Flexible Sliding Windows for Kernel Regression Based Bus Arrival Time Prediction

Hoang Thanh Lam(✉) and Eric Bouillet

IBM Research, Building 3, IBM Technology Campus, Damastown, Dublin 15, Ireland
{t.l.hoang,bouillet}@ie.ibm.com

Abstract. Given a set of historical bus trajectories D and a partially observed bus trajectory S up to position l on the bus route, kernel regression (KR) is a non-parametric approach which predicts the arrival time of the bus at location $l+h$ ($h > 0$) by averaging the arrival times observed at same location in the past. The KR method does not weights the historical data equally but it gives more preference to the more similar trajectories in the historical data. This method has been shown to outperform the baseline methods such as linear regression or k-nearest neighbour algorithms for bus arrival time prediction problems [9]. However, the performance of the KR approach is very sensitive to the method of evaluating similarity between trajectories. General kernel regression algorithm looks back to the entire trajectory for evaluating similarity. In the case of bus arrival time prediction, this approach does not work well when outdated part of the trajectories does not reflect the most recent behaviour of the buses. In order to solve this issue, we propose an approach that considers only recent part of the trajectories in a sliding window for evaluating the similarity between them. The approach introduces a set of parameters corresponding to the window lengths at every position along the bus route determining how long we should look back into the past for evaluating the similarity between trajectories. These parameters are automatically learned from training data. Nevertheless, parameter learning is a time-consuming process given large training data (at least quadratic in the training size). Therefore, we proposed an approximation algorithm with guarantees on error bounds to learn the parameters efficiently. The approximation algorithm is an order of magnitude faster than the exact algorithm. In an experiment with a real-world application deployed for Dublin city, our approach significantly reduced the prediction error compared to the state of the art kernel regression algorithm.

1 Introduction

Recently, bus arrival time prediction using GPS data has become an important problem attracting many researchers from both academia and industry labs [9,11]. Beside having important application in urban transportation management systems, GPS data providing accurate real-time locations of buses is very cheap and easy to collect. In such system, GPS devices equipped on-board continuously update real-time locations of the buses in a reasonable fine-grained

© Springer International Publishing Switzerland 2015
A. Bifet et al. (Eds.): ECML PKDD 2015, Part III, LNAI 9286, pp. 68–84, 2015.
DOI: 10.1007/978-3-319-23461-8_5

time resolution (from seconds to minutes). Updated locations of the buses are used to predict bus arrival time at any location of the trajectory. Prediction can be used to provide urban citizens with real-time information about bus arrival time at any bus stop or to estimate the likelihood of bus bunching from which bus operators can direct bus drivers to avoid those unexpected events.

Literature on bus arrival time prediction problem is very rich [2-3,6-14]. Depending on type of data used for prediction, different methods were proposed. However, when only GPS data is available, the *state-of-the-art* algorithm is relied on the *kernel regression* (KR) method [9]. The KR algorithm exploits the similarity of the currently observed trajectory of the bus and the historical trajectories to make prediction. Since behaviour of buses travelling on a fixed bus route is highly repeated, KR approach has been shown to outperform the baseline approaches such as linear regression or k-nearest neighbour [9].

Example 1 (Kernel regression). In order to illustrate the intuition behind the KR method we show an example with 4 different historical spatio-temporal trajectories A,B,C,D and a partially observed trajectory S at location l in Figure 1. The y-axis shows the time offsets since the time when the buses start and the x-axis shows the distance (in meters) from the departure stop.

Prediction of the arrival time of the bus at location $l + h$ $(h > 0)$ denoted as $\hat{S}(l + h)$ is done by averaging the arrival time of the bus at location $l + h$ observed in the historical data:

$\hat{S}(l + h) = \alpha_A * A(l + h) + \alpha_B * B(l + h) + \alpha_C * C(l + h) + \alpha_D * D(l + h)$

Where the weight values $\alpha_A, \alpha_B, \alpha_C, \alpha_D$ summing up to 1 and can be calculated as follows:

$$\alpha_A = \frac{sim(S,A)}{sim(S,A)+sim(S,B)+sim(S,C)+sim(S,D)}$$
$$\alpha_B = \frac{sim(S,B)}{sim(S,A)+sim(S,B)+sim(S,C)+sim(S,D)}$$
$$\alpha_C = \frac{sim(S,C)}{sim(S,A)+sim(S,B)+sim(S,C)+sim(S,D)}$$
$$\alpha_D = \frac{sim(S,D)}{sim(S,A)+sim(S,B)+sim(S,C)+sim(S,D)}$$

Function $sim(S, A)$ denotes the similarity between two trajectories S and A till position l. The larger the value of $sim(S, A)$ the more similar the two trajectories are.

