variability* and *measurement error*. The top curve shows the underlying distribution of average vehicle speeds with solid arrows representing the true average speed of four vehicles. The second level shows the measurement uncertainty for each vehicle with the observed values represented as dots at the bottom of the graph.

as L gets large, so we consider each road segment ℓ *independently* (this assumption is revisited in section 4.2.3).

Vehicles travelling along road segment ℓ at time t_c have an underlying average speed of $\beta_{\ell,c}$ m/s, which changes with an average rate of q_ℓ m/s^2. Assuming the underlying road state is a Markov process, the model for the evolution of traffic speed along a road with a maximum velocity (speed) of $\mathbb{V}_\ell$ meters per second is

$$\beta_{\ell,c} \,|\, \beta_{\ell,c-1}, q_\ell \sim \mathcal{N}_T\left(\mathcal{F}_c(\beta_{\ell,c-1}),\, (\Delta_c q_\ell)^2,\, 0,\, \mathbb{V}_\ell\right), \tag{4.3}$$

where $\Delta_c = t_c - t_{c-1}$ is the time since the last update and $\mathcal{N}_T$ the truncated Normal density on the interval $(0, \mathbb{V}_\ell)$. This model allows traffic speed to follow temporal trends through the time-dependent transition function $\mathcal{F}_c$ (peak hour traffic, for example) as well as react to real-time events, such as accidents or weather events.

As already mentioned, there are many factors which can affect the speed of a vehicle along a given road segment. To encapsulate this uncertainty, we introduce a single parameter $\boldsymbol{\psi}_\ell$, which describes the *between-vehicle variability*, or the width of the distribution at the top of figure 4.1. The true average speed of bus m along road segment ℓ at time t_c is denoted $B^m_{\ell,c}$, and is assumed to be an observation

from the population distribution with mean $\beta_{\ell,c}$ and variance $\boldsymbol{\psi}_\ell^2$, again truncated appropriately:

$$B_{\ell,c}^m \,\big|\, \beta_{\ell,c}, \boldsymbol{\psi}_\ell \sim \mathcal{N}_T\left(\beta_{\ell,c},\ \boldsymbol{\psi}_\ell^2,\ 0,\ \mathbb{V}_\ell\right). \tag{4.4}$$

Once a vehicle has traversed a segment, we estimate its average speed as $b_{\ell,c}^m$, as estimated in equation (3.34) on equation (3.34). We assume the measurement is made with uncertainty $e_{\ell,c}^m$, as demonstrated in figure 4.1 as the second level of the hierarchy, which is expressed as

$$b_{\ell,c}^m \,\big|\, B_{\ell,c}^m \sim \mathcal{N}\left(B_{\ell,c}^m,\ (e_{\ell,c}^m)^2\right). \tag{4.5}$$

To assess the model, we performed a simulation where all the parameter values are known (based off of the values estimated in section 4.3). Figure 4.2 shows the three stages of the model hierarchy.

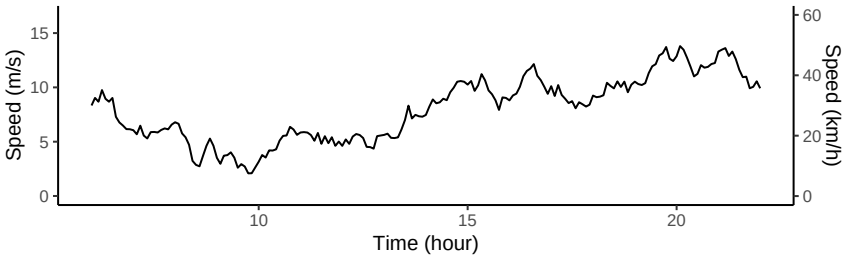
Observed vehicle speeds for a real segment, displayed in figure 4.3, shows some similarities to the simulated data, with one notable difference: the peak-hour spike. This spike is due, in part, to the location of the road, the Symonds Street overbridge:[1] there is often a (very long) queue of buses from many routes converging there on the central city. In this section, we use the simulated data to evaluate the model under "nice" conditions and then test it on this real data to see how well the model copes with these types of phenomenon. The map in figure 4.10 on figure 4.10 shows the location of this and several other road segments.

4.2 Real-time network model

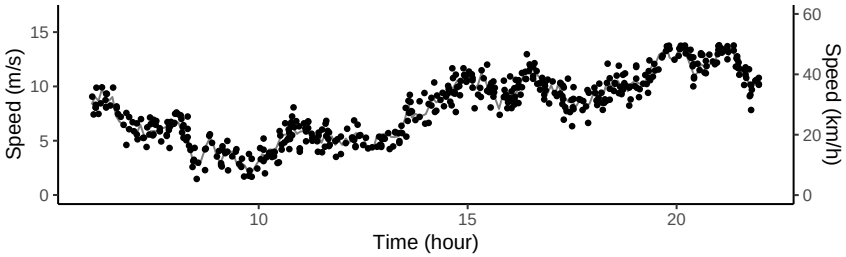
Due to the time constraints of real-time applications, our framework uses a Kalman filter to implement the model described above, which, as previously discussed, is a highly efficient estimation method. This requires the transition and measurement matrices as described in section 2.4.1 on section 2.4.1. The measurement matrix is the identity matrix $\mathbf{I}$ since the data are direct observations of the underlying state (average vehicle speed),

$$\beta_{\ell,c} = \mathbf{H} b_{\ell,c} + w_{\ell,c} = b_{\ell,c} + w_{\ell,c}, \tag{4.6}$$

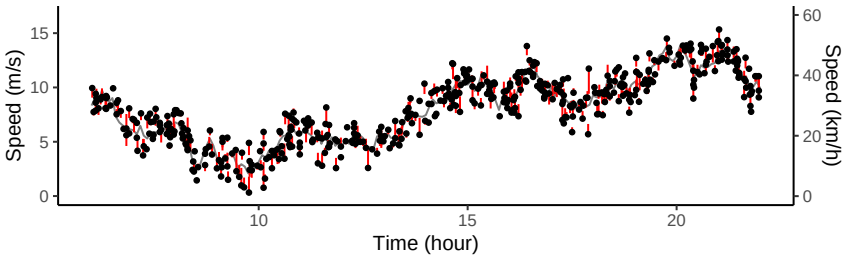
[1] Since a lot of traffic arriving into the CBD from the south must use this bridge, it can get very congested.



(A) Underlying average speed along segment



(B) Average speeds of individual buses



(C) Observed average speeds of buses, with vertial lines to denote measurement error.

FIGURE 4.2: Simulated data of average vehicle speed along a road segment. The right hand axis is shown as a guide for readers not familiar with speeds measured in meters per second.

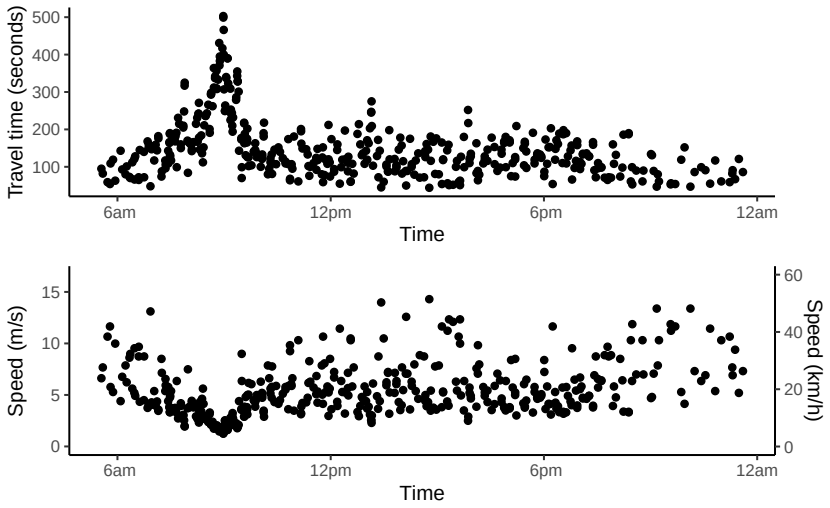


FIGURE 4.3: Road state observations along a single road segment over time, comparing travel time (top) to speed (bottom).

where the measurement error $w_{\ell,c} \sim \mathcal{N}\left(0,\ \boldsymbol{\psi}_\ell^2 + e_{\ell,c}^2\right)$ consists of between-vehicle and observation errors, which are assumed to be independent. We also use the identity matrix for the transition matrix $\mathbf{F}$ as our best guess of the current traffic state is the previous state. So, assuming that $\boldsymbol{\psi}_\ell$ and q_ℓ are known, for now, we have everything needed to implement a Kalman filter on $\beta_{\ell,c}$.

4.2.1 Predict step

In the examples presented, we use a stationary transition function, that is, $\mathbf{F}_c = 1$, which implies an assumption that traffic speed is constant over short periods (less than five minutes). The vector of all segment speed observations up to and including time t_c is defined as

$$b_{\ell,1:c}^{\cdot} = \bigcup_{t=1}^{c} \bigcup_{v \in V_{\ell,t}} b_{\ell,t}^{v}, \tag{4.7}$$

where $V_{\ell,c}$ is the set of all vehicles traversing segment ℓ at time t_c. Then the estimated segment state, conditional on all observations up to time t_{c-1}, has mean

$$\hat{\beta}_{\ell,c|c-1} = \mathbb{E}\left[\beta_{\ell,c} \,\middle|\, b_{\ell,1:c-1}^{\cdot}\right] \tag{4.8}$$

and variance

$$\mathbf{P}_{\ell,c|c-1} = \mathrm{Var}\left[\beta_{\ell,c} \,\middle|\, b_{\ell,1:c-1}^{\cdot}\right], \tag{4.9}$$

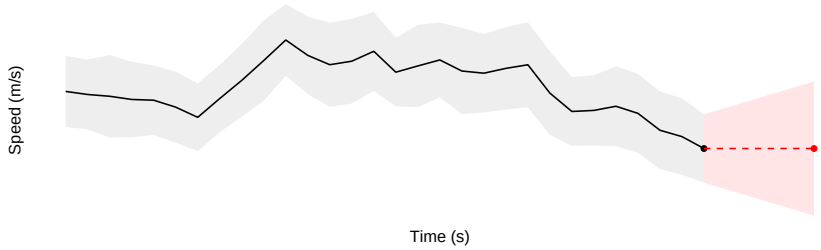
which are predicted using the following equations:

$$\begin{aligned} \hat{\beta}_{\ell,c|c-1} &= \hat{\beta}_{\ell,c-1|c-1}, \\ \mathbf{P}_{\ell,c|c-1} &= \mathbf{P}_{\ell,c-1|c-1} + \left(\Delta_c q_\ell\right)^2. \end{aligned} \tag{4.10}$$

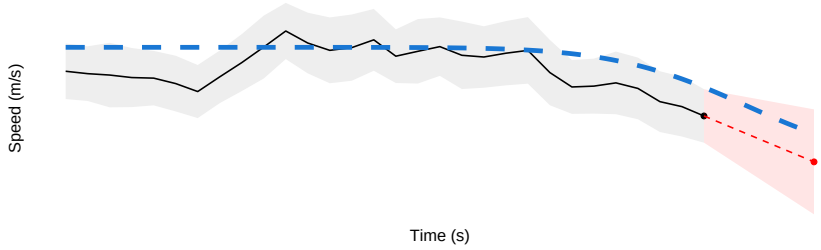
This prediction is shown in figure 4.4a. Alternatively, were forecast information available, we could use a transition matrix $\mathbf{F}_c$ to describe how traffic might change over time, as is shown in figure 4.4b.

4.2.2 Update step

Updating the Kalman filter involves taking the predicted state and updating it using *observations* of vehicle speeds along road segments. These are obtained from the



(A) Constant speed model. The prediction is equal to the previous state with an increased variance.



(B) Historical change based model. The dashed blue line represents historical average speed, which the prediction accounts for under this model.

FIGURE 4.4: Network state prediction depends solely upon the current state (black dot). The mean (solid black line) and uncertainty (shaded grey region) represent previous states. The predicted state has mean (red point) and uncertainty (shared pink region) dependent on the chosen model.

particle filter (section 3.2). However, it is possible to have multiple observations per road segment in one update period, as it is common for buses to travel one behind the other, particularly along bus lanes. Therefore, we have an *observation vector* for a set of vehicles $V_{\ell,c}$ passing through segment ℓ in the time interval $(t_{c-1}, t_c]$.

When updating the network, each observation must be accounted for. One way would be to combine the observations into a single estimate; however, this involves averaging observations and uncertainties. An alternative is to use an *information filter*, which allows the summation of information from multiple observations (Mutambara and Durrant-Whyte, 2000). The information filter involves inverting the state uncertainty $\mathbf{P}$; however, this is a simple computation due to having a one-dimensional state—if we were to estimate the state of all segments simultaneously, inverting the $L \times L$ uncertainty matrix would be computationally demanding, or even impossible, and we would be unable to use the approach.

The first step converts the predicted state vector and covariance matrix into information space by inversion of the covariance matrix, leading to the information

matrix

$$\mathbf{U}_{\ell,c|c-1} = \mathbf{P}^{-1}_{\ell,c|c-1} \tag{4.11}$$

and information vector

$$\boldsymbol{u}_{\ell,c|c-1} = \mathbf{P}^{-1}_{\ell,c|c-1}\hat{\beta}_{c|c-1}. \tag{4.12}$$

Converting the observations into information follows the same formula. Note first that the error needs to account for both measurement error and between-vehicle variation, which are assumed Gaussian and independent, so the total variance is their sum. The observation information matrix is

$$\mathbf{I}^{v}_{\ell,c} = \frac{1}{\boldsymbol{\psi}^2_{\ell} + (e^{v}_{\ell,c})^2} \tag{4.13}$$

and the observation information vector is

$$\boldsymbol{i}^{v}_{\ell,c} = \frac{\hat{b}^{v}_{\ell,c}}{\boldsymbol{\psi}^2_{\ell} + (e^{v}_{\ell,c})^2}. \tag{4.14}$$

Combining these by summation over vehicles yields the complete information matrix and vector for the time period $(t_{c-1}, t_c]$, which are, respectively,

$$\mathbf{I}_{\ell,c} = \sum_{v \in V_{\ell,c}} \mathbf{I}^{v}_{\ell,c} \tag{4.15}$$

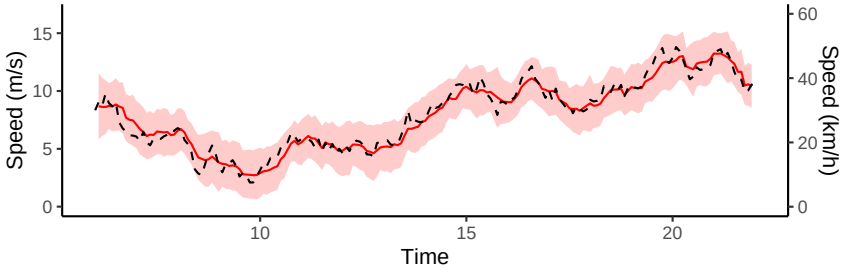
and

$$\boldsymbol{i}_{\ell,c} = \sum_{v \in V_{\ell,c}} \boldsymbol{i}^{v}_{\ell,c}. \tag{4.16}$$

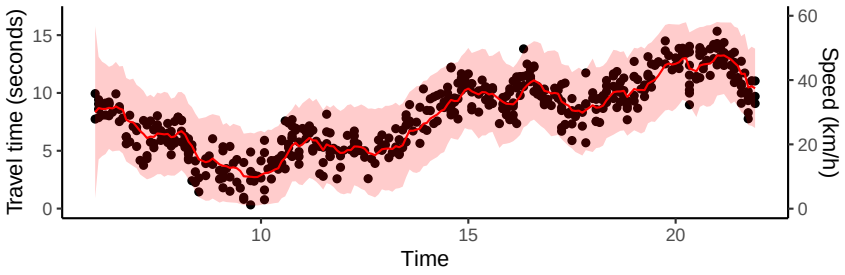
The state update is now just a case of adding the observation information in equations (4.13) and (4.14) to the predicted state information in equations (4.11) and (4.12):

$$\begin{aligned} \mathbf{U}_{\ell,c|c} &= \mathbf{U}_{\ell,c|c-1} + \mathbf{I}_{\ell,c}, \\ \boldsymbol{u}_{\ell,c|c} &= \boldsymbol{u}_{\ell,c|c-1} + \boldsymbol{i}_{\ell,c}. \end{aligned} \tag{4.17}$$

Note that, in situations where no data is observed for a given segment, the information for that segment is zero, so there is no further change to the predicted state value. This could be useful at peak hour, for example, if the transition function predicts



(A) Kalman filter estimate of the state mean (red line) and variance (shaded region) along with the true value (dashed black line).



(B) Predictive distribution of average vehicle speed accounting for state uncertainty and between-vehicle uncertainty ($\boldsymbol{\psi}^2$). Observations represented by black points.

FIGURE 4.5: Results of fitting the Kalman filter to the simulated data.

changes based on historical trends.

Finally, we back-transform the information into the state space,

$$\begin{aligned} \hat{\beta}_{\ell,c|c} &= \mathbf{U}_{\ell,c|c}^{-1}\boldsymbol{u}_{\ell,c|c}, \\ \mathbf{P}_{\ell,c|c} &= \mathbf{U}_{\ell,c|c}^{-1}. \end{aligned} \tag{4.18}$$

The primary constraint on the model is the dependence on $\boldsymbol{\psi}_\ell$ and q_ℓ; however, before considering the estimation of these values, we apply the Kalman filter model to the simulated data in figure 4.2 for which the parameter values are known.

The Kalman filter was fitted to the simulated data using the same values of q_ℓ and $\boldsymbol{\psi}_\ell$ used to generate the data, with the estimate of $\beta_{\ell,1:c}$ shown by the solid red line in figure 4.5a along with the associated uncertainty as estimated by $\mathbf{P}_{\ell,1:c}$ (shaded region). A dashed black line represents the simulated true mean. We see that the 95% credible region contains the true values of $\beta_{\ell,1:c}$. Figure 4.5b shows the posterior estimate of $\beta_{\ell,1:c}$ along with the posterior predictive distribution of $b_{\ell,c}^m$; that is, using the 95% region defined by the sum of $\mathbf{P}_{\ell,1:c}$ and $\boldsymbol{\psi}_\ell^2$, the latter of which is known for the simulation. Approximately 99.8% of the observations lie within the 95% predictive

region. Given the network parameters q_ℓ and $\boldsymbol{\psi}_\ell$ are known, the underlying network state can be recovered using a Kalman filter.

4.2.3 Limitations of the implementations

The main limitation of using the information filter is the need to calculate the inverse of the covariance matrix. If we want to improve the model by including segment interactions, the dimensionality of $\mathbf{P}$ would quickly become too large to compute $\mathbf{P}^{-1}$ easily. Thus, the method presented here is only appropriate for independent segments. Fortunately, however, using the Kalman filter instead would not be too difficult a task, since most of the time, only one vehicle will pass through a segment during an iteration. In cases where there is more than one, the estimates and their errors could be combined using a sample mean and variance, for example.

4.3 Estimating network parameters

To use the Kalman filter in any meaningful way, we first need to estimate the system noise, q_ℓ, and the between-vehicle variance, $\boldsymbol{\psi}_\ell^2$. To estimate values for these parameters, we fit the same model from section 4.1 to the historical data shown in figure 4.3 using Markov chain Monte Carlo sampling methods as implemented by JAGS (Plummer, 2003).

The first step in the modelling process fits the following hierarchical model:

$$
\begin{aligned}
b_{\ell,c}^m \,\big|\, B_{\ell,c}^m &\sim \mathcal{N}\left(B_{\ell,c}^m,\ (e_{\ell,c}^m)^2\right), \\
B_{\ell,c}^m \,\big|\, \beta_{\ell,c}, \boldsymbol{\psi}_\ell &\sim \mathcal{N}_T\left(\beta_{\ell,c},\ \boldsymbol{\psi}_\ell^2,\ 0,\ \mathbb{V}_\ell\right), \\
\beta_{\ell,0} &\sim \mathcal{U}\left(0,\ \mathbb{V}_\ell\right), \\
\beta_{\ell,c} \,\big|\, \beta_{\ell-1,c}, q_\ell &\sim \mathcal{N}_T\left(\beta_{\ell-1,c},\ (\Delta_c q_\ell)^2,\ 0,\ \mathbb{V}_\ell\right), \\
q_\ell &\sim \mathcal{G}\left(0.01,\ 0.01\right), \\
\boldsymbol{\psi}_\ell &\sim \mathcal{G}\left(0.01,\ 0.01\right).
\end{aligned}
\tag{4.19}
$$

The observation error, as in the previous section, is assumed to be constant for all observations, $e_{\ell,c}^m = 0.8$ meters per second (which is about 10 km/h). The model implementation was performed from R using the 'rjags' package (Plummer, 2018).

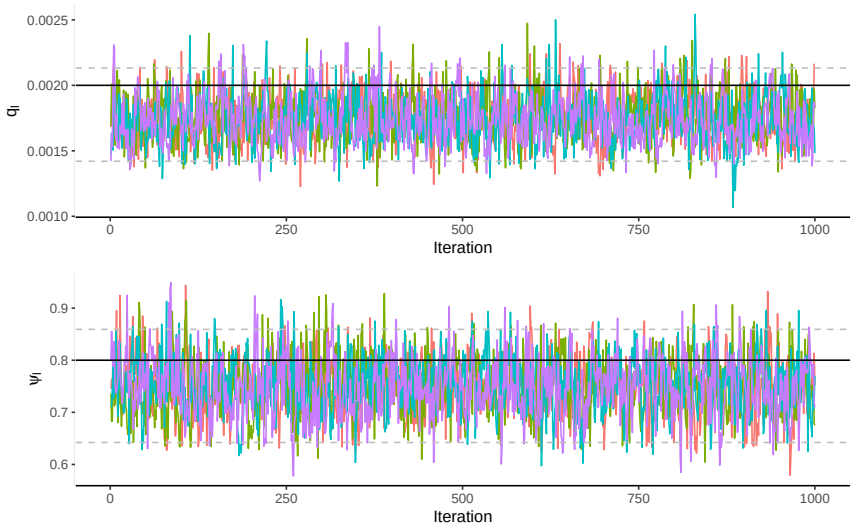


FIGURE 4.6: Traceplots of model parameters for the simulated data. Each of the four chains are coloured separately. Dashed grey lines show the 95% posterior credible interval and the solid lines represent the true values or each parameter.

TABLE 4.1: MCMC results for the Bayesian model fitted to the simulated travel time data. The multivariate $\hat{R}$, including for all of the β parameters, was 1.04, indicating that convergence had been achieved after 15,000 iterations.

Parameter	True Value	Mean	2.5%	50%	97.5%	$\hat{R}$
q_ℓ	0.002	0.0017	0.0014	0.0017	0.0021	1.001
$\boldsymbol{\psi}_\ell$	0.8	0.75	0.64	0.75	0.86	1.001

The model output was processed using the 'tidybayes' package (Kay, 2019) and graphed using 'ggplot2' (Wickham, 2016). The 'coda' package was used to assess the model's convergence results (Plummer et al., 2006).

4.3.1 Simulated data

To assess the model, we start by fitting it to the simulated data (figure 4.2), for which we know the values of q_ℓ and $\boldsymbol{\psi}_\ell$. Trace plots of the model parameters shown in figure 4.6, with the true values overlaid with a dashed line, show that the chains mixed well. Table 4.1 gives a summary of the simulation values and their posterior estimates, along with convergence statistics. The 95% credible intervals for the two parameters contain the true values, so there is nothing to suggest the model is inadequate for modelling the simulated data and estimating the network parameters.

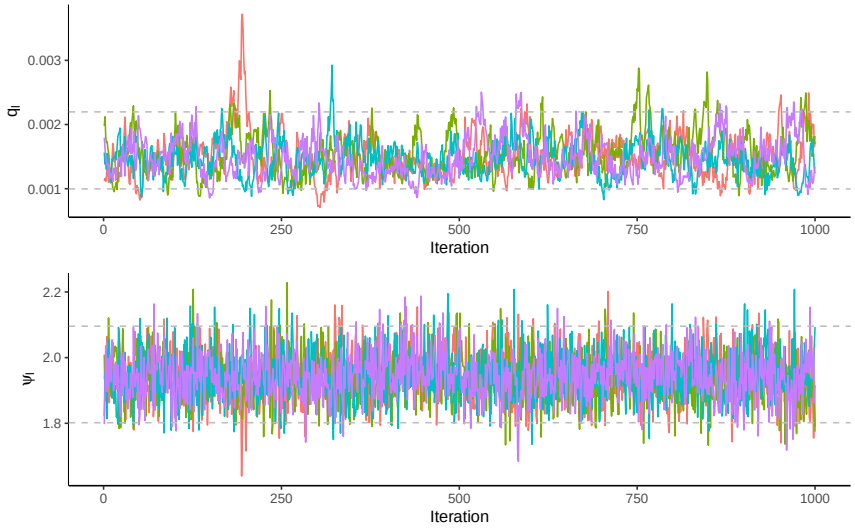


FIGURE 4.7: Traceplots of model parameters q and $\boldsymbol{\psi}$ fitted to the segment data. Each of the four chains are coloured separately. The 95% posterior credible region is denoted by dashed grey lines.

TABLE 4.2: MCMC results for the Bayesian model fitted to the road segment travel time data. The multivariate $\hat{R}$, including for all of the β parameters, was 1.08, indicating that convergence had been achieved after 15,000 iterations.

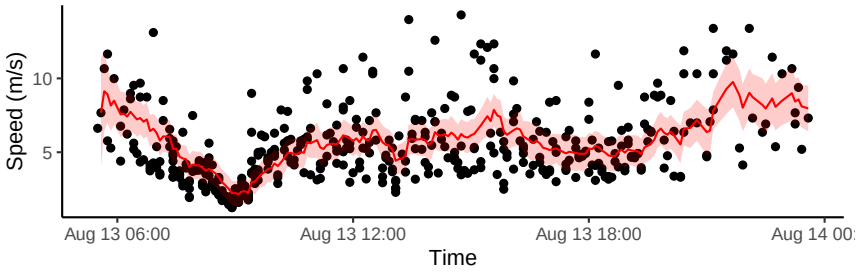
Parameter	Mean	2.5%	50%	97.5%	$\hat{R}$
q_ℓ	0.0015	0.001	0.0015	0.0022	1.013
$\boldsymbol{\psi}_\ell$	1.94	1.8	1.94	2.1	1.002

4.3.2 Real data

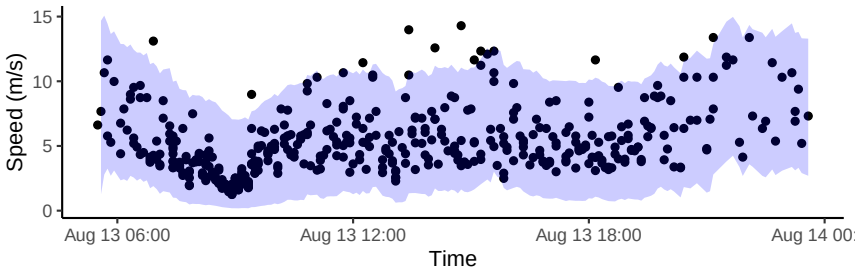
Having shown that the JAGS model is a valid way of estimating the network parameters q_ℓ and $\boldsymbol{\psi}_\ell$, we model the real travel time data from figure 4.3. Trace plots of the network parameters, as shown in figure 4.7, show again that the chains mixed well, which is reaffirmed by the convergence results in table 4.2. Not shown here are the summaries of the β's (there are 187 of them); however, the Gelman diagnostic $\hat{R}$ was less than 1.03 for all of them, indicating they reached convergence.

To assess the adequacy of these parameters, we fit the Kalman filter to the same data[2] to see how well the mean and predictive distributions fit the data. The estimates of $\beta_{\ell,1:c}$, along with associated uncertainties, are shown in figure 4.8a, where we see that the mean approximately follows the center of the data. The predictive vehicle speed distribution, which accounts for both $\mathbf{P}_{\ell,1:c}$ and between-vehicle uncertainty,

[2]In the next section we use two weeks of data, one for each of testing and training.



(A) Estimated mean speed, showing state estimates along with the 95% credible region.



(B) Predictive distribution of vehicle speeds, which includes between-vehicle variability.

FIGURE 4.8: Results of fitting a Kalman filter to the segment data with parameters estimated using the hierarchical model.

TABLE 4.3: Comparison of the timings for the MCMC and Kalman filter estimation methods. Times are reported in seconds.

	User	System	Total
JAGS	124.828	0	124.847
Kalman filter	0.052	0	0.056

$\boldsymbol{\psi}_\ell$, is shown in figure 4.8b, and shows most of the points contained within the 95% posterior predictive region.

