On Centralized and Decentralized Architectures for Traffic Applications

Nikola Mitrovic, *Student Member*, *IEEE*, Aditya Narayanan, Muhammad Tayyab Asif, *Student Member*, *IEEE*, Ansar Rauf, Justin Dauwels, *Senior Member*, *IEEE* and Patrick Jaillet

Abstract—**The role of smartphones in traffic applications is typically limited to front end interface. Although smartphones have significant computational resources, which are most likely to increase further in the near future, most of the computations are still performed on servers. In this paper, we study the computational performance of centralized, decentralized and hybrid architectures for Intelligent Transportation Systems applications. We test these architectures on various Android devices. For implementation, we consider Android Software Development Kit (SDK) and Android Native Development Kit (NDK). Numerical results show that recent smartphones take less than one second to estimate the speed for each road segment in a network of 10,000 links from speed measurements at 1,000 links. The proposed decentralized architecture significantly reduces the overhead of the communication network and paves the way for new cooperative traffic applications and operations.**

Index Terms—**Android applications, NDK, Low-dimensional models, speed estimation, traffic prediction.**

I. Introduction

Smartphones currently play a vital role in everyday traffic management operations. Being equipped with global positioning system (GPS) and Internet access, mobile phones are frequently used to collect location and (instantaneous) speed of vehicles [1]. Internal sensors of smartphones such as accelerometers and GPS devices can further provide information about the behavior of drivers and the road conditions [2]. High penetration rate and powerful visualization capabilities of smartphones facilitate delivery of valuable traffic information to the end users (e.g., drivers) [3]. In most traffic applications, all of the computational operations are performed on servers, while the role of smartphones is often restricted as a front end interface. We refer to this as centralized (or traditional) system architecture. Although a typical smartphone possesses significant computational resources, which are most likely to increase in the future, this potential has hardly been explored in traffic applications. The only exceptions are a few safety related applications [4] where all the computations are internally performed with the help of native programming languages.

In this paper, we explore how traffic speed estimation and prediction, and travel time prediction can be implemented in a decentralized manner on smartphones, as opposed to a centralized server. Although this method can be applied to many other different domains, we focus on ITS applications. We use a column based (CX) matrix decomposition method to allocate certain computations from the server to the smartphones. The CX method approximates any arbitrary matrix as a product of a subset of its columns and an extrapolation matrix. The extrapolation matrix contains the relationship functions (coefficients) that expresses every column of the matrix in terms of the basis provided by a small subset of the original columns. The extrapolation matrix is inferred from the training data set, for a given subset of columns. This subset of the columns can be selected according to different criteria [5]. The CX method has proven to be efficient in traffic networks since traffic parameters (e.g., speed, volume) across different roads tend to be related [6]. The CX method has been deployed for applications of compression, estimation and prediction where the traffic conditions in the entire network (represented as a matrix) are assessed using the information from a few road segments (columns of the matrix) [7]. However, this method has previously not been deployed and systematically tested in an Android environment.

For the purpose of the study, we have developed an Android app that aims to assess the traffic conditions of the entire network from the data of a subset of the road segments. The developed app estimates the traffic conditions in the entire network through vector-matrix multiplications where: (i) the row vector contains the traffic data (e.g., speed) for a subset of the links; (ii) the extrapolation matrix (or relationship matrix) encompasses relationship functions between this subset of the links (defined in (i)) and the entire network [7]. The extrapolation matrix can be pre-computed on the server and stored in the phone memory. In this case, the server sends the traffic data for the predefined subset of the links to the smartphones where the extrapolation is performed. We refer to this as the Decentralized Non-Adaptive (D-NA) mode of operations. The D-NA architecture aims to reduce the overhead of the communication network since only the subset of the predefined links is explicitly monitored.

In the D-NA mode of operations, we assume that traffic

This paper was presented in part at the IEEE Intelligent Transportation Systems Conference, Qingdao, 2014

Nikola Mitrovic, Aditya Narayanan, Ansar Rauf and Justin Dauwels are with the School of Electrical and Electronic Engineering, College of Engineering, Nanyang Technological University, Singapore, 639798; nikola001@e.ntu.edu.sg, jdauwels@ntu.edu.sg.