An benefit of using KR in practice is that there is no need to build a model for every location along the bus route as does with parametric approaches such as linear regression. This property is a big plus in an industrial setting when fast deliver of solution is required. However, KR is sensitive to the choice of the similarity function which usually considers the whole trajectory for similarity evaluation [9]. Under the context of bus arrival time prediction application, bus journeys might be influenced by hidden spatial or temporal contextual factors such as changes from crowded to less traffic locations or unplanned events like accidents. In such cases, the entire trajectory is no longer relevant for making prediction because most of the information is out of date and does not reflect

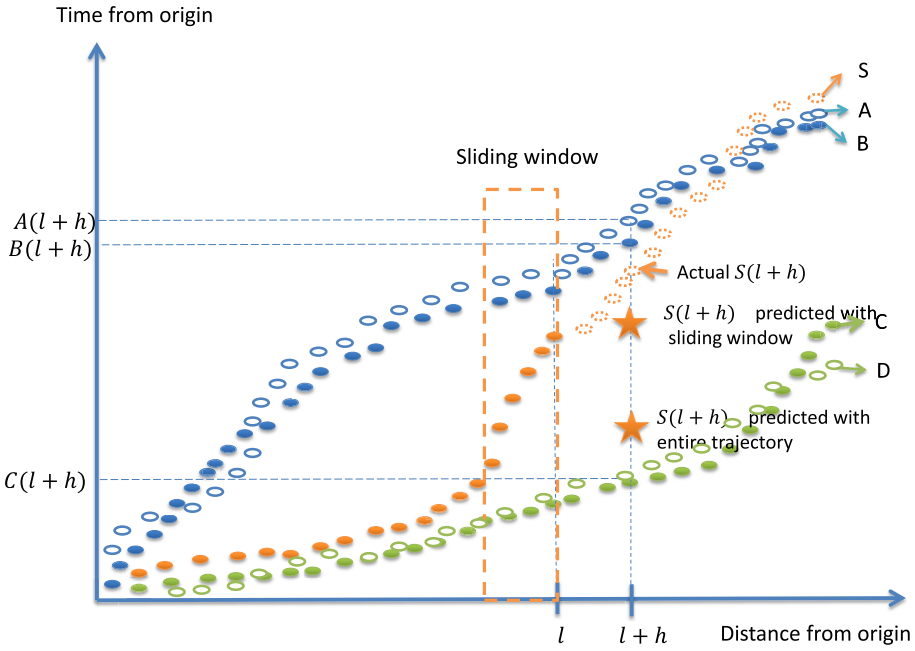

Fig. 1. Four trajectories A, B, C and D in a historical log and a partially observed (orange solid points) bus trajectory S up to location l (the orange dotted points are future positions). As we can observe on the figure, the bus was moving very close to C and D from the beginning but due to some unplanned events happening at location l the bus was delayed and started moving closer to A and B during the last part of the journey. At location l, considering the whole trajectory, S is much closer to C and D than to A and B. On the other hand, in the sliding window with only recent data, S is much closer to A and B. Therefore, making prediction based on recent data is more accurate as it captures recent behaviours of the bus.

the recent behaviour of the buses. The following example shows a situation in which the KR method is sensitive to the choice of similarity evaluation methods.

Example 2 (Sensitivity to similarity function). Figure 1 shows four historical trajectories A, B, C and D and a partially observed (orange solid points) trajectory S at location l (the orange dotted points are future positions). Prediction of bus arrival time at location $l+h$ is made by copying information about arrival times of the bus at location $l+h$ from similar historical trajectories.

Different predictions (marked with star-like symbols) of bus arrival time $\hat{S}(l+h)$ at location $l+h$ are made using either the entire trajectory or only recent data in a sliding window with size $w > 0$. As we can observe on the figure, the bus on the journey S was moving very close to C and D from the beginning but due to some reasons, e.g. an unplanned event happening at location l, the bus was delayed and started moving closer to A and B during the last part of the journey. At location l, considering the whole trajectory, S is much closer to

C and D than to A and B. On the other hand, in the sliding window (marked with a dotted orange box) with only recent data, S is much closer to A and B. Therefore, making prediction based on recent data in that window is more accurate as it captures the recent behaviour of the bus.

The key assumption in Example 2 is that each location along the bus route is associated with a hidden spatial or temporal context that determines how long we should look back to the past for predicting the future bus arrival times. An important question to ask is: how do we discover relevant window size for each location along the bus route to make prediction better? In this work, we try to answer this question by proposing a method that automatically learns appropriate window sizes from training data. The key technical contributions of this work can be summarized as follows:

- A method to optimize the kernel regression based prediction algorithm for a real-world application relied on flexible sliding window length learning.
- An approximation algorithm (with error bound guarantees) for speeding up the parameter learning process when the training size is large.
- Our method reduced the prediction error from 40-60 %.
- The approximation algorithm proposed for the window size learning task achieved a speeds up of 15-20x while preserving high accuracy of the prediction as the brute-force learning method does.
- Our approach speeds up real-time evaluations of the similarity between trajectories as only short parts of the trajectories are considered for evaluation.

2 Problem Definition

In this section, we first recall the kernel regression algorithm, and then formally define the problem. All the notations are shown in Table 2.

Notation	Meaning
$S = t_1 t_2, \cdots, t_n$	A trajectory with arrival times at n locations
$S(l)$	Arrival time at location l
$\hat{S}(l)$	Predicted arrival time at location l of trajectory S
$\hat{S}^w(l)$	Predicted arrival time when window length is w
$S(l, h)$	A sequence of arrival times from location l to h
$D = \{A_1, A_2, \cdots, A_m\}$	A reference set with m historical trajectories
$Sim(A, B)$	A similarity function between two sequences A and B
$e_l(w)$	Prediction error at location l when window size is w
$E[e_l(w)]$	Expectation of prediction error when window size is w
w_l^*	The best window length at location l

2.1 Kernel Regression

A bus trajectory is a sequence of pairs $S = (p_1, t_1), (p_2, t_2), \cdots, (p_n, t_n)$ where p_i is a location on the bus route and t_n is the arrival time at that location counting from the beginning of the journey. In our bus arrival time prediction application, bus route is fixed beforehand and GPS-based locations are interpolated such that the trajectory contains arrival time for every location on the route at the distance one meter. Therefore, a bus trajectory can be considered as a sequence of arrival times $S = t_1 t_2, \cdots, t_n$ because the corresponding location can be implicitly inferred from the context with the index of the arrival time.

