EURO Journal on Transportation and Logistics 9 (2020) 100006

Contents lists available at ScienceDirect

EURO Journal on Transportation and Logistics

journal homepage: www.journals.elsevier.com/euro-journal-on-transportation-and-logistics

Travel speed prediction based on learning methods for home delivery

Maha Gmira [a,c,d], Michel Gendreau [a,d], Andrea Lodi [a,c,d], Jean-Yves Potvin [b,d,*]

[a] Département de mathématiques et de génie industriel, Polytechnique Montréal, C.P. 6079, Succ. Centre-Ville, Montréal, Québec, H3C 3A7, Canada
[b] Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, H3C 3J7, Canada
[c] Chaire d'excellence en recherche du Canada sur la science des données pour la prise de décision en temps réel, Polytechnique Montréal, C.P. 6079, Succ. Centre-Ville, Montréal, Québec, H3C 3A7, Canada
[d] Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Québec, H3C 3J7, Canada

A B S T R A C T

The travel time to proceed from one location to another in a network is an important consideration in many urban transportation settings ranging from the planning of delivery routes in freight transportation to the determination of shortest itineraries in advanced traveler information systems. Accordingly, accurate travel time predictions are of foremost importance. In an urban environment, vehicle speeds, and consequently travel times, can be highly variable due to congestion caused, for instance, by accidents or bad weather conditions. At another level, one also observes daily patterns (e.g., rush hours), weekly patterns (e.g., weekdays versus weekend), and seasonal patterns. Capturing these time-varying patterns when modeling travel speeds can provide an immediate benefit to commercial transportation companies that distribute goods, since it allows them to better optimize their routes and reduce their environmental footprint.

This paper presents the first part of a project aimed at optimizing time-dependent delivery routes in an urban setting. It focuses on the prediction of travel speeds using as input GPS traces of commercial vehicles collected over a significant period of time. The proposed algorithmic framework is made of a number of macro-steps where different machine learning and data mining methods are applied. Computational results are reported on real data to empirically demonstrate the accuracy of the obtained predictions.

1. Introduction

Travel speed prediction is a major issue in most urban transportation models, whether they address the transportation of people or goods. Most variations in travel speeds, and hence in travel times, result from traffic congestion, which can be classified as recurrent, due to well-known patterns, and non-recurrent, due to accidents, construction, emergencies, special events and bad weather, among others. Accordingly, there is a need for accurate predictions of travel speeds (times) under recurrent and non-recurrent congestion. It is important to note that better predictions may significantly help individuals and transportation companies operating in urban areas in planning their activities, and possibly lead to substantial reductions in actual travel times and greenhouse gas emissions.

With the increasing amount of available data collected from probe vehicles, smart phone applications, and other location technologies, the challenge in travel speed prediction is no longer related to the quantity of data, but rather to the modeling and extraction of useful information from such data. In this context, there is a great potential for models and algorithms based on real data that can accurately predict travel speeds at various times of the day for different types of streets and roads.

The research presented in this paper is part of an integrated project aimed at capturing predictable patterns in travel speeds for commercial vehicles in the city of Montreal with the objective of optimizing time-dependent routes for a fleet of delivery vehicles. This paper focuses on making the best travel speed predictions possible using data collected from mobile electronic devices. The data used in this study come from a software development company that produces vehicle routing algorithms to plan the home delivery of large items (appliances, furniture) to customers. This partner has delivery routes with more than 2,500,000 delivery points, serviced by nearly 200,000 routes. Data are collected using automatic vehicle location systems where GPS receivers are interfaced with Global System for Mobile communications (GSM) modems. The system records point locations as latitude-longitude pairs, in addition to instantaneous speed, date and time.

From a methodological standpoint, the problem-solving approach is made of a number of algorithmic ingredients that realize the following macro-steps:

* Corresponding author. Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, Québec, H3C 3J7, Canada.
E-mail address: potvin@iro.umontreal.ca (J.-Y. Potvin).

https://doi.org/10.1016/j.ejtl.2020.100006
Received 28 January 2020; Accepted 15 February 2020

1. Data preparation and representation
2. Size reduction and clustering
3. Missing data imputation
4. Prediction

The literature for addressing those macro-steps in the context of speed prediction is not unified and leads to a number of possible algorithmic choices. Thus, the best choices had to be determined with special attention to the context and to our specific data. In other words, our contribution is tailored to the data: we had to solve the problem for the company that provided us with the data, as any other data-driven predictive or prescriptive application. Although the algorithmic ingredients are not really novel, the contribution of the paper lies in their effective combination to create a methodology, as defined by the four macro-steps above, that is general for speed prediction and works well on our dataset. The algorithmic ingredients resulting in the best performances may not be the same for different datasets, but this does not affect the methodology itself. We expect other researchers and practitioners to follow such a methodology and potentially augment it with additional macro-steps if their datasets require so.

The rest of the paper is organized as follows. We first review the literature related to our work in Section 2. Then, our methodology for travel speed prediction is reported. The creation of a database of speed patterns from GPS traces is described in Section 3. Then, techniques to reduce the size of the database and cluster arcs into similarity classes are explained in Section 4. This is followed in Section 5 by a description of the neural network model used for travel speed prediction. Computational results are reported in Section 6 on a number of real-world instances. Finally, a framework for a real-time delivery system that integrates the neural network model is proposed in Section 7. The conclusion follows.

2. Literature review

Most urban traffic control systems rely on short-term traffic prediction and a huge literature has been published on this topic in the last decades due, in particular, to the advent of intelligent transportation systems. Given that these systems are highly dependent on accurate traffic information, they must collect a large amount of data (locations, speeds and individual itineraries).

Travel speed prediction at a given time typically relies on historical travel speed values and a set of exogenous variables. Methods to predict traffic information are classified in (Van Hinsbergen et al., 2007) as 1) naive (i.e., without any model assumption), 2) parametric, 3) non-parametric and 4) a combination of the last two, called hybrid methods. The first three methods are described in the following.

2.1. Naive methods

Naive methods are by far the easiest to implement and to use because they do not require an underlying model. However, one main drawback is their lack of accuracy. In (Van Hinsbergen et al., 2007), naive methods are divided into instantaneous methods (based on data available at the instant the prediction is performed), historical methods (based on historical data) and mixed methods, where the latter combine characteristics of historical and instantaneous methods. As a baseline for comparison with other parametric methods, the work reported in (Smith et al., 2002) uses a simple hybrid method where the travel speed forecast is a function of the current traffic flow rate, as well as its historical average at a given time of the day and day of the week. In (Wu et al., 2004), the authors compare two methods for travel time prediction, one based on data available at the instant the prediction is performed and one based on historical data. Using the Relative Mean Error (RME) and the Root Mean Squared Error (RMSE), these two approaches showed similar performance, but were clearly outperformed by a more sophisticated approach called Support Vector Regression (see Section 2.3).

2.2. Parametric methods

Parametric methods use data to estimate the parameters of a model, whose structure is predetermined. The most basic model is linear regression, where the traffic variable V_t to be predicted at a given time t is a linear function of independent variables:

$$V_t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n \tag{1}$$

Different techniques are available to estimate the parameters β_i, $i = 1,\ldots, n$. Among parametric methods, we consider the Autoregressive Moving Average model (ARMA) and its Integrated variant (ARIMA), different smoothing techniques and the Kalman filter. They are presented below.

2.2.1. Kalman filter

The Kalman filter is a very popular short-term traffic flow prediction method. It allows the state variables to be updated continuously, which is very important in time-dependent contexts. Some works related to traffic state estimation are based on the original Kalman filter (Kalman and Bucy, 1961), as well as its extension for non-linear systems called the Extended Kalman Filter (EKF) (Julier and Uhlmann, 1997). The latter is particularly relevant since the travel times depend on traffic conditions that are highly non-linear and dynamic, changing over time and space. In (Wang and Papageorgiou, 2005), a freeway state estimator is obtained by solving a macroscopic traffic flow model with EKF. A new EKF based on online-learning is used in (Van Lint, 2008) to provide travel time information on freeways. Also, a dynamic traffic assignment model, which is a non-linear state-space model, is solved by applying three different extensions of the Kalman filter (Antoniou et al., 2007).

In (Jula et al., 2008), the authors use traffic data on California highways to predict travel times on arcs and estimate the arrival time at a destination. First, travel times on arcs are predicted by feeding the Kalman filter with historical data. Then, this prediction is corrected and updated with real time information using the Kalman filter's corrector-predictor form.

2.2.2. ARMA

To predict short-term traffic characteristics such as speed, flow or travel time, ARMA time series models have been widely used. The ARIMA(p',q) model combines p' autoregressive terms AR and q moving average terms MA, also known as the orders of the AR and MA components. If we consider travel time prediction, the general formulation of the ARMA(p',q) model is:

$$T(t) - \sum_{i=1}^{p'} \alpha_i T(t-i) = \varepsilon_t + \sum_{j=1}^{q} \beta_j \varepsilon_{t-j} \tag{2}$$

where the travel time $T(t)$ at departure time t is a linear function of the travel times at previous instants, $\varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_{t-q}$ are noise variables andα_i, β_j are parameters, $i = 1,\ldots p'; j = 1,\ldots, q$.

The ARIMA(p,d,q) model generalizes ARMA by addressing non-stationary time series. It has non-negative integer parameters p, d, and q, where p and q correspond to the AR and MA orders, and d is the degree of differencing (i.e., the number of times the data are differenced). To obtain the ARIMA(p,d,q) model, we first start with the ARMA(p',q) model, which can be rewritten as:

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) T(t) = \left(1 + \sum_{j=1}^{q} \beta_j L^j\right) \varepsilon_t \tag{3}$$

where $L^i T(t) = T(t-i)$ is the lag operator. In particular $LT(t) = T(t-1)$. Assuming that the polynomial on the left-hand side of Equation (3) has a unit root (i.e., factor $(1 - L)$) of multiplicity d, that is:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)=\left(1-\sum_{i=1}^{p'-d}\alpha_i L_i\right)(1-L)^d \tag{4}$$

we obtain the ARIMA(p,d,q) model by setting $p = p' - d$:

$$\left(1-\sum_{i=1}^{p}\alpha_i L^i\right)(1-L)^d T(t)=\left(1+\sum_{j=1}^{q}\beta_j L^j\right)\varepsilon_t \tag{5}$$

In (Hamed et al., 1995), the authors use the Box-Jenkins approach (Box and Jenkins, 1976) to develop a forecasting model based on ARIMA from data collected in urban streets (i.e., 1-min traffic-volume on each street during peak periods). After comparing several ARIMA models, the one of order (0,1,1) yielded the best results in terms of traffic volume forecasts. The authors in (Smith et al., 2002) compare a seasonal ARIMA model, called SARIMA, with a non-parametric regression model where the forecast generation method and neighbor selection criteria are heuristically improved. The tests showed that SARIMA performed better than the improved non-parametric regression.