Muhammad Tayyab Asif was with Nanyang Technological University when this work was done. He is now with IBM Research Collaboratory - Singapore.

Patrick Jaillet is with the Department of Electrical Engineering and Computer Science, School of Engineering, and also with the Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is also with the Center for Future Urban Mobility, Singapore-MIT Alliance for Research and Technology, Singapore (e-mail:jaillet@mit.edu).

The research described in this project was funded in part by the Singapore National Research Foundation (NRF) through the Singapore MIT Alliance for Research and Technology (SMART) Center for Future Mobility (FM).

data, for the predefined subset of road segments, will be available during all time instances. However, this may not be a reasonable assumption due to malfunctions of fixed sensors and the nature of mobile sensors (e.g., probe vehicles) [8]. In such cases, we can still use the server to collect data from a subset of available sensors and send it to the smartphone. Then, the smartphone can compute the corresponding extrapolation matrix for the obtained data and use it to perform network extrapolation. We refer to this as the Decentralized ADaptive (D-AD) mode of operations. The proposed D-AD architecture is a stepping stone toward cooperative (or server-free) mode of operations, since information about traffic conditions may be provided directly by other probes.

We use the decentralized modes of operations for other application domains, beyond estimation, such as traffic and travel time predictions. In these applications, the server performs traffic prediction for a subset of the links and sends the predicted data to the smartphone where the extrapolation is performed. Similar to the application of network estimation, we can also compute the relationship matrix on the server (D-NA mode) or the smartphones (D-AD mode). With the help of the relationship matrix, the smartphone app can compute the future conditions for the entire network and different prediction horizons. This information can then be used for travel time calculations of potential routes. For each road segment of these routes, the algorithm estimates when the driver will be traveling through it, and chooses the prediction horizon accordingly. Then, the smartphone infers the link travel time by dividing the predicted traffic speed by the length of the segment. Finally, the smartphone computes the total travel time as a sum of the link travel times.

We evaluate the execution time of the decentralized architecture for different Android platforms and smartphone devices using traffic data from the Singapore network. For development platforms, we consider the commonly used native development kit (NDK) and software development kit (SDK). We also consider different generations of smartphone devices, and we evaluate the performance of different smartphones with the help of the Android emulator [9]. We measure the execution time of the app by inserting monitoring functions into the Android operating system. Since the execution time depends upon the size of the underlying network, we assess the app performance in three test network comprising 2,156 (highway network), 5,000 and 10,000 links.

Our experiments show that the proposed modes of operations represent promising alternatives to the commonly used centralized framework. Decentralized mode of operations can be efficiently deployed if the development platform of the traffic app relies on native development kit (NDK) platform where the physically complex operations are coded in native languages. In particular, the NDK development platform outperforms the traditional software development kit platform (SDK) by 93-95% in the case of the complex problems. Furthermore, the NDK execution time of the app is highly acceptable for all tested modes of operations, even in the case of large traffic networks. The experimental results also show that memory requirements of the traffic app are satisfactory in most of the tested cases.

Nowadays, traffic applications are increasingly relying on various sensors to collect data about mobility conditions. These applications then use these large data sets to improve user's mobility experience. The centralized architecture, however, can potentially become a bottleneck in terms of scalability, connectivity and service delivery, in the future. At the same time, smartphones offer great untapped potential to explore decentralized and hybrid architectures for traffic related applications. These architectures can also prove useful for device-to-device communication and vehicle-to-vehicle communications. In this study, we consider the problem of traffic state estimation in the context of above mentioned architectures. Our main contributions in this regard are as follows: We investigate a decentralized approach to traffic estimation and prediction, implemented on smartphones. We also discuss the choice of different development platforms for Android OS for traffic related applications.

The rest of the paper is structured as follows. In Section II we briefly review the relevant literature. In Section III we introduce the low-dimensional traffic models and explain how we deploy them on smartphones for applications of traffic estimation and prediction, and travel time prediction. In Section IV we explain the different modes of operations and programming models that we use to test our app. In Section V we explain the tested scenarios and describe the analyzed data sets in this paper. In Section VI we provide and discuss results of our experiments. In Section VII we summarize our contributions and propose topics for future work.