A historical dataset is a collection of bus trajectories $D = \{A_1, A_2, \cdots, A_m\}$ each has length equal to n. Given a trajectory A_i, denote $A_i(l)$ as the bus arrival time at location l and $A_i(l, h)$ $(h > l)$ as the sequence of arrival times starting from location l to location h: $A_i(l+1)A_i(l+2)\cdots, A_i(h)$. Let denote $Sim(A, B)$ as a similarity function between two sequences with the same length A and B. The larger value of $Sim(A, B)$ is the more similar the sequences are.

Let S be a partially observed trajectory up to location l, from now on we call S the *target trajectory* while the trajectories in historical data are called as *reference trajectory*. Assume that our main goal is to predict the arrival time at location $l + h$ for a prediction horizon $h > 0$ of the target trajectory, i.e predict the unobserved value of $S(l + h)$. The kernel regression algorithm estimates the value $\hat{S}(l + h)$ with the help of the historical dataset D as follows:

$$\hat{S}(l+h) = \frac{\sum_{i=1}^{m} Sim(S(0,l), A_i(0,l)) * A_i(l+h)}{\sum_{i=1}^{m} Sim(S(0,l), A_i(0,l))} \tag{1}$$

The prediction is made by averaging observations of arrival time at location $l + h$ in the historical data. Every observation $A_i(l + h)$ is weighted by the similarity between the current trace S and the historical trace A_i. For the bus arrival time prediction application, Sinn et al. [9] suggested to use the *Gaussian kernel* $Sim(A, B) = e^{-\sum_{i=0}^{l} \frac{1}{b} \frac{(A(i)-B(i))^2}{\sigma_i}}$ where σ_i denotes the variance of arrival time at location i and b is a bandwidth parameter.

2.2 Problem Definition

As we have discussed earlier, the KR algorithm is sensitive to the way we evaluate the similarity between the target trajectory and the reference trajectories. The state-of-the-art algorithm considers the entire trajectory for evaluating the similarity score. In this work, we introduce a method that uses recent data for evaluating the similarity score. Let w_l $(w_l \leq l)$ denote the length of the sliding window which we use to calculate the similarity score at position l. In particular, the $Sim(S(0, l), A_i(0, l))$ function in the kernel regression method is replaced by $Sim(S(l - w_l, l), A_i(l - w_l, l))$. Therefore, for a bus route with length n we have a vector of n window lengths $W = (w_1, w_2, \cdots, w_n)$

An important question is how we determine an appropriate window length w_l for every location l along the bus route. Finding optimal window lengths can

Algorithm 1. A brute-force algorithm

1: **Input**: a training set $D = \{A_1, A_2, \cdots, A_m\}$ and a prediction horizon h
2: **Output**: A vector of window lengths $W = \{w_1^*, w_2^*, \cdots, w_n^*\}$
3: **for** l=1 to n **do**
4: **for** w=1 to l **do**
5: **for** i=1 to m **do**
6: $e_l^i(w) \leftarrow (A_i(l+h) - \hat{A}^{w_i}(l+h)^2$
7: $e_l(w) \leftarrow e_l(w) + e_l^i(w)$
8: **end for**
9: $e_l(w) \leftarrow \frac{e_l(w)}{m}$
10: **end for**
11: $w_l^* = argmin_w e_l(w)$
12: **end for**

be formulated as an optimization problem as follows. We use the set of historical trajectories $D = \{A_1, A_2, \cdots, A_m\}$ as a training set and evaluate the set of window lengths through a leave-one-out cross-validation process.

In particular, let h be a horizon for prediction, l be a location. We pick A_i from the training set and consider it as a target trajectory while the set $D \backslash \{A_i\}$ is considered as a reference set. The prediction error at location l when the window length is set to w can be calculated as follows:

$$e_l^i(w) = (A_i(l+h) - \hat{A}_i^w(l+h))^2 \tag{2}$$

Where $\hat{A}_i^w(l+h)$ denotes the predicted arrival time at location $l+h$ when the sliding window length is set to w. The leave-one-out cross-validation average prediction error over the entire training set D can be estimated as : $e_l(w) = \frac{\sum_{i=1}^m e_l^i(w)}{m}$. The problem of learning the optimal window length at location l can be formulated as follows:

Problem 1 (Window length learning). For every location l, find the window length w_l^* which minimizes the expectation of the leave-one-out cross-validation average prediction error $e_l(w)$, i.e. $w_l^* = argmin_w E[e_l(w)]$

3 A Brute-Force Algorithm

In this section, we introduce a brute-force search approach that solves Problem 1. It evaluates all possible values of the window length at every location in the bus route. At location l, since $0 < w < l$ we simply calculate $e_l(w)$ for every possible value of w and choose w_l^* that minimizes the accumulative prediction error $e_l(w)$. The same task can be repeated for every location $1 \leq l \leq n$ to obtain a complete set of window lengths for every location along the bus route.

