Traveling Time Prediction in Scheduled Transportation with Journey Segments

Avigdor Gal[a], Avishai Mandelbaum[a], Francois Schnitzler[a], Arik Senderovich[a,*], Matthias Weidlich[b]

[a]*Technion - Israel Institute of Technology, Haifa, Israel*
[b]*Imperial College London, London, United Kingdom*

Abstract

Urban mobility impacts urban life to a great extent. To enhance urban mobility, much research was invested in traveling time prediction: given an origin and destination, provide a passenger with an accurate estimation of how long a journey lasts. In this work, we investigate a novel combination of methods from Queueing Theory and Machine Learning in the prediction process. We propose a prediction engine that, given a scheduled bus journey (route) and a 'source/destination' pair, provides an estimate for the traveling time, while considering both historical data and real-time streams of information that are transmitted by buses. We propose a model that uses natural segmentation of the data according to bus stops and a set of predictors, some use learning while others are learning-free, to compute traveling time. Our empirical evaluation, using bus data that comes from the bus network in the city of Dublin, demonstrates that the snapshot principle, taken from Queueing Theory works well yet suffers from outliers. To overcome the outliers problem, we use machine learning techniques as a regulator that assists in identifying outliers and propose prediction based on historical data.

Keywords: Traveling Time Prediction, Queue Mining, Machine Learning, Prediction in Networks

1. Introduction

Urban mobility impacts urban life to a great extent. People, living in cities, plan their daily schedule around anticipated traffic patterns. Some wake-up early to "beat" rush hour. Others stay at home and work during days when a convention comes to town. The pleasure of a night in the city may be hampered by the unexpected traffic jam in a theater vicinity and sometimes, even a minor fender bender at an urban highway may wrack havoc the schedule of numerous people.

To enhance urban mobility, much research was invested in traveling time prediction (c.f. [1] and the references within). That is, given an origin and destination, provide a passenger with an accurate estimation of how long a journey lasts. In particular, the ability to predict traveling time in scheduled transportation, *e.g.,* buses, was shown to be feasible [2, 3].

In this work, we investigate a novel use of methods from Queueing Theory and Machine Learning in the prediction process. We propose a prediction engine that, given a scheduled bus journey (route) and a 'source/destination' pair, provides an estimate for the traveling time, while considering both historical data and real-time streams of information that are transmitted by buses. To do so, we model buses as clients that go through a journey of segments that are interpreted as a network of queues. We propose a model that uses natural segmentation of the data according to bus stops and a set of predictors, some use learning while others are learning-free, to estimate traveling time.

We test the proposed predictors using bus data that comes from the bus network in the city of Dublin. Our empirical analysis shows that the snapshot principle, taken from Queueing Theory works well yet suffers from outliers. To overcome the outliers problem, we use machine learning techniques that are based on historical data as boosting methods for the non-learning snapshot principle. To summarize, this work provides the following contributions:

- On a conceptual level, we propose to model scheduled transportation journeys as bus-stop segments.
- We extend an empirical results for the well-known Queueing Theory predictor (the snapshot principle) to a sequential network of segments.
- We introduce new learning-based techniques for the bus arrival problem. These techniques are based on ensembles of regression trees, and prove valuable when combined with the snapshot principle.
- We offer a new technique for data preprocessing, specifically, completing missing events by applying basic laws of kinematics, relating distances to velocity and time.
- We test and analyze the proposed predictors, and their combination, using a real-world dataset of Dublin buses.

The rest of the paper is organized as follows. Section 2 discusses related work. We develop the journey log in Section 3, followed by problem specification in Section 4. The proposed model is given in Section 5 followed by two prediction methods

*Corresponding author
Email addresses: avigal@ie.technion.ac.il (Avigdor Gal), avim@ie.technion.ac.il (Avishai Mandelbaum), francois@ee.technion.ac.il (Francois Schnitzler), sariks@tx.technion.ac.il (Arik Senderovich), m.weidlich@imperial.ac.uk (Matthias Weidlich)

(Section 6). Empirical evaluation is given in Section 7, followed by concluding remarks and future work (Section 8).

2. Related Work

Over the past decade, the problem of predicting traveling times of vehicles, and in particular of buses in urban areas, has received a significant attention in the literature. Most of the work on the subject includes applying various Machine Learning techniques such as Artificial Neural Networks [2], Support Vector Machines [1, 4], Kalman Filter models [5], and Non-Parametric Regression models [6]. A thorough literature review of the above techniques can be found in [3].

In recent work, some of the Machine Learning methods were applied to the bus data that we used for the current work, c.f. [7, 8]. Specifically, in [8], Kernel Regression was used on the Dublin bus data in order to predict the traveling time for bus line number $046A$, the same line that we have used for our evaluation. Due to the non-continuous nature of this data (see Section 3), a spatial segmentation of the route into 100-meter segments was proposed. In contrast to [8], the proposed segmentation in the current work is grounded in the data, since each record of the Dublin bus data relates the bus to a certain stop. Moreover, to better accommodate for the non-continuous structure of the data, and in alignment with our proposed segmentation, we apply advance learning techniques, *e.g.*, regression trees and boosting.

Traditionally, most state-of-the-art approaches to bus arrival-time prediction consider a single bus line at a time. In [3], Machine Learning models were applied to predict the traveling time of buses to given stops, by using data from *multiple lines* that travel through that same stop. Results show that further information regarding similar bus routes adds value to prediction. In the current work such a multi-line approach is enabled via our model of segmented journeys. Specifically, we take into consideration all bus lines that share bus stops with the journey whose time we aim to predict. To empirically demonstrate the value of the multi-line approach, our evaluation combines traveling times of several bus lines that share stops with line $046A$.

Another contribution of the current paper is the non-learning prediction method that base on Queueing Theory. The application of Queueing Theory to solve transportation problems is not new in the literature [9, 10]. However, in most of the works the traffic-flow (*e.g.*, flow of cars) is considered, with traffic modeled as 'customers' in a queueing system. In our work most customers are 'unobserved', while data recordings contain only information on bus travels (i.e. we do not have information regarding cars and other types of transportation). Nonetheless, we apply the *snapshot predictor*, which is a non-learning prediction method that relies on the travel time of the last bus to go through a segment. The motivation for the applications is derived from *queue mining* [11, 12], techniques that come from the domain of business process management, applying Queueing Theory to event data that stems from business processes. The value of these techniques was demonstrated in single-and-multi class queueing systems. Therefore, when considering buses as a class of transportation (that share routes with other classes, *e.g.*, cars), queue mining techniques and specifically, the snapshot predictor, are applicable.

A line of work combines Queueing Theory and Machine Learning [13, 14], where the structure of Web services is approximated via queueing networks. Then, the parameters of these networks are estimated from transactional data in low-traffic via Machine Learning techniques such as Monte-Carlo Markov-Chains. Lastly, the results are extrapolated into heavy-traffic scenarios, and are used to asses performance via *e.g.*, response-time analysis. Following this spirit, we propose predictors that integrate the snapshot approach into the regression tree model to create a more accurate prediction method.

3. The Journey Log

Prediction of traveling time may exploit historical data on scheduled journeys or real-time streams of information on recent movements of vehicles. This section defines a common model for these types of information by means of the notion of a *journey log* (J-Log). A J-Log is a set of sequences of recorded journey events of scheduled bus trips, each sequence being partially ordered by the timestamp that indicates the occurrence time of an event. As such, a J-Log is a particular type of an event log, as they are known, for instance, in the field of business process automation, see [15, Ch. 4].

The presented notion of a J-Log refers to buses that emit *journey events*. Then, a journey is characterized as a sequence of journey events, which, for instance, signal that a particular bus reached a bus stop.

