

Travel-Time Prediction using Gaussian Process Regression: A Trajectory-Based Approach

Tsuyoshi Idé Sei Kato
IBM Research, Tokyo Research Laboratory
{goodidea, seikato}@jp.ibm.com

Abstract

This paper is concerned with the task of travel-time prediction for an arbitrary origin-destination pair on a map. Unlike most of the existing studies, which focus only on a particular link (road segment) with heavy traffic, our method allows us to probabilistically predict the travel time along an unknown path (a sequence of links) if the similarity between paths is defined as a kernel function. Our first innovation is to use a string kernel to represent the similarity between paths. Our second new idea is to apply Gaussian process regression for probabilistic travel-time prediction. We tested our approach with realistic traffic data.

1 Introduction

Recent advances in sensing and information technologies make it possible to track moving objects such as vehicles and humans over a wide area of a city. An Intelligent Transportation System (ITS) is a large-scale information system designed to optimize transportation traffic and to offer guidance information to drivers by analyzing the vehicle traffic over an entire city. Just as dynamic information flows on the hyperlink structure of the Internet have created the new research field of Web mining, traffic data is bringing new challenges in data mining because of the variety and volume of the data.

This paper is concerned with the task of *travel-time prediction* for a given origin-destination (OD) pair on a map. This is one of the fundamental tasks in traffic modeling, and much effort has been devoted since the 90s when ITS appeared. In general, there are two views in traffic modeling, which we call the observer's view and driver's view [12]. The *observer's view* watches the traffic on a particular link, viewing the traffic as a flow of many moving objects. Here a *link* is a road segment between neighboring intersections. The *driver's view*, on the other hand, focuses on a particular moving object, and tracks the object along all of the links of a particular path. Most of the existing approaches are based on the observer's view. Recent examples include multivariate autoregressive (AR) modeling [17] and "nonparametric" approaches [29] for the traffic over particular links. This is

not surprising since there was no way to track the trajectories of individual objects until recently. Thus traffic modeling was possible only for links having sufficient traffic.

However, with the rich information collected using a modern GPS (Global Positioning Systems), the traditional time-series modeling approaches are less satisfactory. Specifically, these approaches are not capable of making predictions for links with little traffic history data. To be concrete, imagine that a control center in a city broadcasts rerouting information to avoid traffic jams in a particular region. In this case, what we really want is a predictive capability for many path candidates rather than accurate predictive models for a limited number of main roads (a *path* is a sequence of links for a given OD pair). For a given OD pair, a path may go through side roads without sensors for time-series modeling.¹ For such links, the traditional methods must be supplemented with static information such as legal speed limits. It is clear that the overall accuracy is greatly degraded when such sensor-free links are included in a path, since, for example, waiting times at intersections are neglected.

Unlike most of the existing work, this paper is based on the driver's view. Our task is to predict the travel time for an arbitrary OD pair, based on a data set as $\{(x^{(n)}, y^{(n)}) | n = 1, 2, \dots, N\}$. Here $x^{(n)}$ is the identification of the n -th path, and $y^{(n)}$ is the observed travel time for the path $x^{(n)}$. N is the number of observed paths. Note that an $x^{(n)}$ can include many links, and some of links might be minor roads whose traffic volume is insufficient for time-series modeling. Also, some of the $x^{(n)}$ s can be the same, but the number of possible paths is exponential with the total number of links in the map.

Our task can be viewed as a regression problem where the target variable is y and the predictor variable is x . However, the nature of this problem is quite different from standard regression problems. First, the predictor x is not

¹In fact, only 22% of the total length of trunk lines in Nagoya, Japan is covered by beacons of the VICS (vehicle information and communication system) [16]. The headquarters of Toyota is located in Nagoya, which probably has the world's most advanced ITS system.

where the goal is to learn $p(y|t)$, and the predictor variable is the time rather than the paths.

With $p(y|x, \mathcal{D})$, we get information on how much time will be spent on a path x . The basic quantities will be the expected travel time for a path x

$$(2.2) \quad m(x) \equiv \int dy y p(y|x, \mathcal{D}),$$

and the variance

$$(2.3) \quad s(x)^2 \equiv \int dy [y - m(x)]^2 p(y|x, \mathcal{D}).$$

Since GPR is a linear-Gaussian model, we can readily find these as sufficient statistics.

2.2 Travel time data. To obtain the travel time data as Eq. (2.1), we need two types of data: *road network data* and *vehicle tracking data*. For the road network data, organizations such as the Federal Geographic Data Committee², European Umbrella Organization for Geographic Information³, or Geographical Survey Institute⁴ provide publicly available data in a digital format collected through Geographic Information Systems (GIS). GIS is an information system that allows capturing, storing, and analyzing map-related information, such as is typically managed by governmental organizations.

Figure 2 shows a map of a downtown section of Kyoto City, Japan, using intersection data contained in the digital map distributed by the Geographical Survey Institute. In general, digital road network data offered by GIS is given as a weighted graph, where each node represents an intersection and each link represents a road segment between neighboring intersections. In this particular set of Kyoto data, each node is attached to a label in the form of (n_{id}, n_1, n_2) , such as

$$(55406, 135.75370659722222, 35.00014322916667),$$

where n_{id} represents a unique identification number and n_1 and n_2 represent the longitude and the latitude of the intersection, respectively. Similarly, each link has a label in the form of $e_{id}, e_1, e_2, e_3, \dots$, where e_{id} denotes a unique identification number, and the other entries represent the length of the link, a legal speed limit, the mean direction measured with respect to the equator, etc.