In (Williams and Hoel, 2003), the authors model traffic flow with SARIMA (1,0,1) and SARIMA(0,1,1) models. Another SARIMA model is reported in (Guo, 2005) to forecast traffic conditions over a short-term horizon, based on 15-min traffic flow data. In this case, SARIMA outperformed the k-Nearest Neighbor method (k-NN), described in Section 2.3.3.

2.3. Non-parametric methods

Non-parametric methods include non-parametric regression and different types of neural networks. Non-parametric methods are also known as data-driven methods, because they need data to determine not only the parameters but also the model structure. Thus, the number and types of parameters are unknown a priori. The main advantage of these methods is that they do not require expertise in traffic theory, although they need a lot of data.

The most popular non-parametric methods for traffic prediction are the support vector machine, neural networks and non-parametric regression.

2.3.1. Support vector machine

The Support Vector Machine (SVM) was first introduced in (Vapnik, 1999; Vladimir, 1995) and was used in many classification and regression studies. SVM is popular because it guarantees global minima, it deals quite well with corrupted data and works for complex non-linear systems. Support Vector Regression (SVR) is the application of SVM to time-series forecasting.

In (Wu et al., 2004), the authors analyze the application of SVR for travel time prediction. They used traffic data, obtained from an Intelligent Transportation System, over a five weeks period. The first four weeks correspond to the training set and the last week corresponds to the testing set. Using a Gaussian kernel function and a standard SVR implementation, their method improved the Relative Mean Error (RME) and the Root Mean Squared Error (RMSE) when compared to instantaneous and historical travel time prediction methods. Due to its promising results, SVR has been used to predict traffic parameters such as traffic flow or travel time. However, the classical SVR, like the one used in (Wu et al., 2004), cannot be applied in real time because it requires a complete model training each time a new record (data) is added. There are also some variants of SVM for traffic prediction. In (Wang and Shi, 2013), the authors report a hybrid model called the chaos-wavelet analysis SVM that overcomes the need to choose a suitable kernel function. In the context of traffic flow prediction for a large-scale road network (Yang et al., 2014), SVM parameters are optimized by a parallel genetic algorithm, thus yielding a Genetic Algorithm-Support Vector Machine (referred to as GA-SVM).

2.3.2. Neural networks

When considering data-driven methods, neural networks are among the best for traffic forecasting because of their ability to learn non-linear relationships among different features without any prior assumption.

The most widely used neural networks are called Multi-Layer Perceptrons (MLPs). They are typically made of an input layer, one hidden layer and an output layer, where each layer contains one or more units (neurons). The units in the input layer are connected to those in the hidden layer, while the units in the hidden layer are connected to those in the output layer. The weights on these connections are adjusted during the learning process using (input, target output) example pairs, so as to produce an appropriate mapping between the inputs and the target outputs. In (Chen et al., 2001), an MLP is used to predict traffic flow from input data (speed, flow, occupancy) collected by detection devices on a highway around the city of London. In this application, the MLP and a radial basis function network performed better than all ARIMA models considered. In (Vlahogianni et al., 2005), the authors exploit a genetic algorithm to fine tune the parameters and the number of hidden units in an MLP. Their model showed better generalization abilities when tested with new inputs. More recently, the authors in (Habtie et al., 2017) report the performance of an MLP with 15 hidden units, trained with the Levenberg-Marquardt backpropagation algorithm. Based on different error performance indicators, the MLP showed good accuracy when predicting road speeds. In (Moniruzzaman et al., 2016), an MLP is used to predict the time needed to cross the Ambassador bridge, one of the busiest bridges at the Canada-US border. A database of GPS records for a full year was used to train and test the neural network.

As indicated in (Ma et al., 2015), Recurrent Neural Networks (RNNs) are better suited for traffic forecasting tasks due to their ability to account for sequential time-dependent data. Typically, the signal sent by the hidden layer to the output layer at some time t is also sent back to the hidden layer. This signal is processed with the input signal at time $t+1$ to determine the internal state of the hidden layer. This internal state acts as a memory and remembers useful time-dependent relationships among data.

A specific class of RNNs, called Long Short-Term Memory network (LSTM), is now widely used in the literature due to its proven ability to learn long-term relationships in the data. In (Ma et al., 2015), LSTM is compared to various RNNs and other statistical methods, namely: Elman neural network, non-linear autoregressive with exogenous inputs (NARX) neural network, Time-Delay Neural Network (TDNN), SVM, ARIMA and Kalman filter. The LSTM achieved the best performances in terms of accuracy and stability. The work in (Tian and Pan, 2015) proposes an LSTM model for short-term traffic flow prediction, which is compared with random walk, SVM, MLP and Stacked AutoEncoder. The results showed the superiority of LSTM in terms of prediction accuracy, ability to memorize long-term historical data and generalization capabilities. In (Fu et al., 2016), LSTM and a neural network model made of Gated Recurrent Units (GRUs) (Cho et al., 2014) are applied to traffic data in California. The study showed that GRUs behave slightly better than LSTM, while both outperformed an ARIMA model in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE).

2.3.3. Non-parametric regression

Another class of non-parametric methods is called Non-Parametric Regression (NPR). It is suitable for short-term traffic prediction since it can deal with the uncertainty in traffic flows. The objective of NPR is to estimate a regression function without relying on an explicit form, unlike the traditional parametric regression models (where the model parameters are estimated). The forecasting ability of NPR relies on a database of past observations. It applies a search procedure to find observations in this database that are similar to the current conditions. Then, it transfers these observations to the forecast function to estimate the future state of the system.

record identifier	latitude	longitude	speed	mobile identifier	driver identifier	date time

Fig. 1. Record structure of a GPS point.

The k-Nearest Neighbor (k-NN) is a widespread class of non-parametric regression methods made of two components:

(i) Search procedure: the nearest neighbors (historical data most similar to the current input) are the inputs to the forecast function aimed at generating an estimate. The nearest neighbors are found using a similarity measure, which is usually based on the Minkowski distance metric:

$$L_r = \left(\sum_{i=1}^{n} |p_i - q_i|^r \right)^{1/r} \tag{6}$$

where n is the vector dimension, p_i is the i^{th} element of the historical record currently considered, q_i is the i^{th} element of the current input, and r is a parameter with values between 1 and 2. The most common implementation uses a sequential procedure to find the nearest neighbors. However, as the number of historical observations increases, the sequential search becomes very time consuming.

(ii) Forecast function: the most general approach to generate a prediction is to compute an average of the dependent variable values over the nearest neighbors. However, the neighbors closer to the current input should probably have more impact on the forecast.

The authors in (Davis and Nihan, 1991) were among the first to use NPR to estimate short-term traffic flows. Their work highlighted the importance of a large and representative dataset. NPR was then applied to estimate traffic volumes from two sites located in Northern Virginia Capital Beltway, based on five months of observations (Smith, 1995). The results showed that the proposed method can generate more accurate predictions than the other tested approaches (including a neural network model). The work in (Smith and Demetsky, 1997) compares historical averages, time series, back-propagation neural networks and non-parametric regression models using a performance index that includes absolute error, error distribution, ease of model implementation and model portability. Overall, NPR proved to be better than the other models and was also easier to implement.

The interested reader is referred to (Ermagun and Levinson, 2018) for a more exhaustive review on traffic forecasting. The following sections will now describe the macro-steps outlined in the introduction to produce travel speed predictions from a huge historical data base. Together, these macro-steps define a novel methodology, centered on data, that none of the references above presents in the form that we propose here.

3. Data preparation and representation

Our industrial partner provided us with one year and a half of GPS data transmitted by mobile devices installed in delivery vehicles (extending over the years 2013, 2014 and 2015). These GPS points were first mapped to the underlying Montreal metropolitan community network to generate daily speed patterns for each individual arc, where a daily speed pattern for a given arc is made of 96 average speeds taken over time intervals of 15 min.

In the following, the main issues related to the derivation of speed patterns from GPS traces are briefly discussed.

3.1. Data cleaning

The record of each GPS point contains an identifier, a latitude-longitude pair, an instantaneous speed, a mobile identifier, a driver identifier, a date and a time stamp (Fig. 1).

arc identifier	date	day	season	00:00AM	.....	11:45PM

temporal characteristics | 96 avg. speed values

Fig. 2. Database structure after geomatic analysis.

The available data had first to be cleaned by deleting GPS points with aberrant speeds (values less than 0 km/h or greater than 150 km/h), aggregates of GPS points associated, for example, with parking stops for deliveries, etc. Then, a map-matching algorithm was applied to the remaining GPS points, as described below.

3.2. Map-matching algorithm

Due to the relatively low accuracy of GPS systems, assigning a GPS point to an arc of the underlying network is a difficult problem, particularly in dense urban road networks. Thus, a good map-matching algorithm is required. To this end, we used a recent algorithm reported in (Hashemi and Karimi, 2016) which was slightly adapted to our context. The algorithm works as follows:

Step 1. Identification of trips. A total of 170 and 327 different vehicle and driver identifiers were found over all GPS points. Within a single day, it is possible to find one driver associated with one vehicle, one driver associated with two vehicles or more, and two drivers or more associated with one vehicle. Thus, there are clearly different trips within a day, where a trip corresponds to a vehicle/driver pair.

Step 2. For each trip, all GPS points associated with it are considered for assignment to arcs of the network. This is done in three main phases:

2.1 In the initialization phase, candidate arcs are those that are adjacent to the three nearest nodes of the first GPS point. A score is calculated for each arc based on a weighted sum of distance and heading difference with the GPS point. Then, the arc with the best score is selected and a confidence level is calculated to account for the uncertainty of that choice (due to inherent uncertainty in positioning sensors and digital maps). Here, the confidence level is based on the difference between the score of the selected arc and the second best score. Clearly, a larger difference implies a higher confidence level. If the confidence level is above a given threshold, the GPS point is assigned to the selected arc, otherwise it is skipped and the next GPS point is considered. Once a GPS point is assigned to the first arc, the same-arc phase starts.

2.2 In the same-arc phase, the next GPS point is assigned to the same arc than the previous one, unless some conditions are not satisfied anymore (e.g., when crossing an intersection). In the latter case, the algorithm switches to the next-arc phase.