Definition 1 (Journey event, Journey)**.** *Let $\mathcal{E}$ denote a set of journey events. The set of all possible journeys is given as the set of finite sequences of journey events, denoted by $\mathcal{E}^*$.*

In the remainder, for a specific journey $j = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $n \in \mathbb{N}^+$, we overload set notation and write $e \in j$ to denote that event e is an element of the sequence $\langle e_1, \ldots, e_n \rangle$. We will also refer to the i-th event of a specific journey j by the notation e_i^j.

Journey events are associated with attributes, e.g., *timestamps*, *journey patterns*, and *bus stops*. We model such an attribute as a function that assigns an attribute value to a journey event. A set of such attribute functions, in turn, defines the schema (aka structure or event type) over the set of journey events.

Definition 2 (Attribute function, Event schema)**.** *Let A be the domain of an event attribute. Then, an attribute function $\alpha : \mathcal{E} \to A$ assigns values of this domain to journey events. A finite set $\{\alpha_1, \ldots, \alpha_k\}$ of attribute functions is called a* schema*.*

A journey log comprises *observed* journeys, such that each journey is formed of *observed* journey events emitted by a particular bus. Here, a function τ in the schema captures the timestamp of a journey event, i.e., the time at which the event $e \in \mathcal{E}$ occurred is denoted by $\tau(e)$. Further, journey events indicate the progress of a bus in its route; they represent the points in time that a bus reaches a specific bus stop. Such bus stops are modeled as a set $\mathcal{S} \subseteq \mathbb{N}^+$. Finally, journeys shall follow a predefined schedule, referred to as a *journey pattern*. It is modeled as a

Table 1: Notations for J-Log and the Online Traveling-Time Prediction Problem

Notation	Meaning
$\mathcal{E} \subseteq \mathbb{N}$	Set of all journey events
$\mathcal{E}^*$	Set of all sequences of journey events
$j \in \mathcal{E}^*$	A journey, i.e., a sequence of journey events
$\mathcal{S} \subseteq \mathbb{N}$	Set of bus stops
$\mathcal{S}^*$	Set of all journey patterns, i.e., sequences of bus stops
$\tau : \mathcal{E} \to \mathbb{N}^+$	Function assigning timestamps to journey events
$\xi : \mathcal{E} \to \mathcal{S}$	Function assigning bus stops to journey events
$\pi : \mathcal{E} \to \mathcal{S}^*$	Function assigning journey patterns to journey events
$T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$	Random variable for the traveling time from stop $\omega_1 \in \mathcal{S}$ to stop $\omega_n \in \mathcal{S}$ via the sequence of stops $\langle \omega_1, \ldots, \omega_n \rangle \in \mathcal{S}^*$, departing at time $t_{\omega_1} \in \mathbb{N}^+$

sequence of bus stops at which a bus should stop. Formally, a journey pattern belongs to the set of finite sequences over $\mathcal{S}$, which we denote by $\mathcal{S}^*$.

Generalizing the *functional* definition of an event log as presented in [16], a J-Log is defined as follows:

Definition 3 (J-Log). *A journey log is a tuple (J, α_J), consisting of a set of journeys $J \subseteq \mathcal{E}^*$, which are defined by journey events of schema $\alpha_J = \{\tau, \xi, \pi\}$, such that*

- *$\tau : \mathcal{E} \to \mathbb{N}^+$ assigns timestamps to journey events,*
- *$\xi : \mathcal{E} \to \mathcal{S}$ assigns bus stops to journey events,*
- *$\pi : \mathcal{E} \to \mathcal{S}^*$ assigns journey patterns to journey events,*

and it holds $\tau(e_i) \leq \tau(e_{i'})$ for all journeys $\langle e_1, \ldots, e_n \rangle \in J$ and $1 \leq i < i' \leq n$.

An overview of the introduced notations, along with notations for concepts introduced later, can be found in Table 1.

EXAMPLE 1. *We illustrate the notion of a J-Log using data of the bus network in the city of Dublin.*[1] *Here, location of buses is sampled in intervals of 5 to 300 seconds (20 seconds on average), depending on the current location of the bus. For each event the following data is submitted to a monitoring system:*

- *A timestamp of the event.*
- *A vehicle identifier for the bus.*
- *A bus stop relating the bus to the stop on its journey with maximal proximity. Hence, every event has a bus stop identifier, even when the bus is not at the stop.*
- *A journey pattern that defines the sequence of bus stops for a journey.*

Based on this input data, the construction of a J-Log as defined above is trivial; timestamps, bus stops and journey patterns are given directly in the input. To partition the events into journeys, a combination of the vehicle identifier and the journey pattern is used. An excerpt of the J-Log for the bus data from the city of Dublin is presented in Table 2. It features intuitive values for the attributes (e.g., stop "Parnell Square") as well as their numeric representation according to our formal model (e.g., 264 identifies the stop "Parnell Square").

[1] See also `http://www.dublinked.ie/` and `http://www.insight-ict.eu/`

Table 2: Example J-Log from buses in Dublin

Event Id	Journey Id	Timestamp	Bus Stop	Journey Pattern
1	36006	1415687360	Leeson Street Lower (846)	046A0001
2	36012	1415687365	North Circular Road (813)	046A0001
3	36009	1415687366	Parnell Square (264)	046A0001
4	36006	1415687381	Leeson Street Lower (846)	046A0001
5	36009	1415687386	O'Connell St (6059)	046A0001
6	36012	1415687386	North Circular Road (814)	046A0001
7	36006	1415687401	Leeson Street Upper (847)	046A0001
8	36009	1415687406	O'Connell St (6059)	046A0001

4. The Traveling Time Prediction Problem

Traveling time prediction is an essential tool for providing integrated solutions for urban mobility, reaching from the creation of individual journey itineraries over capacity planning to car lift sharing [7]. In general, given a source and a destination, travel time prediction answers the question of how long the respective journey lasts. However, in most scenarios, traveling times vary greatly over time, *e.g.*, due to different levels of traffic load. Consequently, prediction is inherently time dependent, so that a specific predictor is a function of the source, the destination, and the time at which the prediction is made.

In this work, we address the problem of online travel time prediction in the context of a bus journey. That is, a journey may be ongoing in the sense that journey events already indicated the progress of the bus on its route. For such an ongoing journey, we are interested in the current prediction of the traveling time from the current bus stop to some destination via a particular sequence of stops, which is defined by the respective journey pattern. Using the model introduced in the previous section for journeys and bus stops, we represent this *traveling time*, from stop $\omega_1 \in \mathcal{S}$ to stop $\omega_n \in \mathcal{S}$ via the sequence of stops $\langle \omega_1, \ldots, \omega_n \rangle \in \mathcal{S}^*$ and departing at time $t_{\omega_1} \in \mathbb{N}^*$ by a random variable $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.

The traveling time prediction problem relates to the identification of a precise predictor for $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.

Problem 1 (Online Traveling-Time Prediction). *For a random variable $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$ representing a traveling time, the* online traveling-time prediction problem *is to find a precise predictor θ for $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.*

Various measures that quantify the precision of prediction have been presented in the literature. In this work, we measure prediction precision by e.g.: the root-mean squared error (RMSE), the mean absolute relative error (MARE), and the median absolute relative error (MdARE). These performance measures will be further defined in Section 7.2.

5. The Model of Segmented Journeys

To address the problem of traveling time prediction, as a first step, we construct a model that establishes a relationship between different journeys by means of visited bus stops. To this end, journeys are modeled as *segments of stops*.