Finally, the timing comparison is displayed in table 4.3. The hierarchical model fit using JAGS took about 2200 times as long as the Kalman filter implementation. So, even if we reduced the number of chains, the number of iterations, and tried to speed up the JAGS model, the Kalman filter would still be significantly faster.

4.3.3 Hierarchical model over multiple road segments

To assess how effective our approach is for estimating $\boldsymbol{\psi}_\ell$ and q_ℓ, we selected five other road segments around Auckland. However, instead of fitting the same model independently to each segment, we use a hierarchical Bayesian model on the parameters, since it seems reasonable that, while they will not be the same for all segments,

TABLE 4.4: MCMC results for the hierarchical Bayesian model fitted to six road segments to estimate the top-level variance parameters. The multivariate $\hat{R}$, including for all of the β parameters, was 2.03.

Parameter	Mean	2.5%	50%	97.5%	$\hat{R}$
μ_ψ	0.383	-0.191	0.389	0.926	1.00
σ_ψ	0.608	0.315	0.549	1.24	1.00
q	0.0014	0.0012	0.0014	0.0017	1.02

there will be an underlying population distribution. This allows us to obtain estimates of the parameter values without explicitly needing to model every road (of which there are 8,151).

The hierarchical model used to estimate the network parameters is:

$$\begin{aligned}
b^m_{\ell,c} \,|\, B^m_{\ell,c} &\sim \mathcal{N}\left(B^m_{\ell,c},\ \left(e^m_{\ell,c}\right)^2\right), \\
B^m_{\ell,c} \,|\, \beta_{\ell,c}, \boldsymbol{\psi}_\ell &\sim \mathcal{N}_T\left(\beta_{\ell,c},\ \boldsymbol{\psi}^2_\ell,\ 0,\ \mathbb{V}_\ell\right), \\
\beta_{\ell 0} &\sim \mathcal{N}\left(0,\ 10^2\right), \\
\beta_{\ell,c} \,|\, \beta_{\ell,c-1}, q_\ell &\sim \mathcal{N}_T\left(\beta_{\ell,c-1},\ (\Delta_c q_\ell)^2,\ 0,\ \mathbb{V}_\ell\right), \\
q &\sim \mathcal{G}\left(0.001,\ 0.001\right), \\
\log\left(\boldsymbol{\psi}_\ell\right) \,|\, \mu_\psi, \sigma_\psi &\sim \mathcal{N}\left(\mu_\psi,\ \sigma^2_\psi\right), \\
\mu_\psi &\sim \mathcal{N}\left(0,\ 10^2\right), \\
\sigma_\psi &\sim \mathcal{G}\left(0.001,\ 0.001\right).
\end{aligned} \tag{4.20}$$

We initially attempted to fit a hierarchical segment noise parameter q_ℓ with hyper-parameters, but the values were all approximately equal for all segments and convergence was very slow (100,000's of iterations). Instead, we opted for a single common system noise parameter across all segments. The speed data for six segments on two consecutive Tuesdays is shown in figure 4.9, with the locations of the roads displayed in figure 4.10.

The model was fit using JAGS to the first day of data for the six segments, while the second was reserved for testing the validity of the estimated parameters. Each of the four chains was run with a 100,000 iteration burn-in phase, followed by 50,000 iterations with a thinning interval of 50. Trace plots of the top-level parameters are displayed in figure 4.11, which demonstrates good mixing of the chains. Table 4.4

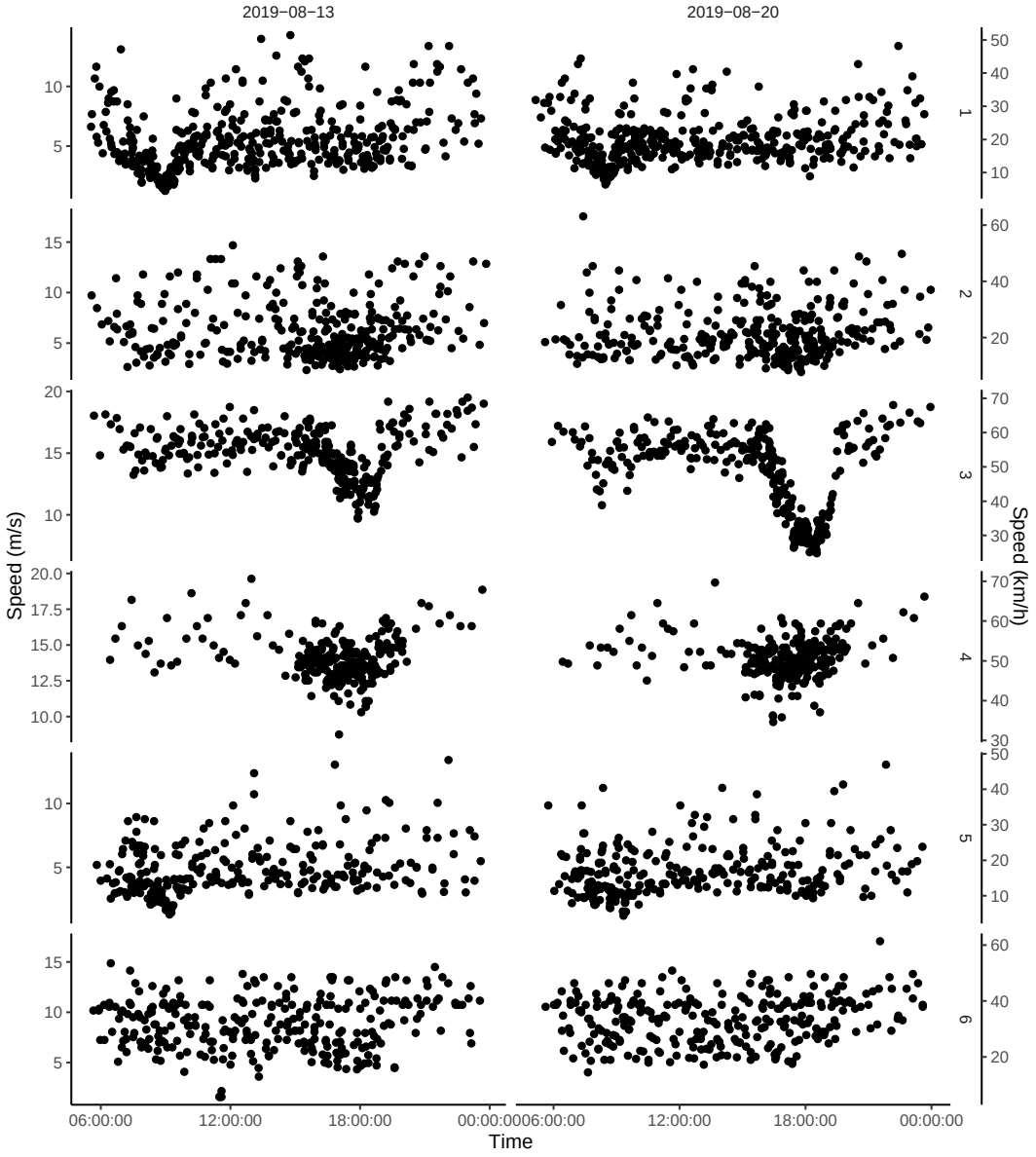


FIGURE 4.9: Observed average bus speeds along six road segments on two consecutive Tuesdays.

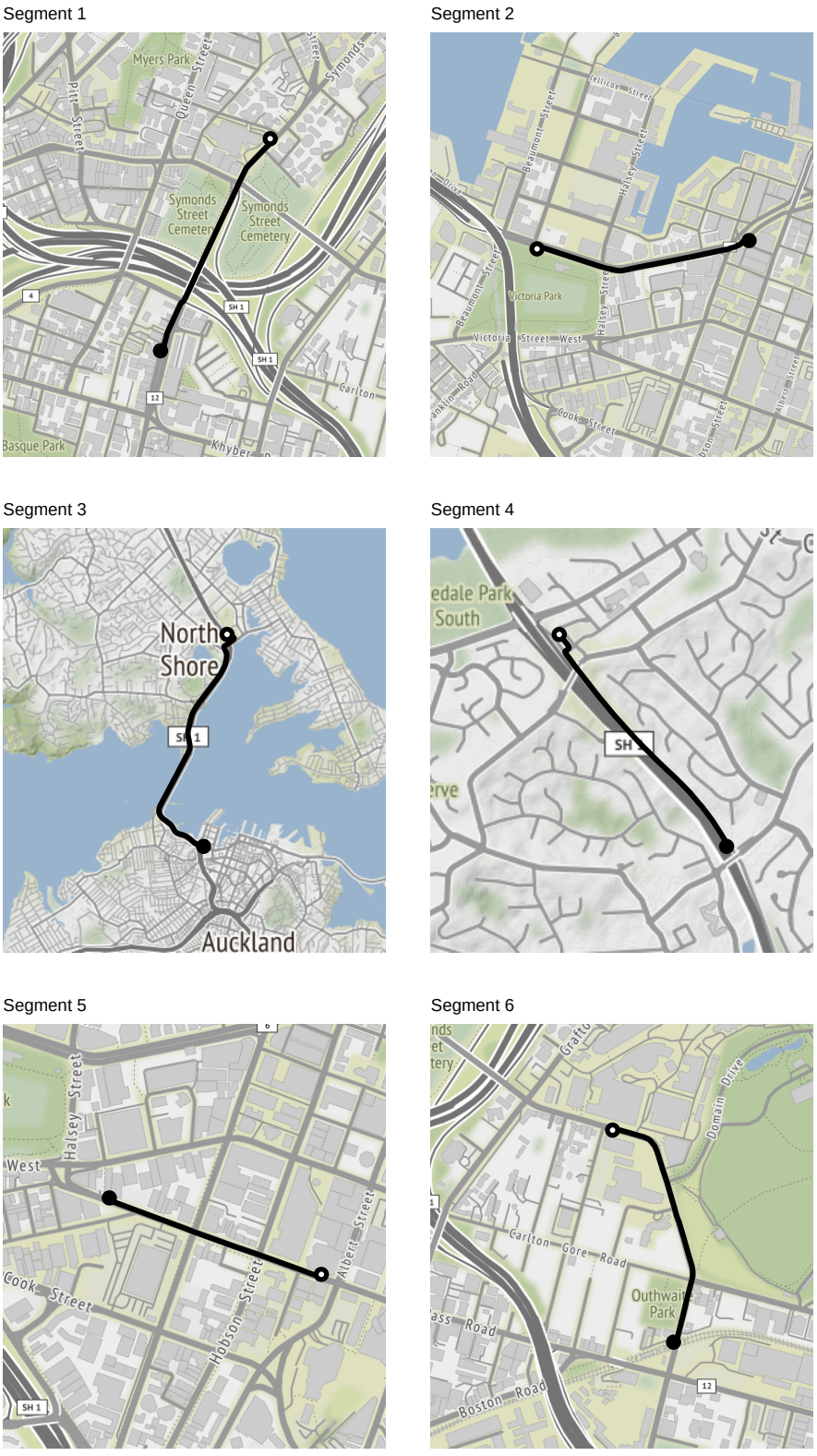


FIGURE 4.10: Segment locations, drawn using the 'ggmap' package (Kahle and Wickham, 2013). Segments begin at the solid point and terminate at the open point.

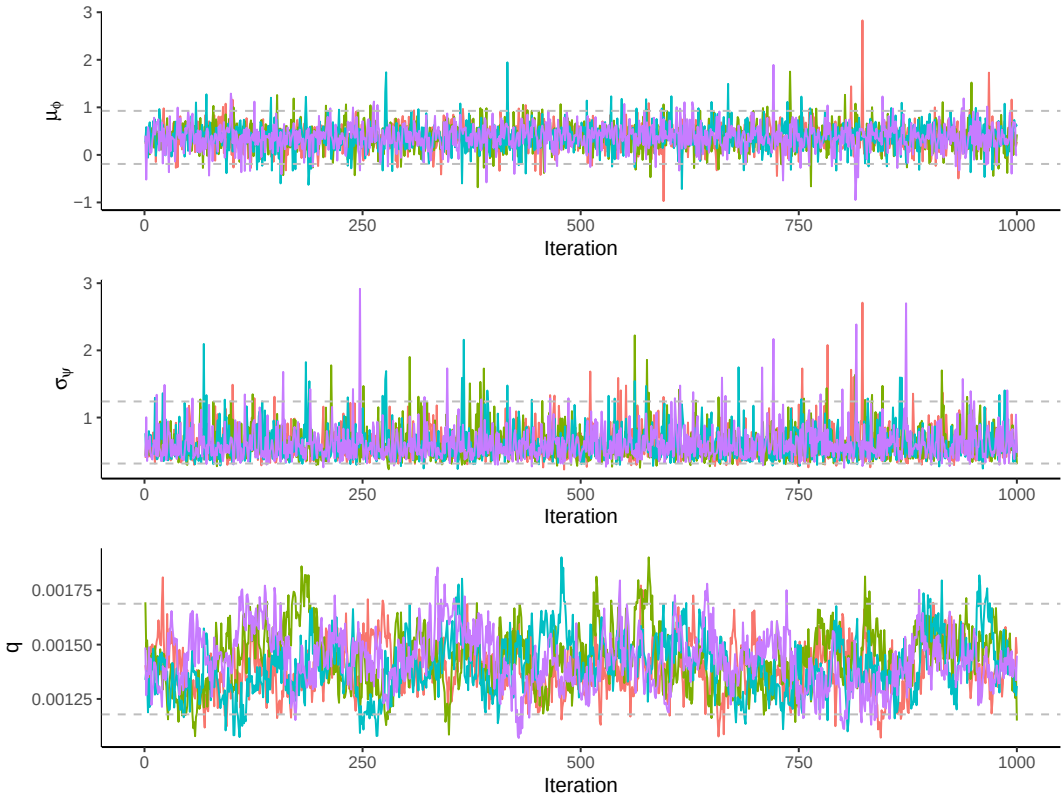


FIGURE 4.11: Traceplots of top-level network parameters estimated by the hierarhical model. The four chains are coloured individually, and 95% credible interval indicated by dashed gray lines.

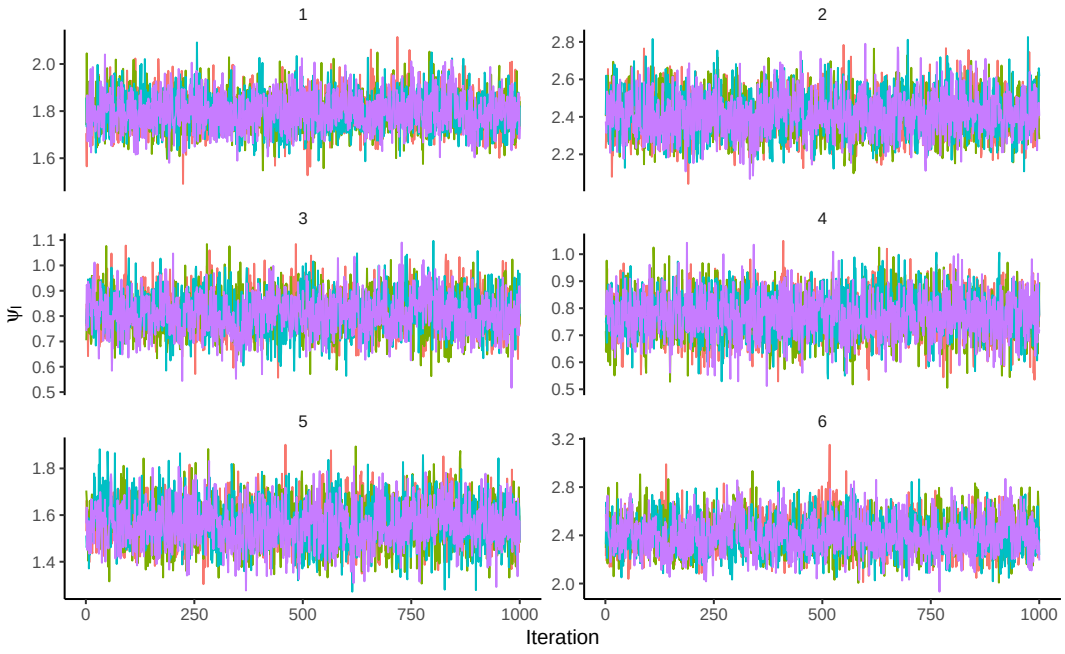


FIGURE 4.12: Traceplots of between-vehicle variance parameters, $\boldsymbol{\psi}_\ell$, for each segment. The four chains are coloured individually.

shows the posterior mean and quantiles for these parameters, along with their Gelman convergence diagnostics.

Trace plots for the segment-specific variance parameters are shown in figure 4.12, and also demonstrate good mixing. The Gelman convergence diagnostic is again very close to unity, indicating that running the chain for longer would not decrease the posterior variance by much.

The Kalman filter was fit to each segment using the posterior mean of $\boldsymbol{\psi}_\ell$ and the population parameters. Figure 4.13 shows the original data superimposed with a posterior sample of β's, along with the next week's data with results from the Kalman filter, including the 95% credible region for the mean road speed and the 95% posterior predictive region for individual vehicles' average speeds.

The results show that the β values estimated with JAGS nicely fit the data, including peak congestion, as do the Kalman filter estimates. The posterior predictive region covers most of the observations for all the segments.

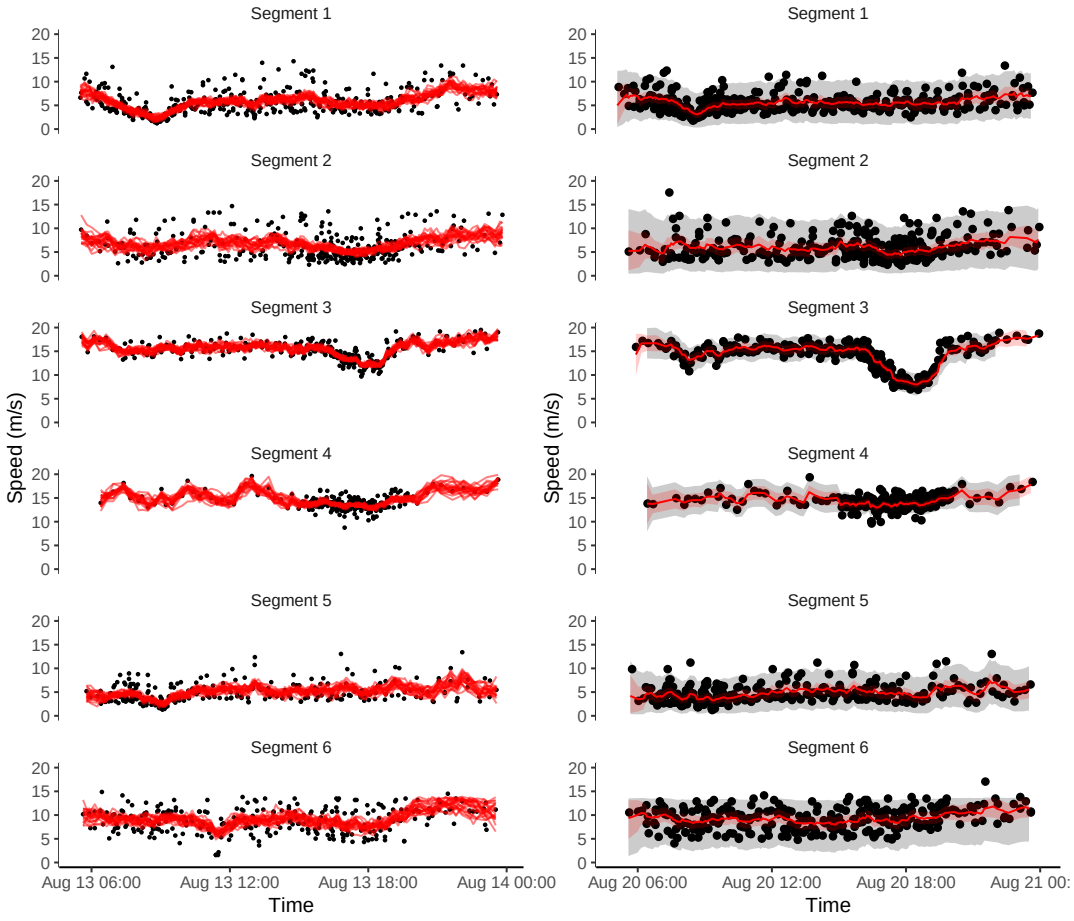


FIGURE 4.13: Results for the hierarchical approach to modelling road speed. Left: the training data with a sample of posterior fits for JAGS. Right: the test data with the Kalman filter estimation results, showing the speed estimate (red line), its uncertainty (shaded in red), and the posterior predictive region for vehicle speeds (shaded in grey).

4.4 Improving forecasts with history

The model presented in sections 4.1 to 4.3 is adequate for estimating the *real-time* state of the network, but is unuseful for forecasting vehicle speeds, particularly just before or after a peak period. In chapter 5, we explore the prediction of arrival times, which involves making short-term forecasts of speed. Using historical data is one way of improving speed forecasts, particularly around peak times, which are notoriously difficult to predict due to high levels of uncertainty (He et al., 2020). Perhaps because of this difficulty, such long-term travel time prediction has seldom been explored in the transit literature (Moreira-Matias et al., 2015).

In section 4.3.2, I showed that most roads have a "peak effect" in the morning or the evening, while some exhibit both. It seems reasonable therefore to have a model which allows a segment to have zero, one, or two peaks, each with varying magnitude (the size of the decrease in speed) and width (how long the peak period is). The temporal location of these peaks is likely to be related, but variable: some roads will experience peak traffic earlier than others, for example.

Early work fitting Bayesian models to these data were not promising, as it involved too much manual work tuning the parameters for each segment and checking the results. Since our goal is to obtain a generalised framework that runs with minimal manual effort, we sought an alternative.

From figure 4.14 it is clear that along each segment, there is a consistent pattern, with some variability between days as far as the temporal location and magnitude of the "peak" on weekdays; weekends are fairly consistent, although this is only two days' worth of data. Here I present a nearest-neighbour grid, which takes time and road state (traffic speed) as input, and outputs the mean (and uncertainty) of travel time in 10, 20, or 30 minutes from the current time. Similar "speed maps" have been used by the likes of Cathey and Dailey (2003), Čelan and Lep (2017), and Chen and Rakha (2014). The first step is to aggregate observations into 5 minute intervals and take the mean traffic speed in each interval. Next, we bind lagged duplicates of these speeds (10, 20, 30 minutes, for example), such that each row includes current and future states. Any missing periods are interpolated from the mean of the adjacent states. The speed map for 30 minute predictions is shown in figure 4.15. The lighter

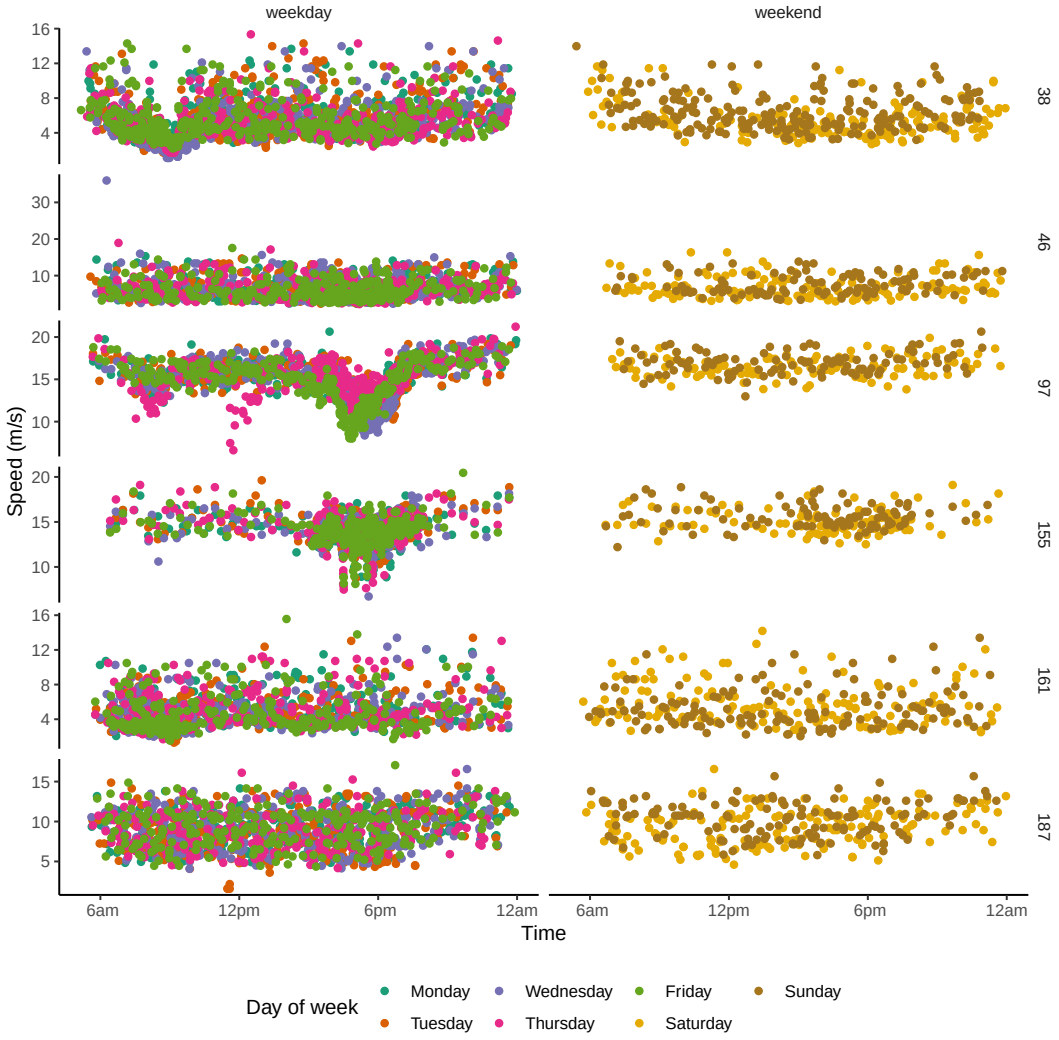


FIGURE 4.14: Vehicle speeds along six road segments over one week, coloured by the day of the week.

TABLE 4.5: RMSE results comparing the predictive performance of two forecast methods for predicting road state in 30 minutes.

Method	RMSE (m/s)
Naïve	2.48
Forecast	1.99

areas represent higher speeds.

Predictions are made from the "speed map", which is stored in the database, by finding the row corresponding to the current time and road state, $\beta_{\ell,c}$. This row then contains the necessary forecast value for (in this case) 30 minutes in the future based on historical data.

The next step is to evaluate the predictive power of this approach. We do this by taking the second week of data (shown in figure 4.16) and use the fitted estimates to predict the state in 30 minutes, and then compare this to the actual state in 30 minutes. RMSE (equation (3.35) equation (3.35)) is used to compare the predicted and observed speeds.

Forecasts 30 minutes into the future are shown in figure 4.17 for both the grid-search method, as well as the naïve method of using the current average traffic speed as the predictor. The comparative RMSE values are displayed in table 4.5. However, these are not too different because there is a lot of inherent uncertainty, but demonstrate a slight increase in predictive performance when historical trends are used.

Most importantly, however, is the linear curve (shown in blue in figure 4.17) compared to the line of equality, in red. The interpretation is that the more accurate the predictions on average, the closer the blue trend line will be to the black line of equality. For our forecast model, we see that in 5 out of 6 segments the two lines are very close, compared to the naïve estimate in which case the predictions are often too high or too low.

Further work is needed to improve the forecasting abilities of our model. However, we have designed the framework in such a way that any such improvements can easily be integrated into both the network modelling stage, and the upcoming arrival time prediction stage (see section 4.5.3 for details).