2.3 In the next-arc phase, candidate arcs are those connected to the previously selected arc plus those that are adjacent to the three nearest nodes of the current GPS point. A score is calculated for each arc based on a weighted sum of three criteria: distance and heading difference with the current GPS point, plus direction difference between the arc and the line connecting the previous GPS point to the current one. Again, the arc with the best score is selected, its confidence level is calculated, and topological constraints are checked (e.g., arc not connected to the previous one, turn restrictions, etc.). Depending on the confidence level and satisfaction of the topological constraints, the current GPS point can either be skipped or assigned to the new arc. In the latter case, the algorithm returns to the same-arc phase. It should be noted that considering arcs that are close to the current GPS point, but not necessarily connected to the previous arc, allows

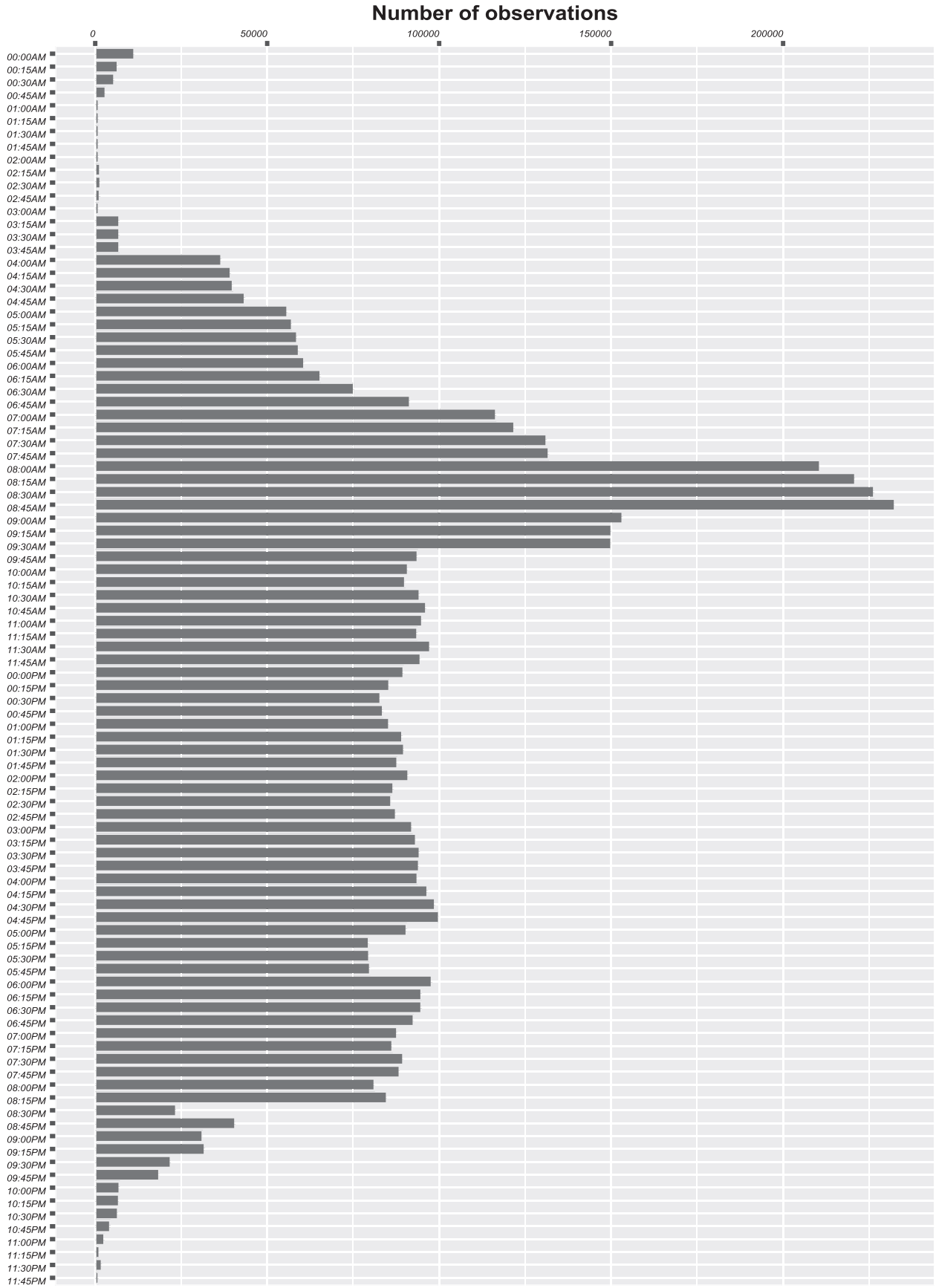

Fig. 3. Number of observations per time interval of 15 min.

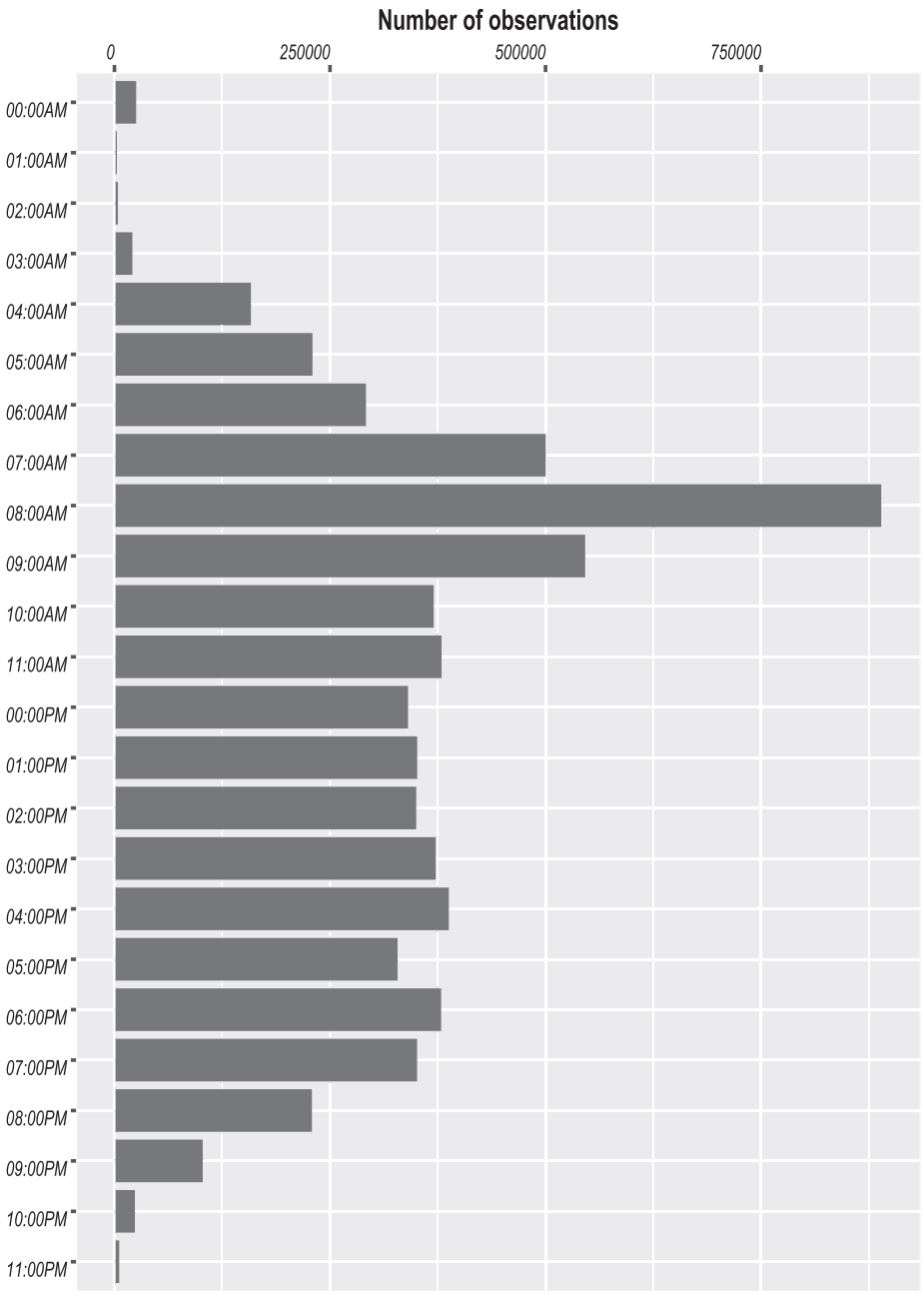

Fig. 4. Number of observations per time interval of 1 h.

the algorithm to account for discontinuities in GPS traces due to obstacles (e.g., tunnels, bridges).

The interested reader is referred to (Hashemi and Karimi, 2016) for details about this algorithm.

When the assignment of GPS points to arcs is completed for each day, average travel speeds can be calculated over time slots of 15 min. Thus, a new database is obtained where each record has the structure shown in Fig. 2. The next section will now explain how this database was exploited to fit our purposes.

4. Size reduction and clustering

Due to the size of our database, size reduction techniques were applied. After eliminating speed patterns and hours with too many missing data, a "prediction-after-classification" approach was used to cluster arcs with similar speed patterns into classes before predicting travel speed values. This is explained in the following.

4.1. Database reduction

With 233,914 arcs and 515 days in the database, we have a total of $233{,}914 \times 515 = 120{,}506{,}910$ speed patterns. Originally, the database was constructed with time intervals of 15 min. That is, a speed pattern for a given arc on a given day is made of 96 average speed values taken over time intervals of 15 min, thus covering an horizon of 24 h. Furthermore, the speed limit was stored when no observation was recorded within a 15-min time interval.

An elimination procedure was first applied to get rid of speed patterns or time intervals with too few data, as described below.