A trip between two stops consists of segments, with each segment being represented by a 'start' stop and an 'end' stop, see

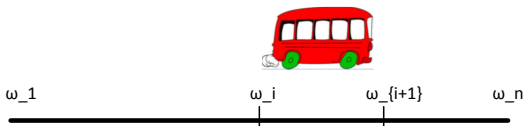

Figure 1: A segmented model of traveling times

Figure 1. Given the first stop of a trip $\omega_1 \in \mathcal{S}$ and the last stop of a trip $\omega_n \in \mathcal{S}$, the intermediate stops are known in advance since each bus follows a predefined journey pattern. Therefore, a trip can be described by segments that are characterized by a pair of stops of the form $\langle \omega_i, \omega_{i+1} \rangle$ (Figure 1). This segmented model, in turn, allows for fine-granular grounding of the prediction of traveling time $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$: instead of considering only journeys that follow the same sequence of stops $\langle \omega_1, \ldots, \omega_n \rangle$, all journeys that share some segments can be used for prediction.

Below, we first describe the segmented model for journeys (Section 5.1). Subsequently, we show how to transform a J-Log into a Segmented J-Log, a log that contains information on segments (Section 5.2).

5.1. The Segmented Model

While the benefit of segmentation of journeys for the purpose of time prediction has been widely acknowledged [8], various approaches may be applied to identify segments. Our work defines segments based on information about the bus stops of a journey pattern. Such segmentation is naturally derived from the structure of the data commonly available in traveling time prediction for bus networks.

Using information on bus stops, the prediction of the journey traveling time $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$ is traced back to the sum of traveling times per segment. The traveling time per segment, in turn, is assumed to be independent of a specific journey pattern (and, thus, also independent of a specific journey):

$$T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1}) = \\ T(\langle \omega_1, \omega_2 \rangle, t_{\omega_1}) + \ldots + T(\langle \omega_{n-1}, \omega_n \rangle, t_{\omega_{n-1}})$$

where

$$t_{\omega_{n-1}} = t_{\omega_1} + T(\langle \omega_1, \omega_{n-1} \rangle, t_{\omega_1}).$$

5.2. Deriving Segments from a Journey Log

A journey log (J-Log) is built of multiple journeys, where a journey is a sequences of events that are emitted by a bus as part of a particular trip. In order to rely on the aforementioned segmented model as the basis for the traveling time predictors, in a first step, we transform the J-Log into a Segmented J-Log that is built of timing information for segments.

A Segmented J-Log is a sequence of *segment events* that capture information on the start and end bus stop of the segment, the journey from which the segment event was derived, the respective journey pattern, and the start and end timestamps observed for the segment. The last two elements are computed using the earliest time the particular journey reached the start and end bus stop.

Definition 4 (Segment Events, Segmented J-Log)**.** *Let $\mathcal{G}$ be a set of segment events and let $\mathcal{G}^*$ be the set of all possible sequences of segment events. A Segmented J-Log is a tuple (G, α_G) where $G \in \mathcal{G}^*$ is a sequence of segment events of schema $\alpha_G = \{\xi_{start}, \xi_{end}, \epsilon, \pi_G, \tau_{start}, \tau_{end}\}$:*

- *$\xi_{start} : \mathcal{G} \to \mathcal{S}$ and $\xi_{end} : \mathcal{G} \to \mathcal{S}$ assign start and end stops to segment events,*
- *$\epsilon : \mathcal{G} \to \mathcal{E}^*$ assigns a journey to segment events,*
- *$\pi_G : \mathcal{G} \to \mathcal{S}^*$ assigns journey patterns to segment events,*
- *$\tau_{start} : \mathcal{G} \to \mathbb{N}^*$ and $\tau_{end} : \mathcal{G} \to \mathbb{N}^*$ assign start and end timestamps to segment events,*

A Segmented J-Log is constructed from the journey events of all journeys in a J-Log. That is, a segment event is derived from two journey events of the same journey, such that (1) the journey events refer to two successive bus stops of the journey pattern, and (2) the journey events are the earliest events referring to these two bus stops. We capture this construction as follows. Let (J, α_J) be a J-Log with $\alpha_J = \{\tau, \xi, \pi\}$. Let $j \in J$ be a journey serving journey pattern $\pi(e) = \langle \omega_1, \ldots, \omega_n \rangle$, for all $e \in j$. For a stop ω_i, $1 \leq i \leq n$, let

$$e_i = \text{argmin}_{e \in j, \xi(e) = \omega_i} \tau(e)$$

be the earliest event for stop ω_i of this journey. Then, for each pair of successive stops ω_i, ω_{i+1}, $1 \leq i < n$, a segment event s of schema $\alpha_G = \{\xi_{start}, \xi_{end}, \epsilon, \pi_G, \tau_{start}, \tau_{end}\}$ is created, such that

- $\xi_{start}(s) = \omega_i$ and $\xi_{end}(s) = \omega_{i+1}$,
- $\epsilon(s) = j$,
- $\tau_{start}(s) = \tau(e_i)$ and $\tau_{end}(s) = \tau(e_{i+1})$.

A Segmented J-Log (G, α_G) for (J, α_J) is constructed as a sequence $G = \langle s_1, \ldots, s_m \rangle \in \mathcal{G}^*$ over all segment events of all journeys $j \in J$, such that $\tau_{start}(s_k) \leq \tau_{start}(s_{k'})$ for $1 \leq k < k' \leq n$.

A Segmented J-Log can be trivially constructed from a J-Log. However, in many real-world applications, the recorded data is incomplete due to data loss, unavailability of data recording devices, or data sampling. Then, it may be impossible to construct a segment event for each pair of successive bus stops of all journeys. For instance, for the data of the bus network in the city of Dublin described in Example 1, due to data sampling, the raw data does not necessarily contain a journey event for each bus stop of the respective journey pattern.

In the remainder of this section, therefore, we outline how to use complementary information on the geographical distances between bus stops and principles of *kinematics* (relating distances to velocity and time) to impute missing journey events and their corresponding timestamps. We assume such distances to be given as a function $\delta : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+$ assigning distance values to pairs of bus stops.

For a journey pattern and a journey recorded in the J-Log, we consider the following three cases of missing sequences. There is a sequence of missing journey events:

I) between two consecutive recorded journey events.

II) that includes the first bus stop.
III) that includes the last bus stop.

We shall first demonstrate the approach to construct the missing journey events for case I) and then demonstrate its refinement for the other two cases.

Let $j = \langle e_1, \ldots, e_u, m_1, \ldots, m_k, e_v, \ldots, e_n \rangle$ be a journey with $m_1, \ldots, m_k$ representing missing journey events, events that according to the journey pattern $\pi(e_1)$ must exist between the stops $\xi(e_u)$ and $\xi(e_v)$. To reconstruct the journey events $m_1, \ldots, m_k$, we need to estimate their timestamps, since information on their route pattern and the respective bus stops are known.