Algorithm 1 describes the main steps of the brute-force algorithm. It iterates over every location l along the bus route (lines 3-12) and evaluate the accumulative prediction error $e_l(w)$ for every candidate window length w (lines 4-10). Evaluation of the accumulative prediction error is done by summing up the

prediction error made when each trajectory A_i is picked as a target and the remaining trajectories are combined to a reference set (lines 5-8). In line 11, the best window length w_l^* for location l is selected as the one that minimizes the accumulative prediction error.

The complexity of Algorithm 1 is $O(m^2n^2)$ because of the three outer loops and the computation of the prediction error in lines 5-8 (requiring another loop over the training set). It is important to notice that the calculation of the prediction error at location l in lines 5-8 requires a full pass over the training data which incurs a complexity of $O(mn)$. However, this number can be reduced to $O(m)$ when the computation only needs to update from the result of the prior step at location $l-1$. In general, the brute-force algorithm is not scalable especially when the training size is large because of the quadratic complexity. In the next subsection, we will discuss an approximation algorithm that speeds up the brute-force method significantly.

4 An Approximation Algorithm

The complexity of Algorithm 1 is $O(m^2n^2)$, where m is the training size and n is the trajectory length. Since trajectory length is usually fixed, it can be considered as a large constant number. The training size m is equal to the number of historical trajectories collected so far. The training size increases as long as data is collected everyday, so we propose a method called FLOW (flexible sliding window for kernel regression) which approximates the solution of Problem 1 by optimizing the expensive computation caused by the training size m.

4.1 Approximation Algorithm

Recall that in Algorithm 1, for every candidate window length w, the prediction error $e_l(w)$ is accumulated by looping over every trace A_i in the training set. Our key intuition is that at a certain iteration i where i is large enough, we can estimate how likely a window length w is the optimal window length for the given location. For instance, if we observe that the value of $e_l(w_1)$ is significantly less than $e_l(w_2)$ then we can conclude that w_2 has a very low chance of being the optimal window length. In that case, we can stop evaluating $e_l(w_2)$ as long as the accuracy of the learning process is not too much sacrificed.

Algorithm 2 is not much different from Algorithm 1. It iterates over every location l on the bus route and find the best window length among a set of candidates stored in a list L (lines 3-4). Different from the brute-force algorithm, in each iteration of the leave-one-out cross-validation (lines 5-15), FLOW checks if the remaining candidates in the list L have very low probability (less than a user defined parameter ϵ) of being the best window (line 11). If the answer is yes then the candidates are removed from the list (line 12) and never be considered in the following iterations. In experiments, we will show that Algorithm 2 is much more efficient than the brute-force algorithm because it

Algorithm 2. FLOW algorithm

1: **Input**: a training set $D = \{A_1, A_2, \cdots, A_m\}$, a parameter ϵ and a prediction horizon h
2: **Output**: A vector of window lengths $W = \{w_1^*, w_2^*, \cdots, w_n^*\}$
3: **for** l=1 to n **do**
4: $L \leftarrow = \{1, 2, \cdots, l\}$
5: **for** i=1 to m **do**
6: **for** each w in L **do**
7: $e_l(w) \leftarrow \frac{e_l(w)*(i-1)+(A_i(l+h)-\hat{A}_i^w(l+h))^2}{i}$
8: **end for**
9: $w^* \leftarrow argmin_w e_l(w)$
10: **for** each w in L **do**
11: **if** $e_l(w) > e_l(w^*) + \Delta(i, \epsilon)$ **then**
12: L.remove(w)
13: **end if**
14: **end for**
15: **end for**
16: $w_l^* = argmin_w e_l(w)$
17: **end for**

prunes the computation significantly. An important point need to explain is the condition used for pruning a candidate w:

$$e_l(w) > e_l(w^*) + \Delta(i, \epsilon) \tag{3}$$

Where w^* is the current best candidate (up to the current iteration of the loop in line 5) with the smallest accumulative prediction error, i.e. $w^* \leftarrow argmin_w e_l(w)$, and $\Delta(i, \epsilon)$ is a function that depends on the tolerance parameter ϵ and the current iteration i.

The tolerance $\epsilon << 1$ is given as an input parameter which tells the program the desired bounded probability of missing the best window length during the search process. In the next subsection we will show how to derive the value of the function $\Delta(i, \epsilon)$ in each iteration. The key meaning of the pruning condition is: if the accumulative error of the candidate w is far deviated from the accumulative prediction error of the best candidate by a large number $\Delta(i, \epsilon)$ then it is safe to prune w from the list of candidates with a small chance of missing the optimal candidate (less than ϵ).

4.2 Theoretical Analysis

Recall that at iteration i, $e_l(w) = \frac{\sum_{k=1}^{i} e_l^k(w)}{i}$, where $e_l^k(w)$ is the prediction error when the A_k trajectory is picked as a target. Our main assumption in the analysis is that each $e_l^k(w)$ is a random variable with the same mean value and they are independent to each other. Therefore, from the definition of $e_l^k(w)$ for any k: $E[e_l^k(w)] = E[e_l(w)]$.

Let us denote the maximum and the minimum value of the arrival time at location $l+h$ as max and min. The following theorem shows how to calculate the

function $\Delta(i, \epsilon)$ to guarantee that the probability of missing the optimal window length is less than ϵ:

Theorem 1 (Error Bound). *In Algorithm 2, if in each iteration we choose* $\Delta(i, \epsilon) = \frac{\sqrt{2}(max-min)^2}{\sqrt{i}} \log \frac{2}{\epsilon}$ *then the probability that FLOW misses the optimal candidate is upper-bounded by* ϵ.