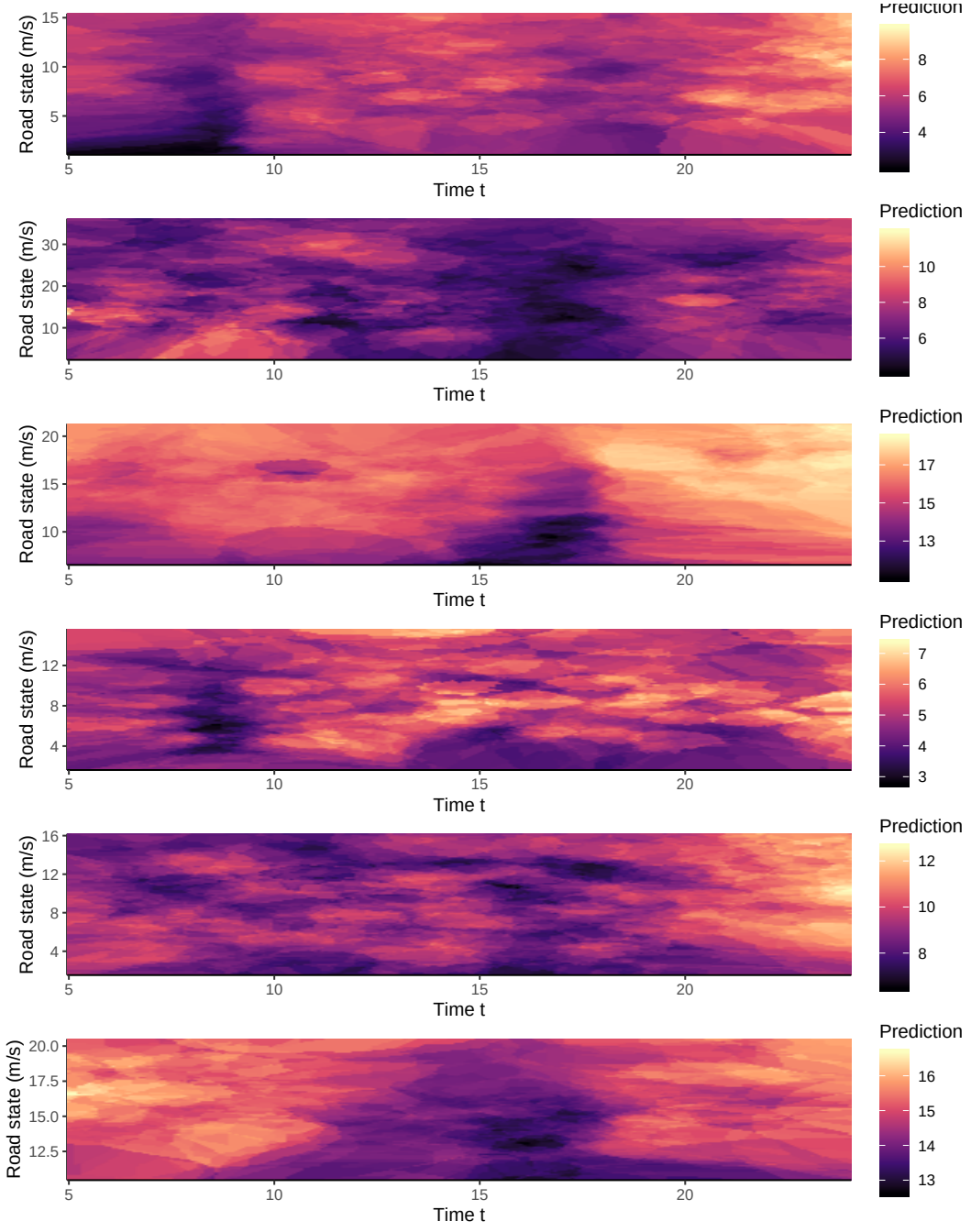


FIGURE 4.15: Speed maps to predict segment speed in 30 minutes given the current average speed (y-axis) and time (x-axis).

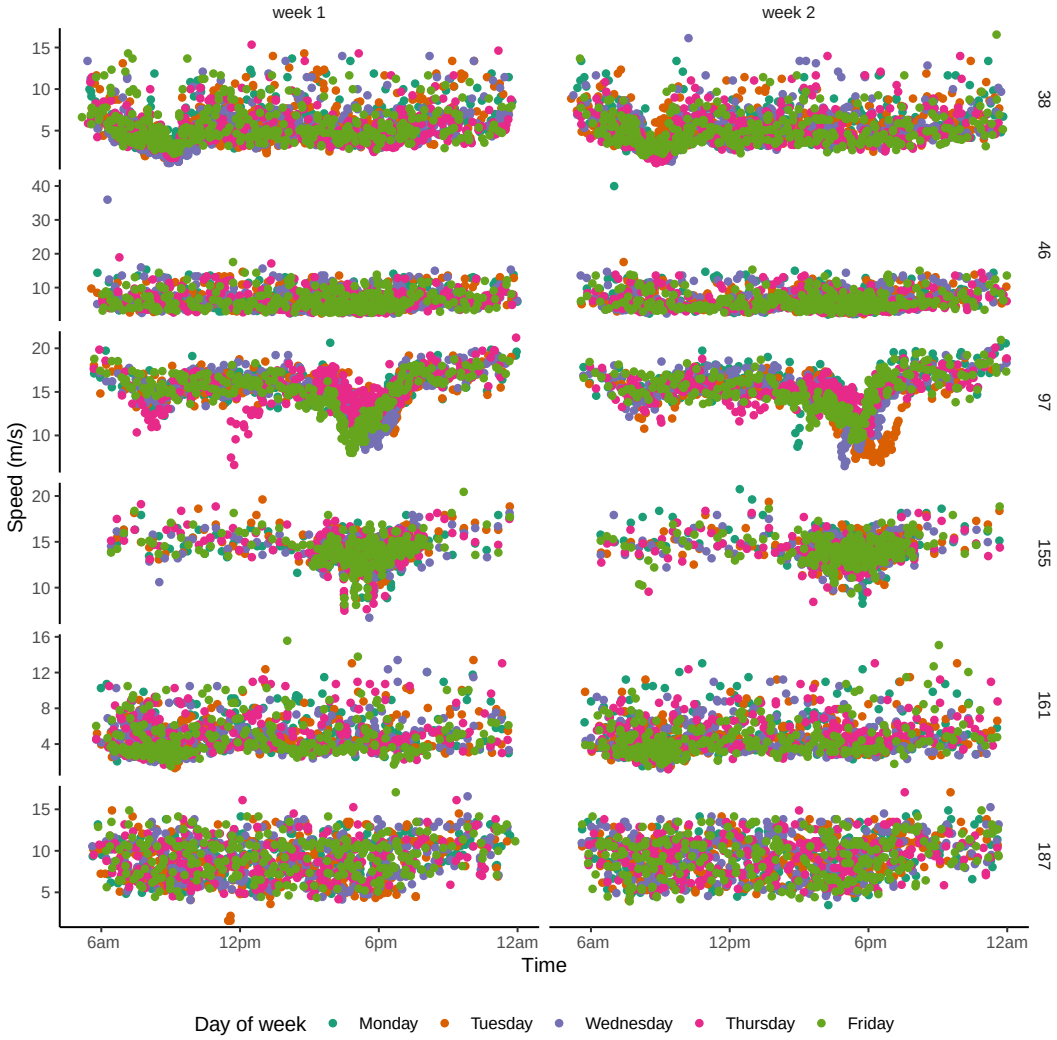


FIGURE 4.16: Vehicle speeds along six roads over the course of two weeks, showing only week days, for training (left) and testing (right).

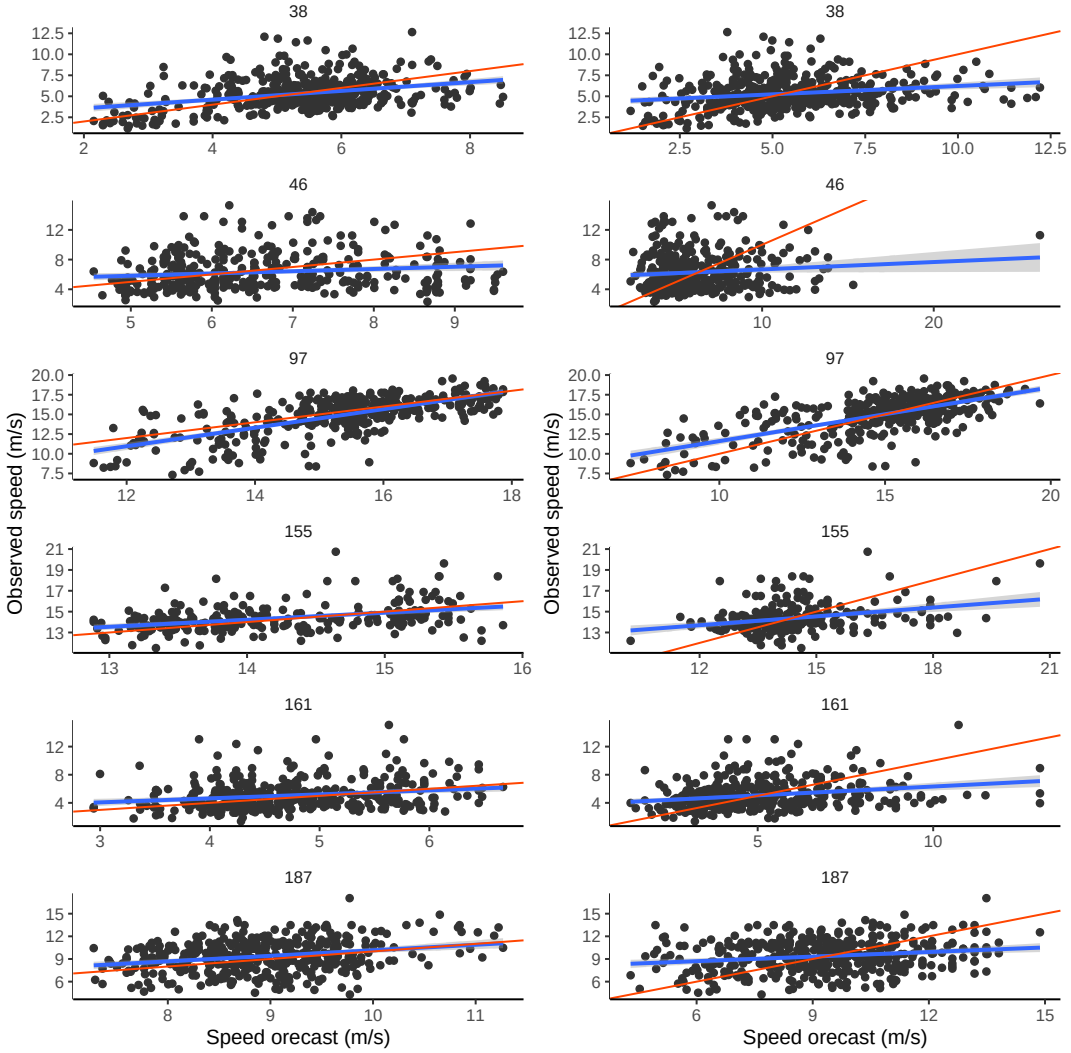


FIGURE 4.17: Forecasts of average vehicle speeds in 30 minutes versus actual average speed in 30 minutes along segments. The line of equality (red) indicates a perfect prediction. The closer the best fit line (blue) is to the line of equality, the better the forecasts.

4.5 Particulars of the real-time implementation

After stepping through the real-time model, estimation of its parameters, and improvement of forecasts using historical data, the last remaining piece of the network model is its real-time implementation within the 'transitr' package. The first step is, of course, to load the historical data and model the real-time parameters using JAGS. These are then inserted into the same database in which we store the GTFS and network data.

The network is implemented as independent objects of class `Segment`, which are initialised when the program is first started with their model parameter values loaded—if available—from the database. The network state itself is implemented using the 'Eigen' C++ library, allowing us to store the state vector and uncertainty matrix as an `Eigen::Vector` or `Eigen::Matrix`, respectively. This library includes all the necessary vector and matrix operations, for example, multiplication. Even though the current state is one-dimensional, we may in future wish to extend this to include, say, the rate of change of speed, which could better model peak periods. This would involve a length-two state vector and 2×2 uncertainty matrix. By using 'Eigen', this change would require minimal code changes. The following sections discuss the initialisation, update, and prediction steps as implemented within 'transitr'.

4.5.1 Initialisation

The initialisation phase of the network is quite important since in many cases it will be the only piece of information available for speed and arrival time predictions, at least until a vehicle traverses it and provides a real-time observation. Therefore, to initialise a `Segment`, we check if there is any historical data available for it: if there is, we set $\beta_{\ell,0}$ equal to the historical mean speed of all observations over time. To initialise the uncertainty, we use $\mathbf{P}_{\ell 0} = \frac{30}{3.6^2}$ meters per second (which is approximately 30 km/h).

Another part of initialisation is setting the constraint for the *maximum speed*. Since we do not know the speed limit along most road segments, the best we can do is to use the maximum speed limit, 100 km/h, which is approximately 30 m/s. For

segments with enough observations, we can use the maximum observed speed to estimate the posted speed limit along the road.

4.5.2 State update

The state update step is simple due to the segments being independent with one-dimensional states. First, `Segment`s are updated after processing the `Vehicle`s to collect any new observations (section 3.2). Then we perform a parallel iteration over `Segment`s: if there are new observations, the update step (section 4.2) is executed. The 'Eigen' library makes this implementation straightforward. Once the state has been updated, the data vector is cleared and the `Segment`'s timestamp set to the current time.

4.5.3 State forecasts

While not part of the network state model, forecasting is an essential component of our real-time application. Each `Segment` has a `forecast()` method which returns the forecasted speed given the current timestamp[3] and segment traffic speed. The forecasts are available in 10 minute intervals, so a forecast for 5 minutes uses the current travel time state, while a forecast for 25 minutes uses the 20-minute forecast, and so on, capped at 60 minutes. As for the uncertainty, this comes from the historical data too, but rather than forecasting, we use the historical variance of vehicle speeds at the desired time.

Currently, the forecast method uses the naïve constant-speed predictor. Since the arrival time prediction component (chapter 5) accesses forecasts through the `Segment::forecast()` method, an implementation of the historical-based forecasts[4] will automatically be integrated into the forecast model with no change needed to the prediction component.

[3] This is the time the `Segment` was updated, not the wall clock time.
[4] Or indeed any other desirable forecast method.

4.6 Chapter contributions

- This chapter presents a hierarchical approach to fitting a real-time Kalman filter to transit vehicle speed observations, using historical data to estimate the intermediate noise parameters.
- The network model uses real-time observations obtained from the particle filter (average vehicle speeds along road segments) to estimate the current network state. Similar methods have been presented in the literature, but the primary different in this case is that the entire process works using only GTFS data (no external input is required).

Chapter 5

Predicting arrival time

So far, we have discussed the structure of transit data and its use in constructing a *transit road network* (chapter 2) through which we model vehicles in real-time using a particle filter to estimate average vehicle speed along roads (chapter 3). These speeds are, in turn, used to update a Kalman filter model of real-time road state (chapter 4). We now have everything needed to begin predicting the arrival time of buses at stops.

In this chapter, we introduce one final state which combines all of the information we have gained thus far. *Trip state* provides a means of combining schedule information with real-time vehicle data to simplify the process of predicting stop arrival times, which also involves the network state and dwell-time information. The first step, therefore, is to estimate trip state (section 5.1) based on the GTFS schedule and estimated vehicle states from chapter 3. Other frameworks have used a similar process, such as Cathey and Dailey (2003), who used a *trip tracker* to prescribe real-time vehicle observations to trip states for use in the *prediction* step.

The remainder of this chapter involves comparing a selection of four models—two using the real-time vehicle and network states, and two others for comparison (including the method currently in use in Auckland). Section 5.2 presents the methods before we examine the results of arrival time prediction for a full day of observations for which we know the actual arrival time, allowing us to compare the performance of the methods (section 5.3). Lastly, we discuss the real-time implementation and performance results (section 5.4) and assess the practicality of using a particle filter for arrival time prediction. This chapter focuses on the statistical properties of the prediction methods; practical issues are discussed in chapter 6.

5.1 Trip state

At any one time, there are numerous scheduled trips within the transit network we want to obtain the *state* of, which can then be updated with any real-time vehicle or network information, if available. Occasionally there is no vehicle associated with a given trip, so having a trip state makes it possible to combine schedule data with real-time network information to obtain estimated times of arrival (ETAs). That is, it enables real-time arrival time prediction in the absence of real-time vehicle data.

At any time t, we obtain a list of scheduled trips such that the trip's start time T_{start} is less than 30 minutes before t, and the trip's end time (arrival at last stop) is less than 60 minutes after t. The state of each trip within this window is represented by: t_k, the time of the previous vehicle observation associated with it (if there is one); s_k, the current stop index; d_k, an indicator of whether the vehicle has departed from that stop; ℓ_k, the current segment index; and $\hat{p}_k$, the progress (as a proportion) along the segment. These are stored in the trip state vector

$$\mathcal{T}_k = [t_k, s_k, d_k, \ell_k, \hat{p}_k]^\top . \tag{5.1}$$

The initial state is $\mathcal{T}_0 = [T_{\text{start}}, 0, 0, 0, 0]^\top$. Trips use the same time subscript k as vehicle locations since they are initialised based on the schedule, and then only updated when vehicle data is received.

Estimation of the various state components depends on the type of the most recent observation associated with the trip. The three possible scenarios are:

1. the last observation was a *trip update* for stop s,
2. the last observation was a *GPS position*, or
3. no data is available for the trip.

Scenario 1: On arrival at stop s at time t_k, the trip's state is

$$\mathcal{T}_k = \begin{cases} [t_k, s, 0, \ell(s), 0]^\top & \text{if arrival,} \\ [t_k, s, 1, \ell(s), 0]^\top & \text{if departure,} \end{cases} \tag{5.2}$$

where $\ell(s)$ is the segment index of stop s.

Scenario 2: Here, the vehicle has completed travel along part of the segment, so the *remaining distance* is needed. From the particle filter in chapter 3, the segment along which the vehicle is travelling at time t_k is identified as ℓ_k, and the most recently visited stop is s_k. The proportion of the segment travelled at time t_k, $\hat{p}_k$, is easily calculated using the vehicle's current distance x_k and the segment's start distance $\mathcal{D}_{\ell_k}$ and length $\mathcal{L}_{\ell_k}$:

$$\hat{p}_k = \frac{x_k - \mathcal{D}_{\ell_k}}{\mathcal{L}_{\ell_k}}. \tag{5.3}$$

Scenario 3: There are three main reasons for there being no observations associated with a scheduled trip:

1. the trip has been cancelled, so no vehicle is servicing the trip, but the cancellation has not been entered into the real-time system;
2. the vehicle is late and has not yet started the trip; or
3. a vehicle is servicing the trip but its automatic vehicle location system is either not working or is registered with the wrong trip.[1]

There is no way to differentiate these three situations without physical investigation, so we always assume there is a bus travelling the route. However, if no bus has been observed, it is desirable to display a warning to passengers so that they are aware and can choose to catch an alternative bus instead.

It is common for passengers waiting at the first stops along a route to have no available real-time information until the bus is a few minutes away. This is because the bus does not register with the server until it has begun the trip. Until this happens, the digital message sign displays the default scheduled time, which can be problematic if the bus is late to start the route. In extreme cases, once the scheduled time has passed the service disappears from the digital message sign, leaving passengers wondering whether the bus will eventually come, which is why it is necessary to display to commuters if no real-time information is available for a trip.

[1] This happens more often than one might first imagine.

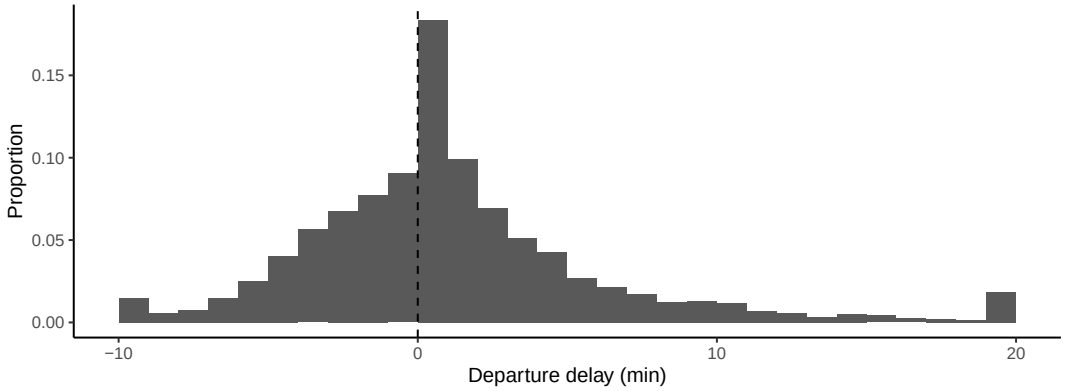


FIGURE 5.1: Vehicle delays at layover stops truncated to 10 minutes early and 20 minutes late. Negative values indicate non-adherance by drivers, which make up 40% of cases.

Additionally, historical data of arrival times can be used in conjunction with the real-time network state to provide a prior estimate of arrival time, which is particularly useful for trips prone to starting late. In these situations, a single prediction is made when first initialising the trip based on historical information. Once the trip is observed (or cancelled) the prediction can be updated. The goal here is not to predict arrival times for unobserved vehicles accurately, but instead to smooth the arrival time estimates at the beginning of a trip's schedule for those situations where the vehicle does finally start its route.

5.2 Arrival time prediction methods

The estimated trip state (and any associated vehicle) together with the road network can now be combined to forecast how long it will take for the vehicle to reach all remaining stops along the route. Dwell times can also affect arrival time uncertainty (Hans et al., 2015; Meng and Qu, 2013; Robinson, 2013; Shalaby and Farhan, 2004; Shen and Li, 2013; Wang et al., 2016). Some trips have the added complexity of layovers at specific stops—these are common at major points of interest, such as at a mall or transit station, where many routes connect so passengers can transfer between services (see section 2.1). In theory, drivers wait at these stops until the scheduled departure time before continuing the trip; however, in 40% of cases over two weeks, the driver left before the scheduled departure time, as is demonstrated in figure 5.1, and 31% departed more than one minute early.

There are various ways to forecast arrival times, each with associated drawbacks and advantages. This section presents four arrival time prediction methods:

- $\mathcal{A}_{\text{p}}$ uses the particle filter estimate of vehicle state and the network state to obtain arrival time distributions;
- $\mathcal{A}_{\text{n}}$ uses a Normal approximation of the vehicle's state, along with trip state, network state, and dwell-time distributions;
- $\mathcal{A}_{\text{h}}$ uses the schedule and historical delay data; and
- $\mathcal{A}_{\text{s}}$ uses the schedule and current real-time delay, as currently used by Auckland Transport.

Methods $\mathcal{A}_{\text{p}}$ and $\mathcal{A}_{\text{n}}$ use the network state, which is simplified for each route by including only the segments (in order) used by route r with mean and variance

$$\boldsymbol{\beta}^r = \begin{bmatrix} \beta_1^r \\ \beta_2^r \\ \vdots \\ \beta_L^r \end{bmatrix} \quad \text{and} \quad \boldsymbol{\zeta}^r = \begin{bmatrix} \zeta_1^r & \rho_{1,2}^r & \cdots & \rho_{1,L}^r \\ \rho_{2,1}^r & \zeta_2^r & \ddots & \rho_{2,L}^r \\ \vdots & \ddots & \ddots & \vdots \\ \rho_{L,1}^r & \rho_{L,2}^r & \cdots & \zeta_L^r \end{bmatrix}, \tag{5.4}$$

respectively, where β_i^r is the average vehicle speed along the ith segment of route r, ζ_i^r is the variance for that segment, and $\rho_{i,j}^r$ is the covariance between segments i and j. Note that the current implementation does not include segment covariances, but the above set-up demonstrates that the model can include them, if available. To simplify notation, I have dropped the r superscript for the remainder of this chapter since only one route is considered at any one time.

For bus stop dwell times, we collected two weeks of data (as in section 4.3) and calculated the mean and variance of dwell time for all stops along each trip. It is possible to determine if a bus *did* stop (there is both an arrival and a departure time), but not that a bus *did not* stop. For example, if the bus reports only an arrival time, it is unclear whether this is because the bus truly did not stop, or if it is a deficiency with the data. The bus may not have reported both observations, or, more likely, the polling interval (30 seconds) did not see both of them (Auckland Transport's real-time feed

only includes the most recent observation). Therefore, stopping probabilities cannot be estimated from the data, so the same value of $\pi_j = 0.5$ is used as in chapter 3.

5.2.1 Particle filter ($\mathcal{A}_\mathrm{p}$)

Each active trip is associated with a single vehicle, itself associated with a set of N particles approximating the vehicle's state. Perhaps the most straightforward method—at least conceptually—of predicting arrival times is to let each particle progress to the end of the route and record its arrival times at stops along the way. Computationally, this is easy to implement but very intensive, significantly increasing iteration time for the application. However, the main reason for needing large N (we use $N = 5000$ in chapter 3) is due to a lot of uncertainty in the vehicle's location and the need to sample many trajectories to prevent degeneration. Now, only arrival times need to be estimated. It is, therefore, possible to use $N^\star < N$ particles to estimate arrival times, providing a significant performance improvement. $N^\star$ can be adjusted dynamically, such as by how many stops remain; however, the present application used a fixed value of $N^\star = 200$.

To implement the particle filter forecast method, denoted $\mathcal{A}_\mathrm{p}$, we take a (weighted) subsample of $N^\star$ particles at time t_k (the time of the most recent vehicle observation). For each particle, we calculate the arrival time at each upcoming stop. To do so, each particle samples a speed for each segment from the network state, which could include correlations if these are available. Computing arrival times follows an iterative process.

Step 1: complete the current segment If the particle is at a stop, the (remaining) dwell time is sampled from the dwell time distribution (this is the same as in chapter 3), and the particle's departure time is stored. If the particle is not at a stop, and $\hat{p}_k > 0$, then the time to completion of the current segment ℓ is needed. When the bus is near the end of the segment (less than 200 m remaining), the particle's current speed is used, otherwise the speed is simulated from

$$v_\ell^{(i)} \sim \mathcal{N}_T\left(\beta_\ell,\, \zeta_\ell^2 + \boldsymbol{\psi}_\ell^2,\, 0,\, \mathbb{V}_\ell\right), \tag{5.5}$$

where $\mathbb{V}_\ell$ is the maximum speed along segment ℓ. Thus, the particle's speed along the remainder of segment ℓ is

$$v_\ell^{(i)\star} = \begin{cases} v_\ell^{(i)} & (1-\hat{p}_k)\mathcal{L}_\ell > 200, \\ \dot{x}_k^{(i)} & \text{otherwise.} \end{cases} \tag{5.6}$$

The travel time to the end of the current segment is thus obtained by

$$\tilde{z}^{(i)} = \frac{(1-\hat{p}_k)\mathcal{L}_\ell}{v_\ell^{(i)\star}}. \tag{5.7}$$

The overall *travel-time-so-far* is stored as $\eta^{(i)} = \tilde{z}^{(i)}$.

Step 2: compute arrival time at the next stop If the bus is not at a stop, segment speeds are forecast using the current network state and the particle's travel-time-so-far, $\eta^{(i)}$. In section 4.4, I demonstrated the possibility of using historical data to make forecasts, which could be used at this stage. However, as this infrastructure was not available at the time of writing, we simulate a speed from the following distribution instead, where P_{max} is the overall variance of speeds along all segments obtained from historical data:

$$v_j^{(i)} \sim \mathcal{N}_T\left(\hat{\beta}_j,\ \left[(\zeta_j + \eta^{(i)}q)^2 + \boldsymbol{\psi}_j^2\right] \wedge P_{\text{max}},\ 0,\ \mathbb{V}_j\right), \tag{5.8}$$

Travel-time-so-far $\eta^{(i)}$ is incremented by the travel time $z_j^{(i)}$ along segment j:

$$\eta^{(i)} = \eta^{(i)} + z_j^{(i)}, \quad \text{where} \quad z_j^{(i)} = \frac{\mathcal{L}_j}{v_j^{(i)}}. \tag{5.9}$$

This set-up allows forecasts for segments that are further away to increase in variability up to the maximum (this prevents speeds becoming unnecessarily variable).

Step 3: compute stop dwell time Once the particle arrives at the next stop, its dwell time is sampled from the dwell time distribution (section 3.1.1). If the stop is a *layover*, and the particle arrives before the scheduled departure time, then with probability 0.6

(figure 5.1) the particle waits to depart, otherwise, it leaves after the sampled dwell time and $\eta^{(i)}$ is incremented.

Step 4: repeat steps 2–3 This occurs until the particle reaches the end of the route.

Step 5: obtain arrival time distributions Having obtained arrival times for all $N^\star$ equally-weighted particles, the predictive distribution for stop j—as in chapter 3—is

$$p(\alpha_j \mid \mathcal{T}_k, \boldsymbol{\beta}_\mathcal{T}) \approx \sum_{i=1}^{N^\star} w^{(i)} \delta_{\alpha_j^{(i)}}(\alpha_j) = \frac{1}{N^\star} \sum_{i=1}^{N^\star} \delta_{\alpha_j^{(i)}}(\alpha_j)\,. \tag{5.10}$$

Estimates or quantiles are obtained from the particle approximation of the distribution of arrival times—for example, the mean, median, or a 90% credible interval. As discussed in appendix B.4, computing quantiles is a computationally demanding task for large numbers of particles due to the sorting of particles in order from earliest to latest arrival times. Using $N^\star < N$ helps to reduce this complexity, but particles need to be re-sorted at each stop. In chapter 6, I demonstrate an alternative, computationally simpler method of computing quantiles.