We estimate the timestamps for missing events based on the average velocity $v : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$ of the respective bus between two stops. It is derived from the events that signal that a bus has been at a certain stop and the distance between the stops:

$$\begin{aligned} v(\omega, \omega') =& \frac{\delta(\omega, \omega')}{\tau(e') - \tau(e)} \\ e =& \operatorname{argmin}_{\hat{e} \in j, \xi(\hat{e}) = \omega} \tau(\hat{e}) \\ e' =& \operatorname{argmin}_{\hat{e} \in j, \xi(\hat{e}) = \omega'} \tau(\hat{e}) \end{aligned}$$

For journey $j = \langle e_1, \ldots, e_u, m_1, \ldots, m_k, e_v, \ldots, e_n \rangle$ with missing journey events $m_1, \ldots, m_k$, we then assume that the bus travels at a constant velocity through every segment on the way between $\xi(e_u)$ and $\xi(e_v)$, which gives rise to the following recursive approximation of the timestamps of the journey events:

$$\begin{aligned} \tau(m_1) &= \tau(e_u) + \frac{\delta(\xi(e_u), \xi(m_1))}{v(\xi(e_u), \xi(e_v))}, \\ \tau(m_i) &= \tau_{m_{i-1}} + \frac{\delta(\xi(m_{i-1}), \xi(m_1))}{v(\xi(e_u), \xi(e_v))}, \quad 1 < i \leq k. \end{aligned}$$

Next, we target the cases II and III, in which the sequence of missing journey events includes the first bus stop, or the last bus stop, respectively. In both cases, we adapt the approach presented above and calculate the velocity of the bus at the segment that still appears in the J-Log after (case II) or before (case III) the recorded journey events, and, as before, assume constant speed of travel throughout the unobserved traveling time. We detail this approach for case II. Case III is handled analogously.

Let $j = \langle m_1, \ldots, m_k, e_u, e_v, \ldots, e_n \rangle$ be a journey with missing events $m_1, \ldots, m_k$. Then, the timestamps of the missing events are determined by the following recursive approximation:

$$\begin{aligned} \tau(m_k) &= \tau(e_u) - \frac{\delta(\xi(m_1), \xi(e_u))}{v(\xi(e_u), \xi(e_v))}, \\ \tau(m_i) &= \tau_{m_{i+1}} - \frac{\delta(\xi(m_i), \xi(m_{i+1}))}{v(\xi(e_u), \xi(e_v))}, \quad 1 \leq i < k. \end{aligned}$$

6. Prediction Methods and Algorithms

The prediction methods we present can be divided into two types. The first type consists of a method that comes from Queueing Theory and approximates systems in heavy-traffic. The predictor is non-learning, in the sense that it does not generalize prediction from historical events, but rather uses recent events to predict future traveling times. The second type comes from Machine Learning and is based on decision trees. The feature space includes also recent information that is considered by the first-type predictor. Both prediction methods make use of the segmented model of journeys (Section 5.1) and on the Segmented J-Log (Section 5.2).

6.1. The Snapshot Principle for Non-Learning Prediction of Traveling Times

We now introduce the snapshot principle for traveling time prediction. Section 6.1.1 provides an overview of the method. The algorithm is detailed in Section 6.1.2.

6.1.1. Method

The *snapshot principle* [17], p. 187, is a heavy-traffic approximation that refers to the behavior of a queue model under limits of its parameters, as the workload converges to capacity. In our context it means that a bus that passes through a segment, *e.g.*, $\langle \omega_i, \omega_{i+1} \rangle \in \mathcal{S} \times \mathcal{S}$, will experience the same traveling time as another bus that has just passed through that segment (not necessarily of the same type, line, *etc*.). Based on the above, we define a single-segment snapshot predictor, Last-Bus-to-Travel-Segment (LBTS), denoted by $\theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1})$.

In real-life settings, the heavy-traffic approximation is not always plausible and thus the applicability of the snapshot principle predictors should be tested ad-hoc, when working with real-world data sets. Results of synthetic simulation runs [18] show that snapshot-based predictors are indeed appropriate for predicting delays. Moreover, it was shown that the snapshot principle predict delays well even when the system is not under heavy load and for a multi-class scenario [11, 12].

In our case however, the LBTS predictor needs to be lifted to a network setting. In [17, p.187], it is stated that the snapshot principle holds for networks of queues, when the routing through this network is known in advance. Clearly, in scheduled transportation such as buses this is the case as the order of stops (and segments) is predefined. Therefore, we define a multi-segment (network) snapshot predictor that we refer to as the Last-Bus-to-Travel-Network or $\theta_{LBTN}(\langle \omega_1, ..., \omega_n \rangle, t_{\omega_1})$, given a sequence of stops (with ω_1 being the start stop and ω_n being the end stop). According to the snapshot principle in networks we get that:

$$\theta_{LBTN}(\langle \omega_1, ..., \omega_n \rangle, t_{\omega_1}) = \sum_{i=1}^{n} \theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1}).$$

We hypothesize that the snapshot predictor performs better whenever recent buses are in time proximity to the current journey. We test this hypothesis in Section 7.

6.1.2. Algorithm

We shall now demonstrate the algorithm to obtain the LBTS and thus the LBTN from the Segmented J-Log. Following the snapshot principle for networks, a bus that is currently at ω_1 is to travel the sum of past traveling times of the last buses that traveled through each of the segments $\langle \omega_i, \omega_{i+1} \rangle$ during that day,

prior to time t_{ω_1}. Therefore, for each pair $\langle \omega_i, \omega_{i+1} \rangle$ we are to search (in the Segmented J-Log) for the previous bus that passed through that segment (prior to time t_{ω_1}) and use its traveling time as an estimate for $T(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_i})$. Hence, the algorithm for extracting the snapshot predictor per segment (LBTS) is as follows. First, given a Segmented J-Log, (G, α_G), we obtain the index of the segmented event that was last to travel through $\langle \omega_i, \omega_{i+1} \rangle$ prior to time t_{ω_1}:

$$i_{LB}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1}) = \\ \text{argmax}_{s \in G, \xi_{start}(s)=\omega_i, \xi_{end}(s)=\omega_{i+1}, \tau_{end}(s)<t_{\omega_1}} \tau_{end}(s). \quad (1)$$

Then, we calculate the estimator as:

$$\theta_{LBTS} = \tau_{end}(s_{i_{LB}}) - \tau_{start}(s_{i_{LB}}).$$

This definition characterizes θ_{LBTS} as a non-learning predictor—it only requires the traveling times of the last buses that went through particular segments.

6.2. Machine Learning for Prediction of Traveling Times

In this section we describe the use of Machine Learning and, more specifically, of regression techniques to predict the bus traveling time $T(\langle \omega_1, \ldots \omega_n \rangle, t_{\omega_1})$. As opposed to the snapshot method described above, machine learning techniques exploit past journey logs to learn a prediction model, and then use this model to make a prediction on new instances of the problem, in our case, traveling times as part of current journeys.

Below, we first formalize the traveling times prediction problem as a regression problem and discuss the features we use. Then, we briefly describe the generic regression algorithms that we apply to solve the problem. Finally, we integrate the snapshot predictor with machine learning methods.

6.2.1. Formalization and Features

Exploiting the introduced model of segmented journeys, we construct a (Machine Learning) predictor, θ_{ML},

$$\theta_{ML}(\langle \omega_i, \omega_{i+1} \rangle, t, \hat{t}_{\omega_i}) : S \times S \times \mathbb{N}^+ \times \mathbb{N}^+ \rightarrow \mathbb{R}^+ \ ,$$

for every traveled segment $\langle \omega_i, \omega_{i+1} \rangle$ along the route. The predictors takes as input the prediction time $t = t_{\omega_1}$ and the estimated time the bus enters the segment, $\hat{t}_{w_i}$. Based on these predictors, a travel time $T(\langle \omega_1, \omega_n \rangle, t_{\omega_1})$ is estimated by

$$\theta_{ML}(\langle \omega_1, \omega_n \rangle, t_{\omega_1}) = \sum_{i=1}^{n-1} \theta_{ML}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1}, \hat{t}_{\omega_i}) \quad (2)$$

$$\hat{t}_{\omega_i} = t_{\omega_1} + \theta_{ML}(\langle \omega_1, \ldots \omega_i \rangle, t_{\omega_1}, t_{\omega_1}) \quad (3)$$

The Machine Learning approach is formalized in two steps. First, we define a feature constructor $\phi : S \times S \times \mathbb{N}^+ \times \mathbb{N}^+ \rightarrow F$ where F is the feature space. The features are used as input for the second step, which is the construction of a regression model $\psi : F \rightarrow \mathbb{R}^+$ that outputs a traveling time prediction. The same ϕ is used for every segment while a model ψ is learned for each segment, anew. The features we consider are

(1) $\theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1})$, the travel time of the last bus that used that segment (see Eq. 1);
(2) $\hat{t}_{\omega_i} - \tau_{end}(s_{i_{LB}})$, the interval between the time the last bus left the segment, and $\hat{t}_{\omega_i}$;
(3) $d(\hat{t}_{\omega_i})$, the day of the week; and
(4) $hms(\hat{t}_{\omega_i})$, the time of the day (hours, minutes, seconds) corresponding to $\hat{t}_{\omega_i}$.