Proof. Because of space limit, please refer to the link[1] to see the proof.

Theorem 1 shows that the value of function $\Delta(i, \epsilon)$ decays linearly with the value of $\sqrt{i}$. Therefore, when i is large enough FLOW can prune the computation substantially.

4.3 Further Optimization

In order to find the best window length for any location l along the bus route, we need to evaluate all values of window length $1 \leq w \leq l$. Our observation with the bus dataset shows that the prediction error in the leave-one-out cross-validation process is a smooth function of window lengths. Its value does not change significantly when the window length is slightly modified.

Therefore, instead of considering all values of w, we limit the search for the best window length to a subset with maximum $\log(l)$ values: $\{l, \lceil \frac{l}{2} \rceil, \lceil \frac{l}{2^2} \rceil, \cdots, 1\}$. This well-known technique in data stream called as pyramidal time frame [1] usually used for reducing the search space with preference to recent part of data. With this minor change to the FLOW and the brute-force algorithm we can reduce the worst-case complexity to $m^2 n \log(n)$. This techniques allows us to speed up the search algorithm especially for the case when n is large. In experiments, the results were reported when both brute-force and FLOW algorithm used this technique to reduce the search space.

5 Experiments

We will first start with a subsection that describes detail information about the dataset used in the paper and the experiment settings. Subsequently, we will report the empirical results including the prediction accuracy and the effectiveness of the approximation method.

The original implementation of the KR algorithm was chosen for comparison because it is considered as the state-of-the-art algorithm for this application. Finally, we performed several analyses on the distribution of the optimal window lengths chosen by our learning method for every location along the bus route to understand the hidden contexts existing in the data.

[1] https://drive.google.com/file/d/0BwWtvZfA5UCSUHpfS0d3a2pMVTQ/view?usp=sharing

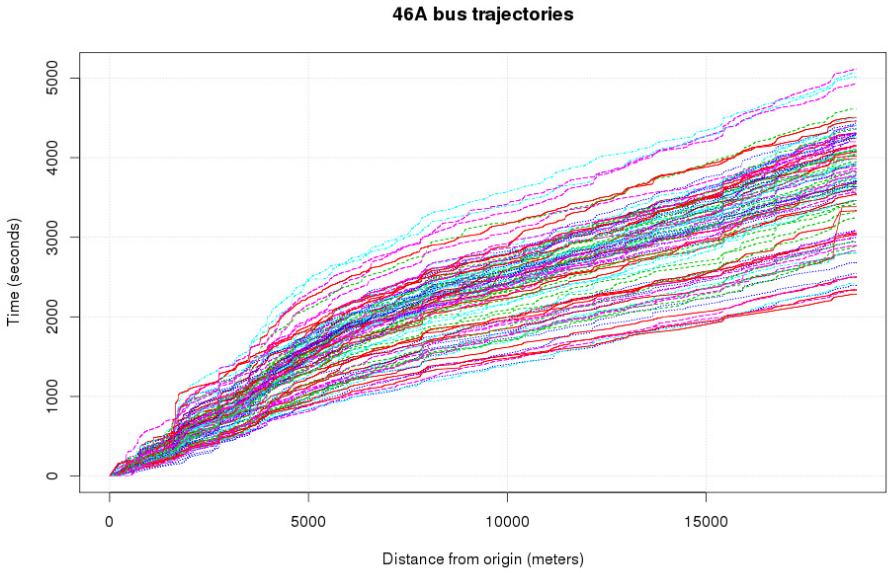

Fig. 2. One hundred bus trajectories sampled from the 46A bus dataset. The length of the journey is about 18 Km, in average, buses need one hour to complete the journey.

5.1 Dataset and Experiment Settings

The dataset used for empirical study consists of 1500 bus traces from the bus route 46A (outbound) in Dublin city in a period of one month. This route was chosen because it is the most frequent route in the city. The dataset is available for download at Dublinked[2]. Each bus trace was created from GPS data updated every 20-30 seconds. The GPS trajectories were projected to the known bus route and the positions were interpolated at one-meter long resolution.

All traces can be considered as sequences with the same length with 18587 data points corresponding to a 18 Km long bus route. More detail about the dataset and the preprocessing steps can be found in [4,9]. All the source codes were written in Java and the program was run on a Linux machine with 4GB of RAM. The implementation of the KR algorithm for comparison follows the description in Sinn et al. [9]. In that implementation, a parameter needs to set concerning the bandwidth of prediction. We fixed that value to 1 as suggested in the original work to ensure a fair comparison [9].