5.2.2 Normal approximation ($\mathcal{A}_\mathbf{n}$)

Due to the computational demand of the particle filter, significant speed improvements can be obtained by using a Normal approximation instead. The network state is a multivariate Normal random variable, so the issue lies with stop dwell times having a point mass at zero, resulting in a mixture predictive distribution. For each stop the vehicle passes, there are twice as many components, so after m stops, there are 2^m components. However, these regularly converge after a few stops, as shown in figure 5.2.

A mixture of Normal distributions can approximate the arrival time distribution (Wang and Chee, 2012) by expressing the mean and uncertainty as vectors $\tilde{\mu}$ and $\tilde{\sigma}^2$, respectively, along with a third vector $\tilde{\pi}$ denoting the $\tilde{N}$ mixture weights,[2] such that

$$\tilde{\pi}_i > 0, i = 1, \ldots, \tilde{N} \text{ and } \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i = 1. \tag{5.11}$$

[2] I place a tilde over parameters related to the Normal approximation, e.g., $\tilde{x}$, to help distinguish them from others used throughout the thesis.

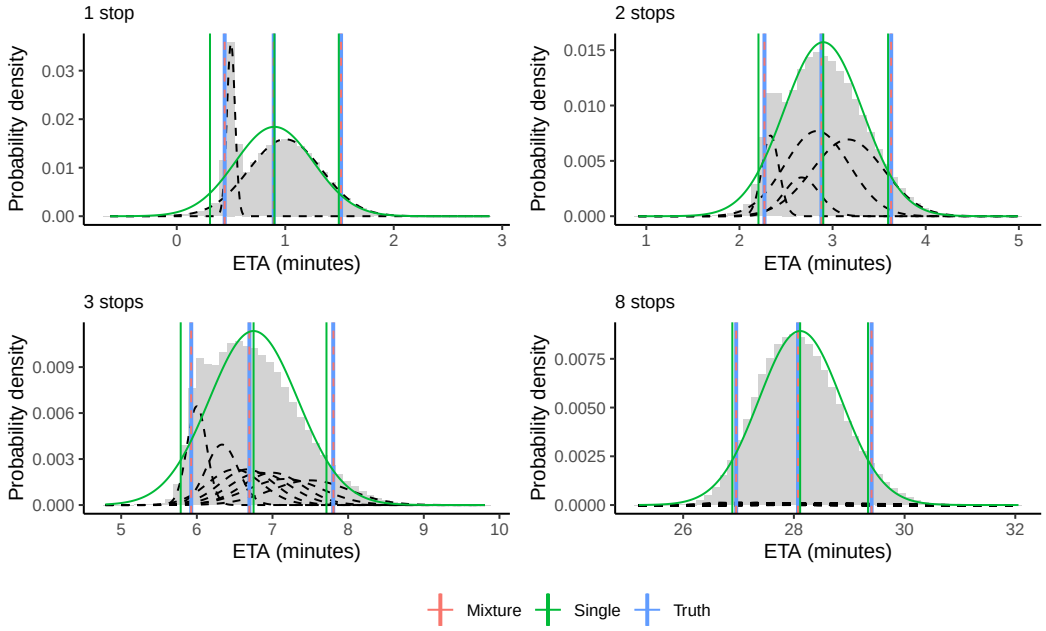


FIGURE 5.2: Normal approximation for 1, 2, 3, and 8 stops ahead. The full distribution is shown by the histogram with each components superimposed (dashed curves). The vertical lines represent the quantiles (2.5, 50, and 97.5%) computed using the samples (blue), the Normal mixture (red), and a single Normal approximation (green). The green curve is the single Normal approximation.

The arrival time at stop $j+n$ is then given by

$$\alpha_{j+n}|\tilde{\mu},\tilde{\sigma}^2,\tilde{\pi},\boldsymbol{\beta} = \sum_{\ell=j}^{j+n-1}\beta_\ell + \sum_{i=1}^{\tilde{N}}\tilde{\pi}_i z_i, \quad z_i \sim \mathcal{N}\left(\tilde{\mu}_i,\,\tilde{\sigma}_i^2\right). \tag{5.12}$$

Each component i has an indicator $I_{im} = \{0,1\}$ of whether it stopped at stop m, so the total dwell time has mean and variance

$$\tilde{\mu}_i = \sum_{m=j}^{j+n} I_{im}\tau_m \quad \text{and} \quad \tilde{\sigma}_i^2 = \sum_{m=j}^{j+n} I_{im}\omega_m^2, \tag{5.13}$$

respectively, assuming dwell times at individual stops are independent of each other.

Mixture weights are obtained through the stopping probability at each stop, π_j:

$$\tilde{\pi}_i = \prod_{m=j}^{j+n}\tilde{p}_{im}, \tag{5.14}$$

where

$$\tilde{p}_{im} = \begin{cases} \pi_m & \text{if } I_{im} = 1, \\ 1 - \pi_m & \text{otherwise.} \end{cases} \tag{5.15}$$

The mixture approximation works well for a few stops ahead, but after some time the mixture weights become small, and the components combine, as shown in figure 5.2. To prevent $\tilde{N}$ from becoming too large, the full distribution is simplified into a single component with mean

$$\begin{aligned} \mathbb{E}\left[\alpha_m | \tilde{\pi}, \tilde{\mu}, \tilde{\sigma}^2, \boldsymbol{\beta}\right] &= \mathbb{E}\left[\sum_{\ell=j}^{j+n-1} \beta_\ell + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i z_i\right] \\ &= \sum_{\ell=j}^{j+n-1} \mathbb{E}\left[\beta_\ell\right] + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i \mathbb{E}\left[z_i\right] \\ &= \sum_{\ell=j}^{j+n-1} \hat{\beta}_\ell + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i \tilde{\mu}_i \end{aligned} \tag{5.16}$$

and variance

$$\begin{aligned} \mathrm{Var}\left[\alpha_m | \tilde{\pi}, \tilde{\mu}, \tilde{\sigma}^2, \boldsymbol{\beta}\right] &= \mathrm{Var}\left[\sum_{\ell=j}^{j+n-1} \beta_\ell + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i z_i\right] \\ &= \sum_{\ell=j}^{j+n-1} \mathrm{Var}\left[\beta_\ell\right] + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i^2 \mathrm{Var}\left[z_i\right] \\ &= \sum_{\ell=j}^{j+n-1} \hat{\zeta}_\ell + \sum_{i=1}^{\tilde{N}} \tilde{\pi}_i \tilde{\sigma}_i^2, \end{aligned} \tag{5.17}$$

assuming segment travel time and dwell time are independent—assuming otherwise makes this model impossible to work with. Indeed, this model versus the particle filter (which makes no such assumption) is effectively testing the viability of this assumption.

An optimisation is used to obtain quantiles q_α such that

$$\left[p\left(\alpha \leq \alpha_m | \tilde{\pi}, \tilde{\mu}, \tilde{\sigma}^2, \boldsymbol{\beta}\right) - q_\alpha\right]^2 = 0. \tag{5.18}$$

This is straightforward using Brent's Algorithm (Brent, 1971), implemented in the 'Boost' C++ library.

When the 2.5%, 50%, and 97.5% quantiles for the single approximation are within

30 seconds of the same quantiles computed for the mixture distribution, the mixture is replaced with one single component with mean and variance defined by equations (5.16) and (5.17). In some situations, the mixture may not converge into a single distribution quick enough, so to prevent the number of components $\tilde{N}$ from exceeding $2^8 = 256$, all components with weights less than a predefined threshold (we used $\frac{1}{2}\max_i(\tilde{\pi}_i)$) are combined into a single component.

5.2.3 Historical arrival delays ($\mathcal{A}_\mathbf{h}$)

Another way to make predictions is to use historical data instead of real-time. We collected two weeks of data and recorded arrival time delays by stop and route to obtain a distribution which can then be used to make predictions. The arrival time at stop j along route r has an average delay of $\bar{d}_{rj}$ seconds with uncertainty v_{rj}, so the predicted arrival time at stop j on route r with scheduled arrival time S_{rj} is

$$\hat{\alpha}_j \sim \mathcal{N}\left(S_{rj} + \bar{d}_{rj},\, v_{rj}^2\right). \tag{5.19}$$

5.2.4 Schedule delays ($\mathcal{A}_\mathbf{s}$)

The currently deployed prediction method uses the scheduled arrival time at stop j along route r, S_{rj}, along with the arrival or departure time at the most recently visited stop, A_{rm} or D_{rm}, respectively, giving a current delay, in seconds, of

$$\kappa_r = \begin{cases} A_{rm} - S_{rm} & \text{if arrival,} \\ D_{rm} - S_{rm} & \text{if departure,} \\ 0 & \text{if neither observed for trip.} \end{cases} \tag{5.20}$$

The arrival time is then predicted as

$$\hat{\alpha}_{rj} = S_{rj} + \kappa_r. \tag{5.21}$$

Note, however, that if a trip has not yet been observed, the default delay is $d_r = 0$. This happens if there is not a vehicle servicing it, the vehicle is running late, or the trip has been cancelled altogether.

5.3 Assessing predictive performance

To compare the four prediction methods, we implemented the first two in 'transitr'. The program estimated and saved all arrival time estimates over a full day of data, along with uncertainty and the 5% and 90% quantiles. The accuracy of each method is assessed by comparison with the reported arrival times using RMSE, MAE, and mean absolute percentage error (MAPE). For each true arrival time X_a we have N estimates $\{X_n : n = 1, \ldots, N\}$. Then we calculate the assessment criteria as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(X_a - \hat{X}_n\right)^2}, \tag{5.22}$$

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}\left|X_a - \hat{X}_n\right|, \tag{5.23}$$

$$\text{MAPE} = \frac{100}{N}\sum_{n=1}^{N}\left|\frac{X_a - \hat{X}_n}{X_a}\right|. \tag{5.24}$$

RMSE and MAE are only consider the size of the difference between the predicted and observed values, and so can be skewed by very long-term predictions with low accuracy. Thus we also include MAPE, which scales the errors by the actual time until arrival. Therefore, it compares accuracy as a percentage of the actual time until arrival. Since predictions change over time, we also computed the summary values by *time until (actual) arrival*, allowing the comparison of models at different time points, as well as by stop sequence and time of day.

Another useful criteria is the prediction interval coverage probability (PICP),

$$\text{PICP} = \frac{1}{N}\sum_{n=1}^{N} I_n, \quad I_n = \begin{cases} 1 & X_a \in (X_{n,\ell}, X_{n,u}), \\ 0 & \text{otherwise.} \end{cases} \tag{5.25}$$

which is only available for the three methods that provide an 85% prediction interval, and allows us to assess how well the model accounts for uncertainty in the predictions. For the particle filter, prediction intervals are obtained by sorting the particles in order of arrival time and taking the $\lfloor 0.05N^\star \rfloor^{\text{th}}$ particle as the lower bound, and the $\lceil 0.9N^\star \rceil^{\text{th}}$ particle as the upper bound (more details in appendix B.4). For the Normal approximation and historical arrival methods, the inverse cumulative density

TABLE 5.1: Comparison of the predictive performance of the four methods after estimating arrival times for a full day of historical data. RMSE, MAE, and MAPE measure the accuracy of the mean or median, and PICP measures the accuracy of the distribution: the nominal coverage is 85% (PICP is not available for the schedule-delay method).

	RMSE (s)	MAE (s)	MAPE (%)	PICP (%)
$\mathcal{A}_p$: Particle filter	232	146	19	77
$\mathcal{A}_n$: Normal approximation	489	349	38	91
$\mathcal{A}_h$: Historical delays	244	166	47	84
$\mathcal{A}_s$: Schedule-delay	238	164	27	

function provides the required quantiles. The results are displayed in table 5.1 and figures 5.3 to 5.5 and described in section 5.3.1.

Additionally, we want to compare the reliability of the various forecast methods, namely *the probability of arriving before the bus*, and hence not missing it, as well as the expected waiting time given a passenger arrives at the stop by a certain time. Table 5.2 and figures 5.6 to 5.9 use the point estimate (mean or median, depending on the forecast method) and the 5% quantile to calculate the probability that the bus arrives after each estimate. Section 5.3.2 discusses these results.

5.3.1 Comparing the accuracy of arrival time prediction

The accuracy measurements (RMSE, MAE, and MAPE) shown in table 5.1 immediately show that the Normal approximation ($\mathcal{A}_n$) estimates are, on average, about half as accurate as the other methods. Overall, the particle filter ($\mathcal{A}_p$) demonstrates the greatest accuracy by all criteria, indicating that its estimates are (on average) closer to the true value in both absolute and relative terms. The historical delays approach ($\mathcal{A}_h$) has similar accuracy to the schedule-delay approach ($\mathcal{A}_s$) in absolute terms (RMSE and MAE), but the least accurate overall in relative terms (MAPE). This indicates that $\mathcal{A}_h$ has worse accuracy for short-term forecasts, which is not surprising as it uses only historical data, ignoring real-time information about vehicle location and network state.

As for the PICP, the theoretical coverage is 85%. Model $\mathcal{A}_p$ underestimates arrival time uncertainty, indicating that the model is not capturing enough uncertainty: this could be any combination of dwell time, travel time, or the unknown. Conversely, $\mathcal{A}_n$ overestimates uncertainty by about 5%. Model $\mathcal{A}_h$ has close to the desired PICP,

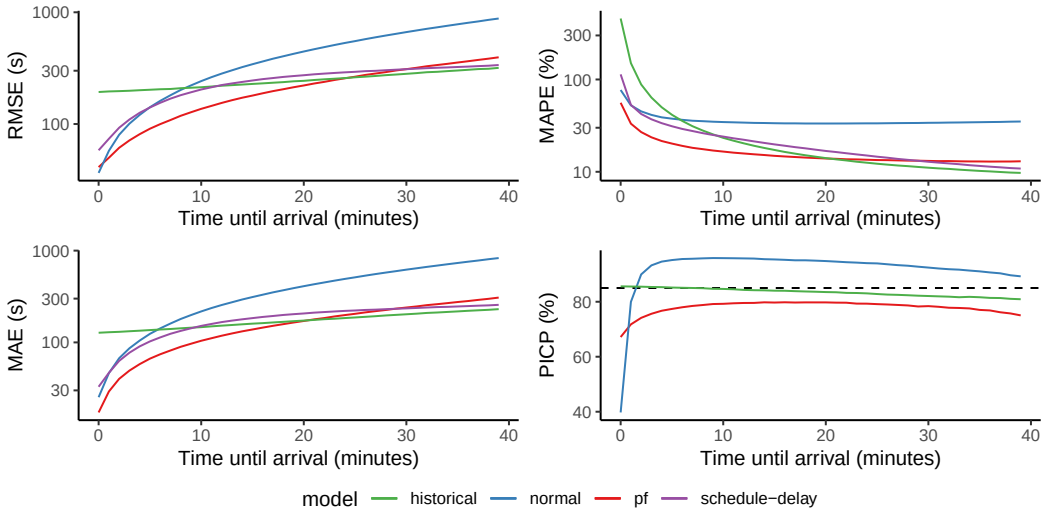


FIGURE 5.3: Model comparative statistics as a function of time-until-arrival. Note the log-scale for RMSE, MAE, and MAPE. The dashed line in the graph of PICP indicates the nominal coverage of 85%.

which demonstrates that arrival time delays at each stop show a certain level of consistency from day-to-day.

The main finding of table 5.1 is that the Normal approximation is inadequate for this problem. However, this was not unexpected: in previous chapters, I emphasised the non-Gaussian nature of a vehicle's state. We now explore the effects of time-until-arrival, stop sequence, and time-of-day on prediction accuracy to more closely inspect the relative performance of the methods.

Time until actual arrival

We would expect arrival time predictions for a given stop to become more accurate as the bus approaches. To assess this, arrival time estimates were binned into one-minute intervals, and for each, the summary statistics were estimated, which are displayed in figure 5.3. Note the use of log scales for RMSE, MAE, and MAPE.

The RMSE and MAE for all methods increase with time-until-arrival as expected. The rate of this increase, however, varies between them. The three methods that use real-time information ($\mathcal{A}_\text{p}$, $\mathcal{A}_\text{n}$, and $\mathcal{A}_\text{s}$) demonstrate small absolute error when the bus is near, which increases the farther out the bus gets, while $\mathcal{A}_\text{h}$ shows a more constant error and far poorer accuracy when the bus is less than 5–10 minutes away. Model $\mathcal{A}_\text{p}$ has the lowest error up until 20 minutes before arrival and has greater

accuracy than $\mathcal{A}_s$ up until 25–30 minutes. This indicates that the current delay is only a useful predictor when the bus has almost arrived, and that accounting for real-time traffic conditions does improve the accuracy of arrival time prediction. Model $\mathcal{A}_n$ quickly shows large prediction errors for all but the nearest buses.

The MAPE for all methods decreases with time-until-arrival, with $\mathcal{A}_h$ showing the highest relative error when the bus is less than about 5 minutes away. Model $\mathcal{A}_n$ remains somewhat constant with a relative error of about 50%. Again, for the first 20 minutes, $\mathcal{A}_p$ has the smallest relative error, and outperforms $\mathcal{A}_s$ up until the bus is 25–30 minutes away. After 30 minutes, $\mathcal{A}_p$ converges with slightly lower accuracy than $\mathcal{A}_h$ and $\mathcal{A}_s$, which may continue to decrease beyond 40 minutes. However, 89% of routes in Auckland are less than one hour, so the number of arrival time predictions made when the bus is more than 40 minutes away is small so results will be strongly affected by individual routes.

The PICP is not available for $\mathcal{A}_s$ since that method only provides point estimates. The coverage for the three remaining methods is reasonably constant across time-until-arrival (figure 5.3). As seen earlier, $\mathcal{A}_p$ underestimates arrival time uncertainty, noted by lower than expected coverage. Model $\mathcal{A}_n$ overestimates uncertainty when the bus is more than a few minutes away, and thoroughly underestimates uncertainty when the bus is near, likely because it does not account for enough vehicle state uncertainty. Model $\mathcal{A}_h$ exhibits good coverage, although this drops slightly as time-until-arrival increases.

Stop sequence

Figure 5.4 shows the RMSE, MAE, MAPE, and PICP versus stop sequence. The first stop is excluded since arrival times are only predicted once the bus begins the trip. Early stops along a route seldom have long until the bus arrives since ETAs are only made once the trip starts. Stop sequence is truncated to 50 (92% of trips have fewer than that many stops).

Both RMSE and MAE increase with stop sequence for all models. Model $\mathcal{A}_n$ shows much higher errors than the other three methods, and $\mathcal{A}_p$ exhibits higher prediction accuracy than $\mathcal{A}_h$ and $\mathcal{A}_s$ up until about stop 30, at which point there is no clear difference between these three methods. However, in terms of relative

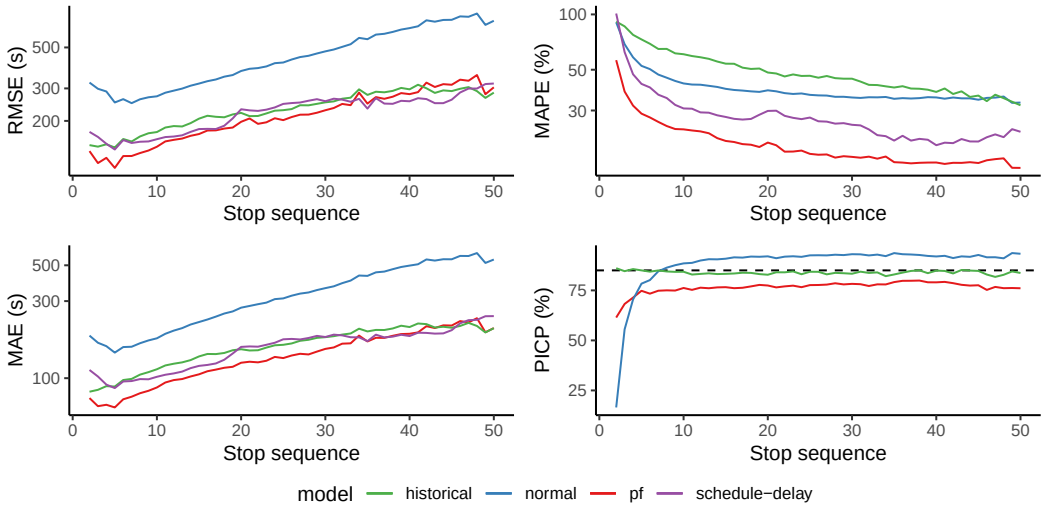


FIGURE 5.4: Model comparative statistics as a function of stop sequence. RMSE, MAE, and MAPE once more use a log-scale. The dashed line in the graph of PICP indicates 85% nominal coverage.

error (MAPE), $\mathcal{A}_p$ outperforms the others for all stops, while $\mathcal{A}_h$ shows the poorest accuracy, particularly for early stops.

PICP shows much the same trend as before with $\mathcal{A}_p$ underestimating arrival time uncertainty at all stops, while $\mathcal{A}_n$ overestimates for all but the first few stops—it performs poorly for those. Model $\mathcal{A}_h$ has the desired coverage for all stops.

Time of day

Observations were binned into 15-minute intervals and summary statistics calculated for each. The results, displayed in figure 5.5, differ from those seen previously, now revealing the peak-hour traffic effect. There is a single morning peak at around 8 am (school and work begin at about the same time) and two evening peaks: one for schools at about 3 pm, and another for workers at around 5 pm.

During off-peak (about 9:30 am until 2:30 pm) $\mathcal{A}_p$ shows the smallest error (RMSE and MAE), with $\mathcal{A}_h$ and $\mathcal{A}_s$ exhibiting slightly larger errors. Model $\mathcal{A}_n$ again has much poorer accuracy.[3] During peak times, all methods show an increase in prediction error as traffic conditions worsen and become more unreliable. Model $\mathcal{A}_s$ is least affected by the peak effect since the schedules do account somewhat for peak congestion, while the current implementation of our method only uses the *current*

[3]Remember to note the log scale.

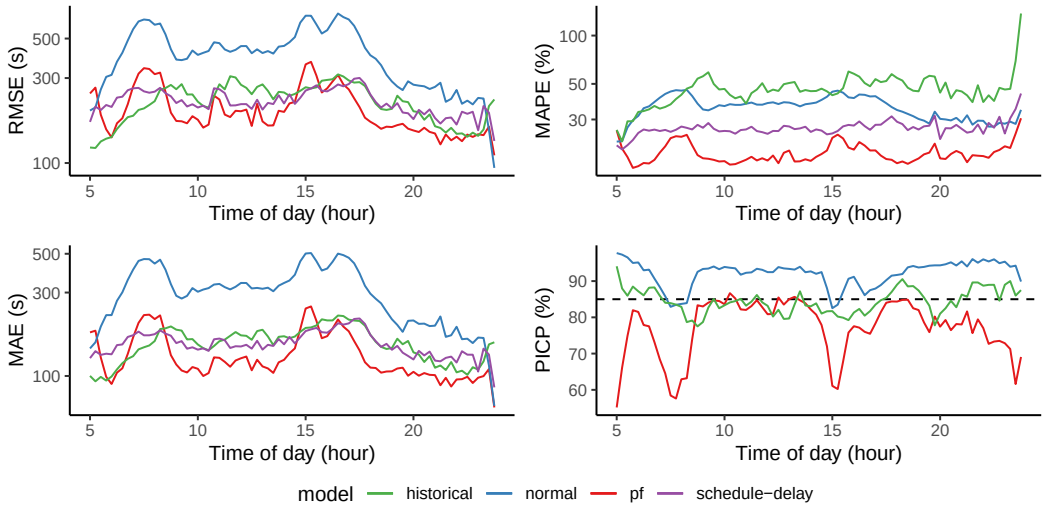


FIGURE 5.5: Model comparative statistics as a function of time of day. RMSE, MAE, and MAPE use log-scales, and the dashed line in the graph of PICP indicates 85% nominal coverage.

traffic state; however, as discussed in section 4.4, improvements could be achieved by implementing a better forecast method.

In terms of relative error (MAPE), we see a less accentuated peak effect, and now $\mathcal{A}_{\text{p}}$ demonstrates the best accuracy throughout the day. During peak time, traffic can quickly become more or less congested, so long-term predictions are prone to inaccuracy as we saw in figure 5.3. Therefore, peak traffic has a noticeable effect on RMSE and MAE, but less so on MAPE. MAPE for $\mathcal{A}_{\text{s}}$ and $\mathcal{A}_{\text{h}}$ seems unaffected by peak hour.

Finally, we look to PICP, where we can truly see the effect of peak traffic on the predicted arrival time distributions, particularly when obtained using $\mathcal{A}_{\text{p}}$. During off-peak, $\mathcal{A}_{\text{p}}$ has very close to the desired coverage of 85%; during peak time, however, coverage drops quite significantly. This drop is likely caused by travel times quickly increasing as congestion builds, meaning initial predictions are too early, and then as travel times decrease again once peak time has passed, the predictions are too late. Our particle filter appears to be able to estimate current travel times accurately, but it would benefit from the forecasting improvement described in section 4.4.

TABLE 5.2: Comparison of the reliability of the four prediction methods based on the results of estimating arrival times for a full day of historical data. $\mathbb{P}_m$ and $\mathbb{P}_\ell$ are proportions of observed events, and $\mathbb{E}_\ell$ and its associated interval are based on the mean and quantiles.

	$\mathbb{P}_m$ (%)	$\mathbb{P}_\ell$ (%)	$\mathbb{E}_\ell$ (m)	5–90% CI
$\mathcal{A}_\text{p}$: Particle filter	64	93	4.4	0.3–10.2
$\mathcal{A}_\text{n}$: Normal approximation	96	98	20.9	2.9–37.6
$\mathcal{A}_\text{h}$: Historical delays	54	98	7.1	1.1–14.1
$\mathcal{A}_\text{s}$: Schedule-delay	38			

5.3.2 Assessing the reliability of arrival time prediction

RMSE, MAE, and MAPE measure the predictive accuracy of the methods, but do not account for the costs associated with inaccurate predictions. In this section, we evaluate the *reliability* of arrival time distributions by examining:

1. the wait time at the bus stop; and
2. the probability of missing the target bus.

In most cases, the latter incurs a much higher cost, but depends entirely on time until the *next* bus arrival: for high-frequency routes, this is small; for low-frequency ones, however, it can become quite high.[4] In this section, I do not differentiate between high and low-frequency routes (I do in section 6.1, however).

The three statistics we compare across the methods are:

- $\mathbb{P}_m = \mathbb{P}\left(A_m \geq \hat{\alpha}_m\right)$, the probability that the vehicle arrives after the point estimate, $\hat{\alpha}_m$, indicating that were a passenger to arrive at the stop by $\hat{\alpha}_m$, they would catch the bus with probability $\mathbb{P}_m$;
- $\mathbb{P}_\ell = \mathbb{P}\left(A_m \geq \hat{\alpha}_{m,\text{lower}}\right)$, the probability that the vehicles arrives after the lower bound of the prediction interval; and
- $\mathbb{E}_\ell = \mathbb{E}\left[A_m - \hat{\alpha}_{m,\text{lower}} \,\middle|\, A_m \geq \hat{\alpha}_{m,\text{lower}}\right]$, the expected waiting time for a passenger arriving at the lower predictive bound, given that the bus arrives after it (that is, conditional on catching the bus).

The overall results are displayed in table 5.2. If a passenger, at any time, looks at the ETA of their bus *once* and arrives at the stop by the indicated time, then using $\mathcal{A}_\text{p}$ the probability of catching the bus is 64% versus only 38% using $\mathcal{A}_\text{s}$. For $\mathcal{A}_\text{h}$, $\mathbb{P}_m$ is

[4] Some routes only run hourly!

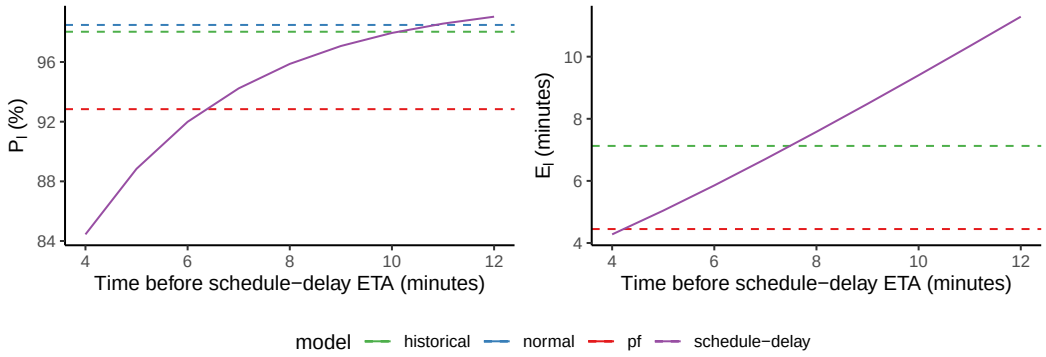


FIGURE 5.6: Capture probability and expected wait times for a traveller arriving x minutes before the schedule-delay prediction. The relevant values for the three other methods are displayed by dashed lines.

about 50%, while for $\mathcal{A}_n$ it is over 95%, indicating that the Normal approximation is underestimating arrival times.