The first two features are computed from the Segmented J-Log and therefore depend on the information available when the prediction is made, at time t.

6.2.2. Generic Algorithms

In this section, we briefly present the regression algorithms applied to the aforementioned features. These algorithms all output ensembles $\Psi_M = \{\psi_m\}_{m=1}^M$ of M regression trees. A regression tree [19] is a tree where each internal node is a test on the value of a feature, and where each leaf corresponds to a value of the target variable, in our case traveling time. An instance goes down from the root to a leaf by selecting at each internal node the branch corresponding to the result of the test of that node. The predicted value for that instance is the value associated with the leaf it reaches. Ensembles are sets of regression trees. The value predicted by the ensemble is typically the (potentially weighted) average of the values predicted by each tree of the ensemble:

$$\Psi_M(.) = \sum_{m=1}^{M} \lambda_m \psi_m(.) \ ,$$

where λ_m is the weight of tree ψ_m. Using an ensemble rather than a single model typically leads to an improvement in accuracy [20]. We briefly describe below the methods we considered to build ensembles. We note that our selection of these methods cannot be comprehensive. However, since our focus is on a comparative analysis with the snapshot-based method for prediction, a selection that covers a variety of different techniques is sufficient.

A **random forest** (RF) [21] is an ensemble built by learning each tree on a different bootstrap replica of the original learning set. A bootstrap replica is obtained by randomly drawing (with replacement) original samples and copying them into the replica. Each tree is learned by starting with a single leaf and greedily extending the tree. Extension consists of considering all possible tests (features and values) at all leafs and splitting the leaf using the test that maximizes the reduction in quadratic error. The tree weights are all equal $\lambda_m = 1/M$.

Extremely randomized trees (ET) [22] is an ensemble where each tree was learned by randomizing the test considered during greedy construction. Instead of considering all values of the features for the split test, only a value selected at random is considered for each feature (and leaf). The tree weights are all equal $\lambda_m = 1/M$.

AdaBoost (AB) [23] builds an ensemble iteratively by reweighting the learning samples based on how well their target variable is predicted by the current ensemble. The worse the prediction is, the higher the weight becomes. Therefore, the next tree constructed focuses on the most 'difficult' samples.

Given the m^{th} model $\psi_m : \mathcal{X} \rightarrow \mathcal{Y}$ learned from a learning set $\{x_k, y_k\}_{k=1}^N$ with weights w_k^m, the next weights w_k^{m+1} are computed as follows:

$$L_k = (y_k - \psi_m(x_k))^2 / \max_j (y_j - \psi_m(x_j))^2$$
$$\bar{L} = \sum_k L_k w_k^m / \sum_j w_j^m$$
$$\beta_m = \bar{L}/(1-\bar{L})$$
$$w_k^{m+1} = w_k^m \beta_m^{1-L_k} \ .$$

The value predicted is the weighted median of the predictions of the trees, where the weight of each tree ψ_m is $-\log \beta_m$. Initial weights are all equal to $1/N$. AB is typically used with weak models, that do not model the data well outside an ensemble. For this reason, the depth (number of tests before reaching a leaf) of regression trees is typically limited when they are combined using AdaBoost. In our experiments, we tried both a depth of 1 and 3. The latter was almost always better, so we will only report the corresponding results. Except for this limitation, trees are learned greedily on the re-weighted learning sets.

Gradient tree boosting (GB) [24] is another boosting algorithm. Instead of weighting the samples, GB modifies modifies the target variable value for learning each tree. The values used to learn the m^{th} tree are given by

$$\tilde{y}_k = y_k - \Psi_{m-1}(x_k) \ . \quad (4)$$

The new tree is trained on the prediction error $\tilde{y}_k$. In this algorithm, the model weights are replaced by leaf weights λ_m^l, where l is a leaf index. The leaf weight is given by $\lambda_m = \nu\gamma_m^l$. ν is a regularization term (equal to 0.1 in our experiments). γ_m^l is optimized by line search:

$$\gamma_m^l = \arg\min_\gamma \sum_{i:reach(x_k,\psi_m,l)} (y_k - [\Psi_{m-1}(x_k) + \gamma\psi_m(x_k)])^2$$

where $reach(x_k, \psi_m, l)$ is true if x_k reaches leaf l in the tree ψ_m. This ensemble is initialized by a tree learned on the unweighted learning set. We also considered a more robust version of this algorithm, denoted by GBLAD, that optimizes the absolute deviation error instead of the mean quadratic error. The most important changes are that each tree is constructed on a learning set $\{x_k, sign(y_k)\}$, and the value of each leaf is the median of the prediction errors of the training samples that reach it.

6.2.3. Combining Snapshot and Machine Learning

The snapshot method stems from Queueing Theory and was demonstrated to perform well in practice, for delay prediction in various settings where heavy traffic (resource queues) produces these delays [12, 11]. Since bus delays are often induced by car traffic, it is tempting to use it as a baseline and try to improve over it. Boosting algorithms appear particularly suited for that task, since they construct ensembles of models sequentially, based on the results of the previous models in the ensemble. Following this line of reasoning, we modify the three boosting algorithms discussed above (AB, GB and GBLAD). That is, the first model considered in the boosting is the snapshot model. We respectively denote the three resulting algorithms S+AD, S+GB and S+GBLAD.

7. Evaluation

In this section, we empirically evaluate the prediction accuracy of the methods that we described in the previous section. The main results of our experiments are:

- Prediction methods that combine the snapshot principle and advanced Machine Learning techniques are superior in quality of prediction to both snapshot predictors and Machine Learning methods (that do not include the snapshot predictor).
- The prediction error increases with the number of bus stops per journey. However, when considering the relative error, it is stable for all trip lengths, i.e. the predictors do not deteriorate proportionally to length of the journey (in stations).
- Surprisingly, the snapshot predictor does not deteriorate for longer trips, therefore contradicting the hypothesis that the snapshot predictor would be more precise for journeys with higher temporal proximity to the current journey.

Below, we first describe the datasets log used for our experiments (Section 7.1), by first going over the training set and then introducing the test set. Then, we describe our experimental setup (Section 7.2). Lastly, Section 7.3 introduces the controlled variables that we selected for explaining the prediction error and reports on our main results.

7.1. Datasets

We shall now describe the construction of two datasets, the training set and the test set that we used for our experiments. First, the training set was used to train our Machine Learning methods and construct learning sets for each of the segments. Then, a test set of all trips (and sub-trips) was constructed to test both the Machine Learning and the Snapshot predictors.

The training set in our experiments consists of 8 days of bus data, between September and October of 2014. Each day contains approximately 11500 traveled segments. Essentially, the learning set is a Segmented Journey Log for those 8 days, with the addition of information on the last bus that traveled through the segment during that day. The data that we add for the last bus is the following: (1) the timestamp of the bus entering the segment, (2) the time that it took for the bus to travel through the segment, and (3) and indicator to whether the timestamp for segment entry was imputed via the algorithm in Section 5.2. Note that the first trip for a certain day will not include the last bus to go through a segment during that day. The data comes from all buses that share segments with line $046A$, since it is the only line that we analyze via the test set. Hence, we exploited traveling time information from additional lines that share segments with $046A$.