Recall that the FLOW algorithm uses the tolerance parameter ϵ to control the early pruning strategy. In our experiments, the tolerance parameter ϵ was set to 0.01 which limits the probability of missing the optimal window length to less than one percent. In addition to that, the Max and Min values in the

[2] http://www.dublinked.ie/

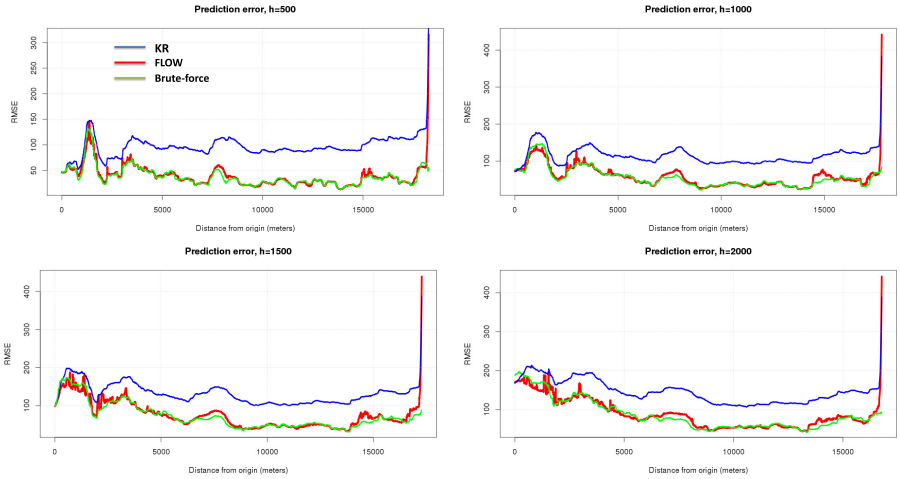

Fig. 3. Prediction error (in seconds, smaller is better) at each location along the bus route. The FLOW algorithm outperformed the KR algorithm. The brute-force algorithm was only slightly better than the FLOW algorithm. This confirmed our theoretical analysis in section 2 which shows that we didn't loose a lot of accuracy when approximation algorithm is used instead of the brute-force search.

formula calculating the pruning factor $\Delta(i, \epsilon)$ were set to the maximum and the minimum value of the arrival time observed in the training data.

5.2 Prediction Accuracy

In this subsection, we show the prediction accuracy calculated as rooted mean square error (RMSE smaller is better) at every location along the bus route. Given a test set $T = \{T_1, T_2, \cdots, T_M\}$ and a prediction horizon value h, the root mean square prediction error at location l is calculated as follows:

$$RMSE_l = \sqrt{\sum_{i=1}^{M} (T_i(l+h) - \hat{T}_i^{w_l^*}(l+h))^2} \tag{4}$$

The experimental settings and the dataset were kept to the same as described in [9]. Since Sinn et al. has shown that the KR method outperformed the linear regression and the k-nearest neighbour algorithm, so in this work we just need to compare FLOW directly with to the original implementation of the kernel regression algorithm denoted as KR.

Figure 3 shows the RMSE (in seconds on the y-axis) at every location along the bus route (x-axis) when the prediction horizon h was set to 500, 1000, 1500

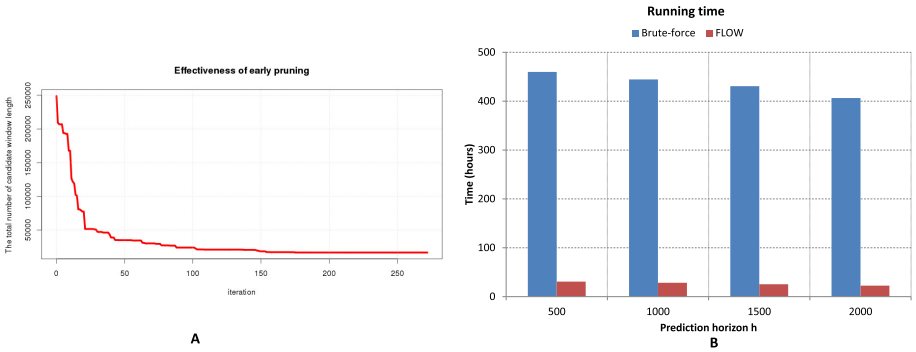

Fig. 4. (A). With the early pruning strategy, FLOW can prune a substantial amount of computation as the number of window length candidates decreases exponentially with the number of iterations. (B). Comparison of running time between the FLOW and brute-force algorithms when the prediction horizon is varied. FLOW achieves about an order of magnitude (15-20x) faster than the brute-force algorithm.

and 2000 meters respectively. The RMSE were reported via a ten-fold cross-validation. The first impression from Figure 3 is that the FLOW algorithm outperformed the KR algorithm with 40-60 % reduction in prediction error. The results are stable across different locations and with various prediction horizons.

Moreover, The difference between the FLOW and the brute-force algorithm is negligible. In fact, as we can see from the plots, the brute-force algorithm was only slightly better than the FLOW algorithm. This empirical results confirmed our theoretical analysis in section 3 that we didn't loose much accuracy when the approximation algorithm was used instead of the brute-force algorithm.

At the locations near to the end of the route, we can see that the errors increase significantly. The reason is that in the set of trajectories there are a few outliers on which the bus needs about more than four hours to complete the journey instead of one hour in average as usual. For those outlier traces, delay happened close to the end of the journey which explains why we see a peak in prediction error at the end of the trajectories.

5.3 Effectiveness of Approximation

In subsection 5.2, we have shown that the prediction accuracy of the approximation algorithm is very similar to the prediction error of the brute-force algorithm. In this subsection, we will show that the FLOW algorithm is significantly more efficient than the brute-force algorithm.