For the vehicle tracking data, there is no widely known and publicly available data repository, but some results have been reported in the literature. For example, Ref. [17] uses probe-car data collected in Nagoya City, Japan, where 1,500 taxis equipped with GPS hardware were used to collect

²<http://www.fgdc.gov/>

³<http://www.eurogi.org/>

⁴<http://www.gsi.go.jp/ENGLISH/index.html>

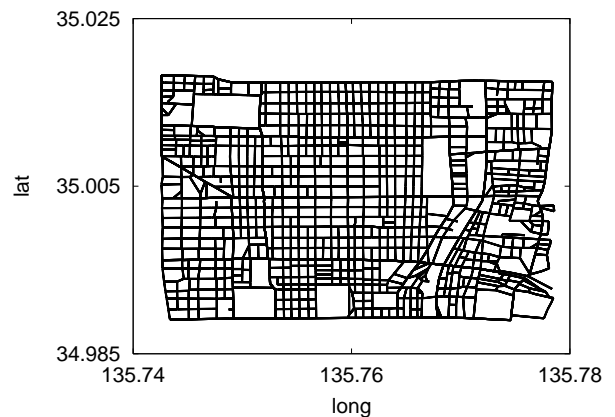


Figure 2: Downtown Kyoto City map plotted using latitude and longitude information of intersections.

time sequences of data chunks, each of which includes a vehicle identification, time, location, and some status data for the vehicles (such as on-off flags for brakes and wipers). These data chunks are either collected periodically or sent via a cellular phone network in an event-driven fashion. Although associating tracking data with a road network data is not trivial, there are established methods for that purpose as in Refs. [20, 15, 4] and references therein. By processing tracking and road network data using a map-matching algorithm, we can get a collection of travel data as in Eq. (2.1).

Because of the limited availability of tracking data, we use an agent-based traffic simulator developed at IBM Tokyo Research Laboratory [10] to generate vehicle tracking data based on road network data. Our simulator generates tracking data by instantiating each vehicle as a Java object, which is arbitrarily programmable to define its individual behaviors and its interaction with other agents. One general test to check if a simulator is realistic is to see if it correctly reproduces a meta-stable state between the free-flow and congestion states [18]. We confirmed that our simulator successfully reproduces the meta-stable state, and therefore, we believe that our travel time data shares essential aspects with the real transportation traffic.

2.3 Notation. This paper employs the following notation. Vectors are represented by bold face such as \mathbf{y}_N . All vectors are assumed to be column vectors. The transpose is denoted by \top . Matrices are represented by sans serif such as K , and the (i, j) element of a matrix is represented as $K_{i,j}$. Identity matrices are represented as I_q , where q is the dimensionality.

3 Related work

As mentioned in the introduction, most of the existing studies on travel time prediction are based on the observer's view. In this section, we first look at studies using this viewpoint. Next, we look at recent studies of trajectory mining, and discuss their relationships with the present paper. Finally, we consider recent studies on GPR.

3.1 Travel time prediction. If a statistically sufficient amount of data is provided for a particular link, time-series modeling with AR (and the related) models seems to be a promising approach. In fact, AR-based modeling was proposed in the 70s [1], and great efforts have been devoted to refining the models since then. One subtle problem in AR-style modeling is how to handle nonstationarities. If extensive amounts of traffic history data are available, then seasonal adjustment techniques known from economics can be used. However, this is true for only a limited number of high-traffic links in the studied city. Even when true, nonstationarities in transportation traffic are so significant in many cases, that compensating for the effects of nonstationarities is a challenging problem.

To overcome the practical limitations of the traditional AR modeling, growing attention has been paid to “nonparametric” prediction techniques in recent years. Examples include k -nearest neighbor (k -NN) regression methods for time-series subsequences [5] and for feature vectors [23]. While these methods report successful improvements in the prediction accuracy in some cases, their utility is still limited to high-traffic links where extensive historical data is available.

Another research trend in the observer's view is to extend a time-series model to include spatial correlations. One natural approach in this direction is to use multivariate AR models. For example, Nakata and Takeuchi [17] showed that the accuracy of travel times can be improved by incorporating the information from neighboring links. Similar research appears in [31], where the traffic for each link is estimated by using the neighboring links defined in a given Bayesian network. On the other hand, Zhang et al. [9] proposed a nonparametric density estimation method, where the density is defined in a two-dimensional space spanned by the time and the position along each single link. Although these studies consider the essential feature of spatial correlation, predicting time for low-traffic (or no sensor) links is still hard, and this can be a drawback in practice.

To predict the travel time along a path including some secondary roads, which may not be equipped with sensors, the simplest approach would be to use static information about the links, such as the lengths and the legal speed limits. However, this type of approach will give large inaccuracies in predicting the travel times, since it lacks actual information such as traffic jams. In addition, it does

not offer any way to estimate the reliability of a prediction. We would like to know the variance, as well as the mean.

3.2 Trajectory mining. Our task is essentially to associate the travel time with a trajectory. Although little has been done in the context of travel time prediction, trajectory modeling is currently attracting attention in the data mining community.

Given a set of trajectories, one of the fundamental tasks is how to define the (dis)similarity between them. For this purpose, one natural approach is to use the idea of elastic matches between trajectories. Examples include classical dynamic time warping [2, 11] (DTW) and the longest common subsequences [26]. To apply these techniques to our problem, however, one has to take account of the constraint that trajectories are defined on a map, not on a translationally uniform space. Tiakas et al. [25] addressed this issue, and defined a mathematically well-defined distance between trajectories, although their goal was not travel-time prediction. In particular, they introduced the idea of sub-trajectory decomposition to compare trajectories of different lengths. Similarly, Lee, Han, and Whang [13] recently proposed a trajectory clustering method, where each trajectory is first partitioned into an optimal number of the sub-trajectories, and then sub-trajectories are grouped according to a heuristic geometric distance function. In a sense, our string kernel can be viewed as a simplified implementation of these “partition and compare” strategies, although these studies do not address travel-time prediction, or the relationship with kernel machines.

In the context of traffic modeling, Krigel et al. [12] proposed a method for estimating traffic density at any location on a map. Their method can be understood as counting the number of shortest paths repeatedly generated under different OD pairs. While their work deals directly with trajectories, and thus is based on the driver's view, it differs from ours since their goal is traffic density modeling and they are based on the assumption of shortest paths.

3.3 Computational issues in Gaussian process regression. GPR is known as a useful probabilistic regression method due to its theoretical simplicity and excellent generalization ability [22]. However, one serious problem in GPR is the computational cost. Naive implementations need $O(N^3)$ time for inverting a kernel matrix. Speeding up GPR is an active research area, and many approaches have been proposed to date. One popular approach is to reduce the size of the problem in some sense. Examples include Nyström's method [28] and pseudo-input-based methods [21, 24]. However, one can say that these methods are not mature enough to use in critical applications such as ours, since most of them are sensitive to a parameter choice [27], or require solving a complicated non-convex optimization

problems.