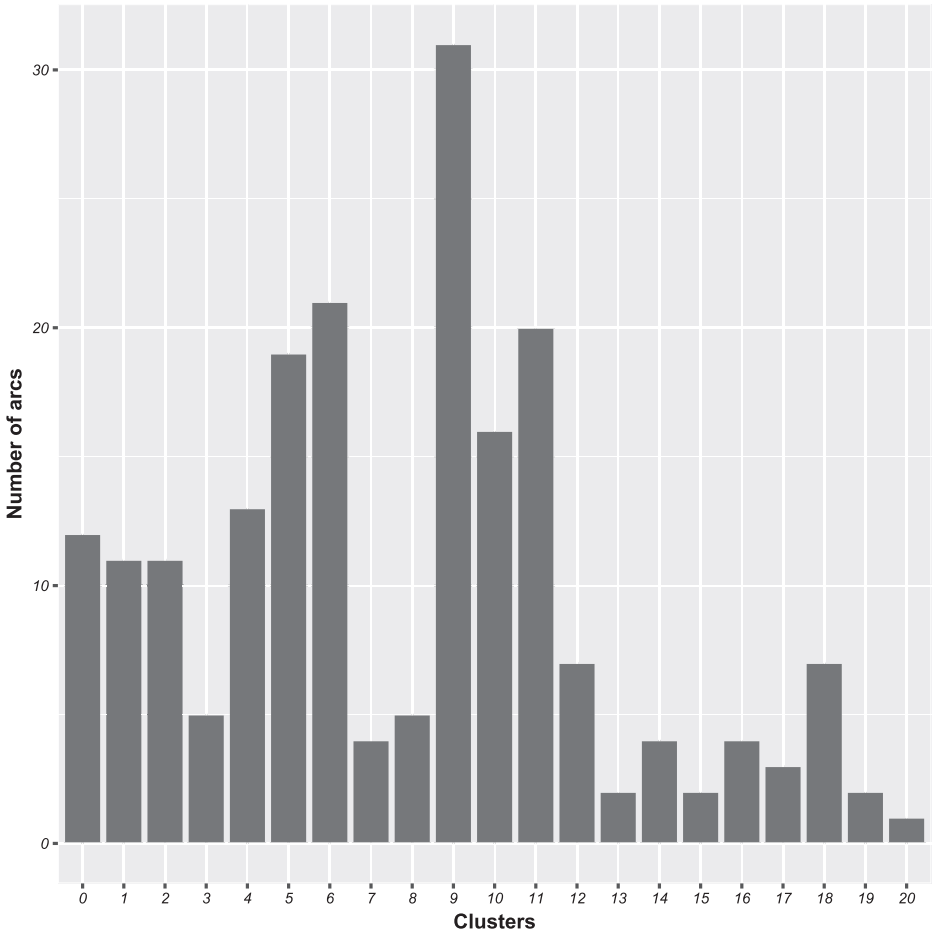

Fig. 5. Number of arcs in each class generated by AP.

(a) *Speed patterns.* To keep only average speed values based on real observations, the time-independent speed limits were removed from the database. Given that a significant proportion of the resulting speed patterns now contained missing data, a speed pattern was automatically discarded when the proportion of real average speed values over the 96 time intervals was less than 5% (note that this threshold is often suggested in the literature). In other words, a speed pattern with only 4 average speeds or less was eliminated. Through this process, we ended up with a total of 6,667,459 speed patterns. It should be noted that only 3485 arcs still had at least one representative speed pattern in this database. The fact that the original GPS traces have been collected from delivery routes in particular sectors of an urban area explains this large reduction in the number of arcs and speed patterns. Finally, the 96 time intervals of 15 min of each remaining speed pattern were aggregated into 24 1-h time intervals, to allow the calculation of average speed values based on more observations in each time interval, see Figs. 3 and 4.

(b) *One-hour time intervals.* After reducing the number of speed patterns in the database, as well as the number of average speed values stored in a pattern, we then examined more closely the 1-h time intervals.

Clearly, there are intervals with no or very few observations over the entire database, like night hours (although some observations can be found in the database because the vehicles are sometimes moved during the night from one location to another). Accordingly, we discarded hours where the proportion of real average speed values over the 6,667,459 speed patterns in the database was under 5%. After this elimination process, the number of 1-h time intervals was reduced from 24 to 13. More precisely, only 1-h time intervals starting from 7:00 a.m. to 7:00 p.m. were kept in every speed pattern.

4.2. Clustering

When this step is reached, we have a database of 6,667,459 speed patterns, where each pattern is associated with a given arc and a given date. A speed pattern can be seen as a vector of 13 average speed values, one for each 1-h time interval between 7:00 a.m. and 7:00 p.m. This number of speed patterns exceeds the capacity of a learning-based prediction algorithm, so we had to group arcs with similar patterns into a number of classes. For this purpose, an average speed pattern was calculated for each arc over all its corresponding speed patterns in the database. Since 3485 arcs are represented in the database, this led to $N = 3485$ average speed patterns. Then, a hierarchical clustering method was developed using first the *K*-means algorithm to get a first classification of arcs, followed by affinity propagation to get the final classes. This hierarchical scheme had to be devised because the number of average speed patterns was too large for a direct application of the affinity propagation algorithm. This novel clustering approach is described in the following.

4.2.1. K-means

To cluster arcs based on their average speed pattern with the K-means algorithm, the distance between two patterns was calculated using the Euclidean metric (see, for example (MacQueen, 1967)). At the end, *K* cluster centroids (classes) were obtained and the average speed pattern of each arc was assigned to the closest centroid. Since the number of classes must be fixed in advance, the latter was purposely set to a large value,

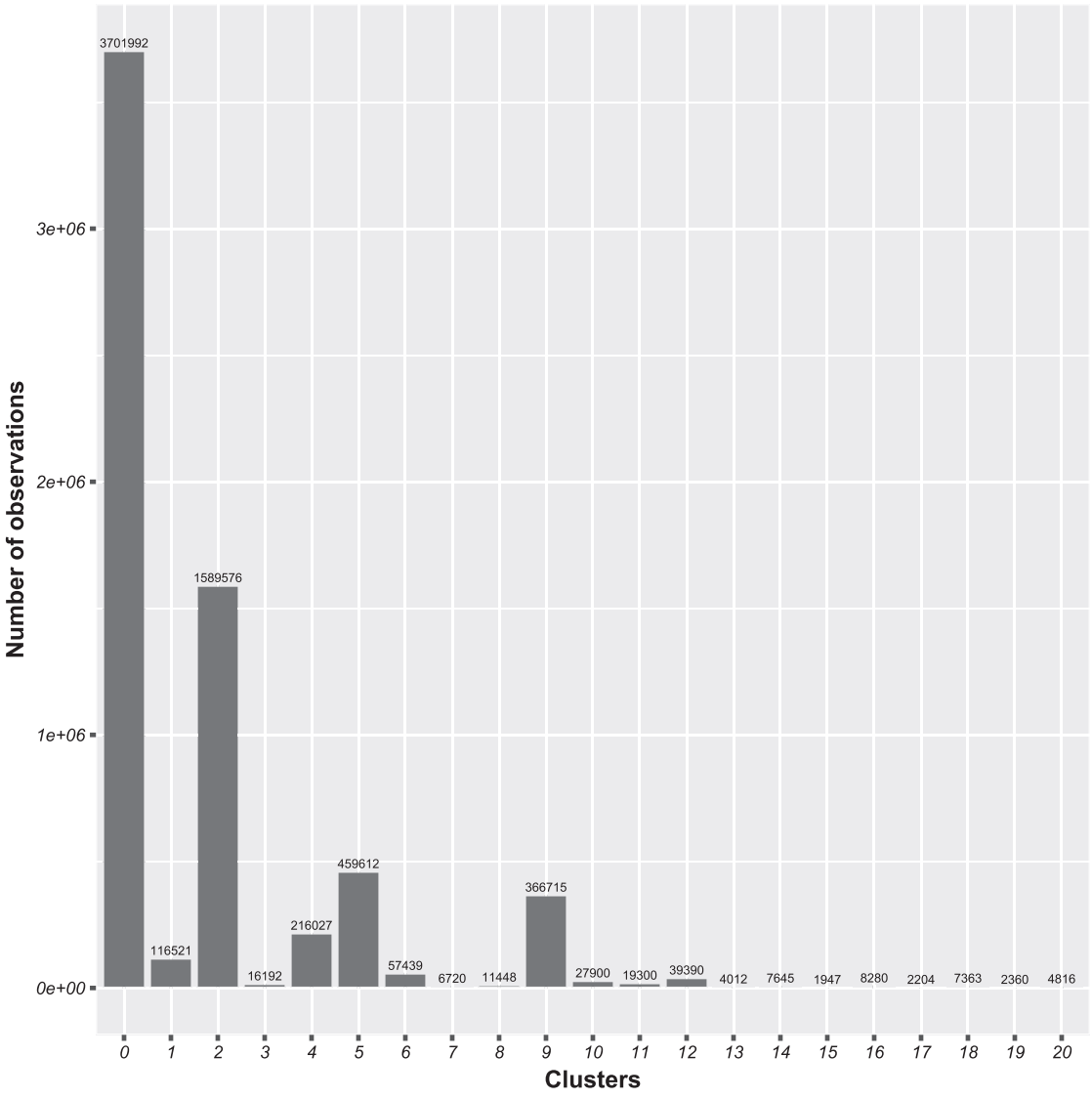

Fig. 6. Number of observations in each class generated by AP.

i.e., $K = 200$. Then, the output of the *K*-means algorithm was fed to the affinity propagation algorithm to further reduce the number of classes (see Section 4.2.2).

The problem solved by the K-means algorithm is summarized below, where C_k stands for class k. The objective is to minimize the sum of the squares of the Euclidean distance between speed pattern x_i of arc i and its closest centroid μ_k, over all arcs. In this objective, the variables are the μ_k's.

$$Min \sum_{k=1}^{K} \sum_{x_i \in C_k} \| x_i - \mu_k \|^2 \tag{7}$$

where:

$$C_k = \{ x_i : \| x_i - \mu_k \| = \min_{l=1,\ldots,K} \| x_i - \mu_l \| \} \tag{8}$$

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \tag{9}$$

An iterative method known as Lloyd's algorithm was used to converge to a local minimum, given that solving the problem exactly is NP-hard. In this algorithm, starting from K randomly located centroids, the following two steps are performed repeatedly until convergence is observed: 1) Cluster assignment: construct the set of classes by assigning each arc to the cluster centroid that is closest to the arc's speed pattern and 2) Update centroids: update the centroid of each cluster by averaging over all speed patterns assigned to it.

4.2.2. *Affinity propagation algorithm*

The Affinity Propagation (AP) algorithm is a clustering procedure proposed in (Frey and Dueck, 2007). As opposed to *K*-means, every data point is considered as a potential centroid. Through the propagation of so-called affinity values among pairs of data points, which reflect the current affinity (or consent) of one data point to consider the other data point as its centroid, some data points accumulate evidence to be centroids. The reader will find in (Frey and Dueck, 2007) the exact mathematical formulas that are used to guide the transmission of affinity values and the accumulation of evidence in data points. At the end, evidence is located only on a certain number of data points that are chosen as cluster centroids. Then, the set of classes is constructed by assigning each data point to its closest centroid. It should be noted that the centroids necessarily correspond to data points and that the number of classes does not need to be fixed in advance. That is, the number of classes will

automatically emerge as the algorithm unfolds. The authors in (Frey and Dueck, 2007) also show that AP approximately minimizes the sum of the squares of the Euclidean distance between each data point and its assigned centroid.

AP was applied using the centroids of the 200 classes produced by *K*-means as data points. At the end, these 200 classes were aggregated into 21 different classes, labeled from 0 to 20. Figs. 5 and 6 show the number of arcs and observations, respectively, in each class.