First, recall that the approximation algorithm works by early pruning the set of candidate window lengths that with high confidence cannot be the optimal window length. In order to evaluate how effective the pruning strategy is in practice we plotted the number of candidate window lengths observed in each iteration (lines 5-15) of Algorithm 2 in Figure 4.A. In that figure, the x-axis

shows the index of the iteration and the y-axis shows the total number of candidate window lengths. If no pruning was used, the total number of candidate window lengths should be always equal to the initial number of candidates at the first iteration. When pruning strategy was used we can see that the number of candidate window lengths decreases exponentially with the iterations. Therefore, after only a few hundred iterations the number of candidate window lengths reaches its minimum value and the searching process can stop early.

The running times of the FLOW and the brute-force algorithms are reported in Figure 4.B. We can see that the FLOW algorithm achieves an order of magnitude (from 15x to 20x) faster than the brute-force algorithm. This result shows that the pruning strategy deployed in the implementation of the FLOW algorithm is very effective.

5.4 Interpretation of the Results

The distribution of the best window length at different locations along the bus route is shown in Figure 5. As we can observe, in most location, FLOW only picks a few recent data points. Thanks to this, evaluation of similarity between the target and the reference trajectories is very efficient because the window length is very short. Interestingly, there are several locations in which the window length suddenly increases. This may concern a hidden spatio-temporal context that causes the change. These shifting contexts might provide bus operators with deep insights about the data.

6 Related Work

Recently, bus arrival time prediction problem attracts a lot of attention because of its useful application in public transport management systems. The most popular methods were relied on artificial neutral networks (ANNs) [2,3,8]. The issue with an ANN is that it is very sensitive to the network structure design and easily overfits data [15]. Besides ANNs, methods based on Kalman filters are also very popular. For instance, Wall et al. [12], Son et al. [10] and Yang et al. [14] combined data from automatic vehicle location services and historical data to make bus arrival time prediction.

Other machine learning approaches have also been proposed for this problem. Li et al. [7] used linear regression model with fused data from different sources such as GPS sensors, wired loop sensors and red radio radar etc. for bus arrival time prediction. Zhou et al. [16] used mobile sensing data from participating users for making prediction. Bin et al. [15] used support vector machine (SVM) using different features extracted from weather condition, type and time of the date, travel time in the previous segments. The SVM method has been shown to be superior to the methods based on ANN.

Under the context of the bus arrival time prediction problem, no method works well for all applications because each of them requires a specific type of

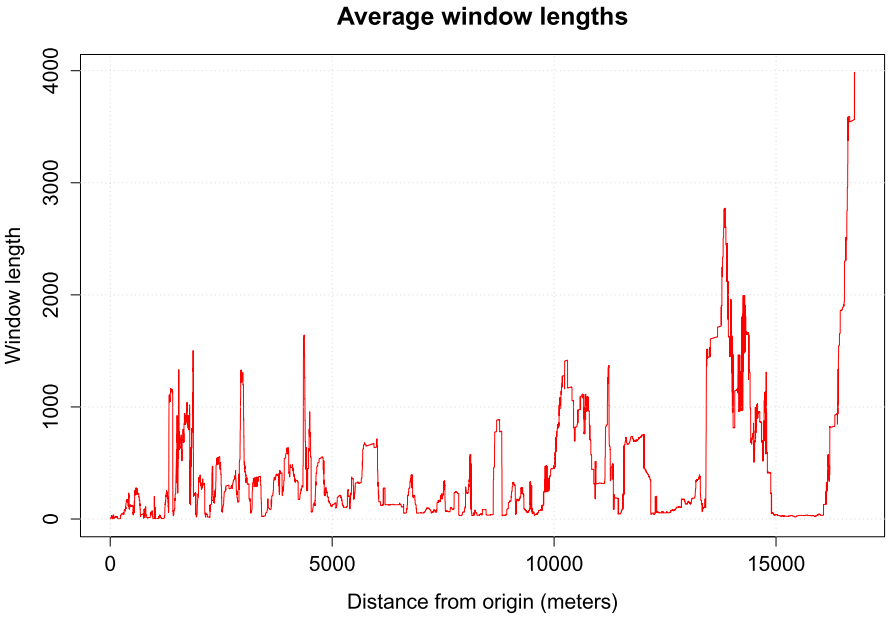

Fig. 5. Distribution of window length (y-axis) as a function of the location (x-axis). We can see that in most case FLOW only picks a few recent data. Interestingly, there are several locations in which the window length suddenly increases. This may concern a hidden spatio-temporal context that causes the change. These shifting contexts might provide bus operators with deep insights about the data.

data used for prediction. In practice, not always different types of data are available, e.g. mobile sensing data is only owned by telco companies while GPS data is collected by bus operator companies. When only GPS data is available, there are several approaches [9,11,13,17]. Nevertheless, the state-of-the-art algorithm for the bus arrival time prediction problem is relied on the kernel regression algorithm [9] in which Sinn et al. showed that the kernel regression approach outperformed the other methods based on linear regression models and k-nearest neighbour prediction algorithms.

Another reason that makes kernel regression attractive is that it is a non-parametric approach. Therefore, we don't need to learn different predictive models for every location along the bus route. It doesn't require intensive human efforts for feature extraction and selection. This property enables us to deploy scalable online prediction algorithms for large-scale applications because it does not require expensive training tasks and bookkeeping a model for every loca-

tions along the bus route. The KR method has been deployed as a service for bus arrival time prediction at the city of Dublin[3].