For relatively stable methods allowing a global solution, Gibbs suggested using the conjugate gradient (CG) method rather than explicit matrix inversion [7]. CG is known to show excellent performance when the matrix is sparse, and is the method of choice in such cases. However, CG can not be used for finding hyperparameters such as the noise level, where a matrix trace and a determinant are involved. Also, CG is known to be somewhat unstable when the matrix is dense and the condition number is very large, due to the nature of Krylov subspace [8].

To summarize, we believe that studies for speeding up GPR are still too immature to use in critical applications. Therefore we employ a simple but stable approach based on Cholesky factorization, as explained later.

4 Regression models for trajectories

In this section, we explain the regression model for travel time prediction for a path. The point is that we use only the similarity between paths rather than using the “coordinates” of the exploratory variable. This can be done by kernel-based regression methods such as GPR. To the best of our knowledge, this is the first attempt to apply kernel machines to the task of trajectory-based travel time prediction.

4.1 Gaussian process regression. As mentioned above, the travel time y is expected to have some complex relationship with the path x . Thus simple parametric models such as linear or polynomial functions will be inappropriate for this problem. We first suppose that the n -th travel time in \mathcal{D} is probabilistically represented through observation noise variance σ^2 as

$$(4.4) \quad p(y^{(n)}|f_n) = \mathcal{N}(y^{(n)} - \bar{y}|f_n, \sigma^2),$$

where $\mathcal{N}(\cdot|f_n, \sigma^2)$ denotes the Gaussian distribution with the mean f_n and the variance σ^2 , and the mean \bar{y} is defined as

$$\bar{y} \equiv \frac{1}{N} \sum_{n=1}^N y^{(n)}.$$

Notice that each data point is attached by a latent variable or a parameter f_n , so that in principle any curve can be represented with this model. This is a general feature of nonparametric models.

To avoid overfitting by this N -parameter model, GPR next assumes a prior distribution on f_n as

$$(4.5) \quad p(\mathbf{f}_N) = \mathcal{N}(\mathbf{f}_N|\mathbf{0}, \mathbf{K}),$$

where $\mathbf{f}_N = (f_1, \dots, f_N)^\top \in \mathbb{R}^N$, and the (i, j) element of the covariance matrix \mathbf{K} is given by the kernel function between paths $x^{(i)}$ and $x^{(j)}$, as denoted by $k(x^{(i)}, x^{(j)})$ (we neglected to denote the dependence on $\{x^{(n)}\}$ above).

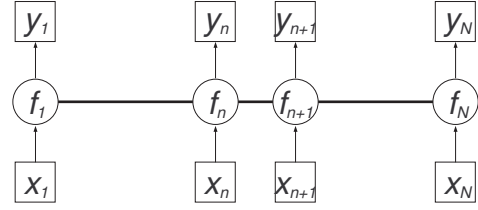


Figure 3: Graphical representation of Gaussian process regression [22]. Notice that latent variables (f_n s) are coupled with each other through the kernel function, while the outputs (the travel times $y^{(n)}$) are independent.

Equations (4.4) and (4.5) represent the only two assumptions in GPR. Figure 3 shows this with a graphical representation of the model. For each element of the target data $y^{(n)}$, we assign a latent variable f_n . While the $y^{(n)}$ s are independent of $y^{(n')}$ ($n \neq n'$), f_n is correlated with $f_{n'}$ through the kernel matrix \mathbf{K} . The kernel matrix \mathbf{K} controls the degree to which the different paths are coupled. Roughly speaking, if $k(x^{(i)}, x^{(j)})$ is very large, the prior strongly favors the situation where $y^{(n)} \simeq y^{(n')}$.

Now let us find the predictive distribution for an out-of-sample path x . Following the Bayesian perspective, it is defined as

$$(4.6) \quad p(y|x, \mathcal{D}) \equiv \int df p(y|f)p(f|\mathcal{D}),$$

where $p(y|f)$ is defined in Eq. (4.4), and $p(f|\mathcal{D})$ is the posterior distribution of the latent variable. Note that the dependence on x is implicitly included within f (see Figure 3). Since we are working with a linear Gaussian model, this integral can be exactly calculated as

$$(4.7) \quad p(y|x, \mathcal{D}) = \mathcal{N}(y|m, s^2)$$

$$(4.8) \quad m = \bar{y} + \mathbf{k}^\top \mathbf{C}^{-1} \mathbf{y}_N$$

$$(4.9) \quad s^2 = \sigma^2 + k(x, x) - \mathbf{k}^\top \mathbf{C}^{-1} \mathbf{k},$$

where \mathbf{y}_N and \mathbf{k} are defined as

$$(4.10) \quad \mathbf{y}_N = (y^{(1)} - \bar{y}, \dots, y^{(N)} - \bar{y})^\top$$

$$(4.11) \quad \mathbf{k} = (k(x^{(1)}, x), \dots, k(x^{(N)}, x))^\top,$$

and $\mathbf{C} \in \mathbb{R}^{N \times N}$ is defined by

$$(4.12) \quad \mathbf{C} = \mathbf{K} + \sigma^2 \mathbf{I}_N,$$

where \mathbf{I}_N represents the N -dimensional identity matrix (See Appendix A for the derivation).

Notice that the predictive mean m is represented as a linear combination of the training data points:

$$m = \bar{y} + \sum_{n=1}^N b_n (y^{(n)} - \bar{y}),$$

where b_n represents the n -th element of the vector $C^{-1}\mathbf{k}$. In this sense, GPR is a generalization of the kernel regression and k -NN regression methods. Unlike these methods, the coefficients b_n are automatically optimized so that the posterior distribution best explains the data. In addition, GPR can systematically produce a probability distribution rather than just a point estimation.