The test set comprises bus data that comes from a single day, September 22nd, 2014. We considered actual trips of line $046A$, which is one of the lengthiest lines in the city of Dublin (58 stops). First, the line travels through city center, where traffic can become extremely hectic and a large number of passengers

may cause stopping delays. Then, it goes through a highway section and lastly it visits a suburban area where the delays are mostly due to frequent get-offs of passengers [8]. During that day, the line has traveled through 111 journeys, all of which were included in our test set.

For our experimental setting we consider all 'real' subsequences of each of the actual 111 trips, i.e. sub-sequences that were actually traveled in reality. In other words, a single journey $\langle \omega_1, ..., \omega_{58} \rangle$ emits the following 'real' sequences: $\{\langle \omega_1, \omega_2 \rangle, \langle \omega_1, \omega_3 \rangle, \langle \omega_1, \omega_4 \rangle, ..., \langle \omega_2, \omega_3 \rangle, ...\}$. In total, the test set contains 172812 trips, since we excluded trips that had more than 50% of its timestamps imputed and early trips that did not have previous buses traveling through the relevant segments. For every source-destination (e.g. $\langle \omega_1, \omega_3 \rangle$), the test set contains: (1) the source (ω_1) of the trip, (2) the destination (ω_3), (3) the traveling time between source and destination, (4) the time of entry to segment, (5) the number of segments between source and destination, and (6) a set of tuples that represent all segments between source and destination, including the entry time into each segment, and the duration of traveling through the segment.

Therefore, for our running example of trip $\langle \omega_1, \omega_3 \rangle$, our test set contains the following tuple: $\langle \omega_1, \omega_3, T(\langle \omega_1, \omega_3 \rangle, t_{\omega_1}), t, 2, \{\langle \omega_2, T(\langle \omega_2, \omega_3 \rangle, t_{\omega_2}), t_{\omega_2} \rangle\} \rangle$, i.e., the traveling time between the two stations, the entry time, the fact that the trip contains two segments, with ω_2 being the next stop after ω_1, the traveling time through $\langle \omega_2, \omega_3 \rangle$, and the entry time into the last segment.

7.2. Setup

7.2.1. Implementation of Machine Learning methods

For the machine learning methods, we used the scikit-learn [25] implementation of the algorithms to create ensembles of regression trees. Also, we relied on ensembles of $M = 100$ trees, unless otherwise stated. The algorithms that combine the snapshot method with machine learning (S+AD, S+GB and S+GBLAD), therefore contain 99 trees in addition to the snapshot model.

7.2.2. Quality of Prediction

We first define our uncontrolled variables, namely the quality measures for the goodness of our predictors. Then, we describe the controlled variables that we selected in an attempt to explain the prediction errors.

We consider several measures for the quality of the predictor θ. The first measure is based on the squared difference between the real traveling time and the predicted value, namely the Root Mean Squared Error (RMSE). Note that the measure can be calculated with respect to an entire trip, e.g. for θ_{LBTN} or for a single segment as in θ_{LBTS}. Formally, the RMSE is defined as follows:

$$RMSE(\theta) = \sqrt{\mathbb{E}[\theta - T]^2},$$

where $\mathbb{E}$ is the expectation of a random variable, T the real traveling time and θ the predictor for T. The RMSE quantifies the error in the time units of the original measurements, i.e., in our case trips are measured in seconds and therefore the error will also be returned in seconds. The theoretical measure of RMSE can be approximated from the *test set* as follows:

$$\widehat{RMSE}(\theta) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(t_k - p_k)^2},$$

where N is the number of trips (or segments), t_k the real travel times through the trip (or segment) and p_k the travel times predicted by θ for the corresponding trip.

The RMSE presents two major issues that require additional performance measures. First, it is sensitive to outliers ([26]) and second it is not normalized with respect to the traveling time. To deal with the latter problem we consider the relative error, i.e. we normalize the error with respect to the real traveling time T. To accommodate for the first problem, we swap the squared error with the absolute error, which is known to be more robust to outliers [26]. Hence, we use the mean absolute relative error (MARE) and the median of absolute relative error (MdARE):

$$\begin{aligned} MARE(\theta) &= \mathbb{E}[\frac{|\theta - T|}{T}], \\ MdARE(\theta) &= \text{median}\left\{\frac{|\theta - T|}{T}\right\} \quad . \end{aligned} \tag{5}$$

For the empirical approximation of the MARE and the MdARE we use the following measures based on the test set:

$$\begin{aligned} \widehat{MARE}(\theta) &= \frac{1}{N}\sum_{k=1}^{N}\frac{|(t_k - p_k)|}{t_k}, \\ \widehat{MdARE}(\theta) &= \text{median}\left\{\frac{|(t_k - p_k)|}{t_k}, k = 1, \ldots, N\right\}, \end{aligned} \tag{6}$$

with N, t_k, p_k, defined as before and the median being calculated as the empirical 50th quantile.

7.3. Results

The first set of controlled variables that we present in the results are the *prediction methods*, i.e. *snapshot predictor* (S), *random forest* (RF), *extremely randomized trees* (ET) etc. As additional covariate we consider the *length of trip*, that may vary between a single segment (length of 1) and a whole trip for $046A$ (58 stops). Moreover, due to the additive structure of the segmented model, we performed an extensive analysis of the *single-segment prediction error*. This follows the paradigm that if the single-segment prediction is improved, the entire model could improve accordingly.

Table 3 presents the accuracy of the different methods tested over all trips. In terms of root-mean square error, the snapshot method (S) is the least accurate, together with the random forest (RF) method. The square error of the combination of the snapshot principle and gradient tree boosting (S+GB) with respect to the square error of S is further detailed in Figure 2. There is a high density of trips whose S square prediction error is higher than the square prediction error of S+GB, showing once again that combining the snapshot method with machine learning method can lead to better predictions. In addition, it

is interesting to note that the improvement is not restricted to large traveling times, which is likely to indicate the existence of outliers.

On a side note, the vertical stripes visible for small prediction errors is due to the typical measurement acquisition frequency of 20 seconds. Because of that, the snapshot method typically make an estimation error that is a multiple of 20, leading to these patterns.

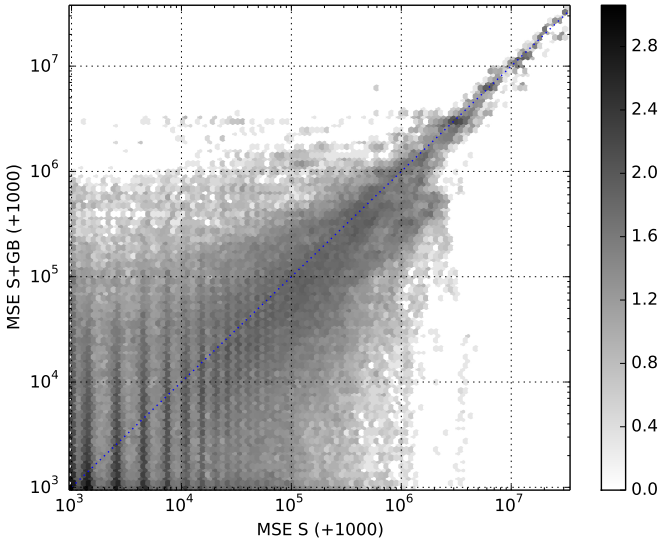

Figure 2: The density of test trips as a function of the square prediction error of S and S+GB shows that the prediction errors of S on many trips is above 10^6 while smaller than 10^6 for S+GB. All scales are logorithmic, and square errors have been increased by 1000 for plotting purposes.