Therefore, our work most relates to [9]. An important difference is that we focused on optimizing the kernel regression methods. Although our proposal makes KR no longer a non-parametric approach, we just need to keep one parameter corresponding to the sliding window length at each location along the route. This approach is still much cheaper than the methods that keeps a set of parameters corresponding to each features used for prediction for each location along the bus route.

Other related work tries to optimize the bandwidth parameter of the KR method [5]. An important difference between those and our work is that our optimization concerns the feature selection problem (how far we should look back into the historical data to make prediction better) while the bandwidth optimization problem more concerns normalization factor optimization. Therefore our approach is orthogonal to the bandwidth optimization problem. In fact, bandwidth selection can be performed in parallel with the window length learning task to improve the prediction further.

Finally, the idea of using a sliding window with predefined size for making prediction is very popular in data stream mining community [1]. Nevertheless, these methods require users to set a fixed window length in advance for all locations. These methods do not work well because of two reasons. First, the users do not know which value they should choose for the window size. Second, the optimal window lengths as observed in Figure 5 vary a lot depending on the location along the bus. Different from those work, our method can automatically learn appropriate window lengths for every location along the bus route.

7 Conclusion and Future Work

In this work, we exploited the implicit spatial or temporal contexts to improve the prediction accuracy of the state of the art prediction algorithm for bus arrival time prediction problem with GPS data. Our algorithm searches for relevant data at each location that needs to use for improving the prediction. The results with a real-world dataset show that our method can improve the prediction accuracy significantly (from 40-60 % reduction in RMSE). Since the learning algorithm is time consuming, we proposed an approximation algorithm for the learning process which reduces the learning time significantly (15x-20x faster).

There are several possibilities to extend the current work. For instance, our algorithm was proposed for static data. It is interesting to discover and search for relevant data in an online settings to capture the real-time effect of several unplanned events such as accidents. Another important problem needs to solve is how to associate the discovered window lengths with the hidden spatial contextual information in order to better understand the behaviour of the bus on each segment of the route.

[3] http://dublinbus.ie/

Acknowledgments. We would like to thank Dublin City Council (DCC) for providing us with the Dublin bus dataset. We also thank Dr. Mathieu Sinn for his useful comments during the development of the work.

References

1. Aggarwal, C.C. (ed.): Data Streams - Models and Algorithms, Advances in Database Systems, vol. 31. Springer (2007)
2. Chen, M., Liu, X., Xia, J., Chien, S.I.: A dynamic bus-arrival time prediction model based on APC data. In: Computer-Aided Civil and Infrastructure Engineering, pp. 364–376, July 2004
3. Chien, S., Ding, Y., Wei, C.: Dynamic bus arrival time prediction with artificial neural networks. Journal of Transportation Engineering **128**(5), 429–438 (2002)
4. Coffey, C., Pozdnoukhov, A., Calabrese, F.: Time of arrival predictability horizons for public bus routes. In: Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science, CTS 2011, pp. 1–5. ACM, New York (2011)
5. Hardle, W., Marron, J.S.: Optimal bandwidth selection in nonparametric regression function estimation. The Annals of Statistics, 1465–1481 (1985)
6. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301), 13–30 (1963)
7. Li, F., Yu, Y., Lin, H., Min, W.: Public bus arrival time prediction based on traffic information management system. In: IEEE International Conference on Service Operations, Logistics, and Informatics (SOLI), pp. 336–341, July 2011
8. Mazloumia, E., Rosea, G., Curriea, G., Sarvia, M.: An integrated framework to predict bus travel time and its variability using traffic flow data. Journal of Intelligent Transportation Systems: Technology, Planning, and Operations (2011)
9. Sinn, M., Yoon, J.W., Calabrese, F., Bouillet, E.: Predicting arrival times of buses using real-time gps measurements. In: 15th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 1227–1232, September 2012
10. Son, B., Kim, H.-J., Shin, C.-H., Lee, S.-K.: Bus arrival time prediction method for ITS application. In: Negoita, M.G., Howlett, R.J., Jain, L.C. (eds.) KES 2004. LNCS (LNAI), vol. 3215, pp. 88–94. Springer, Heidelberg (2004)
11. Sun, D., Luo, H., Fu, L., Liu, W., Liao, X., Zhao, M.: Predicting bus arrival time on the basis of global positioning system data. Transportation Research Record: Journal of the Transportation Research Boardg (2007)
12. Wall, Z., Dailey, D.J.: An algorithm for predicting the arrival time of mass transit vehicles using automatic vehicle location data. In: 78th Anual Meeting of the Transportation Research Board (1999)
13. Xinghaoa, S., Jinga, T., Guojuna, C., Qichongb, S.: Predicting bus real-time travel time basing on both GPS and RFID data. In: 13th COTA International Conference of Transportation Professionals (CICTP 2013), pp. 2287–2299, November 2013
14. Yang, J.-S.: Travel time prediction using the GPS test vehicle and kalman filtering techniques. In: Proceedings of the 2005 American Control Conference, vol. 3, pp. 2128–2133, June 2005

15. Yu, B., Yang, Z., Yao, B.: Bus arrival time prediction using support vector machines. Journal of Intelligent Transportation Systems, 151–158, July 2006
16. Zhou, P., Zheng, Y., Li, M.: How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing (2013)
17. Zhu, T., Ma, F., Ma, T., Li, C.: The prediction of bus arrival time using global positioning system data and dynamic traffic information. In: Wireless and Mobile Networking Conference, pp. 1–5, October 2011