II. Related work

In this paper, we explore the computational capabilities of smartphones for different modes of operations and traffic applications. Being equipped with smart sensors and high visualization capabilities, smartphones have found many applications in the domain of intelligent transportation systems (ITS) [1], [2], [10]–[18]. Numerous smart sensors such as GPS, manometers, barometers etc., have been used to detect the speed, location and activity of the travelers, mode of the transportation and user (driver) behavior [1], [2], [10]–[14]. Conversely, smartphones are frequently seen as a convenient way to deliver real-time traffic information to the travelers. This information encompasses current and future traffic conditions, relevant transit information, as well as step-by-step guidance from origin to the destination [15]–[18]. In these studies, traffic apps often deal with the computationally light operations in a centralized framework and typically do not use all available CPU resources. Although smartphones possess significant computational power nowadays, these resources, to our best knowledge, have not been evaluated for computationally intensive traffic applications.

Development of traffic (and other Android) applications is typically done in the Java programming language with the help of Android Software Development Kit (SDK), third party libraries, and other useful tools [19]. The SDK platform is "easy-programmable", portable, and supported by most Android features such as services and content providers [20]. Unlike the SDK, the Native Development Kit (NDK) platform uses C/C++ libraries for computation components of the intensive smartphone applications such as video games, image processing applications, etc. [20], [21]. Recent studies on the Android platform show that applications written in C/C++ achieve better performance than those in Java [20]–[22]. These benefits of using native languages vary across the applications and underlying architecture of the devices [20]–[22]. If only the front-end of the application is implemented on the mobile device (such as in [1], [2], [10]–[18]), then the improvement in performance may not be that significant [20]–[22]. For the intensive smartphone applications, this improvement might be significant [20]–[22]. Although the NDK development platform may yield a significant improvement in performance, it is still not clear whether it is sufficient for the ITS applications at hand. To answer this question, one needs to implement and test the NDK development platform for various traffic applications.

III. Column based method for traffic applications on smartphone

In this section, we explain how we use the column based (CX) matrix approximation to allocate certain computations from the server to smartphones, for applications of traffic speed estimation and prediction, and travel time prediction.

A. Column based (CX) matrix approximation

The column based method approximates a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ as the product of low dimensional matrices $\mathbf{C} \in \mathbb{R}^{m\times c}$ and $\mathbf{X} \in \mathbb{R}^{c\times n}$, such that $\hat{\mathbf{A}} = \mathbf{CX}$ [23]. Matrix $\mathbf{C}$ contains c columns of the matrix $\mathbf{A}$ (see Fig. 1a). The relationship (or extrapolation) matrix $\mathbf{X}$ expresses every column of $\mathbf{A}$ in terms of the basis provided by the columns of $\mathbf{C}$ [23]. For given matrices $\mathbf{C}$ and $\mathbf{A}$, we compute the matrix $\mathbf{X}$ as:

$$\mathbf{X} = \mathbf{C}^{+}\mathbf{A}, \tag{1}$$

where $\mathbf{C}^{+}$ is the Moore-Penrose pseudo-inverse of matrix $\mathbf{C}$.

There are a few sampling strategies to select c columns [5]. The SVD based sampling technique, which assigns higher selection probability to the columns with larger variations, has the best performance [5], [23]. The SVD strategy assigns the selection probability to each column $\{\mathrm{P}_{a_i}\}_{i=1}^{n}$ in proportion to the Euclidean norm of top k right singular vectors of the matrix:

$$\mathrm{P}_{a_i} = \frac{1}{k}\sum_{j=1}^{k} v_{ij}^2 \quad \forall\ i = 1,...n, \tag{2}$$

where v_{ij} is the i-th coordinate of j-th right singular vector.

The other selection strategy that performs reasonably well is random sampling [5]. This algorithms assigns equal selection probabilities to each column of the matrix $\mathbf{A}$. We refer to [5], [23] for detailed information about the low-dimensional network representation and different sampling methods.