4.2 Choosing the kernel. If one compares two different paths, the simplest approach would be to look only at the lengths (i.e. longer path takes more time). This could be a zeroth approximation for a particular link on a highway, but will be inaccurate for city roads, where traffic signals and intersections greatly affect the travel times. The movement pattern at an intersection can be described with a tuple of the neighboring links around the intersection. For example, if a north-directed link is followed by a west-directed link, we know that the object made a left turn.

Generalizing this idea, we first represent a path using a sequence of symbols. A symbol may be directions, but we associate the identification itself with each link. Next, we focus on subsequences of length p from the paths, and count the co-occurrences of each subsequence. Our definition of the similarity between paths $x^{(i)}$ and $x^{(j)}$ is given by

$$(4.13) \quad k_p(x^{(i)}, x^{(j)}) = \beta \sum_{\mathbf{u} \in \Sigma^p} N_{\mathbf{u}}(x^{(i)}) N_{\mathbf{u}}(x^{(j)}).$$

This similarity was first introduced in bioinformatics [14], and now is known as the p -spectrum kernel in machine learning. Here are the definitions of the symbols:

- Σ is a set of symbols being used to represent links.
- Σ^p is a set of subsequences of p consecutive symbols.
- $N_{\mathbf{u}}(x^{(i)})$ is the number of occurrences of a subsequence \mathbf{u} in a path (sequence of symbols) $x^{(i)}$.

The coefficient β controls the magnitude of the variance of the prior, and is treated as a hyperparameter to be optimized from data.

4.3 Choosing σ and β . So far we have treated the noise level σ^2 and the magnitude β as constants. In the Bayesian framework, one theoretically consistent approach to choosing these is the evidence approximation. In this approach, they are obtained as the maximizer of the marginal likelihood. If we define

$$\gamma \equiv \frac{\sigma^2}{\beta},$$

the marginal likelihood is written as

$$(4.14) \quad \begin{aligned} \psi(\gamma, \beta) &\equiv \ln \int d\mathbf{f}_N p(\mathbf{f}_N) \prod_{n=1}^N p(y^{(n)} | f_n) \\ &= -\frac{1}{2} \ln \det(C_1) - \frac{1}{2\beta} \mathbf{y}_N^\top C_1^{-1} \mathbf{y}_N - \frac{N}{2} \ln \beta, \end{aligned}$$

omitting a constant term. Here we defined C_1 as

$$C_1 \equiv K_1 + \gamma \mathbf{I}_N,$$

where K_1 is the kernel matrix with $\beta = 1$. Using standard formulas for matrix derivatives (see Appendix B.2), the condition of optimality is given as

$$(4.15) \quad 0 = \frac{\partial \psi}{\partial \gamma} = \frac{1}{2\beta} \mathbf{y}_N^\top C_1^{-2} \mathbf{y}_N - \frac{1}{2} \text{tr}(C_1^{-1})$$

$$(4.16) \quad 0 = \frac{\partial \psi}{\partial \beta} = -\frac{N}{2\beta} + \frac{1}{2\beta^2} \mathbf{y}_N^\top C_1^{-1} \mathbf{y}_N.$$

These equations are solved alternately until convergence. Note that the latter is analytically solved as

$$(4.17) \quad \beta = \frac{1}{N} \mathbf{y}_N^\top C_1^{-1} \mathbf{y}_N$$

for a given value of γ .

4.4 Algorithm summary. Here is a summary of our algorithm. For more details, see Appendix C.

In the *training phase*, we determine the best σ and β with the evidence approximation as follows.

1. Input: Kernel matrix K , vector of travel times \mathbf{y}_N , and initial values for σ and β .
2. Algorithm: Solve Eqs. (4.15) and (4.16) alternately until convergence.
3. Output: σ^2 and β that maximize ψ .

For initial values, one reasonable approach is to relate them with the variance σ_y^2 , defined by

$$\sigma_y^2 = \frac{1}{N} \sum_{n=1}^N (y^{(n)} - \bar{y})^2.$$

In particular, one can choose the initial values of σ^2 and the diagonal elements of K , which is proportional to β , on the same order of magnitude as σ_y^2 . See Appendix C for more details.

In the *test phase*, we precompute the Cholesky factor L , where $C = LL^\top$, and its inverse L^{-1} as a side product of the Cholesky factorization. We also precompute a vector $\mathbf{h} \equiv L^{-1} \mathbf{y}_N$.

1. Input: Path x (and precomputed L^{-1} and \mathbf{h}).
2. Algorithm:
 - Compute $\mathbf{l} \equiv L^{-1} \mathbf{k}$.
 - Compute $m = \bar{y} + \mathbf{h}^\top \mathbf{l}$.
 - Compute $s^2 = \sigma^2 + k(x, x) - \mathbf{l}^\top \mathbf{l}$.
3. Output: Predictive mean m and variance s^2 .

5 Experiment

In this section, we test our trajectory-based travel-time prediction method based on a realistic traffic data set.

5.1 Data Generation. To generate travel time data \mathcal{D} , we used real road network data of downtown Kyoto, Japan, as shown in Fig. 2, and vehicle tracking data generated with the traffic simulator [10], as explained in Section 2. Figure 4(a) shows a screenshot of the simulator, where colored dots represent generated vehicles. As shown in the map, Kyoto City has a grid-like road network structure, where the length of each link is about 2 km on average. We picked an origin and destination at the upper left and lower bottom corners in the map, as shown in Fig. 4(a).

We first listed 132 path candidates between the OD pair by using the k shortest path method based on Yen's algorithm [30], some of which are shown in Fig. 4(b). Each path is represented as a sequence of links, and the minimum, maximum, and average number of the size of the sequences is 103, 185, and 140.7, respectively. On simulation, we selected one path out of the 132 candidates for each vehicle in turn.

At each time, the velocity of the vehicles is determined as a function of the vehicular gaps and the legal speed limits at the current positions. The functional form has been determined empirically and we skip describing it here. At intersections, vehicles are programmed to stop for time τ to simulate the waiting times for the traffic lights. We think of this waiting time as an input parameter.