We also tested another clustering algorithm, known as the Mean Shift Algorithm (Fukunaga and Hostetler, 1975), but since it did not perform as well as AP for the two evaluation metrics proposed in the next section, namely the Silhouette coefficient and the Calinski-Harabasz score, we will omit its description.

4.3. Evaluation metrics

After clustering the arcs into 21 classes, the quality of these classes was evaluated using two well-known metrics, namely the Silhouette coefficient and the Calinski-Harabasz score.

The Silhouette coefficient (Rousseeuw, 1987) associates a value between −1 and 1 with each data point. It can be interpreted as follows: if the coefficient is close to 1, then the data point is associated with the right cluster; if it is close to 0, then it lies between two clusters; and if it is close to −1, then the point is not associated with the right cluster. Assuming that a data point corresponds to the average speed pattern associated with an arc, this coefficient is calculated as follows:

- intra-class: for every arc i, calculate the average distance between its speed pattern and the speed pattern of all other arcs in the same cluster; we call this value a_i.
- inter-class: for every arc i, calculate the average distance between its speed pattern and the speed patterns of all arcs in the cluster with the closest centroid; we call this value b_i.
- the Silhouette coefficient s_i of arc i is then:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{10}$$

At the end, the Silhouette coefficient is the average of those s_i coefficients over all arcs.

The Calinski-Harabasz score (Calinski and Harabasz, 1974) is another measure that provides a ratio between intra-class and inter-classes dispersion values. The clusters are better defined when the score is higher. The score is computed as follows:

$$CH^{(K)} = \frac{Tr\left(B^{(K)}\right)}{Tr\left(W^{(K)}\right)} \cdot \frac{N-K}{K-1} \tag{11}$$

where

$$W^{(K)} = \sum_{k=1}^{K} \sum_{x \in C_k} \left(x - c^{(k)}\right)\left(x - c^{(k)}\right)^{\mathrm{T}} \tag{12}$$

$$B^{(K)} = \sum_{k=1}^{K} n_k \left(c^{(k)} - c\right)\left(c^{(k)} - c\right)^{\mathrm{T}} \tag{13}$$

In these equations, a vector should be viewed as a column. Also, K is the number of clusters or classes, N is the number of speed patterns (arcs), $W^{(K)}$ is the intra-cluster dispersion matrix, $B^{(K)}$ is the inter-cluster dispersion matrix, $Tr(M)$ is the trace of matrix M, C_k is the k-th class of speed patterns (arcs) of cardinality n_k, $c^{(k)}$ is the centroid of speed patterns in C_k and c is the centroid of all speed patterns.

Note that each entry (i,j) in matrices $W^{(K)}$ and $B^{(K)}$ corresponds respectively to:

$$W_{ij}^{(K)} = \sum_{k=1}^{K} \sum_{x \in C_k} \left(x_i - c_i^{(k)}\right)\left(x_j - c_j^{(k)}\right) \tag{14}$$

$$B_{ij}^{(K)} = \sum_{k=1}^{K} n_k \left(c_i^{(k)} - c_i\right)\left(c_j^{(k)} - c_j\right) \tag{15}$$

When the Silhouette coefficient was evaluated on the 21 classes produced by AP, a value of 0.85 was obtained. This is quite good, given that 1 is the best possible value. It should be noted that the coefficient for the Mean Shift algorithm was equal to 0.67. Concerning the Calinski-Harabasz score, a value of 10.52 was obtained by AP, which is to be compared with 3.39 for the Mean Shift algorithm.

By focusing on the four classes with the largest number of observations, namely classes 0, 2, 5 and 9, we noted that the average speeds of classes 0, 2 and 5 are very different, ranging from approximately 25 to 45 km/h. The average speed of class 9 is similar to the one of class 2, but it does not evolve in the same way over the day. When we looked at more detailed data, we observed that the arcs of class 9 are not affected by the congestion observed during weekdays. That is, as opposed to classes 0, 2 and 5, there is no significant difference between the weekday average speeds and the weekend average speeds. Thus, the clustering algorithm was successful in identifying classes of arcs with different characteristics.

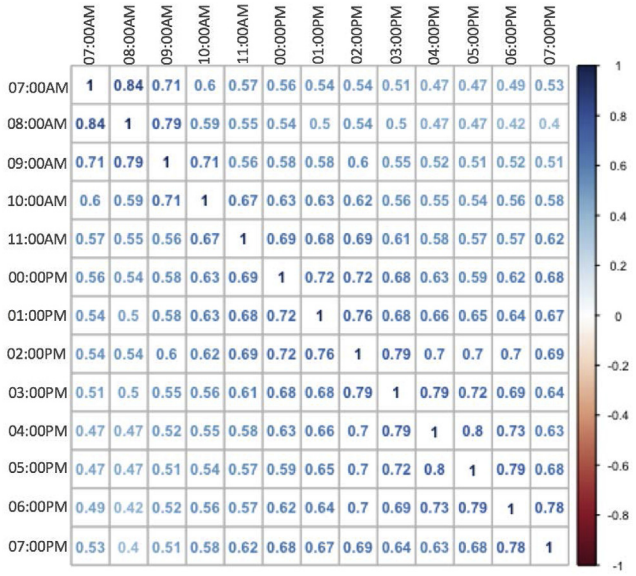

Fig. 7. Correlation matrix of travel speeds.

5. Speed prediction

This section describes the supervised neural network model for predicting travel speeds based on the classes generated by the AP clustering algorithm. Since a data missing issue emerges in this context, we will first explain how this problem is handled. Then, the neural network model will be described.

5.1. Missing data

Given that input vectors for the neural network model are obtained by averaging speeds over all arcs in a class produced by our clustering methodology, there is no missing data in the input (see Section 6.2). However, each target output vector corresponds to one of the 6,667,459 speed patterns in the database. Thus, it is likely for a target output to have one or more missing values.

To handle missing values, we must input plausible estimates drawn from an appropriate model. In this process, the following variables will be accounted for: day (Monday, Tuesday, …, Saturday, Sunday), season (Spring, Summer, Fall, Winter), the arc's class label, and most

importantly, the average speed in each time interval which corresponds to the variables with missing values. Different Multiple Imputation (MI) methods will be applied (Rubin, 2004). These methods generate multiple copies of an incomplete database and replace the missing values in each replicate with estimates drawn from some imputation method. An analysis is then performed on each complete database and a single MI estimate is calculated for each missing value by combining the estimates from the multiple complete databases (Little and Rubin, 2014; Rubin, 2004). The methods considered here are: Multivariate Imputation via Chained Equation (MICE) (Buuren and Groothuis-Oudshoorn, 2011), missForest (which relies on a Random Forest imputation algorithm (Stekhoven and Bühlmann, 2011)) and Amelia (Honaker et al., 2011).

5.1.1. MICE

This algorithm can be described as follows:

1. Perform a mean imputation for every missing speed value by setting it to the average over observed speeds in the same time interval.
2. Select the time interval variable with the largest proportion of missing speed values.
3. Select the explanatory variables from those with a correlation greater than 0.5 with the selected time interval variable (see, e.g., the correlation matrix of travel speeds in Fig. 7).
4. Perform linear regression.
5. Replace the missing speed values for the selected time interval with estimates obtained from the regression model. If this time interval is subsequently used as an explanatory variable in the regression model of other time interval variables, both observed and imputed values are used.
6. Repeat steps 2 to 4 for the remaining time intervals with missing values.
7. Repeat the entire procedure for a number of iterations to obtain multiple estimates or imputations for each missing value.

Since each iteration in step 7 produces an estimate or imputation for each missing value, the number of iterations corresponds to the number of imputations for each missing value. At the end, these multiple imputations are averaged to obtain a single estimate for each missing value.

5.1.2. Random forest

The Random Forest (RF) algorithm (Breiman, 2001) is a machine learning technique that does not require the specification of a particular regression model. It has a built-in routine to handle missing values by weighting the observed values of a variable using a matrix of proximity values, where proximity is defined by the proportion of trees in which pairs of observations share a terminal node. It works as follows:

1. Replace missing values by the average over observed values in the same time interval.
2. Repeat until a stopping criterion is satisfied:
   (a) Using imputed values calculated so far, train a random forest.
   (b) Compute the proximity matrix.
   (c) Using proximity as a weight, impute missing values as the weighted average over observed values in the same time interval.

The algorithm stops when the difference between the new imputed values and the old ones increases for the first time. Note that, by averaging over multiple random trees, this method implicitly behaves according to a multiple imputation scheme. The RF algorithm was implemented using the randomForest R-package (Liaw and Wiener, 2002).

5.1.3. Amelia

Amelia imputes missing data based on different bootstrapped samples drawn from the database (bootstrapped data samples correspond to smaller samples of the same size that are repeatedly drawn, with

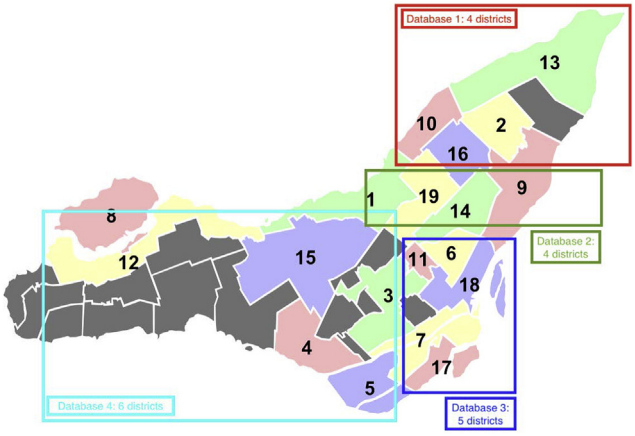

Fig. 8. Test instances.

replacement, from a single original sample, see (Efron and Tibshirani, 1994)). It basically applies the Expectation Maximization (EM) method (Dempster et al., 1977) to find the maximum likelihood estimates for the parameters of a normal distribution. It works as follows:

1. *M* bootstrap samples are drawn from the data base.
2. An estimation of the mean and variance of the distribution is calculated for each one of the *M* samples with EMB (EM with bootstrapping). Thus, we have *M* different distributions at the end of this step.
3. An estimate for each missing value in the database is produced with each distribution.

Thus, *M* different imputations are available for each missing value in the data base at the end. These imputations are then averaged to obtain a final estimate for each missing value.

The complete databases produced by MICE, RF and Amelia are used to provide target outputs during the training and testing phases of our neural network model. The accuracy of the travel speed predictions made by the neural network will be compared for the three imputation methods considered.