Influence of the trip length. Figures 3 and 4 present the RMSE and MARE as function of the trip length (number of stations on the route), respectively. For all methods, the RMSE increases as the trip-length goes up (from [83,96] to [1070,1296]). In contrast, the MARE decreases as trip-length increases, indicating that the error remains *proportionally* constant, regardless of the increase in trip-length. Moreover, methods that are optimizing the absolute error (GBLAD and S+GBLAD) perform better, as expected.

When observing the MARE (in Figure 4), we notice that the snapshot predictor behaves as the rest of the methods. In other words, the last bus that traveled through a long trip predicts the current trip as good as the last bus that traveled through a short trip. Again, this is in contrast to our hypothesis from Section 5. Lastly, we observe that Machine Learning methods that comprise of the snapshot predictor improve over the same methods excluding the snapshot feature.

Single-segment prediction. Figures 5 and 6 shows the evolution of the prediction error and the relative prediction error (respectively), over the duration of a single segment. The horizontal axis of those figures corresponds to the order of segments in the trip, i.e. in a trip that goes between stations ω_k and ω_l, with $l > k$, $index = 1$ is the segment $\langle \omega_k, \omega_{k+1} \rangle$. In other words,

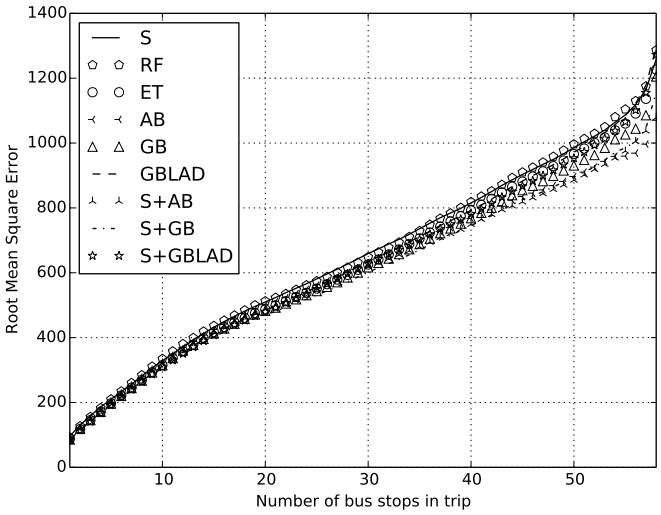

Figure 3: The mean square error on the duration of the whole trip increases with the trip length. S+GB is the best method for all trip lengths except for length [2,3], [44,54] and [55,58] where respectively S+GBLAD, S+AB and AB are more accurate.

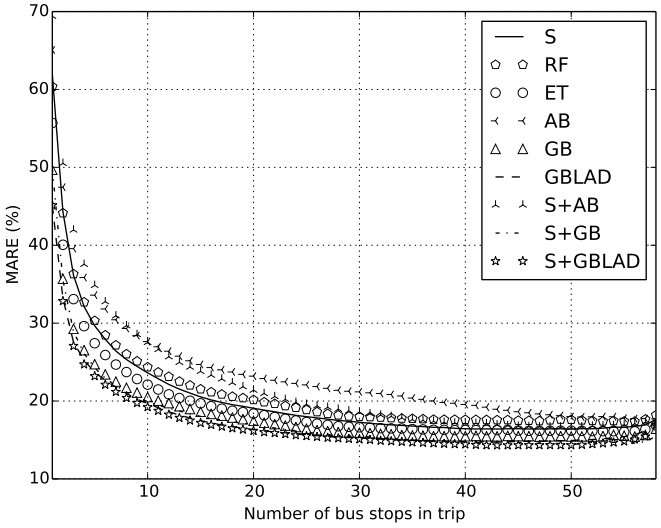

Figure 4: The mean absolute relative error on the duration of the whole trip decreases with the trip length. S+GBLAD is the best method for all trips length except for trips of length 1, where GBLAD is more accurate, and trips of length 52 and [55,58], where S+GB is more accurate.

Table 3: Accuracy of the prediction of the trip length, for the different methods tested over all trips

	S	RF	ET	AB	GB	GBLAD	S+AB	S+GB	S+GBLAD
RMSE	539	539	519	512	508	520	504	**494**	514
MARE (%)	23.37	24.11	22.05	27.08	20.46	19.38	26.32	19.95	**19.06**
MdARE (%)	16.15	16.37	15.23	18.05	13.84	13.86	16.84	**13.53**	13.65

these figures contain the incremental estimation error as number of segments per trip increases.

According to Figure 6 the relative prediction error for all methods remain unchanged as the index of a segment increases, however vary towards the longest trips. For the longer trips, the value of the error grows significantly. We suspect that this occurs either due to the small test set size for trips of these lengths (only 111 samples for trips of 58 stations), or due to a deterioration of the prediction accuracy for lengthier trips. It can be observed that S+GBLAD, which is the combination between the snapshot predictor and the GBLAD learning algorithm, is the most accurate method *per segment*, as well as per trip (Figure 4).

Figure 5 presents a decrease in the RMSE as function of the segment index (further segments have lower errors), with an unexpected increase towards the end. Intuitively, one may expect the accuracy of the snapshot principle, and maybe other regression methods, to decrease with the segment index (far segments). Indeed, segments further away in the trips have a larger time interval between the moment the journey we consider will enter the segment and the time the bus whose travel time is used as a prediction went through it.

This phenomena does not occur due to a difference between the nature of segments, but due to outliers, as can be seen in Figure 7. The figure contrasts box plots of the square estimation error (left) and the root mean square estimation error (right) for the snapshot method and for all segments at a given index in a trip. The drops in the curve of the RMSE correspond to the disappearance of outliers. Outliers are arranged in horizontal lines, because a single outlier in the journey log may affect several trips, i.e. an outlier in the segment $\langle \omega_i, \omega_{i+1} \rangle$ will affect the first segment of the trip whose source is ω_i, the second segment of the trip whose source is w_{i-1} etc.

Figure 5 does not allow to analyze the effect of the segment index on the precision of the snapshot method. However, Figure 7 contains the median of the square estimation error for this method, which is presented in the box-plots (at the lower part of the figure). One may observe that the median is stable with little variation in its values. Hence, the index of the segment does not influence the median of the square estimation error for the snapshot-based methods.