Upon arrival at the destination, the travel time is computed by adding up the transit times of each link and the waiting times at each intersection. In the experiment, 100 out of the 132 paths were randomly selected for training (i.e., $N = 100$), and the remaining 32 data were used for prediction.⁵

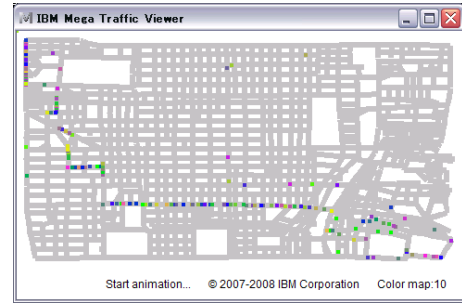
5.2 Methods Compared. We compared three different kernels in this experiment, the ID kernel, direction kernel, and area kernel. Since we are working on the new problem of trajectory-based travel time prediction, traditional time-series models are not directly comparable in our problem setting.

The ID kernel is a p -spectrum kernel defined in Eq. (4.13), where the definition of the alphabet is a set of all the link identifications (IDs). In this case, it is given as

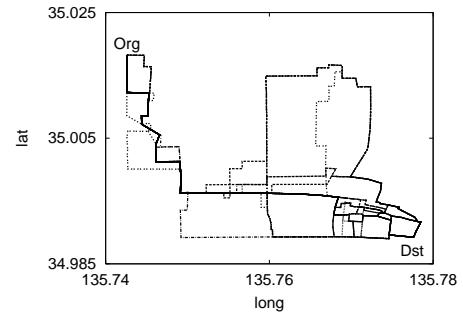
$$\Sigma = \{10150600, 5180612, 5080611, \dots, 11340400\},$$

where each numbers represent an ID of an individual link.

The direction kernel is also a p -spectrum kernel, but the



(a) Screenshot of simulator.



(b) Sample route paths.

Figure 4: (a) IBM's mega traffic simulator. This window visualizes traffic flow with colored rectangles denoting moving vehicles. (b) Example sample route paths between an origin (located at the upper left) and a destination (located at the bottom right).

definition of the alphabet Σ is given by the direction of links

$$\Sigma = \{N, S, E, W\},$$

representing north, south, east, and west directions, which can be identified using the longitude and latitude values of links. Unlike the ID kernel, this kernel only takes account of the directions of links, disregarding the location-specific information contained within the link IDs. In this sense, it compares between paths in a topological manner.

In contrast, the area kernel uses the area enclosed by two paths as the dissimilarity measure. Clearly, the enclosing area is a generalization of the L_1 distance. We use this as a representative of geometrical dissimilarity measures that are based on direct comparison between curve shapes, such as DTW-based dissimilarities (see Sec. 3.2). We define the area kernel as

$$k^{\text{area}}(x^{(i)}, x^{(j)}) \equiv \beta e^{-S(x^{(i)}, x^{(j)})},$$

where the area enclosed by two paths $x^{(i)}$ and $x^{(j)}$ is denoted by $S(x^{(i)}, x^{(j)})$.

⁵The data set is available at http://www.trl.ibm.com/projects/socsim/project_e.htm.

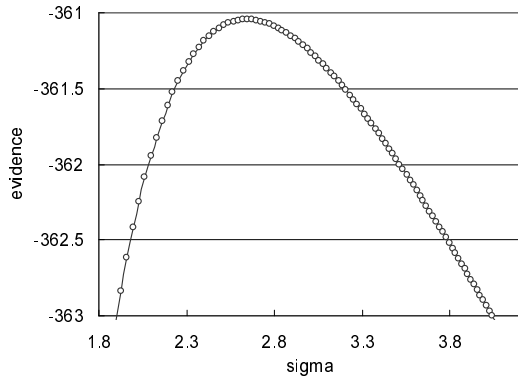


Figure 5: The σ -dependence of the evidence function (ID kernel with $p = 1$).

5.3 Evaluation Metric. We adopt the correlation coefficient, r , between actual and prediction times in the test data. The metric r is simply defined as

$$(5.18) \quad r \equiv \frac{\sum_{n=1}^{N_{\text{test}}} (y^{(n)} - \bar{y})(m(x^{(n)}) - \bar{m})}{\sqrt{\sum_{n=1}^{N_{\text{test}}} (y^{(n)} - \bar{y})^2 \sum_{l=1}^{N_{\text{test}}} (m(x^{(l)}) - \bar{m})^2}}.$$

Here the numerator is proportional to the sample covariance between the predictive mean and the actual travel times in the test data. Quantities with the bar represent the sample mean (over the test data in this case). Note that r is essentially the same as so-called R^2 metric in regression analysis. Since we are solving the new problem of trajectory-based prediction, understandability of the evaluation metric is important, and r is such a one.

5.4 Experimental Results. In our experimental setting, we have two controllable parameters, p in the p -spectrum kernels and τ in the data. In this subsection, we first look at the dependence on p in the ID kernel. Then, with the best value of p , we compare the ID kernel with the other kernels.

Before getting into the details of experimental results using the test data, we briefly look at the training phase. Figure 5 shows the σ -dependence of the evidence function of the ID kernel for $p = 1$ with the optimal value of $\beta = 114.4$. Note that the resulting diagonal elements of K are on the same order of magnitude as $\sigma_y^2 = 96.5^2$. This is reasonable since this means that the variance of the prior was given as a value rather close to the variance of y . As shown in the figure, ψ has a single global maximum at $\sigma = 2.64$. Thanks to the unimodal shape, finding the maxima is numerically easy in this case.

p -dependence. We calculated the predictive mean $m(x)$, changing the value of p with the ID kernel. Figure 6

shows the result, where r is given as a function of p for different values of τ . As shown, the dependence on p is not considerable when $\tau = 0$. This result implies that the total travel time in this case mostly comes from the individual links, and the turning patterns at the intersections give minor effects. In contrast, for $\tau = 10$ and 20 , the figure shows maxima at $p = 2$, showing the fact that moving patterns at intersections play a crucial role in travel-time prediction. In fact, in the case of $\tau = 10$, the turning times account for 28.7% of the total travel time on average.