5.2. LSTM

In this section, we briefly describe the supervised neural network model used to predict travel speeds, once the missing values in the database have been replaced by imputed ones (either using MICE, RF or Amelia). The neural network model is a Long Short-Term Memory network (LSTM). This choice was motivated by the ability of the LSTM to handle sequential data and capture time dependencies. First introduced in (Hochreiter and Schmidhuber, 1997), this special type of Recurrent Neural Network (RNN) alleviates the vanishing gradient problem (Pascanu et al., 2013) (when the gradient of the error function becomes too small with respect to a given weight, the latter cannot change anymore). It is made of an input layer, a variable number of hidden layers and an output layer. Each hidden layer is made of memory cells that store useful information from past input data. Memory cells in a given hidden layer send signals at time *t* to the memory cells in the next hidden layer (or the units in the output layer, if last) but also to themselves. This recurrent

Table 1
Test instances.

Instance	# Arcs	# Patterns
DB1	749	1,132,978
DB2	902	1,125,695
DB3	1178	2,737,257
DB4	656	1,671,529

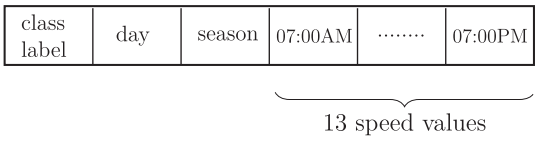

Fig. 9. Input vector I.

Fig. 10. Input vector II.

signal is used by the memory cells to determine their internal state at time $t+1$. Thus, there are connection weight matrices from the input to the first hidden layer, from each hidden layer to itself and to the next hidden layer and from the last hidden layer to itself and to the output layer. Furthermore, memory cells in a given layer have three gates: one for the signal sent from the previous layer, one for the signal sent by the memory cells to themselves and one for the signal sent by the memory cells to the next layer. Gates can be seen as filters that regulate the signals by allowing some parts of it to be blocked (or forgotten). Like the weight matrices mentioned above, the gates have weights that are updated during the learning process. The interested reader will find more details about the LSTM network model in (Greff et al., 2017).

In the next section, computational results obtained with LSTM and comparisons with alternative approaches are reported.

6. Computational study

In this section, we first define the four test instances used in the computational study and describe the input and output vectors of the LSTM. Then, we present the fine tuning of the LSTM hyperparameters on each instance before reporting the prediction results. At the end, a study based on the whole Montreal road network is presented.

Our LSTM was implemented in Python 3.5. The hyperparameter tuning experiments were performed on a Dell R630 server with two Intel Xeon E5-2650V4 of 12 cores each (plus hyperthreading) and 256GB of memory. The server also has 4 NVIDIA Titan XP GPUs with 3,840 CUDA cores and 12GB of memory. However, our code was limited to only 4 cores and one GPU from the server. To obtain more computational power, the LSTM results reported on the whole road network of Montreal were obtained on the Cedar cluster of Compute Canada. We requested 6 cores with Intel E5-2650v4 processors, 32GB of RAM and 1 NVIDIA P100 GPU with 12GB of memory.

6.1. Test instances

To perform the computational study, we used four test instances associated with different sectors of Montreal, as illustrated in Fig. 8. They are denoted as DB1, DB2, DB3 and DB4, where DB stands for database. The size of these instances in terms of number of arcs represented in the database and number of speed patterns is reported in Table 1.

6.2. Input and output vectors

The input vector for the neural network was first designed as illustrated in Fig. 9. Each vector is associated with a class of arcs produced by AP and is made of: the class label, the day (Monday, Tuesday, …, Saturday, Sunday), the season (Spring, Summer, Fall, Winter) and 13 average speeds over all arcs in the corresponding class, that is, one speed value for each 1-h time interval starting from 7:00 a.m. to 7:00 p.m. The target output vector corresponds to a speed pattern among the 6,667,459 available speed patterns, where the missing values are filled with one of the three imputation methods of Section 5.1. Obviously, the target output vector must come from an arc of the same class, and for the same season and day than the vector provided in input. We should also note that the speed values in the patterns were normalized using the scikit-learn object (Pedregosa et al., 2011).

Unfortunately, the results obtained with this approach were unsatisfactory. To better exploit the capabilities of LSTM to handle sequential data, we turned to input vector II shown in Fig. 10. Here, input vector I with 13 speed values is transformed into 13 input vectors II, each with a single speed value. That is, rather than providing at once the whole speed pattern for 1-h time intervals starting from 7:00 a.m. to 7:00 p.m., a sequence of 13 input vectors from 7:00 a.m. to 7:00 p.m. is provided, where each input vector contains a single speed value. The target output vector is modified accordingly and also contains a single speed value taken from an arc of the same class, and for the same season, day and hour than the vector provided in input.

Table 2
Hyperparameter values.

Hyperparameter	Values
Hidden layers	1,2,3,4,5
Units in each hidden layer	1,5,10,15,20,25,30,35,40,45,50
Batch size	10,20,30,40,50,80,100
Training epochs	5,10,20,30,40,50
Learning algorithm	SGD, RMSprop, Adagrad Adadelta Adam, Adamax, Nadam
Activation function	softmax, softplus, softsign, relu, tanh sigmoid, hard sigmoid, linear
Weight initialization	uniform, lecun uniform, normal glorot normal, glorot uniform, he normal, he uniform

6.3. Hyperparameter tuning

Each database instance was divided into a training set (80% of the total) and a testing set (20% of the total), where the latter is made of the most recent observations. Apart from the connection weights, which are adjusted through learning, a neural network model also relies on a number of hyperparameters that must be set before learning takes place. The following were considered:

- Number of hidden layers;
- Number of units in each hidden layer;
- Batch size: Number of training examples provided to the neural network before updating the connection weights;
- Training epochs: Number of passes through the set of training examples;
- Learning algorithm: Algorithm used during the training phase to adjust the weights;
- Weight initialization: Method used to set the initial weights, see (Glorot and Bengio, 2010) for details;
- Activation function: Function used to compute the internal state of a unit from the signals it receives.

A good parameter setting for a neural network has a huge impact on its results, as discussed in (Bergstra et al., 2011). To determine an appropriate combination of the above hyperparameters, different search strategies can be used, in particular grid search and random search (Snoek et al., 2012). In grid search, a systematic evaluation of all possible combinations of parameter values is performed. Random search is much less computationally expensive, given that only a sample of all possible combinations is considered (100 combinations, in our case). Due to the curse of dimensionality, the superiority of random search over grid search is known for high-dimensional parameter spaces (Bergstra and Bengio, 2012). We applied both methods on our LSTM. The values tested

Table 3
Hyperparameter tuning.

Hyperparameter	DB1		DB2		DB3		DB4	
	G	R	G	R	G	R	G	R
Hidden layers	1	1	1	1	1	1	2	2
Units in each hidden layer	10	10	15	15	10	10	5	5
Batch size	10	10	10	10	10	10	10	10
Training epochs	10	10	10	10	10	10	10	10
Learning algorithm	Adam	Adam	Adam	Adam	Adam	Adamax	Adamax	Adamax
Activation function	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid tanh	sigmoid tanh
Weight initialization	normal	normal	normal	normal	normal	normal	uniform	uniform

Table 4
Results with different imputation methods on instance DB3.

Method	# Imput.	MAE	RMSE	Time (min)
MICE	5	4.00	2.85	91
	10	4.00	2.83	178
	15	3.99	2.72	480
	20	3.89	2.64	917
RF	5	6.20	4.19	110
	10	6.12	4.02	182
	15	5.92	3.63	382
	20	5.73	3.61	1018
AMELIA	5	7.29	6.38	150
	10	7.20	6.03	220
	15	6.55	6.05	370
	20	6.02	5.74	500

Table 5
Results on the four instances using MICE with 5 imputations.

Instance	MAE	RMSE	Time (min)
DB1	4.11	2.99	76
DB2	5.86	4.05	83
DB3	4.00	2.85	91
DB4	4.00	2.67	70

for each hyperparameter are shown in Table 2. The final settings obtained by each search method on each database are shown in Table 3, where G and R stand for grid search and random search, respectively.

We first note that the best settings produced by grid search and random search are the same, except for the learning algorithm of DB3, where we decided to go with Adam. Overall, when comparing the best settings for each instance, we observe some differences but also a number of similarities. For example, the activation function is always sigmoid, except for DB4 where two hidden layers have produced the best result, with sigmoid used for the first hidden layer and the hyperbolic tangent tanh for the second one. Also, the best learning algorithm is either Adam or Adamax (a variant of Adam). Adam and Adamax were introduced in (Kingma and Ba, 2015) and proved to be particularly appropriate for complex neural network structures and large datasets, as detailed in (Ruder, 2016).

6.4. LSTM results

Here, we measure the accuracy of the travel speed predictions produced by our LSTM tuned and trained over each database instance. The root mean squared error and the mean absolute error are used to measure the accuracy, where:

$$RMSE = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(y_l - \hat{y}_l)^2} \tag{16}$$

Table 6
Comparison of alternative models.

Test instance	Model	Metric	Input vector I	Input vector II
DB1	LSTM	RMSE	**4.60**	**2.99**
		MAE	**6.61**	**4.11**
	MLP	RMSE	6.26	4.24
		MAE	8.84	6.71
	SVR	RMSE	6.22	3.62
		MAE	8.01	6.11
	k-NN	RMSE	6.93	4.01
		MAE	9.10	7.21
DB2	LSTM	RMSE	**5.27**	**4.05**
		MAE	7.30	5.86
	MLP	RMSE	6.96	5.00
		MAE	7.29	5.16
	SVR	RMSE	6.58	4.71
		MAE	**7.27**	**5.00**
	k-NN	RMSE	7.65	5.86
		MAE	9.06	7.84
DB3	LSTM	RMSE	**4.39**	**2.85**
		MAE	**6.21**	**4.00**
	MLP	RMSE	6.09	4.05
		MAE	8.26	6.32
	SVR	RMSE	6.15	3.49
		MAE	7.87	6.55
	k-NN	RMSE	7.02	5.95
		MAE	7.32	5.11
DB4	LSTM	RMSE	**3.82**	**2.67**
		MAE	**6.16**	**4.00**
	MLP	RMSE	6.32	4.26
		MAE	7.94	6.21
	SVR	RMSE	6.35	3.74
		MAE	7.14	4.38
	k-NN	RMSE	7.32	5.17
		MAE	8.94	7.61

Table 7
Hyperparameter tuning for the Montreal network instance.

Hyperparameter	Grid search	Random search
Hidden layers	3	3
Units in each hidden layer	20	20
Batch size	20	20
Training epochs	10	10
Learning algorithm	SGD	Adam
Activation function	sigmoid tanh tanh	sigmoid tanh tanh
Weight initialization	normal	normal

$$MAE = \frac{1}{L}\sum_{l=1}^{L}|y_l - \hat{y}_l| \tag{17}$$

In these equations, L is the number of (input, target output) pairs in the training or testing set, where the target output corresponds to an observed speed pattern. In pair l, y_l stands for the observed speed pattern

Table 8
RMSE and MAE metrics with different imputation methods.