The concept behind examining the accuracy of the lower quantile, $\mathbb{P}_\ell$, is that this value should give passengers the best chance of catching the bus.[5] We used the 2.5% quantile for the lower estimate, so we would expect the bus to arrive after the predicted time 97.5% of the time. This is the case for $\mathcal{A}_n$ and $\mathcal{A}_h$, but $\mathcal{A}_p$ has a slightly lower probability that expected. Associated with the lower bound is the expected wait time until the bus arrives, conditional on not having missed it ($\mathbb{E}_\ell$). Model $\mathcal{A}_p$ has the shortest expected wait followed closely by $\mathcal{A}_h$, while the wait time using $\mathcal{A}_n$ is about four times as long as that of $\mathcal{A}_p$.

Since $\mathcal{A}_s$ provides only a point estimate, we cannot compare it directly to the other methods, in particular, $\mathcal{A}_p$. To do so, we can *indirectly* compare these methods by proposing that a passenger arrives x minutes before the arrival time predicted by $\mathcal{A}_s$, similar to the method described by Cathey and Dailey (2003), and calculating the probability of capture and expected wait time. The resulting curves for arriving 4–12 minutes before the stated arrival are shown in figure 5.6, with the values for the other three methods overlaid as dashed lines.

First, let us consider the probability of the bus arriving after the estimated arrival time, $\mathbb{P}_\ell$. Based on the data collected, a passenger would need to arrive at least 10 minutes before the ETA predicted by $\mathcal{A}_s$ to have a 97.5% probability of catching the

[5] Without having to arrive before the bus has even left the depot.

bus. To obtain the same probability as $\mathcal{A}_\text{p}$, this would be a little more than 6 minutes before arrival. Now for the expected waiting time: to achieve the targetted 97.5% chance of catching the bus, the passenger would arrive 10 minutes before the stated ETA which, from the figure on the right, has an expected waiting time of about 9 minutes. Arriving 6 minutes before yields an expected wait time of about 6 minutes, which is longer than the expected wait time under $\mathcal{A}_\text{p}$ (a little over 4 minutes), despite having the same probability of catching the bus.

We can also consider the reverse: a passenger would need to arrive no more than 4 minutes before the stated arrival time to have the same expected wait as $\mathcal{A}_\text{p}$, giving them an 85% chance of catching the bus compared to 93% for $\mathcal{A}_\text{p}$. For the remainder of this section, we use 6 minutes before the specified ETA to obtain a lower bound for $\mathcal{A}_\text{s}$, and once more compare the probabilities as a function of time-until-arrival, stop sequence, and time of day.

Time until actual arrival

Figure 5.7 shows $\mathbb{P}_m$, $\mathbb{P}_\ell$, and $\mathbb{E}_\ell$ computed at one-minute intervals for each of the four methods. Model $\mathcal{A}_\text{s}$ used a lower bound of $\hat{\alpha} - 300$ (6 minutes before the point estimate). For the expected wait time, 5% and 90% quantiles are displayed as shaded regions. The dotted line in the expected waiting time graph represents the maximum possible waiting time; that is, the waiting time for a passenger already at the stop, given the bus is 10 minutes away, is 10 minutes.

The probability that the bus arrives after the point estimate (mean or median), $\mathbb{P}_m$, is more or less constant with a slight decrease as the bus nears the stop. Models $\mathcal{A}_\text{p}$ and $\mathcal{A}_\text{n}$ are above the expected value of 50%, though the latter is significantly higher. Model $\mathcal{A}_\text{n}$ performs reasonably well, though it tends to underestimate arrival time the farther out the bus is, while $\mathcal{A}_\text{s}$ overestimates arrival time such that there is less than a 40% chance of catching the bus if it is less than 20 minutes away.

For $\mathbb{P}_\ell$, we observe a different trend: $\mathcal{A}_\text{p}$ and $\mathcal{A}_\text{h}$ increase the further out the vehicle is; on the other hand with $\mathcal{A}_\text{s}$, the probability decreases, an artefact of using a fixed lower bound. Once the bus is about 15 minutes away, the lower bound of $\hat{\alpha} - 300$ ($\mathcal{A}_\text{s}$ minus 6 minutes) is increasingly likely to result in a caught bus, while

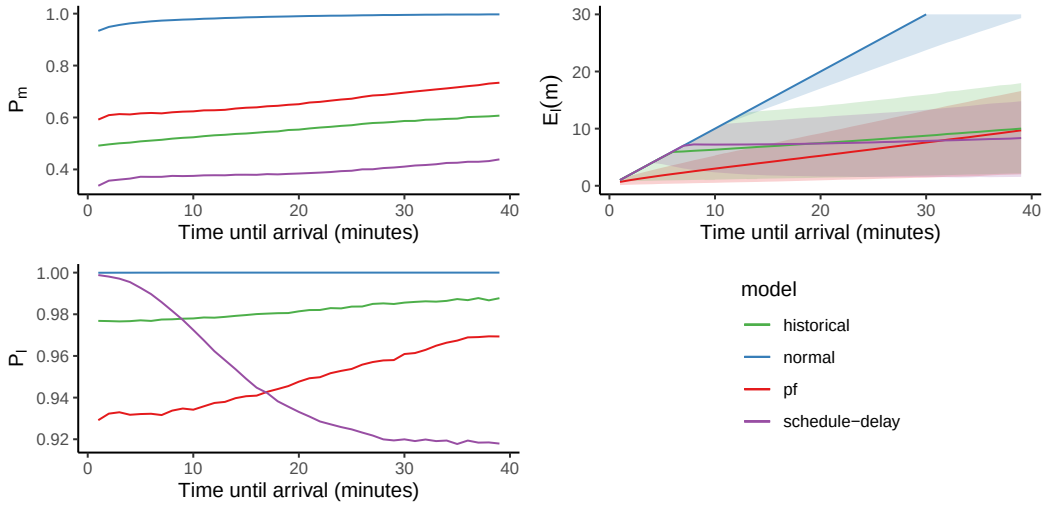


FIGURE 5.7: Capture probabilities and expected wait times for the models by time until the bus's actual arrival. Expected wait time (right) includes a shaded region for the 5 and 90% quantiles of wait times.

under $\mathcal{A}_p$ this probability is lower since the width of the interval decreases as fewer sources of variability remain.

Finally, we examine the expected waiting time given a passenger arrives at the lower bound *and* the bus arrives after it. Model $\mathcal{A}_p$ has a consistently shorter waiting time, though by about 30 minutes until arrival the three methods (excluding $\mathcal{A}_n$) are approximately the same. We see that the expected waiting time for $\mathcal{A}_s$ is more or less independent of time-until-arrival, which is to be expected since the width is fixed. The shaded regions represent 5% and 90% quantiles of waiting time, revealing that $\mathcal{A}_p$ results in shorter waits, on average, when the bus is less than 10 minutes away. The Normal approximation tends towards overestimation of arrival time uncertainty, so expected waiting time is considerable if arriving at the lower estimate, indicating that $\mathcal{A}_n$ is not a reliable choice for arrival time prediction.

Stop sequence

The same values were computed by stop sequence and displayed in figure 5.8. Model $\mathcal{A}_h$ achieves the target probability of 50%, while $\mathcal{A}_p$ is slightly above at about 60%. Most notable is the steady decline in the reliability of $\mathcal{A}_s$, dropping to about a one in three chance of catching the bus after stop 40 (assuming you arrive at the predicted ETA).

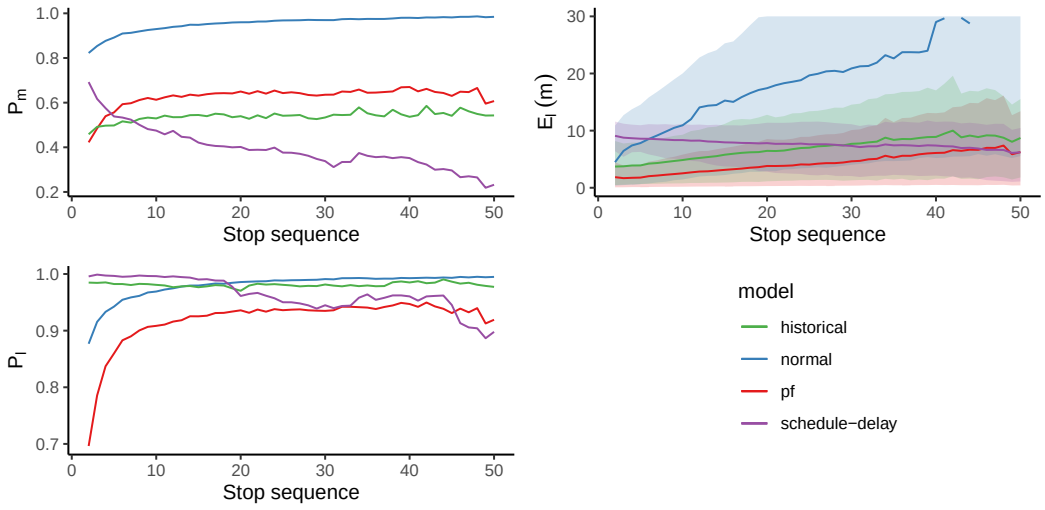


FIGURE 5.8: Capture probabilities and expected wait times for the models by stop sequence. Expected wait time (right) includes a shaded region for the 5 and 90% quantiles of wait times.

Moving on to the lower 2.5% quantile, $\mathbb{P}_\ell$, we expect 97.5% probability of capture. We see that, as before, $\mathcal{A}_\text{p}$ overestimates arrival time slightly, with $\mathbb{P}_\ell$ of about 90%, while for $\mathcal{A}_\text{s}$ it decreases with stop sequence. The associated wait time, $\mathbb{E}_\ell$, increases for all methods with increasing stop sequence *except* for $\mathcal{A}_\text{s}$, which decreases (remembering that the width is fixed). Model $\mathcal{A}_\text{p}$ has the lowest expected wait time, and the upper 90% interval is lower than the mean for $\mathcal{A}_\text{s}$ for the first 20 stops. However, many stops with low indices are serviced within the first 5–10 minutes of the trip, thus inflating $\mathbb{P}_\ell$ under $\mathcal{A}_\text{s}$.

Time of day

The probabilities and expected waiting times were calculated for 15-minute intervals for each of the models, as shown in figure 5.9. The peak hour effects are once more visible and associated with an increased probability of arriving before the bus under $\mathcal{A}_\text{p}$ for both the median and 2.5% quantile. For $\mathcal{A}_\text{s}$, the probability of arriving before the bus decreases slightly during peak times. In the evening, we see the performance of $\mathcal{A}_\text{p}$ drop rapidly, likely due to the much-reduced probability of stopping (many evening services have low occupancy). Future work implementing more specific dwell time models could help to improve this.

We see much the same pattern with the expected wait time $\mathbb{E}_\ell$, which increase by

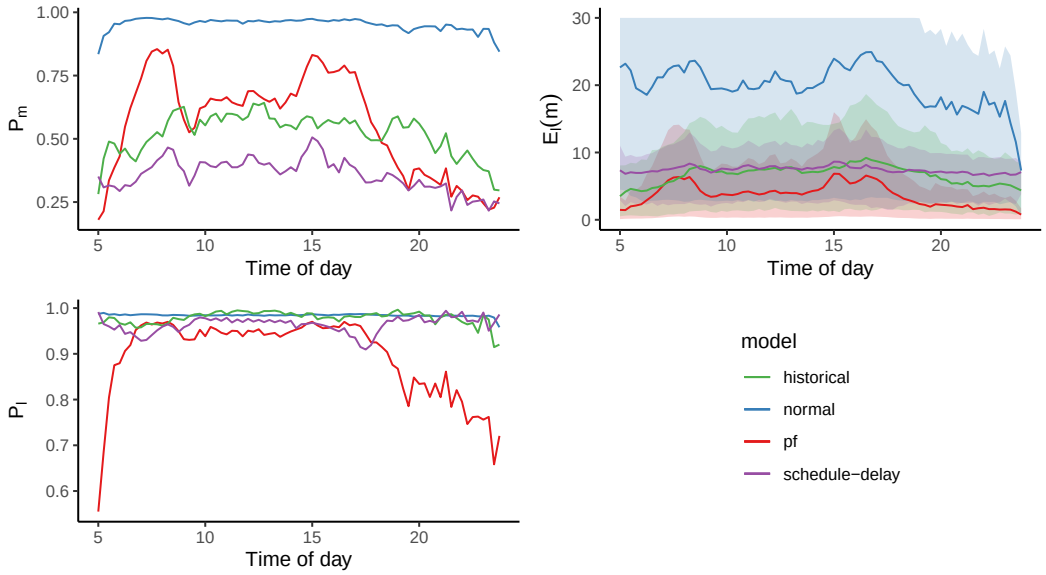


FIGURE 5.9: Capture probabilities and expected wait times for the models by time of day. Expected wait time (right) includes a shaded region for the 5 and 90% quantiles of wait times.

a few minutes under $\mathcal{A}_p$ during peak times. There is little noticeable change for $\mathcal{A}_s$ and $\mathcal{A}_h$. Even so, $\mathcal{A}_p$ consistently provides shorter expected wait times than $\mathcal{A}_s$.

5.3.3 Result summary

The results in sections 5.3.1 and 5.3.2 demonstrate the relative accuracy and reliability of four methods of predicting the arrival time distribution at a stop. The particle filter ($\mathcal{A}_p$) uses a sample of weighted particles representing the vehicle's state, along with the real-time traffic state, to approximate the distribution. On average, it is slightly more accurate than the currently used method ($\mathcal{A}_s$), particularly when the bus is within 20 minutes of arriving and during the daytime off-peak period, and captures most of the uncertainty during this period (as per PICP).

As for the reliability of the estimates—that is, their use as an ETA for commuters—the particle filter tends to underestimate arrival time, whereas the current method tends to *overestimate* it. In terms of the application, the former is preferred, often resulting in a 30% higher probability of catching the bus. The particle filter provides an estimate of the arrival time distribution, allowing easy estimation of quantiles. We have seen that the lower bound is typically an overestimate, indicating that perhaps not enough variability is being included, or perhaps that more particles

are needed. The presented results used $\tilde{N} = 200$, but future work could involve modifying this depending on the number of stops remaining along a route. However, the lower bound still gives a 90–95% chance of catching the bus in most cases and has a significantly shorter wait time compared to the current method (assuming one arrives 6 minutes before the specified time). This difference indicates that the particle filter, as well as providing improved prediction accuracy, can offer more reliability for passengers.

We saw that there were a few scenarios that could be attributed to much of the particle filter's weaker performance, namely peak times and evenings. Further work could be performed, particularly on forecasting network travel times, to improve these predictions. We also presented two other models. The Normal approximation ($\mathcal{A}_{\mathrm{n}}$) demonstrated inferior performance in that it tended to overestimate arrival time uncertainty, a likely result of combining the mixtures. Historical data ($\mathcal{A}_{\mathrm{h}}$), in which the arrival time prediction distribution was based purely on the distribution of arrival time delays at stops along a route, performed admirably. It estimated the distribution quantiles well, but its reliability (at least compared to the other methods) decreased once the trip was underway and real-time information was available. It does, however, suggest that historical data should be used for prior predictions of arrival time: before a trip has registered with the real-time service, the best prediction of arrival time is $\mathcal{A}_{\mathrm{h}}$.

5.4 Real-time performance

The primary constraint on real-time applications is time, especially when the predictions will be out of date once the next vehicle observations are available 30 seconds after the first. We recorded the timings of each component of our application during the simulation described in section 5.3. Table 5.3 presents the mean timing results for each component using both *wall clock* (the time passed as recorded by a clock on the wall), as well as the *CPU clock* (the processing time). The latter represents the overall computational complexity of the problem. Since computations are spread out over multiple cores (in this simulation we used 3) the actual wall clock is up to 3 times faster in the vehicle update, network update, and ETA prediction steps (we

TABLE 5.3: Average timings of the six components of each iteration, running on three cores, along with average total iteration time. Wall clock is the real-world time passed, and CPU time is the processing time, all reported in milliseconds.

	Wall clock	(SE)	CPU time	(SE)
(L) Load data	28.1	(0.22)	16.7	(0.19)
(U) Update vehicle information	3.47	(0.04)	3.32	(0.034)
(V) Vehicle state update	3930	(37)	10400	(96)
(N) Network state update	0.874	(0.12)	1.6	(0.026)
(P) Predict ETAs	1650	(26)	3490	(39)
(W) Write ETAs to protobuf feed	327	(1.5)	341	(1.1)
(T) Total iteration time	5940	(55)	14200	(130)

discussed multithreading in section 2.5). In some steps (such as while loading data), the wall clock time is longer than the CPU time: this is because the step involved downloading data from the server, during which time the processor is mostly inactive while it waits for the file.

From the results in table 5.3, we see that the average iteration time is about 6 seconds, which is well below our original target of 30 seconds. The CPU time is about 14 seconds, so, on average, we see a 50% reduction attributed to multithreading on three cores. The most computationally intensive step is the vehicle state update, which involves transitioning, reweighting, and occasionally resampling 5000 particles for each vehicle, which can exceed 1000 at peak time. Next is the arrival time prediction step, which also involves estimation with a subset of particles (in this simulation, we used 200). The last significant step is writing the ETAs to file, which additionally (in this case) involved computing the median arrival time for each vehicle and stop combination (appendix B.4). The remaining steps each had average times of less than 0.1 seconds.

Since the number of vehicles changes throughout the day, as displayed in the top half of figure 5.10, we also display the timings over time, as shown in the lower half of the figure. We see the two peak periods: morning peak, which includes both school and workers at the same time, and the broader evening peak since schools finish at about 3 pm while most workers finish about 5 pm. When the number of vehicles exceeds 1100, the total iteration time is around 9 seconds and remains at around 6 seconds during daytime off-peak. Again, this is the wall clock iteration time, so deployment on a server with more cores would result in even faster iterations.

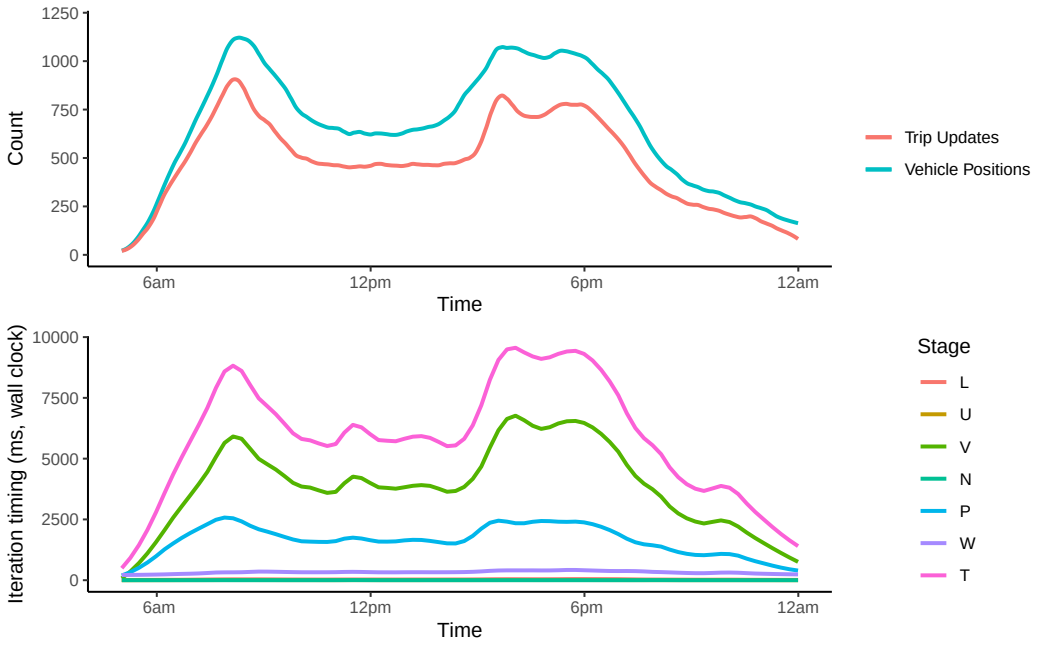


FIGURE 5.10: Top: number of vehicles and trip updates at different times of the day. Bottom: timing results for various stages of each iteration: (L) Load data, (O) Update vehicles information, (V) Vehicle state update, (N) Network state update, (P) Predict ETAs, (W) Write ETAs to protobuf feed, (T) Total iteration time.

5.5 Chapter contributions

- I use a particle filter to incorporate real-time transit network state into arrival time distributions for every bus at every stop in the network, in real-time.
- Comparison with current GTFS method shows improvements and demonstrates that further work (network state forecasts) could make further improvements.
- Demonstration that the real-time application (implemented within the 'transitr' package) runs well within the target timeframe of 30 seconds (10 seconds per iteration at peak time; less during off-peak).

Chapter 6

Arrival time prediction for journey planning

Arrival time prediction is most often used to provide commuters with a countdown while they wait for their bus, and has been shown to reduce experienced wait time (Schweiger, 2003). However, research has also shown that passengers who use real-time information have shortened actual wait times (Lu et al., 2018), demonstrating that, if reliable, passengers can use real-time information to better plan their commutes. In this chapter, we shift our focus from the probabilistic view of arrival time estimation in chapter 5 to a practical one.

Accurately predicting the arrival time of a transit vehicle is a decidedly difficult problem. There are several known sources of variability—traffic lights and intermediate bus stops, for example—but also countless others which we cannot model. Mazloumi et al. (2011) proposed a method of finding prediction intervals of arrival time using neural networks. More recently, Fernandes et al. (2018) explored methods for displaying uncertainty, which included using quantile dot plots, to provide travellers with a means of making informed decisions. We examine the use of a single point estimate in an attempt to balance reliability and accuracy and compare this with the use of an interval estimate to convey uncertainty.

Another desirable aspect of real-time information is journey planning: given an origin and a destination, and optionally one or more constraints, what is the optimal route (or routes for journeys requiring multiple *legs*). Bérczi et al. (2017) developed an approach to journey planning that uses a *probabilistic model* of arrival times to

decide between competing options. Since the initial selection of candidate options is a complicated topic (see Häme and Hakula (2013a,b) and Zheng, Zhang, and Li (2016) for methods of doing so) we do not consider this problem here. Instead, we use the results from chapter 5 to decide between pre-selected options.

Both the arrival time estimates (point and interval) and journey planning decisions use the arrival time distributions estimated in chapter 5. First, however, we need to express the distribution of arrival times in a form that is easy to work with and distribute. The first consideration is that estimated times of arrival are typically displayed in minutes as integers, so any estimates first need to be rounded. Take, for example, an estimated time of arrival of 105 seconds, or 1.75 minutes, which we may round up to 2 minutes, or down to 1 minute. We use Bayesian posterior predictive probabilities to estimate the median arrival time $a_{0.5}$ such that $\mathbb{P}\left(A \geq a_{0.5}\right) = 0.5$. If we round 1.75 in the above example to $\hat{a}_{0.5} = 2$ minutes, then the equality is incorrect: the probability that the actual arrival time A is later than the estimated arrival time $\hat{a}_{0.5}$ is, in fact, *less* than 50%. However, if we round *down* to $\hat{a}_{0.5} = 1$ minute, then, although the equality remains invalid, the probability that the bus arrives after $\hat{a}_{0.5}$ is *at least* 50%.

To compute the CDF of integer-valued arrival times, we first convert each particle's arrival time $\alpha^{(i)}$ to minutes and round down, which is the equivalent of taking its modulus with 60:

$$\breve{\alpha}^{(i)} = \alpha^{(i)} \mod 60. \tag{6.1}$$

Next, we tally the number of particles with each arrival time a and calculate the probability of the bus arriving in the interval $[a, a+1)$,

$$\mathbb{P}\left(A \in [a, a+1)\right) = \frac{1}{N^\star} \sum_{i=1}^{N^\star} I_{\breve{\alpha}^{(i)}=a}, \tag{6.2}$$

where the indicator $I_{a=b}$ is 1 if a equals b, and zero otherwise. Computationally, this is significantly easier than computing quantiles, since the latter involves sorting the particles (see appendix B.4), whereas tallying integers can be performed with a single, unsorted pass over the particle vector. This provides significant performance enhancements, remembering that quantiles require sorting of particles *at every stop*.

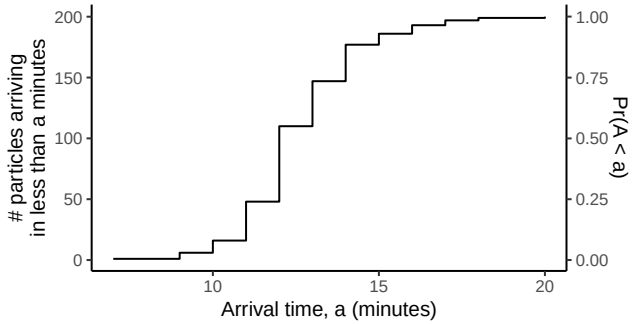


FIGURE 6.1: CDF of arrival time after rounding particle estimates down to the nearest minute.

Using equation (6.2), the CDF is

$$\mathbb{P}\left(A < a\right) = \sum_{x=0}^{x=a-1} \mathbb{P}\left(A \in [x, x+1)\right). \tag{6.3}$$

Figure 6.1 displays the CDF for a single stop. The remainder of this chapter explores the reliability and usefulness of various summary statistics to convey this distribution to commuters.

6.1 Estimates of arrival time

As previously discussed, there is considerable uncertainty in the posterior predictive distribution of arrival time. This uncertainty makes point estimates challenging to provide since there is a strong negative correlation between *accuracy*—how far the estimates are from the actual value—and *reliability*—how *useful* the estimate is. Take, for example, the median, which has, in theory, the best accuracy, but 50% of the time the bus arrives earlier than it predicts, which could result in a passenger missing their bus were they to arrive at the predicted time. An alternative to using point estimates that allows conveyance of uncertainty is to use *prediction intervals*.

The method for obtaining estimates from the CDF in equation (6.3) uses quantiles, such that the q-quantile, $q \in (0, 1)$, is obtained by solving

$$\hat{A}_q = \max\left\{a \in \{0, 1, \ldots\} : \mathbb{P}\left(A < a\right) \leq q\right\}. \tag{6.4}$$

Figure 6.2 shows a single CDF with a horizontal line at $q = 0.5$, where the solution to equation (6.4) is the maximum of the values below this line (the red points) which is,

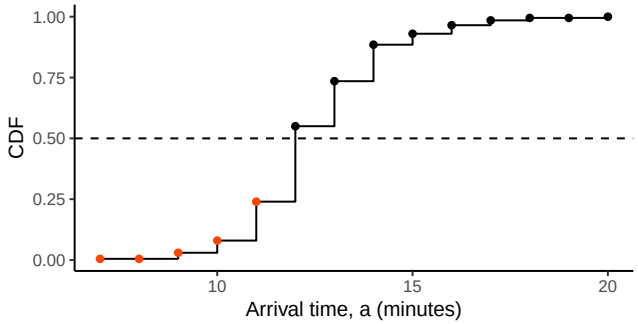


FIGURE 6.2: Quantile estimation from a CDF. ETA values with quantiles below the desired threshold (0.5) are coloured red; the maximum of these is the desired quantile value.

in this case, $\hat{A}_{0.5} = 11$.

6.1.1 Point estimate

In a perfect world, it would be possible to predict precisely when a bus will arrive at a stop and display this single number to passengers. Alas, as we have seen, arrival time prediction is prone to significant levels of uncertainty. However, since much of the infrastructure currently only allows for point estimates, we present some here. We now examine if it is possible to find a single statistic that performs well—on average—and, more importantly, is more reliable than the currently deployed schedule-delay method.