These results can be understood as interplay between intra-link and inter-link dynamics. By definition, the $p = 1$ kernel represents only intra-link behaviors, while the $p = 2$ also covers the inter-link dynamics. A larger value of τ naturally leads to a larger contribution of the inter-link term. Our finding is that this interplay is well described with the string kernel with $p = 2$, at which intra- and inter-link effects are well balanced. Since τ should be always finite in real traffic data, we conclude that the string kernels with $p > 1$ work well in practical travel-time prediction.

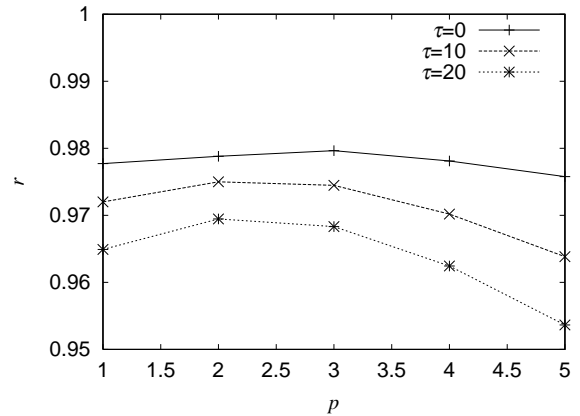


Figure 6: The correlation coefficient as a function of p for $\tau = 0$ (solid line), $\tau = 10$ (broken line) and $\tau = 20$ (dotted line).

Comparing different kernels. Next, we compared the ID, direction, and area kernels. In the ID and direction kernels, p is fixed to 2. Figure 7 shows the scatter plot between the predictive and observed travel times in second. In the figure, we see that the points distribute closely along the diagonal line with the ID and direction kernels. This means a good agreement between the predictions and observations. The r values are summarized in Table 1, showing the highest accuracy 0.980 in the ID kernel. We can say that this value of r is satisfactory enough, if we consider the fact that conventional technologies are not capable of predicting the travel time for the paths including low-traffic links, and our approach is the first one that enables us to do that.

It is interesting to see that the predictive accuracy of the direction string kernel is comparable to that of the ID kernel, in spite of the fact that this kernel uses only four kinds of alphabets representing the direction. On the other hand, in spite of the fact that the enclosing area more accurately calculates the dissimilarity in the geometrical sense, the predictive accuracy of the area kernel is much worse than the others. The accuracy is too inferior to consider that other metrics based on direct comparison between trajectory shapes would produce competitive results. In fact, in the road network we used, some of the main streets are neighboring to narrow side roads, but the legal speed limits for these roads may differ by a factor of 3. In this case, the geometrical dissimilarity will be almost zero, but the expected travel times should be greatly different.

Predictive variance. We computed the predictive variance for each prediction, and showed values of the averaged standard deviation in Table 1, where \bar{s}^2 is defined by

$$\bar{s}^2 = \frac{1}{N_{\text{test}}} \sum_{n=1}^{N_{\text{test}}} s(x^{(n)})^2.$$

The summation runs over the test samples in this definition. In the table, we see that the ID kernel produces a much smaller \bar{s}^2 value.

Although analyzing the risk of the prediction in terms of the predictive variance is of particular interest, we would point out that the calculated predictive variances might be rather small as compared to the order of magnitude of \hat{y} . Specifically, we first listed paths using a k -shortest path algorithm, and then calculated the travel time using a deterministic agent-based simulation. Considering a variety of circumstances that might happen in the real-world situations, it is possible that the data we used captures only a small portion of the variety. Further investigation on this point is an interesting research issue from the simulation side, and we leave that to our future work.

6 Conclusion

We have proposed a method for predicting the travel time for an arbitrary path. We formulated our problem as regression where the target variable is the travel times while the predictor is the paths, or trajectories, not the time as in standard time-series modeling. Our method allows predicting the

Table 1: r and averaged s^2 values for different kernels ($\tau = 10$).

	ID	direction	area
r	0.980	0.933	0.059
$\sqrt{\bar{s}^2}$	4.5	10.0	10.3

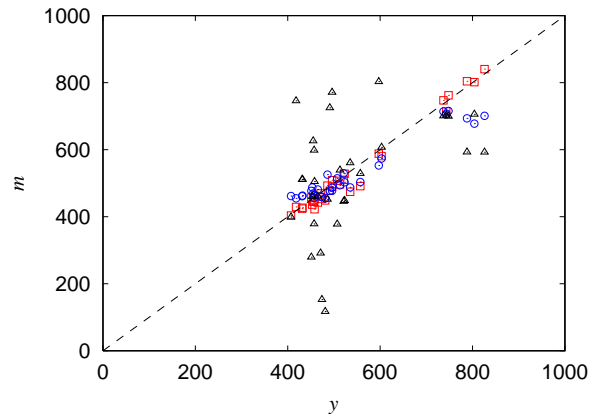


Figure 7: Comparison between the predicted (m) and actual (y) travel times with the ID kernel (\square), direction kernel (\circ), and the area kernel (\triangle). The solid line represents $y = m$, showing perfect agreement.

travel time for paths including links with little traffic history data. This is in contrast to traditional time-series modeling where travel time analysis is performed mainly for particular links with heavy traffic.

Our innovation was to use only the similarity between different paths in regression, and for that purpose, we proposed to use string kernels that are known in bioinformatics. Our second new idea was to use a nonparametric probabilistic kernel machine, Gaussian process regression, for probabilistic travel-time prediction.

Using travel-time data generated with an agent-based traffic simulator and a real road network, we demonstrated that our approach is capable of making accurate probabilistic predictions. In particular, we found that the string kernel with $p = 2$ gives the best performance.

For future work, first, it would be interesting to conduct extensive experiments on more realistic settings in terms of the variety of travel times and trajectories. Second, while we tested only the string and area kernels in this paper, studying other similarity metrics for trajectories would be an interesting research issue. Finally, comparing different kernel regression methods would be also an interesting future work.

Appendix

A Derivation of the predictive distribution.