Method	# Imput.	MAE	RMSE	Running time (min)
MICE	5	13.71	12.91	429
	10	**12.84**	**10.84**	450
	15	12.52	10.32	1117
	20	12.39	10.09	1900
RF	5	13.84	13.28	512
	10	13.02	11.38	649
	15	12.94	11.08	1640
	20	12.68	11.03	2000
AMELIA	5	17.91	17.53	550
	10	17.63	16.97	580
	15	16.76	16.10	940
	20	16.49	16.07	1500

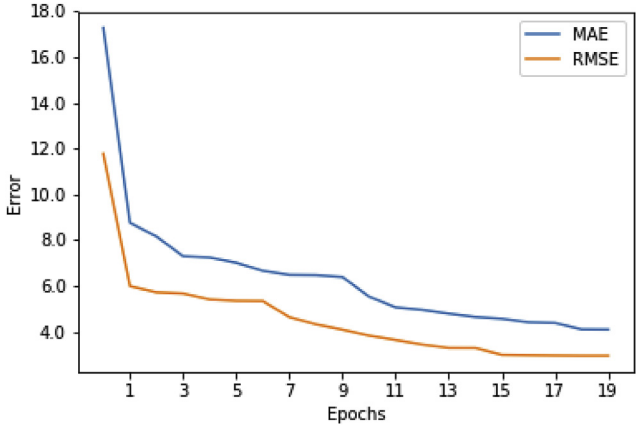

Fig. 11. Evolution of MAE and RMSE during the training phase for instance DB3.

and $\hat{y}_l$ for the speed pattern produced by the neural network for input vector l. RMSE and MAE are two error functions typically used to evaluate how close the output vectors produced by the neural network are to the target outputs. Due to the square in the RMSE formula, large errors have much more impact than small ones. On the other hand, MAE measures the relative error and is more robust to outliers since there is no square in its formula.

Table 4 reports the prediction errors of the trained LSTM in the testing phase for the largest instance DB3, based on the RMSE and MAE metrics, using the three different imputation methods and a variable number of imputations (i.e., 5, 10, 15, 20). Note that the authors in (Schafer and Olsen, 1998) show that 3 to 5 imputations yield good results. But, more recently, the authors in (Graham et al., 2007; Schafer and Graham, 2002) proposed 20 imputations. Thus, we chose to vary this number between 5 and 20. The results clearly show that MICE is the best imputation method (the same is observed for the three other instances). Increasing the number of imputations from 5 to 20 with MICE produces slightly better results, but the marginal improvement does not justify the additional computational cost. Thus, MICE with 5 imputations was chosen. Fig. 11 shows the evolution of the MAE and RSME metrics of LSTM during the training phase on instance DB3, using MICE with 5 imputations. We can see that both RMSE and MAE drop sharply at the beginning and then keep improving, although at a smaller pace, until convergence is observed after approximately 20 passes (epochs) over the training set. Fig. 12 then illustrates the differences between the observed speeds and the speeds predicted by the trained LSTM on instance DB3 using a sample of observations. The figure shows that, in most cases, the predicted speed values follow closely the observed ones. The MAE and RMSE errors on the four instances using MICE with 5 imputations are summarized in Table 5.

6.5. Comparison with other models

In this section, our LSTM is compared with three alternative approaches: Support Vector Regression, k-Nearest Neighbor regression and multi-layer perceptron.

6.5.1. k-Nearest Neighbor regression

As previously mentioned, k-Nearest Neighbor (k-NN) is a non-parametric method that relies on a similarity measure, based here on the Euclidean distance, between data points (Friedman et al., 2001). Assuming a database of (input, target output) data points and a new input data, the method picks the k closest input data points in the database and sets the prediction for the new input to the average of the corresponding output data points.

Parameter k has to be selected carefully. A large k value leads to a smoother fit and lower variance at the expense of a higher bias, and conversely for a small k value. We tested the values $k = \{2, 4, 6, 8, 12, 16, 20\}$ on each instance and found that an improvement is observed up to k

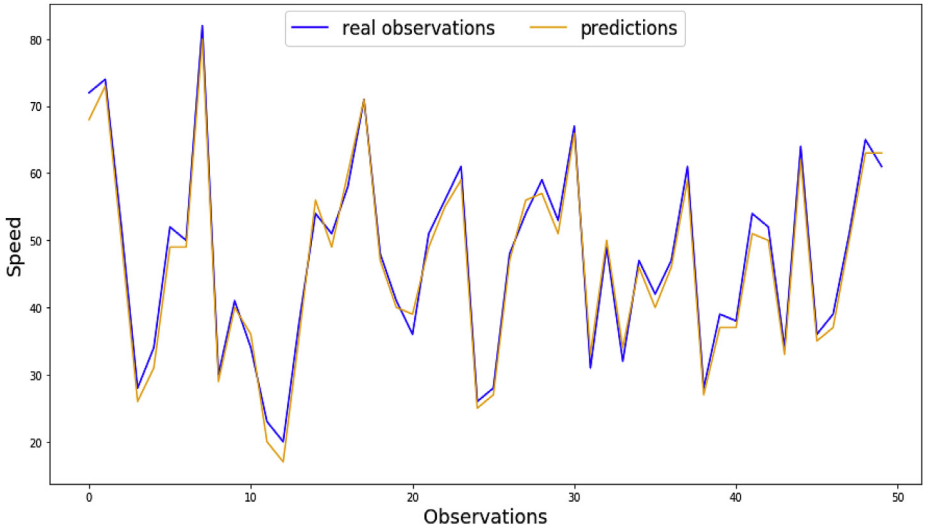

Fig. 12. Comparison between observed and predicted speed values in the testing phase for instance DB3.

Table 9
Comparison of alternative models.

Test instance	Model	Metric	Input vector I	Input vector II
DB0	LSTM	RMSE	**17.22**	**10.84**
		MAE	**22.86**	**12.84**
	MLP	RMSE	21.92	15.17
		MAE	29.57	17.40
	SVR	RMSE	22.85	17.03
		MAE	31.30	14.68
	k-NN	RMSE	25.00	18.51
		MAE	38.13	37.91

$= 8$, after which the error stabilizes. Thus, the results obtained with $k = 8$ are reported below.

6.5.2. Support Vector Regression

Given data points (x_i, y_i), $i = 1,\ldots,M$, where x_i is an input vector and y_i the corresponding output vector, the method known as Support Vector Regression (SVR) (Scholkopf and Smola, 2001) identifies (in the linear case) the hyperplane $w^T x + b$, where w and b are the weights and bias, respectively, that minimizes:

$$\frac{1}{2}||w||^2 + C\sum_{i=1}^{M} |y_i - (w^T x_i + b)|_\varepsilon \tag{18}$$

The first term in the summation involves the norm of the weight vector while the second term is the total penalty for a tolerance ε (i.e., the penalty is incurred only when the error exceeds ε). The constant C is a parameter that provides a tradeoff between the two terms. In the non linear case, kernel functions that map the input space to another feature space are also used in SVR to speed up the calculations. The following values for ε, C and the kernel function were explored through grid search for a total of $5 \times 11 \times 10 = 550$ combinations.

- Kernel functions: Linear Kernel, Polynomial Kernel, RBF Kernel, Sigmoid Kernel and Gaussian Kernel.
- $\varepsilon = 0, 0.1, 0.2, \ldots, 0.9, 1$
- $C = 10, 20, 30, 40, 50, 100, 500, 1000, 1500, 2000$

At the end, the best parameter values were the same on each instance and correspond to a SVR model based on the RBF Kernel with $\varepsilon = 0.1$ and $C = 1000$.

6.5.3. Multi-layer perceptron

The Multi-Layer Perceptron (MLP) is trained with the Levenberg-Marquardt algorithm (Sutskever, 2013). We tested MLP models with one to three hidden layers, using a variable number of units in the hidden layers. Only the results obtained with the best MLP model are reported below.

6.5.4. Results

The results for LSTM, MLP, SVR and k-NN on the four database instances are reported in Table 6, using again the RMSE and MAE metrics. Results with the two input structures are also reported to show the superiority of input vector II over input vector I, see Section 6.2. With regard to RMSE, LSTM provides a better prediction accuracy than the three other models on the four test instances. With regard to MAE, LSTM shows better results on three test instances out of four. It is outperformed by SVR and MLP on DB2.

6.5.5. Montreal network instance

To test the limits of our methodology, we also addressed the whole Montreal network with the complete database of 6,667,459 speed patterns. As for DB1, DB2, DB3 and DB4, the database was divided into a training set (80% of the total) and a testing set (20% of the total), where the latter is made of the most recent observations. The best hyperparameter values obtained in this case with grid search and random search are shown in Table 7. They are the same for both search types, except for the learning algorithm. Since Adam (or its variant Adamax) was previously used, we decided to go with Adam. Note that the activation function differs depending on the hidden layer considered: the sigmoid function is used for the first hidden layer while the hyperbolic tangent tanh is used for the second and third hidden layers.

Table 8 reports the prediction errors of the trained LSTM on the testing set, based on the RMSE and MAE metrics, using the three different imputation methods and a variable number of imputations (i.e., 5, 10, 15, 20). Again, MICE turned out to be best method, although RF was quite competitive. Since most of the improvement occurs between 5 and 10 imputations, MICE with 10 imputations was chosen.

Table 9 compares the trained LSTM with the three alternative prediction methods. After a grid search similar to the one described for DB1, DB2, DB3 and DB4, a value of $k = 12$ was chosen for k-NN, while SRV was again based on the RBF kernel with $\varepsilon = 0.1$ and $C = 1000$. Not surprisingly, LSTM produced the best predictions by a rather wide margin.

7. Real-time system

The work reported in this paper is a first step towards the development of a real-time system for the management of delivery routes. Fig. 13 depicts the general framework of such a system. As shown in the figure, the LSTM neural network model should first be extended to receive real-time data (sensor data, incident reports, weather reports, …), in addition to historical data, to make real-time speed predictions at a given time. These predictions can then be used to improve the current vehicle routes.