To compare several choices of point estimate, I computed—for every stop i along every trip j at each time k—a selection of quantiles

$$q \in \{0.05, 0.25, 0.4, 0.5, 0.6, 0.75, 0.9\}.$$

For each, I calculated MAE and MAPE to assess accuracy, along with the observed proportion of estimates $\hat{A}_{qijk}$ that were *earlier* than the bus's true arrival A_{ijk} and a passenger arriving at the predicted time will have caught the bus:

$$P_{\text{caught}} = \frac{N_{\text{caught}}}{N_{\text{eta}}} = \frac{\sum_{i,j,k} I_{A_{ijk} < \hat{A}_{qijk}}}{\sum_{i,j,k} 1}, \tag{6.5}$$

where N_{caught} is the number of predictions earlier than actual arrival and N_{eta} is the total number of predictions. For those estimates that were earlier than the actual

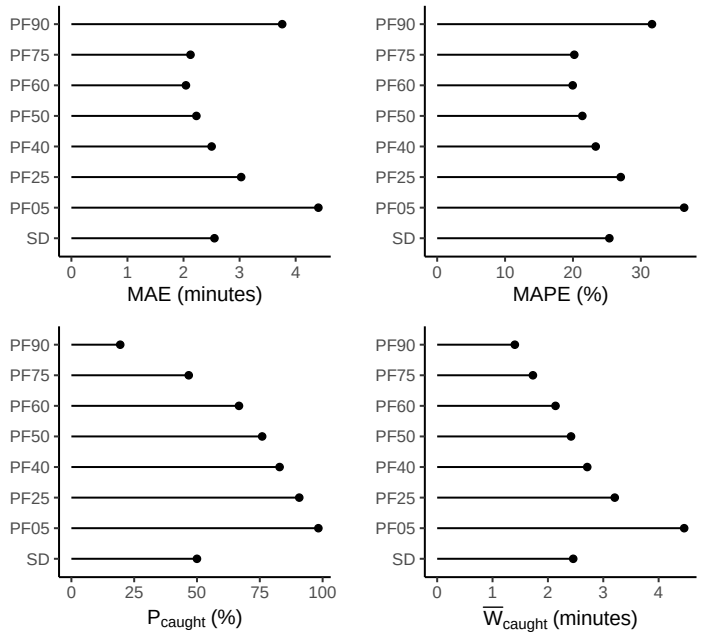


FIGURE 6.3: Comparison of the summary statistics for various quantiles of the predictive distribution (PFX uses the X% quantile) and the currently deployed schedule-delay (SD) method. Results are obtained using off-peak data (between 9h30 and 14h30).

arrival, the average waiting time is

$$\overline{W}_{\text{caught}} = \frac{1}{N_{\text{caught}}} \sum_{i,j,k} \left(A_{ijk} - A_{qijk} \right) . \tag{6.6}$$

The results are displayed in figure 6.3, along with the same values computed using the schedule-delay (SD) arrival time estimates. For the particle filter, the lowest MAE value is achieved using PF60 ($q = 0.6$), for which 67% of predictions were earlier than the true arrival and had an average waiting time of 2.7 minutes. Looking at P_{caught}, we see the effect of rounding estimates down: arriving by the 50% quantile should, in theory, give a passenger a 50% chance of catching the bus. However, referring back to figure 6.2, the estimated 50% quantile is 11 minutes, which is, in fact, the 25% quantile, providing a 75% chance of catching the bus, which is similar to the value of P_{caught} in figure 6.3 for PF50.

An important consideration is the *cost* of missing the bus, which occurs when the predicted arrival time is later than the bus's actual arrival. For each trip, the scheduled time between the current trip and the subsequent one—referred to as *headway*—is used to compute the expected waiting time if a passenger misses the bus. If a bus is

predicted to arrive in 5 minutes but arrives in 3, and the time between buses servicing the same route is $H = 10$ minutes, then the expected waiting time for a passenger arriving at the predicted arrival time in 5 minutes is $10 + 3 - 5 = 8$ minutes.

The total expected wait time can be conditioned on whether the bus was caught,

$$\begin{aligned}\mathbb{E}\left[\text{wait}\right] &= \mathbb{P}\left(\text{catch}\right)\mathbb{E}\left[\text{wait}\,\middle|\,\text{catch}\right] + \mathbb{P}\left(\text{miss}\right)\mathbb{E}\left[\text{wait}\,\middle|\,\text{miss}\right] \\ &= \mathbb{P}\left(A \geq \hat{A}\right)\mathbb{E}\left[A - \hat{A}\,\middle|\,A \geq \hat{A}\right] + \mathbb{P}\left(A < \hat{A}\right)\mathbb{E}\left[H + A - \hat{A}\,\middle|\,A < \hat{A}\right],\end{aligned} \tag{6.7}$$

where $\hat{A}$ and A are the estimated and observed arrival times, respectively. For simplicity, we assume headway H is maintained—that is, if a route has 20-minute headway and a passenger misses a bus by 5 minutes, the next bus is expected to arrive in 15 minutes. This is not true in most situations, however, but predicting headway is a difficult problem (Chen et al., 2012; Hans, Chiabaut, and Leclercq, 2014; Hans et al., 2015).

To estimate $\mathbb{P}\left(\text{catch}\right)$ and $\mathbb{E}\left[\text{wait}\,\middle|\,\text{catch}\right]$ we use the values estimated by equations (6.5) and (6.6), respectively, allowing estimation of the total expected wait times. These wait times are displayed in figure 6.4 for a range of trip headways. Capture success rates are mostly unaffected by headway, while average wait times are, unexpectedly, shorter for less frequent trips. This could be attributed to there being more trips with short headway at peak times when there is more uncertainty in arrival times, and buses tend to be later than predicted. The relationship between headway and wait time if the bus is missed is as expected, with little noticeable difference between predictors.

The overall wait time shows little difference between predictors for trips with a headway less than 10 minutes, but the methods quickly disperse as headway increases beyond this. However, we see that it is possible to maintain an expected wait time below 10 minutes by choosing appropriate quantiles. For low-frequency routes, the most reliable estimate would be the 5% or 25% quantile to minimise the probability of missing the bus. As a comparison, the same values were calculated based on the schedule-delay predictions, and are shown in figure 6.4 as dashed black lines. We see that, in most cases, it is the equivalent to using $q = 0.75$, and has no better than a 50%

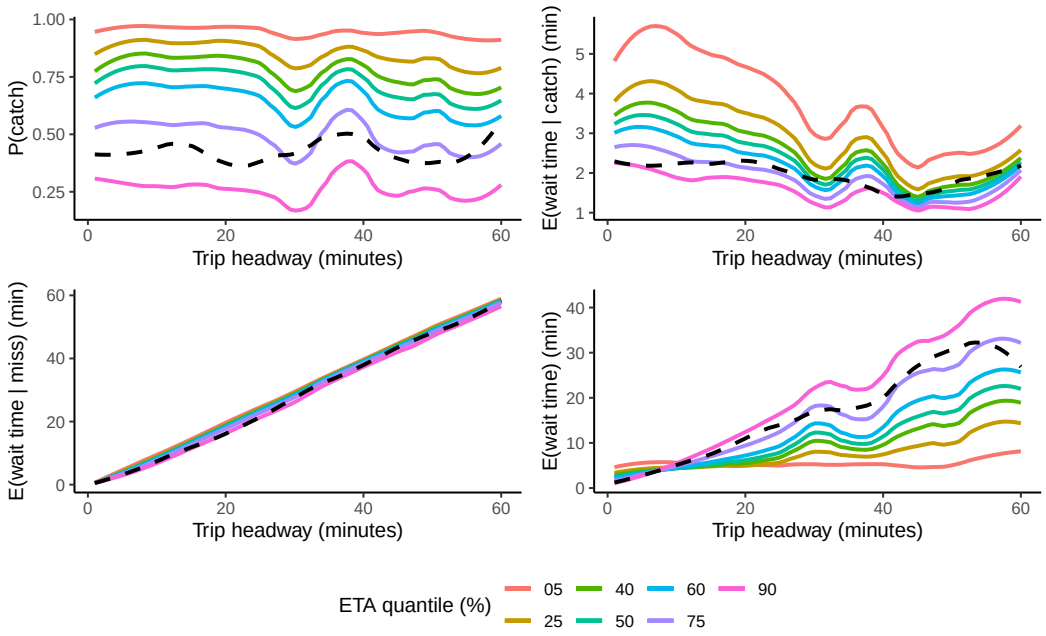


FIGURE 6.4: Capture probabilities and expected wait times by trip headway (time until the next trip) using varying quantiles as point estimates of arrival time. Wait times assume the passenger arrives at the specified time. The dashed black line represents the same values using the schedule-delay prediction.

chance of catching the bus. Consequently, the total expected waiting time is about 50% of the headway.

6.1.2 Interval estimate

Deciding on a "best" single-value estimate of arrival time is exceedingly difficult given the amount of uncertainty involved. The best approach above is to use a small quantile, but this increases expected wait time given catching the bus (top-right of figure 6.4). An alternative approach is to provide *two* estimates—a lower and upper bound—that is to say, a *prediction interval*. Such an interval should be both *reliable* and *useful* for commuters.

Reliability means that, if one arrives by the *lower estimate*, there is only a small probability of missing the bus. The bus should also have a low chance of arriving after the *upper estimate*, particularly when using the prediction to decide which bus to catch to get to a destination on time (more of this in section 6.2).

Usefulness corresponds to the interval's width and expected wait time. These should both be minimised where possible, so for example, if a bus is 5 minutes away,

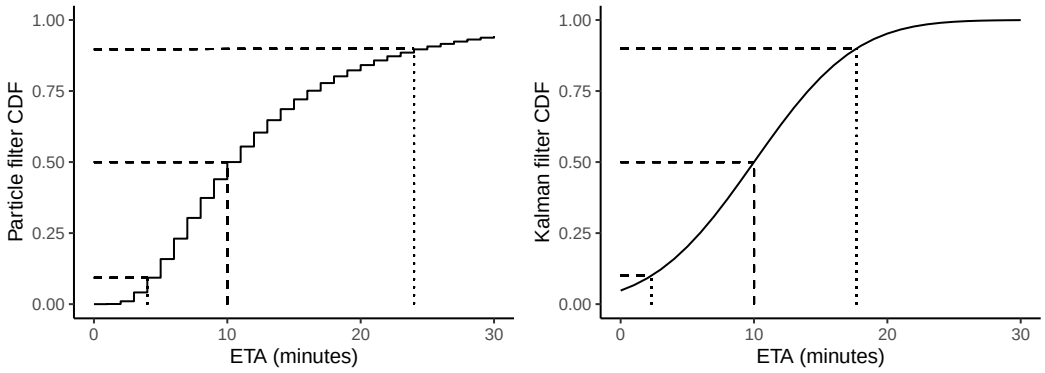


FIGURE 6.5: Symmetry of arrival time prediction intervals. Symmetric intervals on the probabilty scale map to asymmetric intervals on the arrival time scale under the particle filter (left), and symmetric intervals on the arrival time scale under the Kalman filter (right).

providing a 30-minute interval should only be done if there is a valid reason to do so. If there is a layover between the bus and the passenger's stop, for example, the time range could very well be that large since the particles implement the layover behaviour discussed in section 5.2.

Here we consider symmetric $100(1-\alpha)\%$ prediction intervals, $\alpha \in (0,1)$, of the form

$$\left(\hat{A}_{\alpha/2},\ \hat{A}_{1-\alpha/2}+1\right), \tag{6.8}$$

where $\hat{A}_q$ is as defined in equation (6.4). The "plus one" is used to ensure the probability of the bus arriving before the upper bound is *at least* $1-\frac{\alpha}{2}$. Note that although these intervals are symmetric in probability, they are most often asymmetric on the arrival time scale, particularly when the bus is near and the distribution is right-skewed, as shown in figure 6.5. In contrast, a Kalman filter implementation assumes Gaussian errors, so a symmetric interval (y-axis) is also symmetric around the mean (x-axis), as shown, often leading to incorrect intervals (potentially even an estimated time of arrival below zero).

I computed intervals for $\alpha \in \{0.01, 0.05, 0.1, 0.2\}$, and for each evaluated the observed coverage, the proportion of times that the bus arrived before the lower bound, and the average interval width (in minutes). Figure 6.6 shows that the observed coverage drops slightly as the interval width increases (smaller α) and that the probability of the bus arriving before the lower bound is higher than expected. This may indicate that not enough uncertainty is being incorporated; referring back

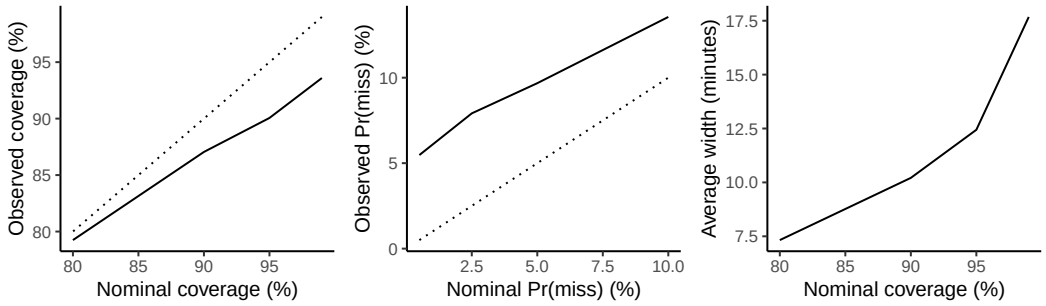


FIGURE 6.6: Evaluation of the observed coverage (left), lower bound (middle), and width (right) of various prediction intervals using the particle filter arrival time CDF. The dotted line indicates the nominal value.

to chapter 5, much of this occurs during peak times. Interval width increases with coverage probability, demonstrating the trade-off between reliability and usefulness.

Other variables—time-until-arrival, time of day, or stop sequence—are not considered here as in the previous chapter since the results are much the same. However, if desired, such relationships could be explored to choose the best point or interval estimate under any given situation. The goal of this section was to demonstrate that reliability and usefulness can be improved upon by using prediction intervals.

6.2 Journey planning

So far we have been concerned with the arrival time of a bus at a single stop with no consideration of the passenger's commute as a whole. The most straightforward journey consists of a single route choice, so the only decision is which trip to catch; a slightly more complex journey may offer two alternative routes (section 6.2.1) between which the passenger may choose. Finally, there may be no single route which goes from the passenger's start location to their destination, in which case a transfer between two (or more) different routes, or *legs*, is necessary (section 6.2.2).

Dynamic routing—the selection of candidate trips and real-time assessment of which is *optimal*—is in itself a difficult problem to solve (Häme and Hakula, 2013a,b; Zheng, Zhang, and Li, 2016). There also exist implementations which can take a probabilistic model of arrival time (such as obtained from our CDFs) as input to determine the optimal route (including the selection of candidate routes), such as proposed by Bérczi et al. (2017). Therefore, for this section, we only consider choosing

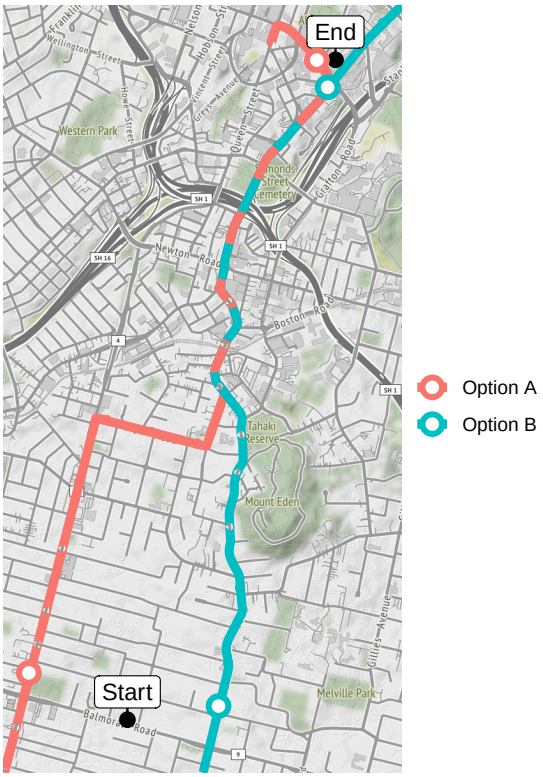


FIGURE 6.7: Route options for a passenger travelling from 'Start' to 'End'. Both options involve a short walk at either end of the journey.

between pre-selected candidate journey options using the arrival time CDFs obtained using our particle filter model. These are compared to the results one might obtain using the currently deployed schedule-delay predictions, which only provide a binary ("Yes" or "No") prediction.

6.2.1 Choosing between two alternative routes

In this scenario, a passenger lives within walking distance of two major roads, along which various routes travel into the central city, as displayed in figure 6.7. The passenger must decide which route option to take before leaving home for an appointment in town[1] at 2:00 pm. At 1:20 pm, they consult the real-time app on their phone before deciding whether to walk to route option A or B. Factors that could influence their decision include:

- how long they will have to wait for the next bus;
- the probability that they will arrive in time for the next bus, and how long until the bus after it;

[1] "In town" means the Auckland CBD, located at the top of figure 6.7.

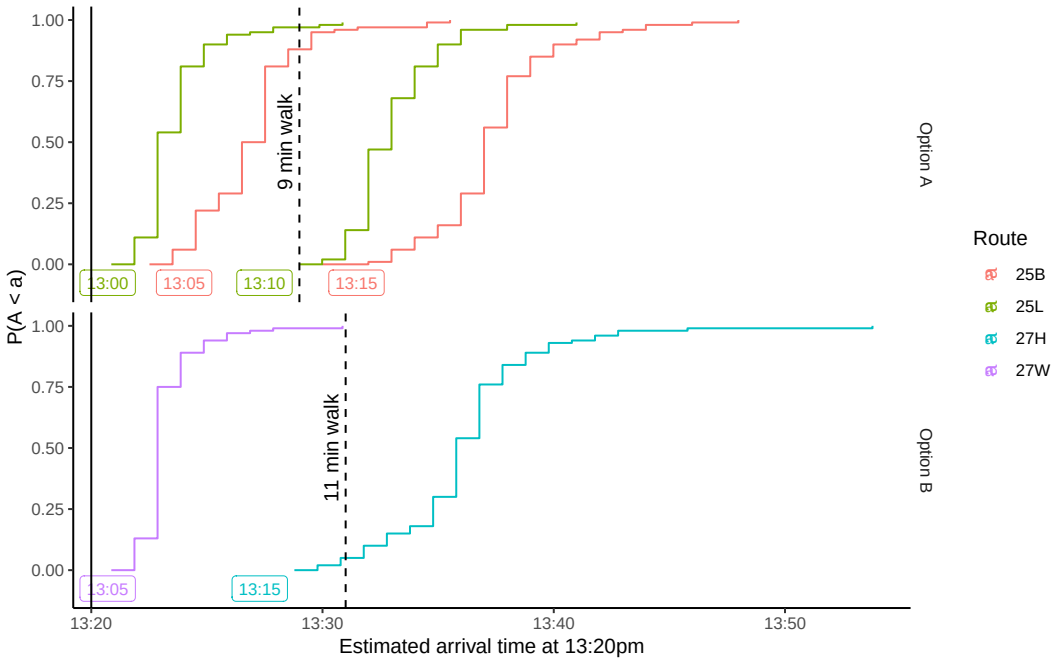


FIGURE 6.8: ETA predictions of bus arrivals at origin stops for two route options. The coloured curves represent the CDFs of arrival time for individual trips made at 1:20 pm. The vertical black lines indicate the estimated walking time (according to Google Maps) from the Start location to each stop.

- the probability of arriving at their destination on-time and how early they will be; and
- the overall length of the journey.

We can provide information relating to these questions using the CDFs of arrival time at the start and end stops. Walking times at either end of the journey can also be accounted for.

Figure 6.8 displays the CDFs of all active trips[2] along the two route options at 1:20 pm when the passenger is ready to leave home (solid line), with dashed lines representing the passenger's arrival time at the stops (accounting for walking time). Below each CDF, which are coloured by route number, is a label indicating the trip's start time (the simplest method of distinguishing trips). For option A, there appears to be a minimal chance of catching the 1:05 pm 25B, but a good chance of catching the 1:10 pm 25L, with the additional safety of another 25B not long after. As for option B, there is a slightly smaller chance of catching the 1:15 pm 27H, but if missed, it is unclear how long the wait for the next bus will be.

[2]Our application currently only estimates arrival times for active trips.

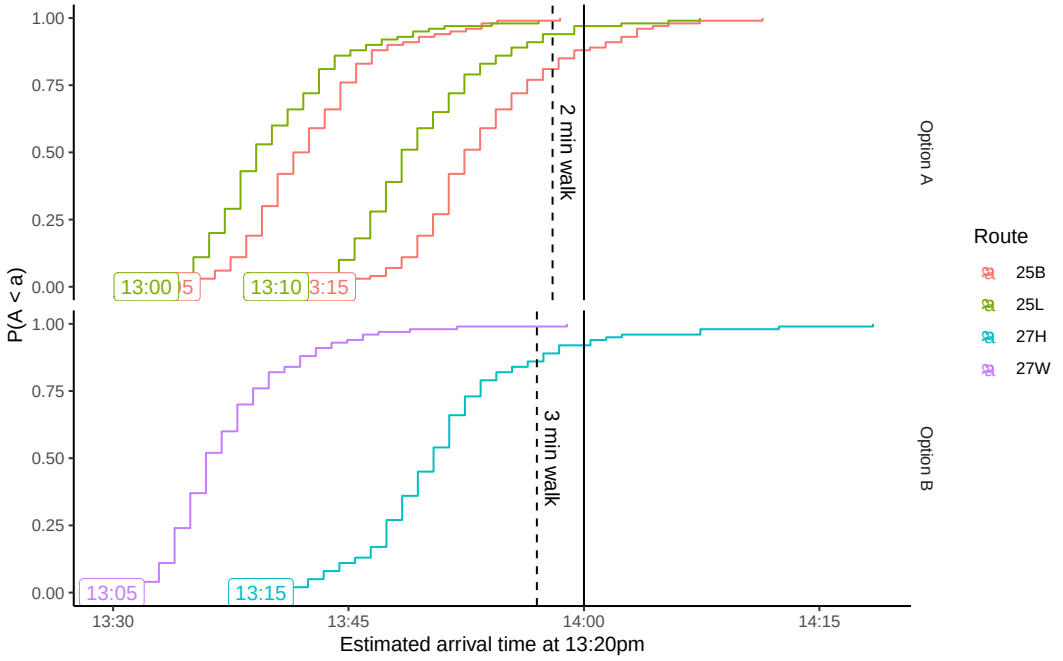


FIGURE 6.9: ETA predictions of bus arrivals at the desination stops for two route options. The coloured curves represent the CDF of arrival time for individual trips made at 1.20 pm. The vertical black lines indicate the estimated walking time (according to Google Maps) from the stops to the End location.

Based on the arrival results alone, I would choose to walk to option A as it has a good chance of a short wait; however, the passenger also needs to consider their appointment at 2:00 pm. Figure 6.9 provides CDFs of each trip's arrival time at the final stop, as well as vertical lines representing the appointment time (solid) and necessary arrival time at the stop to allow for walking time (dashed). None of the trips show definitive signs of being late, although the passenger would likely hope to catch the 1:10 pm 25L to maximise their chances of arriving on-time.

The CDFs allow easy calculation of the predicted probability of success in each

TABLE 6.1: Predicted outcomes for the route selection journey planning problem. The particle filter provides probabilities of success, while the schedule-delay provides only binary ("Yes" or "No") predictions. The observed outcome is displayed on the right.

			Particle filter		Schedule-delay		Outcome	
Option	Route	Trip	P_{catch}	P_{arrive}	Catch	Arrive	Caught	Arrived
A	25L	13:00	0.02	0.99	N	Y	N	Y
	25B	13:05	0.05	0.99	N	Y	N	Y
	25L	13:10	0.98	0.94	Y	Y	Y	Y
	25B	13:15	1.00	0.81	Y	N	Y	Y
B	27W	13:05	0.00	0.99	N	Y	N	Y
	27H	13:15	0.90	0.86	Y	Y	Y	Y

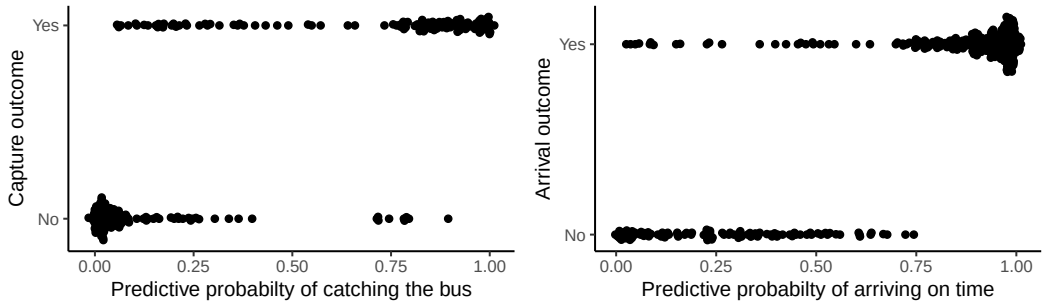


FIGURE 6.10: Results of performing the journey planning prediction with different starting times (from 9:00 am to 3:00 pm), using the same start and end locations. Observations are whether or not the passenger would have arrived at the stop before the bus. Predictive probabilities of 0 and 1 that were correctly estimated have been removed.

scenario, displayed in table 6.1. Additionally, the binary ("Yes" or "No") predictions made using the schedule-delay method are displayed for comparison, as well as the eventual outcome. That is, would the passenger have caught the bus and did it arrive on time? All of the predictions based on the particle filter arrival time CDFs are valid: small probabilities correspond to an outcome of "No" and large probabilities (> 0.8) to "Yes". There was a discrepancy for the 1:15 pm 25B, which arrived at the destination on time: the schedule-delay incorrectly predicted "No", while our method predicted an 81% chance of success.

The results in figures 6.8 and 6.9, and table 6.1, are based on *one single forecast* made at 1:20 pm. To evaluate the overall performance of our prediction method, I repeated the process described above in 5 minute intervals over the off-peak period from 9:00 am to 3:00 pm. For the appointment time, I used 30-minute intervals allowing for 15–45 minutes for the journey: leaving between 9:15 am and 9:45 am had a targetted arrival time of 10:00 am; 9:45 am–10:15 am targetted 10:30 am; and so on. The probabilities of catching the bus and arriving on time were calculated for each case. Figure 6.10 graphs the distribution of predicted probabilities for each of the two outcomes, for both catching the bus (left) and arriving on time (right). For most of the buses that arrived before the passenger, our method predicted capture probability below 50%, though more buses which arrived after the passenger also had capture probabilities below 50%. As for the arrival outcome, the maximum probability of arriving on time for buses which did not was 75%, whereas a few buses which did arrive on time had small ($< 50\%$) probabilities of doing so.

TABLE 6.2: Schedule-delay prediction results for all journeys, with the displayed values representing the proportion of outcomes in each cell (rows are conditioned by the schedule-delay prediction).

	Observed outcome			
	Catch bus		Arrive on time	
Schedule-delay Prediction	No	Yes	No	Yes
No	0.98	0.02	0.78	0.22
Yes	0.09	0.91	0.00	1.00

For comparison, table 6.2 presents a two-way contingency table for the binary schedule-delay predictions, with the predicted outcomes in rows. When the schedule-delay predicted that the bus would arrive after the passenger, it was correct 91% of the time. When it predicted the bus would arrive first, it had a 98% success rate. As for the arrival, in all cases that the schedule-delay predicted the bus would arrive on time, it was correct, versus only 78% when it predicted the bus would be late. Overall, the schedule-delay method correctly predicted the bus capture outcome in 94% of cases and successfully predicted the on-time arrival outcome in 89% of cases.

In both methods, predicting bus capture is more accurate than predicting on-time arrival, which is acceptable considering the difference in forecast length: about 10 minutes for bus capture versus about 30 minutes for arrival. The schedule-delay predictions have a 1-in-10 chance of missing the bus, while our particle filter provides a less definite prediction (such as 75%), opening up the opportunity for passengers to make more informed decisions based on these probabilities and the importance of their constraints. In contrast, the schedule-delay method provides, at best, the predicted arrival time as a singular point estimate with no associated uncertainty, allowing only a "Yes" or "No" prediction to be made.

6.2.2 Planning a multi-stage journey

A more complex scenario is one in which the passenger must transfer from one route to another. Transfer journeys are common among travellers commuting from further afield, and occur when there is no single route between their origin and destination, as is demonstrated in figure 6.7. In this scenario, the passenger must first catch a bus along leg 1[3] to the stop marked "Transfer", at which point they disembark and wait

[3]Each leg of the journey is serviced by one or more unique routes which make the same journey, as can be seen with leg 2 branching in the south-west of figure 6.7.