This Appendix derives Equations (4.7)-(4.9). Introducing additional integral variables \mathbf{f}_N , the definition (4.6) is written as

$$(A-1) \quad p(y|x, \mathcal{D}) = \int d\mathbf{f} d\mathbf{f}_N p(y|\mathbf{f})p(\mathbf{f}|\mathbf{f}_N)p(\mathbf{f}_N|\mathcal{D}).$$

If we use the notations of

$$\begin{aligned}\mathbf{y}_{N+1} &= (y - \bar{y}, y^{(1)} - \bar{y}, \dots, y^{(N)} - \bar{y})^\top \\ \mathbf{f}_{N+1} &= (f, f^{(1)}, \dots, f^{(N)})^\top,\end{aligned}$$

Equation (A-1) is written simply as

$$(A-2) \quad p(y|x, \mathcal{D}) \propto \int d\mathbf{f}_{N+1} p(\mathbf{y}_{N+1}|\mathbf{f}_{N+1})p(\mathbf{f}_{N+1}),$$

where we used the definition of conditional probabilities:

$$p(\mathbf{f}_N|\mathcal{D}) \propto p(\mathbf{y}_N|\mathbf{f}_N)p(\mathbf{f}_N).$$

To get the final expression for $p(y|x, \mathcal{D})$, we note

$$\begin{aligned}p(\mathbf{y}_{N+1}|\mathbf{f}_{N+1}) &= \mathcal{N}(\mathbf{y}_{N+1}|\mathbf{f}_{N+1}, \sigma^2\mathbf{I}_{N+1}) \\ p(\mathbf{f}_{N+1}) &= \mathcal{N}(\mathbf{f}_{N+1}|\mathbf{0}, \Sigma),\end{aligned}$$

where, for the kernel matrix \mathbf{K} and the kernel vector defined in (4.11),

$$\Sigma \equiv \begin{pmatrix} k(x, x) & \mathbf{k} \\ \mathbf{k}^\top & \mathbf{K} \end{pmatrix}.$$

Using the well-known additive nature of the covariance matrices of the Gaussian, we have

$$p(y|x, \mathcal{D}) \propto \mathcal{N}(y|\mathbf{y}_{N+1}|\mathbf{0}, \sigma^2\mathbf{I}_{N+1} + \Sigma).$$

We get the equality by dividing the right hand side by $\mathcal{N}(\mathbf{y}_N|\mathbf{0}, \sigma^2\mathbf{I}_N + \mathbf{K})$. This amounts to partitioning \mathbf{y}_{N+1} into (y, \mathbf{y}_N) , and conditioning y given \mathbf{y}_N . Using the standard partitioning formula for Gaussians (see Eqs.(B-1)-(B-3)), we see that Eqs. (4.7)-(4.9) give the predictive distribution.

B Mathematical formulas

This section summarizes useful mathematical formulas.

B.1 Partitioning formula for Gaussians. Consider a Gaussian distribution $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \Sigma)$. If we partition \mathbf{y} as $\mathbf{y} = (\mathbf{y}_a, \mathbf{y}_b)^\top$, and correspondingly,

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix},$$

the conditional distribution of \mathbf{y}_a given \mathbf{y}_b is given by

$$(B-1) \quad p(\mathbf{y}_a|\mathbf{y}_b) = \mathcal{N}(\mathbf{y}_a|\boldsymbol{\mu}_{a|b}, \Sigma_{a|b}),$$

where

$$(B-2) \quad \boldsymbol{\mu}_{a|b} \equiv \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{y}_b - \boldsymbol{\mu}_b)$$

$$(B-3) \quad \Sigma_{a|b} \equiv \Sigma_{aa} + \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}.$$

See Ref. [3] for the derivation.

B.2 Matrix derivatives. Assume that an $N \times N$ matrix \mathbf{C} contains a scalar parameter α . For derivatives with respect to α , the following holds.

$$\begin{aligned}\frac{\partial}{\partial \alpha} \log \det \mathbf{C} &= \text{tr} \left(\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \alpha} \right) \\ \frac{\partial}{\partial \alpha} \mathbf{C}^{-1} &= -\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \alpha} \mathbf{C}^{-1}\end{aligned}$$

For proof and detailed discussions, see, e.g. [6].

C Implementation details

This section explains implementation details of our approach, and summarized the algorithm.

C.1 Cholesky factorization. As mentioned in Section 3, computational cost is an issue in GPR. Although several methods have been proposed for speeding up GPR [7, 28, 21, 24], these methods are still at a premature stage to apply directly to critical applications. Therefore, we believe that it is more practical to use a relatively stable method for a moderate N that allows a global solution.

Our implementation of GPR is based on Cholesky factorization. In contrast to CG, Cholesky factorization can be used in hyperparameter learning, and is extremely stable for both dense and sparse matrices. Let us look briefly at how it works. Cholesky factorization decomposes a positive definite matrix into the product between lower triangular matrices. For \mathbf{C} , the decomposition is written as

$$\mathbf{C} = \mathbf{L}\mathbf{L}^\top,$$

where \mathbf{L} is a lower triangular matrix. Direct elementwise comparison between both sides leads to

$$\begin{aligned}L_{i,i} &= \left[C_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2 \right]^{1/2} \\ L_{i,j} &= \frac{1}{L_{j,j}} \left[C_{i,j} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k} \right]\end{aligned}$$

for $i = j + 1, j + 2, \dots, N$, from which the entries of \mathbf{L} are computed. As suggested by these simple equations, Cholesky factorization is numerically very stable, although it takes $N^3/6$ operations.

C.2 Optimizing hyperparameters. In the training phase, we need to solve the equation $\partial\psi/\partial\gamma = 0$. Since this is a single-parameter equation, efficient numerical approaches that are much more efficient than gradient methods are available. In our implementation, we use Brent's method [19], which is an extension of the bisection method. Brent's method needs the upper and lower bounds of the search domain. For a value of β , one reasonable choice for

the initial values will be $(\gamma_1, \gamma_2) = (a\sigma_y^2, b\sigma_y^2)/\beta$, where $a \sim 0.5$ and $b \sim 25$ are useful in many cases. Starting with the initial values, the algorithm locates a solution by sequentially narrowing the domain. For the initial value of β , we chose it so that the diagonal elements of K is the same order as σ_y^2 . For instance, in the case of the ID kernel, we used $\beta = 100$ as the initial value.