B. CX method for traffic speed estimation

We start by explaining how we use column based (CX) low-dimensional models to assess current traffic conditions in the entire network through straightforward vector matrix multiplication. For this purpose, we consider the traffic data in the form of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ where the columns of the matrix $\{\mathbf{a}_i\}_{i=1}^{n}$ contain traffic data from different roads $\{s_i\}_{i=1}^{n}$. In that way we can write the traffic data matrix as $\mathbf{A} = [\mathbf{a}_1 \ ... \ \mathbf{a}_n]$. Fig. 1b shows an arbitrary traffic network with the n road segments. Suppose that the matrix $\mathbf{C}$ contains the observed traffic conditions at c specific locations in the network, such that $\{\mathbf{c}_1,...,\mathbf{c}_c\} \subseteq \{\mathbf{a}_1,...,\mathbf{a}_n\}$. These c locations are shown as red rectangles in Fig. 1b where $\mathbf{c}^i$ is a row vector that contains the traffic parameter (e.g., speed) at these c specific locations at time i. Then, at time i we can estimate the traffic conditions of the entire network ($\alpha^i \in \mathbb{R}^{1\times n}$) as:

$$\hat{\alpha}^i = \mathbf{c}^i\mathbf{X}. \tag{3}$$

We estimate the current traffic conditions of the entire network by taking the product of traffic data for the subset of the links (given in the form of a row vector) and the relationship matrix (see (3)). Since the latter remains constant at any time i (and for a given subset $\mathbf{c}$) it motivates us to explore whether network extrapolation can be executed on a smartphone. We propose to store the extrapolation matrix in the phone memory and use it to perform extrapolation once the traffic data (for given subset of the links) is fetched from the server. We refer to this as the Decentralized Non-Adaptive (D-NA) method since the relationship matrix is pre-computed on the server in an offline manner.

The D-NA method assumes that traffic data for the predefined subset of the links in the network is available all the time (see Fig. 1b). However, this may not be a reasonable assumption since the traffic sensors are prone to faulty operations and damages (see Fig. 1c) [8]. One alternative is to use the server to re-compute the extrapolation matrix for the available subset of the links (see green circles in Fig. 1c) and send it to the smartphone. However, this is not a feasible solution since the relationship matrix has to be transmitted for each new subset of the explicitly observed links. Another alternative is to re-compute the appropriate extrapolation matrix $\mathbf{X}_*$ on the smartphone as:

$$\mathbf{X}_* = \mathbf{C}_*^{+}\mathbf{A}, \tag{4}$$

where $\mathbf{C}_*$ contains the historical traffic information for the subset of any $\mathbf{c}_*$ locations where the traffic data is currently available (see green circles in Fig. 1c). In this case, the historical traffic data for the entire network, given as matrix $\mathbf{A}$, has to be stored in the phone memory, as part of the smartphone app (see (4)). Finally, we estimate the traffic conditions of the entire network as $\hat{\alpha}^i = \mathbf{c}_*^i\mathbf{X}_*$. We refer to this as the Decentralized ADaptive (D-AD) scenario since the relationship matrix is calculated in an adaptive manner, using the smartphone resources.

We deploy the proposed method for the more realistic scenario where only up-to-date traffic estimates of several potential routes are provided. We estimate the current traffic conditions of only these links ($j = 1,2,...,L$) that lie along the potential routes, as follows:

$$\hat{\alpha}_j^i = \mathbf{c}^i\mathbf{x}_j, \quad \forall\ j = 1, \ ...\ L, \tag{5}$$

where $\mathbf{x}_j$ is j^{th} column of the relationship matrix $\mathbf{X}$. In this case, the list of L segments has to be either stored in the phone memory (for the commonly used routes by a user) or obtained from the server. The approach (5) can be deployed within any of the described decentralized modes of operations.