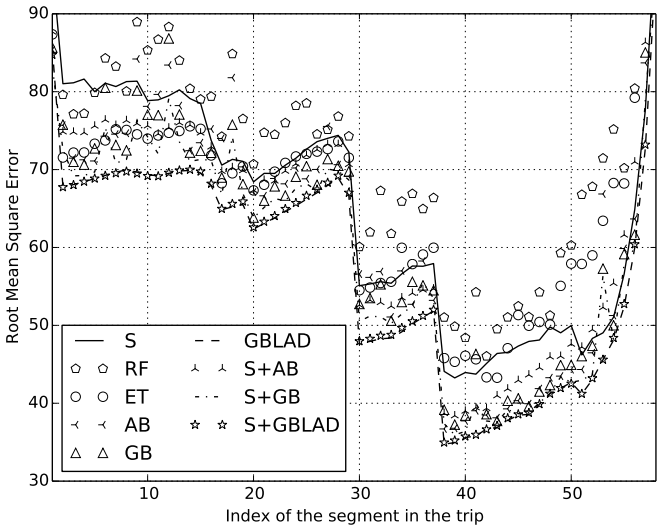

Figure 5: The mean square error on the duration of a single segment

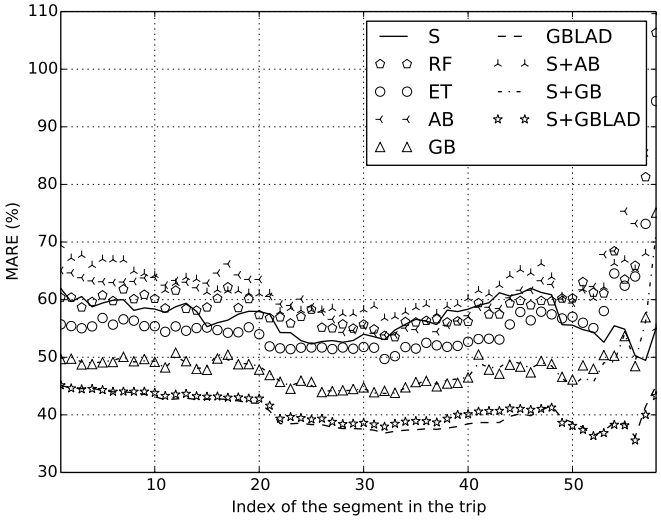

Figure 6: The mean absolute relative error on the duration of a single segment

8. Conclusion

In this work we presented a novel approach towards predicting travel time in urban public transportation. Our approach is based on segmenting the travel time into station-based segments, and combining the use of Machine Learning and Queueing Theory predictors to reduce the effect of outliers. Our empirical analysis

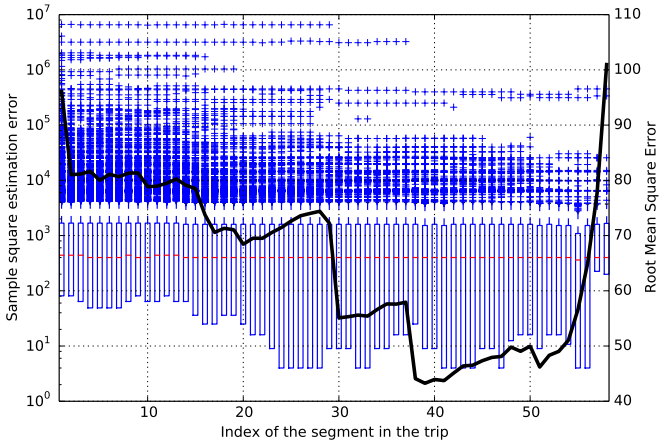

Figure 7: Comparing the boxplots of sample errors (left) to the MSE (right) of the snapshot method as functions of the index of the segment shows that the big drops in the MSE curve are due to the largest outliers disappearing. In particular, look at indexes 30, 38 and 15 to 18.

confirms that the combination of methods indeed improves performance. Moreover, we observe that the snapshot predictor is, counter-intuitively, unaffected by the length of a journey. This leads to positive evidence in favor of applying mixed Queue and Machine Learning predictors in similar settings.

In future work, we intend to extend our methods to support prediction for multi-modal transportation. Also, the mutual support of methods from Queueing Theory and Machine Learning require further investigation to unlock its full potential. Furthermore, to check the appropriateness of the snapshot predictors as function of car traffic, we intend to add data that comes from a real-time coordinated adaptive traffic system (SCATS) of Dublin [27]. The system, along with many other functionalities, counts traffic intensity through junctions (in real-time), and records these counts in an event log. We believe that the combination of the two data sets will enable an improvement of prediction, and lead to a better understanding of the root-cause for the errors in our methods.

Acknowledgment

This work was supported by the EU INSIGHT project (FP7-ICT 318225).

References

[1] C.-H. Wu, J.-M. Ho, D.-T. Lee, Travel-time prediction with support vector regression, Intelligent Transportation Systems, IEEE Transactions on 5 (4) (2004) 276–281.

[2] S. I.-J. Chien, Y. Ding, C. Wei, Dynamic bus arrival time prediction with artificial neural networks, Journal of Transportation Engineering 128 (5) (2002) 429–438.

[3] Bus arrival time prediction at bus stop with multiple routes, Transportation Research Part C: Emerging Technologies 19 (6) (2011) 1157 – 1170. doi:http://dx.doi.org/10.1016/j.trc.2011.01.003.
URL http://www.sciencedirect.com/science/article/pii/S0968090X11000155

[4] Y. Bin, Y. Zhongzhen, Y. Baozhen, Bus arrival time prediction using support vector machines, Journal of Intelligent Transportation Systems 10 (4) (2006) 151–158.

[5] A. Shalaby, A. Farhan, Prediction model of bus arrival and departure times using avl and apc data, Journal of Public Transportation 7 (1) (2004) 41–62.

[6] H. Chang, D. Park, S. Lee, H. Lee, S. Baek, Dynamic multi-interval bus travel time prediction using bus transit data, Transportmetrica 6 (1) (2010) 19–38.

[7] A. T. Baptista, E. Bouillet, F. Calabrese, O. Verscheure, Towards building an uncertainty-aware personal journey planner, in: Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, IEEE, 2011, pp. 378–383.

[8] M. Sinn, J. W. Yoon, F. Calabrese, E. Bouillet, Predicting arrival times of buses using real-time gps measurements, in: Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on, IEEE, 2012, pp. 1227–1232.

[9] T. Van Woensel, N. Vandaele, Modeling traffic flows with queueing models: a review, Asia-Pacific Journal of Operational Research 24 (04) (2007) 435–461.

[10] G. F. Newell, Applications of queueing theory, Tech. rep. (1982).

[11] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining - predicting delays in service processes, in: M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, J. Horkoff (Eds.), Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings, Vol. 8484 of Lecture Notes in Computer Science, Springer, 2014, pp. 42–57. doi:10.1007/978-3-319-07881-6.
URL http://dx.doi.org/10.1007/978-3-319-07881-6

[12] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining for delay prediction in multi-class service processes, Tech. rep. (2014).

[13] C. Sutton, M. I. Jordan, Bayesian inference for queueing networks and modeling of internet services, The Annals of Applied Statistics 5 (1) (2011) 254–282. doi:10.1214/10-AOAS392.
URL http://dx.doi.org/10.1214/10-AOAS392

[14] C. A. Sutton, M. I. Jordan, Inference and learning in networks of queues, in: Y. W. Teh, D. M. Titterington (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010, Vol. 9 of JMLR Proceedings, JMLR.org, 2010, pp. 796–803.
URL http://www.jmlr.org/proceedings/papers/v9/sutton10a.html

[15] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[16] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Mining resource scheduling protocols, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, Vol. 8659 of Lecture Notes in Computer Science, Springer, 2014, pp. 200–216. doi:10.1007/978-3-319-10172-9.
URL http://dx.doi.org/10.1007/978-3-319-10172-9

[17] W. Whitt, Stochastic-process limits: an introduction to stochastic-process limits and their application to queues, Springer, 2002.

[18] R. Ibrahim, W. Whitt, Real-time delay estimation based on delay history, Manufacturing and Service Operations Management 11 (3) (2009) 397–415. arXiv:http://msom.journal.informs.org/content/11/3/397.full.pdf+html, doi:10.1287/msom.1080.0223.
URL http://msom.journal.informs.org/content/11/3/397.abstract

[19] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and regression trees, Wadsworth International (1984).

[20] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[21] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[22] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning 63 (1) (2006) 3–42.

[23] H. Drucker, Improving regressors using boosting techniques, in: ICML, Vol. 97, 1997, pp. 107–115.

[24] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of Statistics (2001) 1189–1232.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel,

M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[26] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[27] B. McCann, A review of scats operation and deployment in dublin.