An incremental reoptimization scheme is envisioned where we first check if the current routes are still feasible with the new travel speeds. If they are, we just keep the solution as it is, while simply updating the arrival and departure times at each customer. Otherwise, we look for better paths to go from one customer to the next in the road network, based on the new travel speeds. If the solution remains infeasible, then we go for a true reoptimization. Since a home delivery application is motivating our work, the reoptimization procedure cannot exchange customers between routes, but the sequence of customers in each route can be modified to account for the new travel speeds. It should be noted that the solution may still be infeasible at the end of the reoptimization procedure, due to lateness at one or more customers or at the depot, but

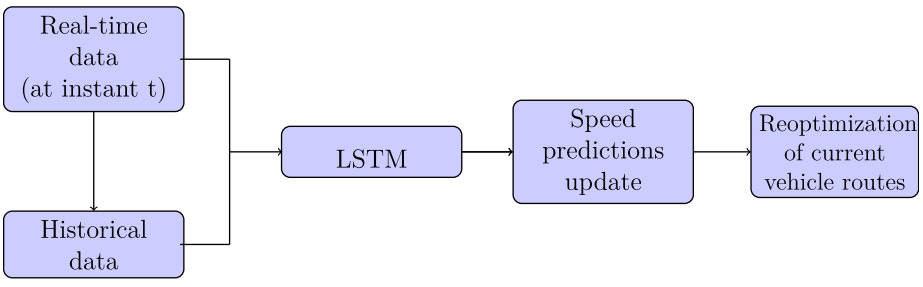

Fig. 13. Real-time delivery system.

this situation cannot be avoided in a context where the speeds are dynamic, unless customers can be canceled.

8. Conclusion

In this work, a methodology for predicting travel speeds exploits a large database of GPS traces collected from mobile devices installed inside delivery vehicles. The methodology features as macro-steps: (1) data preparation and representation, (2) size reduction and clustering of arcs into classes, using unsupervised learning, (3) missing data imputation and (4) speed prediction for the previously identified classes of arcs, using supervised learning.

In particular, the speed prediction macro-step was addressed with an LSTM neural network model whose parameters were adjusted with grid search. The neural network was then tested on four different instances and compared to three alternative prediction approaches, using two error metrics, where it turned out to be the best in all cases but one. LSTM was also the best when considering the whole Montreal road network. A natural extension of this work would be to include real-time data in the speed prediction process, with the ultimate goal of integrating the LSTM neural network model into a real-time system that would adjust the planned delivery routes according to the current status of the road network.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada through the Canada Excellence Research Chair in data science for real-time decision-making. Also, computing facilities were made available to us by Compute Canada. This support is gratefully acknowledged. Finally, we wish to thank Prof. Leandro C. Coelho and his team for their contribution to the analysis of the GPS data, as well as the three reviewers for their useful comments on the manuscript.

References

Antoniou, C., Ben-Akiva, M., Koutsopoulos, H.N., 2007. Nonlinear Kalman filtering algorithms for on-line calibration of dynamic traffic assignment models. IEEE Trans. Intell. Transport. Syst. 8 (4), 661–670.
Bergstra, J., Bengio, Y., 2012. Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305.
Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., 2011. Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems, vol. 24. Curran Associates, Inc., pp. 2546–2554
Box, G.E.P., Jenkins, G.M., 1976. Time Series Analysis: Forecasting and Control, Revised. Holden-Day.
Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.
S. Buuren and K. Groothuis-Oudshoorn. MICE: multivariate imputation by chained equations in R. J. Stat. Software, 45(3), 2011.
Calinski, T., Harabasz, J., 1974. A dendrite method for cluster analysis. Commun. Stat. 3 (1), 1–27.
Chen, H., Grant-Muller, S., Mussone, L., Montgomery, F., 2001. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. Neural Comput. Appl. 10 (3), 277–286.
Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conf. on Empirical Methods in Natural Language Processing Association for Computational Linguistics, pp. 1724–1734 arXiv:1406.1078.
Davis, G.A., Nihan, N.L., 1991. Nonparametric regression and short-term freeway traffic forecasting. J. Transport. Eng. 117 (2), 178–188.
Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. B 1–38.
Efron, B., Tibshirani, R.J., 1994. An Introduction to the Bootstrap. CRC Press.
Ermagun, A., Levinson, D., 2018. Spatiotemporal traffic forecasting: review and proposed directions. Transport Rev. 38 (6), 786–814.
Frey, B.J., Dueck, D., 2007. Clustering by passing messages between data points. Science 315 (5814), 972–976.
Friedman, J., Hastie, T., Tibshirani, R., 2001. The Elements of Statistical Learning. Springer.
Fu, R., Zhang, Z., Li, L., 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In: Proceedings of Youth Academic Annual Conference of the Chinese Association of Automation. IEEE, pp. 324–328.
Fukunaga, K., Hostetler, L., 1975. The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Inf. Theor. 21 (1), 32–40.
Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256.
Graham, J.W., Olchowski, A.E., Gilreath, T.D., 2007. How many imputations are really needed? Some practical clarifications of multiple imputation theory. Prev. Sci. 8 (3), 206–213.
Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhhuber, J., 2017. LSTM: a search space odyssey. IEEE Trans. Neural Network. Learn. Syst. 28 (10), 2222–2232.
Guo, J., 2005. Adaptive Estimation and Prediction of Univariate Vehicular Traffic Condition Series. Ph.D. Thesis. North Carolina State University, NC, USA.
Habtie, A.B., Abraham, A., Midekso, D., 2017. Artificial neural network based real-time urban road traffic state estimation framework. In: Computational Intelligence in Wireless Sensor Networks. Springer, pp. 73–97.
Hamed, M.M., Al-Masaeid, H.R., Said, Z.M.B., 1995. Short-term prediction of traffic volume in urban arterials. J. Transport. Eng. 121 (3), 249–254.
Hashemi, M., Karimi, H.A., 2016. A weight-based map-matching algorithm for vehicle navigation in complex urban networks. J. Intell. Transport. Syst. 20 (6), 573–590.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
Honaker, J., King, G., Blackwell, M., 2011. Amelia II: a program for missing data. J. Stat. Software 45 (7), 1–47.
Jula, H., Dessouky, M., Ioannou, P.A., 2008. Real-time estimation of travel times along the arcs and arrival times at the nodes of dynamic stochastic networks. IEEE Trans. Intell. Transport. Syst. 9 (1), 97–110.
Julier, S.J., Uhlmann, J.K., 1997. New extension of the Kalman filter to nonlinear systems. AeroSense 97, 182–193. International Society for Optics and Photonics.
Kalman, R.E., Bucy, R.S., 1961. New results in linear filtering and prediction theory. J. Basic Eng. 83 (1), 95–108.
Kingma, D.P., Ba, J., 2015. Adam: A Method for Stochastic Optimization. In: Bengio, Y., Le Cun, Y. (Eds.), Proceedings of the 3rd Int. Conf. on Learning Representations. arXiv:1412.6980. ICLR.
Liaw, A., Wiener, M., 2002. Classification and regression by randomforest. R News 2 (3), 18–22.
Little, R.J.A., Rubin, D.B., 2014. Statistical Analysis with Missing Data. John Wiley & Sons.
Ma, X., Tao, Z., i Wang, Y., Yu, H., Wang, Y., 2015. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. Transport. Res. C Emerg. Technol. 54, 187–197.
MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1. University of California Press, pp. 281–297.
Moniruzzaman, M., Maoh, H., Anderson, W., 2016. Short-term prediction of border crossing time and traffic volume for commercial trucks: a case study for the Ambassador bridge. Transport. Res. C Emerg. Technol. 63, 182–194.
Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks. Int. Conf. Mach. Learn. 1310–1318.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.
Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65.
Rubin, D.B., 2004. Multiple Imputation for Nonresponse in Surveys. John Wiley & Sons.
Ruder, S., 2016. An Overview of Gradient Descent Optimization Algorithms arXiv preprint arXiv:1609.04747.
Schafer, J.L., Graham, J.W., 2002. Missing data: our view of the state of the art. Psychol. Methods 7 (2), 147.
Schafer, J.L., Olsen, M.K., 1998. Multiple imputation for multivariate missing-data problems: a data analyst's perspective. Multivariate Behav. Res. 33 (4), 545–571.
Scholkopf, B., Smola, A.J., 2001. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and beyond. MIT press.
Smith, B.L., 1995. Forecasting Freeway Traffic Flow for Intelligent Transportation Systems Application. PhD thesis, Charlottesville, VA, USA.
Smith, B.L., Demetsky, M.J., 1997. Traffic flow forecasting: comparison of modeling approaches. J. Transport. Eng. 123 (4), 261–266.
Smith, B.L., Williams, B.M., Oswald, R.K., 2002. Comparison of parametric and nonparametric models for traffic flow forecasting. Transport. Res. C Emerg. Technol. 10 (4), 303–321.
Snoek, J., Larochelle, H., Adams, R.P., 2012. Practical Bayesian optimization of machine learning algorithms. Adv. Neural Inf. Process. Syst. 2951–2959.
Stekhoven, D.J., Bühlmann, P., 2011. Missforest-non-parametric missing value imputation for mixed-type data. Bioinformatics 28 (1), 112–118.
Sutskever, I., 2013. Training Recurrent Neural Networks. Ph D. Thesis. University of Toronto, Toronto, Canada.

Tian, Y., Pan, L., 2015. Predicting short-term traffic flow by long short-term memory recurrent neural network. In: IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). IEEE, pp. 153–158.
Van Hinsbergen, C.P., Van Lint, J.W., Sanders, F.M., 2007. Short term traffic prediction models. In: Proceedings of the 14th World Congress on Intelligent Transportation Systems. ITS America.
Van Lint, J.W.C., 2008. Online learning solutions for freeway travel time prediction. IEEE Trans. Intell. Transport. Syst. 9 (1), 38–47.
Vapnik, V.N., 1999. An overview of statistical learning theory. IEEE Trans. Neural Networks. 10 (5), 988–999.
Vladimir, V.N., 1995. The Nature of Statistical Learning Theory. Springer Heidelberg.
Vlahogianni, E., Karlaftis, M.G., Golias, J.C., 2005. Optimized and meta-optimized neural networks for short-term traffic flow prediction: a genetic approach. Transport. Res. C Emerg. Technol. 13 (3), 211–234.
Wang, Y., Papageorgiou, M., 2005. Real-time freeway traffic state estimation based on extended Kalman filter: a general approach. Transp. Res. Part B Methodol. 39 (2), 141–167.
Wang, J., Shi, Q., 2013. Short-term traffic speed forecasting hybrid model based on chaos–wavelet analysis-support vector machine theory. Transport. Res. C Emerg. Technol. 27, 219–232.
Williams, B.M., Hoel, L.A., 2003. Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: theoretical basis and empirical results. J. Transport. Eng. 129 (6), 664–672.
Wu, C.-H., Ho, J.-M., Lee, D.-T., 2004. Travel-time prediction with support vector regression. IEEE Trans. Intell. Transport. Syst. 5 (4), 276–281.
Yang, Z., Mei, D., Yang, Q., Zhou, H., Li, X., 2014. Traffic flow prediction model for large-scale road network based on cloud computing. Math. Probl Eng. 2014 (3), 1–8.