Until convergence, we need to repeatedly evaluate the gradient at the bounds and the present value of σ . Cholesky factorization is also useful for this purpose. In Eq. (4.15), we have two terms: $\mathbf{b}^\top \mathbf{b}$ and $\text{tr}(\mathbf{C}^{-1})$. Given the Cholesky factorization $\mathbf{C} = \mathbf{L}\mathbf{L}^\top$, we note

$$\text{tr}(\mathbf{C}^{-1}) = \sum_{i \leq j} (\mathbf{L}^{-1})_{i,j}^2,$$

and that \mathbf{L}^{-1} is readily found as a side product of the Cholesky factorization (see, e.g. Sec. 2.9 in [19]). With \mathbf{L}^{-1} , we can directly compute \mathbf{b} in $O(N^2)$ time. Otherwise, we can solve $\mathbf{L}\mathbf{L}^\top \mathbf{b} = \mathbf{y}_N$ using forward and backward substitutions [19] that take $O(N^2)$ operations at worst.

C.3 Making predictions. With a matrix \mathbf{C} having an optimal σ , we precompute the inverse of the Cholesky factor \mathbf{L}^{-1} as a side product of Cholesky factorization itself. Also, we keep $\mathbf{h} \equiv \mathbf{L}^{-1}\mathbf{y}_N$.

On prediction for an input path x , we first compute the kernel vector \mathbf{k} and a product $\mathbf{l} \equiv \mathbf{L}^{-1}\mathbf{k}$. The predictive mean and variance are easily computed with the vectors \mathbf{h} and \mathbf{l} . For the predictive mean m (Eq. (4.8)), we compute $\mathbf{l}^\top \mathbf{h}$. For the predictive variance s^2 (Eq. (4.9)), we compute $\mathbf{l}^\top \mathbf{l}$ to get $\mathbf{k}^\top \mathbf{C}^{-1} \mathbf{k}$. The computational cost of prediction is $O(N^2)$ when K is dense. Otherwise, thanks to the fact that a resulting \mathbf{L} inherits the sparsity of the original matrix, the cost can be sub-quadratic.

References

- [1] M. S. Ahmed and A. R. Cook. Analysis of freeway traffic time-series data by using Box-Jenkins techniques. *Transportation Research Record*, 722:1–9, 1979.
- [2] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI-94 Workshop on Knowledge Discovery in Databases*. AAAI, 1994.
- [3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
- [4] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proc. Intl. Conf. Very Large Data Bases*, pages 853–864, 2005.
- [5] S. Clark. Traffic prediction using multivariate nonparametric regression. *Journal of transportation engineering*, 129(2):161–168, 2003.
- [6] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 2nd. edition, 1990.
- [7] M. N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, Department of Physics, University of Cambridge, 1997.
- [8] G. H. Golub and C. F. V. Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, 1996.
- [9] C.-M. Hsu and F.-L. Lian. A case study on highway flow model using 2-d Gaussian mixture modeling. In *Proc. IEEE Intl. Conf. Intelligent Transportation System*, pages 790–794, 2007.
- [10] S. Kato, G. Yamamoto, H. Mizuta, and H. Tai. Simulating whole city traffic with millions of multiple vehicle agents. Technical Report RT0759, IBM Research, 2008.
- [11] E. Keogh and M. Pazzani. Scaling up dynamic time warping for data mining applications. In *Proc. ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining*, pages 285–289, 2000.
- [12] H.-P. Kriegel, M. Renz, M. Schubert, and A. Zuefle. Statistical density prediction in traffic networks. In *Proc. SIAM Intl. Conf. Data Mining*, pages 692–703, 2008.
- [13] J. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *Proc. 2007 ACM SIGMOD Intl. Conf. Management of Data*, pages 593–604, 2007.
- [14] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauderdale, and T. E. Klein, editors, *Proc. the Pacific Symposium on Biocomputing*, pages 564–575, 2002.
- [15] T. Miwa, T. Sakai, and T. Morikawa. Route identification and travel time prediction using probe-car data. *International Journal of ITS Research*, 2(1), October 2004.
- [16] T. Morikawa, T. Yamamoto, T. Miwa, and L. Wang. Development and performance evaluation of dynamic route guidance system PRONAVI. *Journal of the Japan Society of Traffic Engineers*, 42(3):65–75, 2007.
- [17] T. Nakata and J. Takeuchi. Mining traffic data from probe-car system for travel time prediction. In *Proc. ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining*, pages 817–822, 2004.
- [18] K. Nishinari, M. Fukui, and A. Schadschneider. A stochastic cellular automaton model for traffic flow with multiple metastable states. *Journal of Physics, A*, 71:2339–2347, 2002.
- [19] H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 2nd. edition, 1992.
- [20] M. A. Qudus, W. Y. Ochieng, L. Zhao, and R. B. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions Journal*, 7(3):157–167, 2003.
- [21] J. Quiñonero–Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.
- [22] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- [23] S. Robinson and J. W. Polak. Modeling urban link travel time with inductive loop detector data by using the k -NN method. *Transportation research record*, (1935):47–56, 2005.
- [24] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt,

editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264, Cambridge, MA, 2006. MIT Press.

- [25] E. Tiakas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djordjevic-Kajan. Trajectory similarity search in spatial networks. In *Proc. Intl. Database Engineering and Applications Symposium*, pages 185–192, 2006.
- [26] M. Vlachos. Elastic translation invariant matching of trajectories. *Machine Learning Journal*, 58(2-3):301–334, 2005.
- [27] C. K. I. Williams, C. E. Rasmussen, A. Schwaighofer, and V. Tresp. Observations on the Nyström method for Gaussian process prediction, http://www.dai.ed.ac.uk/homes/ckiw/online_pubs.html. 2002.
- [28] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688, 2001.
- [29] C.-H. Wu, J.-M. Ho, and D. Lee. Travel-time prediction with support vector regression. *IEEE Trans. Intelligent Transportation Systems*, 5(4):276–281, 2004.
- [30] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.
- [31] C. Zhang, S. Sun, and G. Yu. A bayesian network approach to time series forecasting of short-term traffic flows. In *Proc. IEEE Intl. Conf. Intelligent Transportation System*, pages 216–221, 2004.