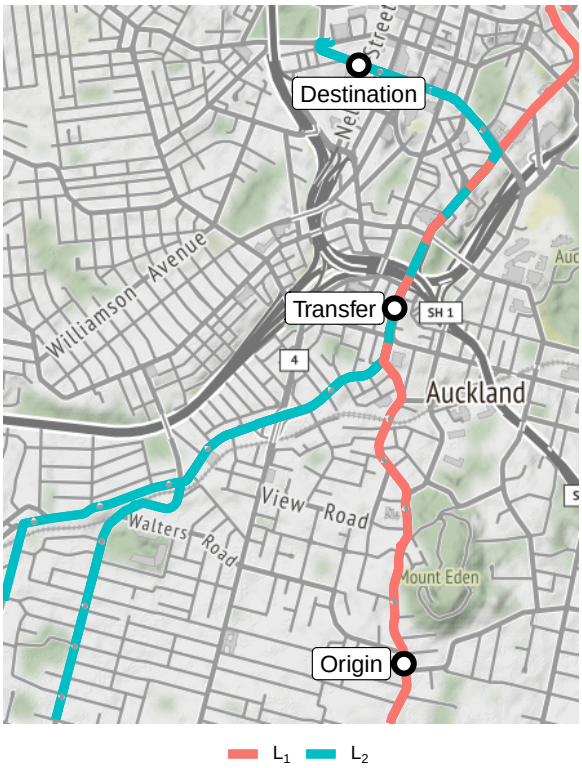


FIGURE 6.11: A passenger travelling from Origin to Destination must change buses at the Transfer stop, resulting in a journey with two *legs*.

for the next bus along leg 2 to get to their final destination.

As with section 6.2.1, we take one single forecast of all trip arrivals at 1 pm. Walking time has been removed to simplify the example, but could easily be included if desired. Figure 6.12 shows, for all active trips at 1:00 pm, their arrival time CDFs at the start and transfer stops (for the first leg), and the transfer and end stops (for the second). Additional constraints, such as arrival by a specific time, could, of course, be added; in this instance, however, we are solely interested in how well the particle filter CDF predicts a successful transfer from leg 1 (L_1) to leg 2 (L_2).

Given CDFs for two trips arriving at a stop, the probability that a transfer can successfully be made from L_1 to L_2 is calculated by

$$\begin{aligned}
\mathbb{P}_{\text{catch}} = \mathbb{P}\left(L_1 < L_2\right) &= \sum_{x=1}^{\infty} \mathbb{P}\left(L_1 < L_2 \,|\, L_2 = x\right) \mathbb{P}\left(L_2 = x\right) \\
&= \sum_{x=1}^{\infty} \mathbb{P}\left(L_1 < x\right) \mathbb{P}\left(L_2 = x\right) \\
&= \sum_{x=1}^{\infty} \mathbb{P}\left(L_1 < x\right) \left[\mathbb{P}\left(L_2 < x+1\right) - \mathbb{P}\left(L_2 < x\right)\right],
\end{aligned} \tag{6.9}$$

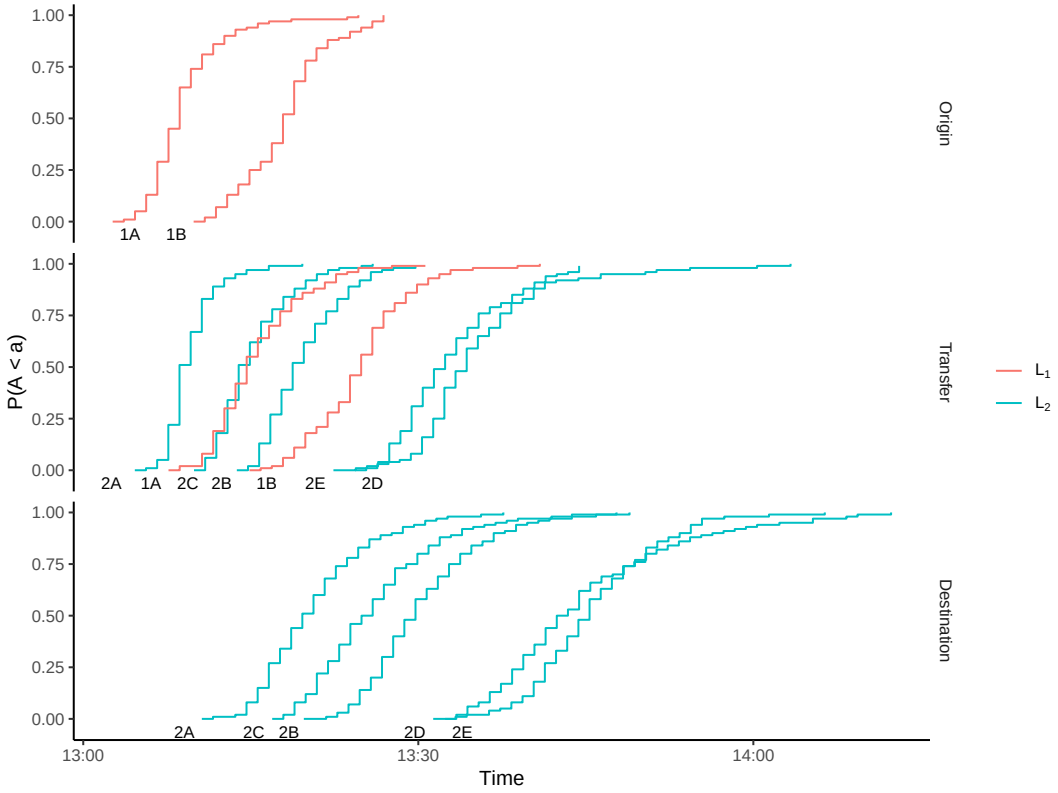


FIGURE 6.12: Arrival times CDFs for buses at the Origin, Transfer, and Destination stops (top to bottom) as at 1:00 pm. The CDFs are coloured by leg (origin to transfer, or transfer to destination).

TABLE 6.3: Predicted outcomes for the transfer journey planning problem. The particle filter provides probabilities of a successful transfer from a trip on leg 1 to a trip on leg 2. The schedule-delay method predicts only a binary ("Yes" or "No") outcome. The observed outcome is displayed on the right.

Leg 1	Leg 2	$\mathbb{P}_{\text{transfer}}$	Schedule-delay	Outcome
1A	2A	0.04	N	N
	2B	0.71	Y	Y
	2C	0.32	Y	N
	2D	1.00	Y	Y
	2E	0.99	Y	Y
1B	2A	0.00	N	N
	2B	0.20	N	N
	2C	0.05	N	N
	2D	0.96	Y	Y
	2E	0.93	Y	Y

which is easily computed from the arrival time CDFs obtained from equation (6.2). Table 6.3 displays the predicted probabilities, along with the binary predictions using the schedule-delay method and the observed outcomes.

The particle filter predicts transfer probabilities with reasonable accuracy: low probabilities result in negative ("No") outcomes. In contrast, the schedule-delay predictions fail to correctly predict the outcome of a transfer between 1A and 2C: it incorrectly predicted a successful transfer while the particle filter predicted only a 32% chance of success. Here a passenger could, where applicable, base their decision on how important it is they make the transfer which might depend on frequency or other constraints. While this example is somewhat trivial—it's a short wait for the next bus—the headway between trips can sometimes be 30–60 minutes. In that case—from personal experience—watching your second bus depart as the one you are on arrives at the station is very frustrating. However, until our application has been updated to make predictions for upcoming (and not just active) trips, we are not yet able to make predictions for such scenarios.

As before, the procedure was repeated in 5-minute intervals between 9:00 am and 3:00 pm, and the probabilities of transfers computed for each. Figure 6.13 shows the predicted transfer probabilities grouped by the outcome. To complement these results, table 6.4 shows the outcomes for the schedule-delay method along with particle filter predictions using three decision rules. For the schedule-delay method, in 10% of cases a predicted successful connection failed versus only 7% using the particle filter with a decision rule of $\mathbb{P}_{\text{catch}} > 0.5$. The false-positive rate was similar for both methods,

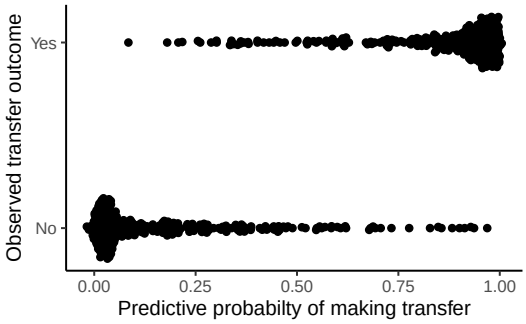


FIGURE 6.13: Results of performing the transfer journey planning prediction with different starting times (from 9h00 to 15h00). Observations are whether or not the first bus would arrive at the transfer stop before the second bus. Predictive probabilities of 0 and 1 that were correct have been removed.

TABLE 6.4: Summary of the results of all predictions between 9:00 am and 3:00 pm. Rows represent the predicted outcome (No, the transfer will be made, or Yes, it will) and the numbers represent the proportion of predictions with each observed outcome. The schedule-delay rule is binary, while the particle filter results are based on transfer probability of at least 0.5, 0.8, or 0.95 (as indicated).

	Outcome							
	Schedule-delay		PF (P > 0.5)		PF (P > 0.8)		PF (P > 0.95)	
Prediction	No	Yes	No	Yes	No	Yes	No	Yes
No	0.97	0.03	0.94	0.06	0.85	0.15	0.7	0.3
Yes	0.1	0.9	0.07	0.93	0.03	0.97	0	1

although slightly higher for the particle filter (6% versus 3%). However, depending on the passenger's requirements, it is possible to reduce the false-positive rate under the particle filter by increasing the threshold. Table 6.4 also shows the results using 0.8 and 0.95 as the decision rule, which result in 3% and 0% false-positive rates, respectively. These do, however, have a significantly higher false-negative rate. By providing probabilities, the passenger has the option to decide what level of risk they are willing to take depending on personal constraints.

In this section, we have seen that having access to the full distribution of arrival times enables calculation of event probabilities as opposed to binary outcome predictions. Probabilities allow for more sophisticated decision making by travellers depending on their own needs, or by journey planning software and dynamic routing methods (Bérczi et al., 2017). A further advantage of using the particle filter to estimate the CDFs is that any improvements to the vehicle or network models will be integrated into the arrival prediction component, making future improvements simple to implement.

6.3 Chapter contributions

- We present a simple, discrete approximation to the arrival time CDF, achieved by "rounding down" from seconds to minutes, which can be sent to users' phones, for example.
- We can use our particle filter CDFs to answer some common journey planning questions and calculate the probabilities of events. This could then be used by dedicated journey planning software to determine the optimal route.

Chapter 7

Discussion

Accurately predicting the arrival time of transit vehicles involves estimating many unknowable factors, and is inevitably prone to high levels of uncertainty. We set out to develop a real-time application framework that sits on top of GTFS—a globally used transit data format—allowing estimation of *network traffic state* to make arrival time predictions more useful for travellers. Our results demonstrate that the proposed *particle filter* method provides the ability to both estimate network state and predict arrival time distributions better, on average, than the currently deployed method. By estimating the full CDF, we can provide quantile-based prediction intervals, but also compute probabilities for journey planning problems, such as the chance of arriving at the destination on time.

It is clear from the literature that successful arrival time prediction requires some form of *traffic state estimation*, be it based on historical trends (Cathey and Dailey, 2003; Čelan and Lep, 2017; Julio, Giesen, and Lizana, 2016; Mazloumi et al., 2012) or real-time information (Ma et al., 2019; Shalaby and Farhan, 2004; Xinghao et al., 2013). However, one of our aims was to build on top of GTFS and require no additional information (such as taxi data) which led us to construct a *transit network*, allowing us to model real-time traffic conditions using the transit vehicles themselves.

Our application uses stops as the nodes, and road segments are identified for each route by their unique sequence of stops. This is similar to the network described by Čelan and Lep (2017, 2018); however, while they showed that including stops is

necessary for accurate arrival prediction, they also demonstrated the need for other potential time barriers—for example, roundabouts or pedestrian crossings—which we do not yet have implemented. Nevertheless, other methods have also used links between stops, often incorporating multiple stops in each link (Shalaby and Farhan, 2004).

Having expressed each route as a sequence of segments which each correspond to a unique one-directional road between stops, it is trivial to assign a vehicle's current or previous location to roads throughout the network. In this way, our framework allows the estimation of not only vehicle speeds, but also of the average speed of vehicles along individual roads. We chose *average vehicle speed* similarly to Čelan and Lep (2017, 2018) instead of travel time (Dai, Ma, and Chen, 2019; Shalaby and Farhan, 2004) as the latter tends to have long tails (the Kalman filter assumes the state is Gaussian, and from our observations found that speeds are more symmetric than travel time). The process of predicting arrival times that incorporate real-time traffic conditions involves three main phases: vehicle modelling to estimate vehicle state and road speeds; network modelling to update network state; and arrival time prediction.

Modelling transit vehicles involves handling vehicle behaviour and other unique features of transit data along with measurement uncertainties and irregularities. We use a particle filter to estimate vehicle state due to its superior handling of *multimodality*, which is common in transit data. It also allows for many trajectories to be explored (Hans et al., 2015), which is particularly an issue under low observation frequency. Not only this, however, but the particle filter allows for a more generalised transition function which can implement the complex behaviour we describe, which includes stopping behaviours at stops and intersections. Using a particle filter to estimate vehicle state provides a straightforward way of estimating vehicle speeds along roads. By expressing each route as a sequence of *road segments*, each particle's travel time along each segment can be calculated. Then, using the Dirac delta measure, the vehicle's estimated average speed along individual segments is obtained with associated uncertainty.

The primary advantage of developing the transit network is that the vehicle speed estimates are assigned to *roads*, rather than links along a given route. In contrast,

other methods either only apply to a given route (Čelan and Lep, 2017; Chang et al., 2010; Yu et al., 2010), or require manual identification of common road segments (Yin et al., 2017; Yu, Lam, and Tam, 2011).

Our particle filter implementation also uses a unique *likelihood function* which, unlike other methods we have seen, allows the particle observations to be compared directly to the observation. Historically this step required the "snapping" of GPS observations to the nearest point along a route to estimate the vehicle's *trip-distance-travelled*, as Cathey and Dailey (2003) describe in detail. Our approach removes this unnecessary source of uncertainty, which is of particular importance where routes may "loop": given a single observation it is impossible to tell whether the bus is entering or exiting the loop (figure 3.5, figure 3.5). We also use *arrival* and *departure times* in the likelihood—where available—to reduce the number of possible trajectories which occur in the vicinity of bus stops (figure 3.9, figure 3.9).

One aspect we have not explored entirely is real-time *dwell-time modelling*. Shalaby and Farhan (2004) demonstrated its significance, as did Hans et al. (2015); however, their applications had access to automatic passenger counter data, which is not available in Auckland. Instead, we chose to use historical estimates of dwell time (computed from arrival and departure data) to obtain a dwell-time distribution for each stop along a route. In doing so, we noted a peculiarity with the data in which dwell times seem to display a 9-second frequency.

Other issues with the Auckland real-time data can be attributed to *way points*, which can result in the bus appearing to go backwards. To minimise the impact of this, we pre-process each observation to detect if it might be erroneous or "pre-emptive", and in those cases ignore it altogether. This issue has not been mentioned in any of the literature, so it might very well only affect Auckland.

Given the ability to *observe* the speeds of vehicles as they travel along roads, it becomes possible to model the transit road network's state in real-time. We use a Kalman filter to model the state (average vehicle speed) of roads throughout Auckland's public transport network. Previous research by Shalaby and Farhan (2004) demonstrated the ability of the Kalman filter to react to real-time traffic events, a result which we were also able to demonstrate. This approach can be contrasted with alternatives which use "real-time" information to choose from historical trends, as

demonstrated by the likes of Čelan and Lep (2017), Chen and Rakha (2014), Julio, Giesen, and Lizana (2016), Yu, Lam, and Tam (2011), and Yu et al. (2010) who used machine learning methods (artificial neural network or support vector machine) to predict travel times or speeds. Yin et al. (2017) demonstrated the difficulty of these types of methods to respond to sudden, large increases in travel time during peak times.

At the end of chapter 4, I demonstrated the use of a simple forecasting method to predict road speeds based on the current state and historical trends. This forecasting approach predicted road speed better than the naïve "current state" prediction. More research is required in this area, however, particularly on the notion of *relationships* or *correlations* between road segments, particularly at peak times. In such cases, a *traffic shockwave* model similar to that employed by Julio, Giesen, and Lizana (2016) might be appropriate.

Under the current implementation of our particle filter, speeds tend to be over-estimated, mostly due to arrival and departure time uncertainty. Particles tend to *travel faster* and *spend longer* at stops than the bus does, which could be improved by implementing a more restrictive likelihood on the arrival and departure time observations. It may also require some exploration of the "9-second" phenomenon and how it may be affecting the reliability of arrival and departure time observations.

Regardless, the real-time network state plays a critical role in arrival time prediction, as this has time and again been shown to contribute to uncertainty (Julio, Giesen, and Lizana, 2016; Shalaby and Farhan, 2004; Yu, Lam, and Tam, 2011; Yu, Yang, and Yao, 2006; Yu et al., 2010). Not only this, but the network state may also be useful to transit providers as a way of monitoring real-time congestion which may be affecting service punctuality (on-time arrival). The primary use of network state, however, is for real-time arrival time prediction.

The primary goal of this thesis is to demonstrate how using real-time traffic information can be used to improve the reliability of arrival time prediction. Vehicle state (as approximated by a set of particles) is combined with network state and historically-based dwell-time distributions to obtain estimates of arrival time at all upcoming stops. Unlike many previous approaches, we are not limited to predicting only a single stop (Yu, Lam, and Tam, 2011). Since the nature of the application

requires distribution of arrival time predictions to travellers, I demonstrate a simple, fast method for computing the full CDF of arrival time by rounding all particles' estimates to integer-minutes. This significantly reduces the complexity of computing and storing the arrival time CDFs, and allows quantile estimates and event probabilities to be calculated directly on a user's phone.

The advantages of providing a probabilistic distribution of arrival time versus a point estimate is that best- versus worst-case scenarios can be evaluated. If you are meeting a friend in town for a coffee, the cost of missing the bus is not very high—you can text your friend that you are running late (and maybe shout them a muffin). However, if you have a job interview or an apartment viewing you cannot be late for, you want to know with more certainty that the bus will be on time. The arrival time distribution allows travellers to make decisions: if the bus only has a 70% chance of getting you to your job interview on time, you might decide to catch a taxi instead. Previous research by Fernandes et al. (2018) supports the idea that people can make more informed decisions given uncertainty information. Besides, point estimates can still be computed from the CDF for those who prefer them.

Of course, the uncertainty is still problematic, and ideally as much as possible should be reduced. Road speed contributes in part to uncertainty, but there is also a significant amount attributed to dwell times, which we have no formal model for due to a lack of associated data. However, another problematic situation is *layovers*, which, as I demonstrated, are poorly maintained by drivers. This contributes to uncertainty, particularly for a route that is running early: we cannot assume the driver will stop. Thankfully, the particle filter makes this scenario easy to handle by merely allowing each particle an independent chance of waiting (which we obtain from observing historical departure delays, figure 5.1, figure 5.1). Thus, the "best" and "worst" case scenarios are when the driver does and does not stop, respectively. This concurs with discussions by Hans et al. (2015) and Ulmke and Koch (2006) noting the particle filter's ability to handle multimodality in the vehicle's trajectory.

One issue impacting the predictive performance of our method is the *overestimation* of road speeds. This results in arrival time estimates that are earlier than they should be, which we saw in chapter 6 with the 60% quantile having the smallest absolute error, although some of this can be attributed to *rounding down* to obtain the CDF.

These issues were particularly noticeable during peak times when traffic conditions are more volatile. We also noticed that often not enough uncertainty was included during peak time: the upper quantile of arrival time, notably concerning journey planning, was frequently too low, resulting in underestimates of the probability that the bus would arrive on time. While not ideal, I believe that this is a better predicament to be in than the other way around, where arrival time predictions are *later* than they should be and could lead a passenger to miss their bus.

As far as we know, no other framework provides the full CDF of arrival times. Work such as that by Čelan and Lep (2017, 2018) presents the possibility for more comprehensive journey planning that uses a probabilistic model of arrival times our application provides. In this situation, a database containing the arrival time distribution could be queried by a journey planning application, which finds candidate routes from the network and computes the probabilities of their success. Most importantly, this uses the *real-time* predictions, rather than scheduled travel and arrival times, which are often unreliable.

The combination of the particle filter to model vehicles with the real-time network state and dwell time distributions allows our application to predict arrival times more accurately and more reliably than the method currently used by Auckland Transport and likely many other agencies around the world. Our application provides a foundation for further research, for example, for more advanced dwell time or travel time models. Of course, any such work must also consider the computational constraints of real-time applications.

Besides reliability, our main concern while developing this application was its real-time feasibility: *is it fast enough for real-time use?* Therefore, we made several important decisions early on, namely using C++ through 'Rcpp' (Eddelbuettel and François, 2011), which resulted in the R package 'transitr'. Our initial goal for an iteration time was *30 seconds or faster*, and as I showed in section 5.4, our application comes in well under that with most iterations taking 6–10 seconds using 3 cores on a desktop computer.[1]

[1] We do have access to an 8 core virtual machine, courtesy of the Center for e-Research at the University of Auckland, but were unable to run the simulations on it due to security configurations that I am still working around.

Our application connects to a public GTFS API and fetches the data. 6–10 seconds later, arrival time estimates are available for travellers.[2] To get an idea of the complexity of the task, let us say there are 1000 vehicles at peak time, each servicing routes with up to 40 or more stops. If we assume that the average number of *remaining stops* per vehicle is 10, then we need to predict arrival times for 10,000 stops during each iteration; at peak time, this is about 1000 predictions per second. Chang et al. (2010) developed a prediction method that could predict *multiple stops ahead*, and they were able to make about 4000 predictions *per minute*. Unfortunately, other methods did not report timings of their methods for us to compare ours with.

7.1 Future work

The foundation of our arrival time prediction framework is the *transit network* constructed on top of GTFS data. This could potentially be further generalised from using *stops as nodes* to identifying overlapping shapes and, more importantly, the locations where non-overlapping routes join (that is, find intersections). Algorithms, such as those proposed by Xie et al. (2016) and Zhang et al. (2017), could be used to find intersection locations from the GTFS shape data. This would improve the modelling capability as many of these intersections will affect vehicle speed as Čelan and Lep (2017) describe, but would also completely reduce segment overlap. This would also mean that express routes (which use the same road but bypass some stops) could contribute to the road states and be able to use them for predictions. Other possibilities include using external data sources to identify intersections, such as OpenStreetMap (OpenStreetMap contributors, 2017).

Using a particle filter to model vehicle behaviour enables a vast array of possibilities, some of which I demonstrated in chapter 3. The main feature is the likelihood, which can become a fairly complicated issue. We presented the distribution of vehicle distance from the shape path, and one theory for this is road width. From manual inspection, many of the shape paths run down the centre line of the road, so we would expect there to be a short distance between where the bus is and the shape location. For single-lane roads, this is about 1 meter, and 2.5–3 meters for double-lane

[2] In theory; currently, because of the server's firewall and security settings, we are working on ways of making the data accessible.

roads. These are approximately the peaks we saw in figure 3.21 on figure 3.21. It may be possible to model this behaviour, such that the actual vehicle location is offset slightly to the left of the shape path, potentially permitting the use of a smaller value for GPS error. Similarly, the arrival time and departure time likelihood could be modified such that the 9-second phasing is accounted for to reduce the possible range of trajectories further.

On the other hand, alternative types of data could also be included in the likelihood. Recently,[3] Auckland Transport began including vehicle speed and odometer readings for some vehicles, which could go a long way to improving the estimation of *average* vehicle speed along roads, particularly for the issue of *overestimated* speed (due to particles "rushing" to the stop). Further, such information could assist distinguishing between buses that have and have not stopped at a bus stop. Each trip update (arrival or departure) is associated with a vehicle position, which may contain speed information. If these are combined, we could detect the approximate speed of the vehicle at the time of reporting: if it is zero, this indicates the bus stopped; if it is greater than zero, it means that the bus *possibly* did not stop. However, it is still unlikely to remove all uncertainty.

One component of the model that we pay less attention to is dwell time. In particular, a *real-time* dwell time model could be based on passenger demand and *time since the last service*. If available, automatic passenger counter data could improve dwell time predictions, particularly for arrival time prediction, removing as much uncertainty as possible. At peak times, for example, there is a high chance of buses stopping at all stops, versus during the evening when they usually have only a few passengers.

In chapter 4, I explained our use of modelling road segments independently. However, this is not a realistic assumption, and not only could real-time predictions be improved by modelling them with a dependent structure, but so too could the accuracy of the network model itself. For example, observations of vehicle speed along a segment could have implications for adjacent segments, similar to the work presented by Julio, Giesen, and Lizana (2016). Alternatively, methods using particle filters to propose traffic state trajectories based on historical data, similar to the work

[3] As of December 2019.

of Chen and Rakha (2014) for forecasting car travel times, could account for historical trends. Either way, I would predict any such improvement would have a positive effect on the accuracy of arrival time prediction, but not necessarily on the application timing—the Kalman filter in our application takes less than 1 millisecond. Further, in the arrival time prediction component, forecasting each segment for each particle could become a very costly procedure.

On the topic of computational effort, in chapter 5, we saw that the application runs much faster during off-peak times (particularly overnight) due to far fewer buses in operation. I would expect most deployments to use a single dedicated server with fixed resources, such that during these off-peak periods, much of the computer's resources are idle. There are several ways we might use this "downtime". It would first be worth exploring whether or not increasing the number of particles during off-peak would improve network state estimation, perhaps by reducing the particle filter degeneration rate. However, this is only likely to make a slight improvement—we still want estimated times of arrival available as soon as possible. A second, more exciting possibility would be to update the database, for example, by incorporating the previous day of data into the historical dataset. This could involve running Markov chain Monte Carlo estimation procedures overnight, allowing the most recent trends in network state to be accounted for the next day. This would greatly improve the application by enabling it to automatically adapt to changes in the network on a daily basis.

7.2 Concluding remarks

The prediction of transit vehicle arrival time at upcoming stops in real-time is not quite as easy as it seems and is prone to uncertainty entering at all points throughout the process. Some of these uncertainties can be modelled: for example, travel time can be measured in real-time to better predict how long the bus will take. Some others are impossible (or too difficult) to model and predict accurately, such as traffic accidents. By using a real-time transit road network, our predictions can, however, react to these events quickly.

The more uncertainty that can be modelled, the more reliable our predictions will be, particularly when choosing to convey uncertainty to commuters. In particular, the *lower quantile*, which is effectively an estimate of *how early the bus might be*, can—for passengers who use the real-time information to make decisions—maximise their chances of catching the bus. Further, it allows us to compute the probabilities of events, from "the bus arrives on time" to "the transfer is successful". Passengers who wish to make use of this information can make informed journey planning decisions, and in doing so, we hope that such results will improve their perception of public transport as previous studies have shown, and indeed increase ridership.

Appendix A

GTFS

GTFS (Google Developers, 2006) describes the way in which transit data should be organised. There are many *dataset files* specified on the GTFS website,[1] each of which can be thought of as a table in a relational database. Here, I describe the tables used throughout this thesis, and even within each table I only cover relevant *fields* (or columns in a database).

Each table consists of an ID column, which corresponds to a *primary key* in a relationship database, and must be unique for each entry. Some tables also include *reference IDs* or *foreign keys* which point to entries in another table. For example, each entry in the trips table has a reference to an entry in the routes table. Since there can be one or more trips per route, this is considered a *many-to-one* relationship in database terminology. Figure A.1 demonstrates the relationships between the tables in GTFS, including the essential fields in each, which are described in the following paragraphs. The quoted descriptions are taken from the GTFS website. Note that these are only some of the fields available under GTFS, which is continually expanding to provide more flexibility for transit providers.

Routes

The routes table contains information about individual *transit routes*, "a group of trips that are displayed to riders as a single service.".

route_short_name generally a number ("110"), a short descriptive word ("OUTER"), or an alphanumeric sequence riders can easily identify ("NX1").

[1] https://developers.google.com/transit/gtfs/reference

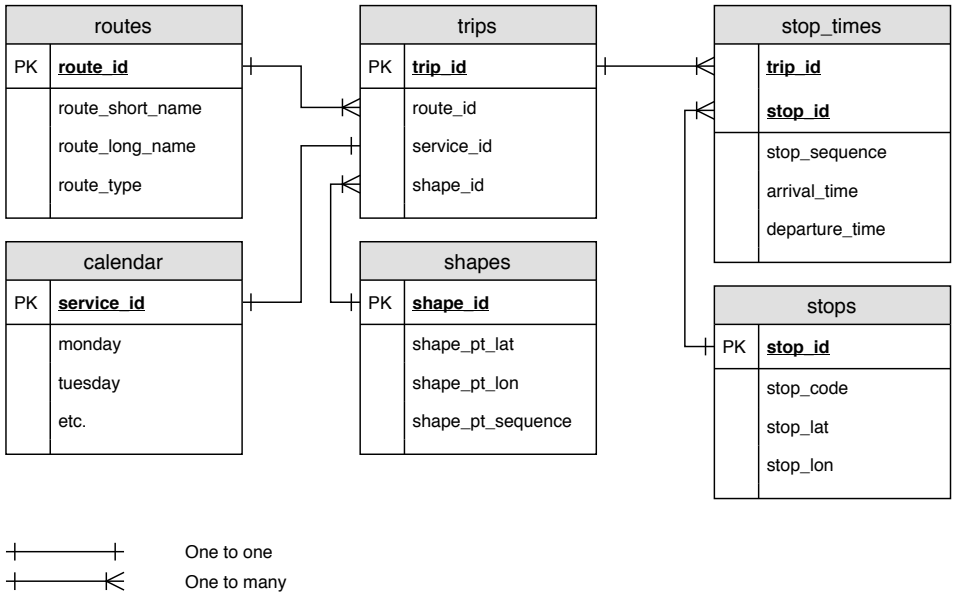


FIGURE A.1: GTFS diagram.

route_long_name describes the route in more detail, usually consisting of the destination and some major points of interest along the way: "Ellerslie To Britomart" or "Three Kings To Britomart Via Mt Eden Rd".

route_type specifies the vehicle type used for the route. For example, 2 is used for trains and 3 is used for buses.

Trips

The trips table describes individual instances of routes: "A trip is a sequence of two or more stops that occur during a specific time period.".

route_id references the route the trip belongs to.

service_id services specify on what days trips run (see Calendar table).

shape_id references the shape that describes the vehicle's path.

Calendar

The calendar table contains "Service dates specified using a weekly schedule with start and end dates.". This means trips can be repeated on multiple days, which is usually divided into *weekday*, *Saturday*, and *Sunday and public holidays* schedules. Each

entry has a column for each day of the week that is either 1 if the trip runs on that day, or 0 otherwise.

Shapes

The shapes table consists of "Rules for mapping vehicle travel paths", which are essentially lines on a map.

shape_pt_lat, shape_pt_lon the latitude and longitude coordinates of each point.

shape_pt_sequence specifies the order of shape points for each shape.

Stops

The stops table describes "Stops where vehicles pick up or drop off riders.", and are an essential component of our transit network used in chapter 3.

stop_code At least in Auckland, all stops have a unique code printed on the sign, which helps riders find them (particularly at stations where there might be multiple stops servicing different trips).

stop_lat, stop_lon The physical latitude and longitude coordinates of the stop.

Stop times

The stop times table consists of *trip-stop* pairs, and is by far the longest table. There is one row for each stop along each trip, providing the "Times that a vehicle arrives at and departs from stops". Most importantly, this is the information used by the "schedule-delay" method currently used by Auckland Transport to predict bus arrival times.

trip_id, stop_id This table does not include its own unique ID, so instead unique rows are identified by unique pairs of trip and stop IDs.

stop_sequence defines the order of stops along a trip.

arrival_time, departure_time specifies the scheduled arrival and departures times at stops. At least in Auckland, these are the same unless the stop is a *layover*

(the bus will, in theory, wait until the scheduled departure time). In this case, the arrival time will be earlier than the departure time.

Appendix B

Computing with particles

The particle filter allows us to approximate any distribution by using a sample of *particles*, $x^{(i)}$, with weights $w^{(i)}$. There are many advantages of the particle filter, but one is that producing summary statistics is very simple. The only real disadvantage is that it is a computationally demanding process. In this appendix, I briefly describe the theory behind using the particle filter to estimate a distribution, as well as give details on its computational demands.

B.1 The Dirac measure

In section 2.4 and chapter 3 I described the particle filter implementation, and noted that we can use the Dirac measure, δ, to approximate the distribution (Benedetto, 1996). We will not get into the theoretical aspects of the function, except to state the following property:

$$\int_X f(x)\delta_y(x)\,\mathrm{d}x = f(y). \tag{B.1}$$

Let x be an unknown quantity for which we have a sample of N particles $\{x^{(i)}\}_{i=1}^N$ with associated weights $\{w^{(i)}\}_{i=1}^N$. The Dirac delta measure allows the approximation of the distribution of x as

$$p(x) \approx \sum_{i=1}^{N} w^{(i)}\delta_{x^{(i)}}(x). \tag{B.2}$$

Using equations (B.1) and (B.2) we can now compute, for example, the mean, using $f(x) = x$,

$$\begin{aligned}\bar{x} &= \int_X xp(x)\,\mathrm{d}x \\ &= \int_X x \sum_{i=1}^{N} w^{(i)} \delta_{x^{(i)}}(x)\,\mathrm{d}x \\ &= \sum_{i=1}^{N} w^{(i)} \int_X x\delta_{x^{(i)}}(x)\,\mathrm{d}x \\ &= \sum_{i=1}^{N} w^{(i)} x^{(i)}.\end{aligned} \tag{B.3}$$

B.2 Calculating particle coordinates

An important aspect of the model used in chapter 3 is the *measurement function*. I mentioned that we are able to obtain GPS coordinates for individual particles, allowing us to compare them directly to the observation. Doing so requires first that we have a *shape path* consisting of a sequence of coordinates (as shown in figure B.1), for which we can compute the distance between each. This, in turn, allows us to compute the *cumulative distance* of each point within the shape, referred to as *shape distance travelled* (in meters).

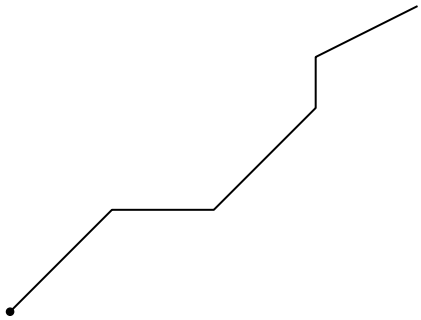


FIGURE B.1: A shape path, starting at the point (lower left).

To find the coordinates of a point 40 meters along this line, we find the maximum point along the route with *shape distance travelled* less than 40, which is the point in red in figure B.2, which is 31 meters along the route. There remains $40 - 31 = 9$ meters to travel to get to 40 meters.

To compute the desired coordinates, we use the *destination point* formula, which takes a start point, bearing, and distance to travel and returns a final point. The

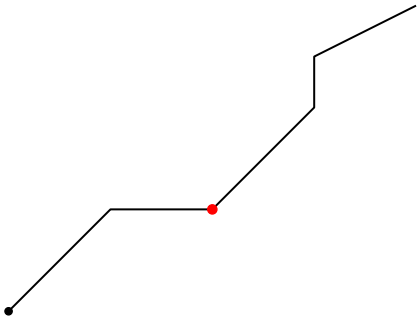


FIGURE B.2: The farthest point along the route less than 40 meters along the route (red).

function `destPoint()` is available in the 'geosphere' package (Hijmans, 2019). The *bearing* required is shown in figure B.3:

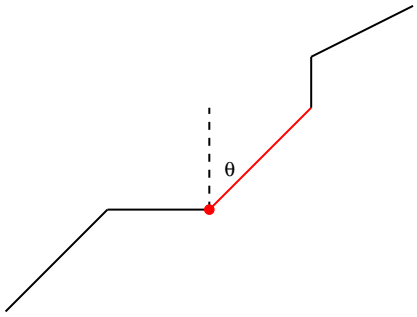


FIGURE B.3: The bearing, θ, is the angle between the dashed line (North) and the red line.

Travelling 9 meters from the red point at a bearing of $\theta°$ (degrees) yields the GPS coordinates of the particle $x = 40$ meters along the route, indicated by the point in figure B.4 below, and is calculated using the 'geosphere' package:

```
particle_coord <- geosphere::distGeo(start, bearing, dist)
```

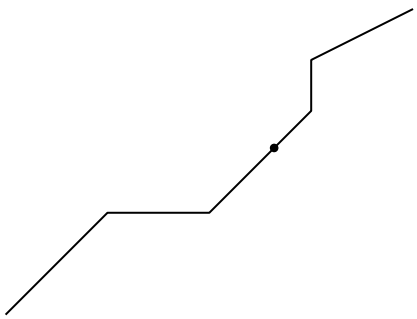


FIGURE B.4: The final position of the particle is obtained by travelling 19 meters along the red line.

B.3 Resampling

Part of the particle filter update step is to, when required, resample the particles according to their weights. This involves generating random numbers to determine which particles to include in the sample, replacing the vector of particle states with the new sample. The latter is straightforward: we simply copy the sampled particles into a new vector and, once finished, replace the original vector with the new one.

We will now describe unweighted resampling first before discussing weighted. Given a sample of size N, we use a random number generator to obtain uniform numbers $u \in (0, 1)$. Since each particle has equal weight, we multiply u by N and round down (using the *floor* function), to get

$$j = \lfloor uN \rfloor. \tag{B.4}$$

Thus, we have sampled particle j.

When the particles are weighted, however, we cannot simply use uN. For example, here we have five particles, where each is represented by a rectangle whose width is equal to the particle's weight (the total weight is unity), as shown in figure B.5.

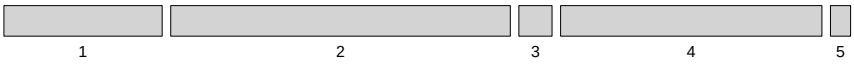


FIGURE B.5: Visualisation of cumulative particle weights as used during resampling.

As before, we use a random number generator to obtain a uniform random number u; to map the number to a particle, we compute *cumulative weights* and find the first particle with cumulative weight greater than u:

$$j = \min \left\{ k = 1, \ldots, N : \sum_{i=1}^{k} w^{(i)} \geq u \right\}. \tag{B.5}$$

B.4 Summary statistics

Another core part of the program is computing of summary statistics, for example the mean and variance. However, these are easily computed using `std::accumulate`, for example, a vehicle's mean speed is obtained by:

```
double mean = std::accumulate (
    x.begin (), x.end (),
    // initial value
    0.,
    // lambda function, where 'a' is the current value
    [](double a, Particle& p) {
        return a + p.get_weight () * p.get_speed ();
    }
);
```

Similarly, the variance can be calculated by passing a reference to the value of `mean` to the lambda function (within `[]`), which becomes

```
[&mean](double a, Particle& p) {
    return a + p.get_weight () * pow (p.get_speed () - mean, 2.);
}
```

More complicated, however, is estimation of *quantiles*, since this requires sorting the vector of particles in order of the desired value. In the above example, to get the median speed, we need to first sort the particles by speed (slowest to fastest) and then sum their weights, in order, until the sum exceeds 0.5: the speed of the particle that does this provides the median speed.

```
std::sort (x.begin (), x.end (),
    (Particle& p1, Particle& p2)
    {
        return p1.get_speed () < p2.get_speed ();
    }
);
double wt = 0.;
int i = 0;
while (wt < 0.5) wt += x.at (i++).get_weight ();
double median = x.at (i).get_speed ();
```

The issue here is the first step, sorting, which has computational complexity of $\mathcal{O}(N \log N)$.[1] If we wish to estimate the median distance and the median speed, we need to sort the particles *twice*. In chapter 6, we used multiple quantiles for each stop: this requires sorting each vehicle's particles once for each upcoming stop. However,

[1] Described on the documentation page for `std::sort`, https://en.cppreference.com/w/cpp/algorithm/sort.

since we use a subsample to compute ETAs ($N^\star$), the complexity is significantly less, $\mathcal{O}(N^\star \log N^\star)$; if we attempted to use all N particles (in the simulation used in chapter 5, we used $N = 10000$) the application would have taken far longer.

B.5 Calculating ETA CDFs

It is also possible to summarize the cumulative density function by using the definition in equations (6.1) to (6.3) (starting on equation (6.1)) and rounding estimates to minutes. This is the equivalent of computing the modules of an ETA in seconds with 60. Then we simply tally the number of particles with each ETA, which is straightforward in C++ by incrementing the `eta`th index of the ETA vector.

```
int Amax, eta;
std::vector<int> cdf;
Amax = std::max (state.begin (); state.end ();
    [](Particle& p) { return p.arrival_times.at (stop_index) % 60; });

// create vector as long as max ETA and set all values to 0:
cdf.resize (Amax, 0);

for (auto p : state)
{
    eta = p.arrival_times.at (stop_index) % 60;
    cdf[eta]++;
}
```

Bibliography

Anderson, B. and Moore, J. (1979). *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall.

Auckland Transport (2019). *Auckland Transport Monthly Indicators Report 2019/20*. URL: https://at.govt.nz/media/1981080/item-11-attachment-1-open-22-october-2019-auckland-transport-monthly-indicators-report-august-2019.pdf.

Balogh, S. and Smith, R. (1993). "London Transport's Countdown system-A leader in the bus transport revolution". In: *IEE Colloquium on Public Transport Information and Management Systems*, pp. 7/1–7/3.

Benedetto, J. J. (1996). *Harmonic analysis and applications*. Ed. by B. Raton. Studies in advanced mathematics. CRC Press.

Bérczi, K., Jüttner, A., Laumanns, M., and Szabó, J. (2017). "Stochastic Route Planning in Public Transport". In: *Transportation Research Procedia* 27, pp. 1080–1087. DOI: 10.1016/j.trpro.2017.12.096.

Brent, R. P. (1971). "An algorithm with guaranteed convergence for finding a zero of a function". In: *The Computer Journal* 14.4, pp. 422–425. DOI: 10.1093/comjnl/14.4.422.

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). "Improved particle filter for nonlinear problems". In: *IEE Proceedings - Radar, Sonar and Navigation* 146.1, pp. 2–7. DOI: 10.1049/ip-rsn:19990255.

Cathey, F. W. and Dailey, D. J. (2003). "A prescription for transit arrival/departure prediction using automatic vehicle location data". In: *Transportation Research Part C: Emerging Technologies* 11.3-4, pp. 241–264. DOI: 10.1016/s0968-090x(03)00023-8.

Cats, O. and Loutos, G. (2015). "Real-Time Bus Arrival Information System: An Empirical Evaluation". In: *Journal of Intelligent Transportation Systems* 20.2, pp. 138–151. DOI: 10.1080/15472450.2015.1011638.

— (2016). "Evaluating the added-value of online bus arrival prediction schemes". In: *Transportation Research Part A: Policy and Practice* 86, pp. 35–55. DOI: 10.1016/j.tra.2016.02.004.

Čelan, M. and Lep, M. (2017). "Bus arrival time prediction based on network model". In: *Procedia Computer Science* 113, pp. 138–145. DOI: 10.1016/j.procs.2017.08.331.

— (2018). "Bus-arrival time prediction using bus network data model and time periods". In: *Future Generation Computer Systems*. DOI: 10.1016/j.future.2018.04.077.

Chang, H., Park, D., Lee, S., Lee, H., and Baek, S. (2010). "Dynamic multi-interval bus travel time prediction using bus transit data". In: *Transportmetrica* 6.1, pp. 19–38. DOI: 10.1080/18128600902929591.

Chen, H. and Rakha, H. A. (2014). "Real-time travel time prediction using particle filtering with a non-explicit state-transition model". In: *Transportation Research Part C* 43, pp. 112–126. DOI: 10.1016/j.trc.2014.02.008.

Chen, W., Yang, C., Feng, F., and Chen, Z. (2012). "An improved model for headway-based bus service unreliability prevention with vehicle load capacity constraint at bus stops". In: *Discrete Dynamics in Nature and Society* 2012. DOI: 10.1155/2012/313518.

Dagum, L. and Menon, R. (1998). "OpenMP: An Industry-Standard API for Shared-Memory Programming". In: *IEEE Comput. Sci. Eng.* 5.1, pp. 46–55. DOI: 10.1109/99.660313.

Dai, Z., Ma, X., and Chen, X. (2019). "Bus travel time modelling using GPS probe and smart card data: A probabilistic approach considering link travel time and station dwell time". In: *Journal of Intelligent Transportation Systems* 23.2, pp. 175–190. DOI: 10.1080/15472450.2018.1470932.

Dailey, D., Maclean, S., Cathey, F., and Wall, Z. (2001). "Transit Vehicle Arrival Prediction: Algorithm and Large-Scale Implementation". In: *Transportation Research*

Record: Journal of the Transportation Research Board 1771, pp. 46–51. DOI: 10.3141/1771-06.

Daum, F. (2005). "Nonlinear filters: beyond the Kalman filter". In: *IEEE Aerospace and Electronic Systems Magazine* 20.8, pp. 57–69. DOI: 10.1109/maes.2005.1499276.

Davidson, P., Collin, J., and Takala, J. (2011). "Application of particle filters to a map-matching algorithm". In: *Gyroscopy and Navigation* 2.4, pp. 285–292. DOI: 10.1134/s2075108711040067.

Doucet, A., Godsill, S., and Andrieu, C. (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". In: *Statistics and Computing* 10.3, pp. 197–208.

Eddelbuettel, D. and François, R. (2011). "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8, pp. 1–18. DOI: 10.18637/jss.v040.i08.

Fernandes, M., Walls, L., Munson, S., Hullman, J., and Kay, M. (2018). "Uncertainty Displays Using Quantile Dotplots or CDFs Improve Transit Decision-Making". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery. DOI: 10.1145/3173574.3173718.

Gong, J., Liu, M., and Zhang, S. (2013). "Hybrid dynamic prediction model of bus arrival time based on weighted of historical and real-time GPS data". In: *2013 25th Chinese Control and Decision Conference (CCDC)*, pp. 972–976. DOI: 10.1109/CCDC.2013.6561064.

Google Developers (2006). *What is GTFS?* https://developers.google.com/transit/gtfs/.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEE Proceedings F Radar and Signal Processing* 140.2, pp. 107–113. DOI: 10.1049/ip-f-2.1993.0015.

Gustafsson, F. (2010). "Particle filter theory and practice with positioning applications". In: *IEEE Aerospace and Electronic Systems Magazine* 25.7, pp. 53–82. DOI: 10.1109/maes.2010.5546308.

Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., and Nordlund, P.-J. (2002). "Particle Filters for Positioning, Navigation, and Tracking".

In: *IEEE Transactions on Signal Processing* 50.2, pp. 425–437. DOI: 10.1109/78.978396.

Häme, L. and Hakula, H. (2013a). "Dynamic Journeying in Scheduled Networks". In: *Trans. Intell. Transport. Sys.* 14.1, pp. 360–369. DOI: 10.1109/TITS.2012.2213817.

— (2013b). "Dynamic journeying under uncertainty". In: *European Journal of Operational Research* 225.3, pp. 455–471. DOI: 10.1016/j.ejor.2012.10.027.

Hans, E., Chiabaut, N., and Leclercq, L. (2014). "Investigating the irregularity of bus routes: highlighting how underlying assumptions of bus models impact the regularity results". In: *Journal of Advanced Transportation* 49.3, pp. 358–370. DOI: 10.1002/atr.1275.

Hans, E., Chiabaut, N., Leclercq, L., and Bertini, R. L. (2015). "Real-time bus route state forecasting using particle filter and mesoscopic modeling". In: *Transportation Research Part C: Emerging Technologies* 61, pp. 121–140. DOI: 10.1016/j.trc.2015.10.017.

He, P., Jiang, G., Lam, S.-K., and Sun, Y. (2020). "Learning heterogeneous traffic patterns for travel time prediction of bus journeys". In: *Information Sciences* 512, pp. 1394–1406. DOI: 10.1016/j.ins.2019.10.073.

Hijmans, R. J. (2019). *geosphere: Spherical Trigonometry*. R package version 1.5-10. URL: https://CRAN.R-project.org/package=geosphere.

Horn, M. E. (2004). "Procedures for planning multi-leg journeys with fixed-route and demand-responsive passenger transport services". In: *Transportation Research Part C: Emerging Technologies* 12.1, pp. 33–55. DOI: 10.1016/j.trc.2002.08.001.

Jeong, R. and Rilett, L. (2005). "Prediction Model of Bus Arrival Time for Real-Time Applications". In: *Transportation Research Record: Journal of the Transportation Research Board* 1927, pp. 195–204. DOI: 10.3141/1927-23.

Julio, N., Giesen, R., and Lizana, P. (2016). "Real-time prediction of bus travel speeds using traffic shockwaves and machine learning algorithms". In: *Research in Transportation Economics* 59, pp. 250–257. DOI: 10.1016/j.retrec.2016.07.019.

Kahle, D. and Wickham, H. (2013). "ggmap: Spatial Visualization with ggplot2". In: *The R Journal* 5.1, pp. 144–161. URL: https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf.

Kay, M. (2019). *tidybayes: Tidy Data and Geoms for Bayesian Models*. R package version 1.1.0. DOI: 10.5281/zenodo.1308151. URL: http://mjskay.github.io/tidybayes/.

Lu, H., Burge, P., Heywood, C., Sheldon, R., Lee, P., Barber, K., and Phillips, A. (2018). "The impact of real-time information on passengers' value of bus waiting time". In: *Transportation Research Procedia* 31, pp. 18–34. DOI: 10.1016/j.trpro.2018.09.043.

Ma, J., Chan, J., Ristanoski, G., Rajasegarar, S., and Leckie, C. (2019). "Bus travel time prediction with real-time traffic information". In: *Transportation Research Part C: Emerging Technologies* 105, pp. 536–549. DOI: 10.1016/j.trc.2019.06.008.

Mazloumi, E., Rose, G., Currie, G., and Moridpour, S. (2011). "Prediction intervals to account for uncertainties in neural network predictions: Methodology and application in bus travel time prediction". In: *Engineering Applications of Artificial Intelligence* 24.3, pp. 534–542. DOI: 10.1016/j.engappai.2010.11.004.

Mazloumi, E., Moridpour, S., Currie, G., and Rose, G. (2012). "Exploring the Value of Traffic Flow Data in Bus Travel Time Prediction". In: *Journal of Transportation Engineering* 138.4, pp. 436–446. DOI: 10.1061/(ASCE)TE.1943-5436.0000329.

Meng, Q. and Qu, X. (2013). "Bus dwell time estimation at bus bays: A probabilistic approach". In: *Transportation Research Part C: Emerging Technologies* 36, pp. 61–71. DOI: 10.1016/j.trc.2013.08.007.

Moreira-Matias, L., Mendes-Moreira, J., Sousa, J. F. de, and Gama, J. (2015). "Improving Mass Transit Operations by Using AVL-Based Systems: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4, pp. 1636–1653. DOI: 10.1109/tits.2014.2376772.

Mutambara, A. G. O. and Durrant-Whyte, H. E. (2000). "Estimation and control for a modular wheeled mobile robot". In: *IEEE Transactions on Control Systems Technology* 8.1, pp. 35–46. DOI: 10.1109/87.817690.

Okunieff, P. (1997). *AVL Systems for Bus Transit*. Synthesis of Transit Practice 24. Washington, DC: Transportation Research Board.

OpenStreetMap contributors (2017). *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org.

Plummer, M. (2003). *JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling*.

— (2018). *rjags: Bayesian Graphical Models using MCMC*. R package version 4-8. URL: https://CRAN.R-project.org/package=rjags.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). "CODA: Convergence Diagnosis and Output Analysis for MCMC". In: *R News* 6.1, pp. 7–11. URL: https://journal.r-project.org/archive/.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Reinhoudt, E. M. and Velastin, S. A. (1997). "A Dynamic Predicting Algorithm for Estimating Bus Arrival Time". In: *IFAC Proceedings Volumes* 30.8, pp. 1225–1228. DOI: 10.1016/S1474-6670(17)43988-7.

Robinson, S. (Dec. 2013). "Measuring Bus Stop Dwell Time and Time Lost Serving Stop with London iBus Automatic Vehicle Location Data". In: *Transportation Research Record: Journal of the Transportation Research Board* 2352, pp. 68–75. DOI: 10.3141/2352-08.

Schweiger, C. L. (2003). *Real-Time Bus Arrival Information Systems*. Synthesis of Transit Practice 48. Washington, DC: Transportation Research Board.

Schweiger, C. L. and Shammout, K. (2003). *Strategies for improved traveler information*. TCRP report 92. Washington, DC: Transportation Research Board.

Shalaby, A. and Farhan, A. (2004). "Prediction Model of Bus Arrival and Departure Times Using AVL and APC Data". In: *Journal of Public Transportation* 7.1, pp. 41–61. DOI: 10.5038/2375-0901.7.1.3.

Shen, J. and Li, W. (2013). "Cluster analysis of larger-scale discrete data with application to estimating dwell time of bus route". In: *Pakistan Journal of Statistics* 29, pp. 873–886.

Snyder, J. P. (1998). *Flattening the Earth*. The University of Chicago Press.

Statistics New Zealand (2019). Subnational population estimates (RC, SA2), by age and sex, at 30 June 1996, 2001, 2006-13, 2018-19 (2019 boundaries). Retrieved from www.stats.govt.nz.

Ulmke, M. and Koch, W. (2006). "Road-map assisted ground moving target tracking". In: *IEEE Transactions on Aerospace and Electronic Systems* 42.4, pp. 1264–1274. DOI: 10.1109/TAES.2006.314571.

Viggiano, C, Weisbrod, G, Jiang, S, Homstad, E, Chan, M, and Nural, S (2019). *Data Sharing Guidance for Public Transit Agencies—Now and in the Future*. Pre-publication draft of TCRP Research Report 213. Washington, D.C: Transportation Research Board.

Vuurstaek, J., Cich, G., Knapen, L., Ectors, W., Yasar, A.-U.-H., Bellemans, T., and Janssens, D. (2018). "GTFS bus stop mapping to the OSM network". In: *Future Generation Computer Systems*. DOI: 10.1016/j.future.2018.02.020.

Wall, Z. and Dailey, D. (1999). "An algorithm for predicting the arrival time of mass transit vehicles using Automatic Vehicle Location Data". In: *Proceedings of the Transportation Research Board Annual Meeting*. DOI: 10.1.1.579.2083.

Wang, C., Ye, Z., Wang, Y., Xu, Y., and Wang, W. (2016). "Modeling Bus Dwell Time and Time Lost Serving Stop in China". In: *Journal of Public Transportation* 19.3, pp. 55–77. DOI: 10.5038/2375-0901.19.3.4.

Wang, Y. and Chee, C.-S. (2012). "Density estimation using non-parametric and semi-parametric mixtures". In: *Statistical Modelling* 12.1, pp. 67–92. DOI: 10.1177/1471082X1001200104.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. R package version 3.1.1. Springer-Verlag New York. URL: https://ggplot2.tidyverse.org.

Xie, X., Liao, W., Aghajan, H., Veelaert, P., and Philips, W. (2016). "Detecting Road Intersections from GPS Traces Using Longest Common Subsequence Algorithm". In: *ISPRS International Journal of Geo-Information* 6.1, p. 1. DOI: 10.3390/ijgi6010001.

Xinghao, S., Jing, T., Guojun, C., and Qichong, S. (2013). "Predicting Bus Real-time Travel Time Basing on both GPS and RFID Data". In: *Procedia - Social and Behavioral Sciences* 96, pp. 2287–2299. DOI: 10.1016/j.sbspro.2013.08.258.

Yin, T., Zhong, G., Zhang, J., He, S., and Ran, B. (2017). "A prediction model of bus arrival time at stops with multi-routes". In: *Transportation Research Procedia* 25, pp. 4623–4636. DOI: 10.1016/j.trpro.2017.05.381.

Yu, B., Lam, W. H. K., and Tam, M. L. (2011). "Bus arrival time prediction at bus stop with multiple routes". In: *Transportation Research Part C: Emerging Technologies* 19.6, pp. 1157–1170. DOI: 10.1016/j.trc.2011.01.003.

Yu, B., Yang, Z.-Z., and Yao, B. (2006). "Bus Arrival Time Prediction Using Support Vector Machines". In: *Journal of Intelligent Transportation Systems* 10.4, pp. 151–158. DOI: 10.1080/15472450600981009.

Yu, B., Yang, Z.-Z., Chen, K., and Yu, B. (2010). "Hybrid model for prediction of bus arrival times at next station". In: *Journal of Advanced Transportation* 44.3, pp. 193–204. DOI: 10.1002/atr.136.

Zhang, Y., Liu, J., Qian, X., Qiu, A., and Zhang, F. (2017). "An Automatic Road Network Construction Method Using Massive GPS Trajectory Data". In: *ISPRS International Journal of Geo-Information* 6.12. DOI: 10.3390/ijgi6120400.

Zhao, Y. (1997). *Vehicle location and navigation systems*. Boston, Mass: Artech House.

Zheng, Y., Zhang, Y., and Li, L. (2016). "Reliable Path Planning for Bus Networks Considering Travel Time Uncertainty". In: *IEEE Intelligent Transportation Systems Magazine* 8.1, pp. 35–50. DOI: 10.1109/MITS.2015.2473475.