It is noteworthy that $\mathbf{c}$ (in D-NA case) and $\mathbf{c}_*$ (in D-AD case) are subsets of $\hat{\alpha}$ and this helps to reduce the computational load of the matrix multiplication as follows. Let us assume that c_j ($j \in \{1,\ 2,\ \dots,\ c\}$) is j^{th} element of $\mathbf{c}$ and ℓ-th ($\ell \in \{\ 1,\ 2,\ \dots,\ n\}$) entry of $\hat{\alpha}$. The ℓ-th column of the extrapolation matrix $\mathbf{X}$ has all zeros except the ℓ-th entity which is equal 1. We improve the computation efficiency by storing the c positions of these columns, instead of performing c vector-vector multiplications.

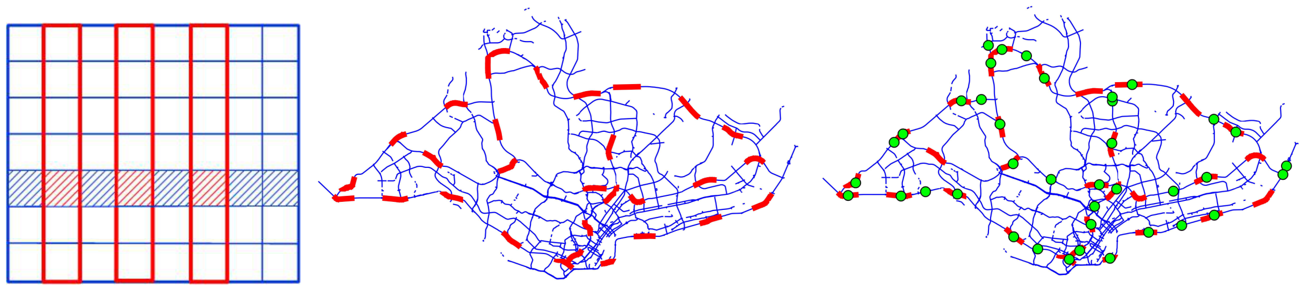

(a) Column based data matrix. (b) Decentralized non-adaptive (D-NA) scenario. (c) Decentralized adaptive (D-AD) scenario.

Fig. 1: *Left:* An example of a matrix (blue) and a subset of columns (red). The explicitly observed red shaded values are used to infer the blue shaded values. *Middle:* An analogous example of traffic network (blue) with the predefined subset of locations where the traffic is explicitly monitored (red). *Right:* Locations of the currently available sensors (green) may not overlap with the subset of predefined locations (red).

Operation Modes		Server Operations				Smartphone Operations				Smartphone Memory Requirement	
		Data Acquisition				Computations					
		Entire Network	SVD Sampling	Uniform Sampling	Prediction	Extrapolation Matrix	Network Extrapolation	Travel-Time Calculation	Data Visualization	Extrapolation Matrix	Historical Data
Centralized		✓			✓				✓		
Decentralized	Non Adaptive		✓		✓	✓	✓	✓	✓	✓	
	Adaptive			✓	✓	✓	✓	✓	✓		✓

Fig. 2: List of the computations performed on the server (orange) and smartphone (green), for different operation modes (red). Memory requirement of the traffic app is given in blue.

C. CX method for speed prediction

Similar to (3), we can obtain the future traffic conditions of the entire network for the k-th prediction horizon as:

$$\hat{\alpha}^{i+k\Delta t} = \hat{\mathbf{c}}^{i+k\Delta t}\mathbf{X}, \tag{6}$$

where Δt is the sampling interval (e.g., 5 minutes). The row vector $\hat{\mathbf{c}}^{i+k\Delta t}$ contains the predicted traffic speed for c selected locations and k^{th} ($k = 1, 2, \ \ldots, \ p$) prediction horizon. These predicted values $\hat{\mathbf{c}}^{i+k\Delta t}$ are computed on the server by means of a state-of-the-art prediction algorithm (e.g., support vector regression (SVR)). Since the extrapolation matrix $\mathbf{X}$ remains unchanged for different prediction horizons, we can use (6) to perform network extrapolation for multiple prediction horizons $(k_{i+\Delta t} \ \ldots \ k_{i+p\Delta t})$. In this case the matrix $\hat{\mathbf{C}} \in \mathbb{R}^{p\times c}$ $([\hat{\mathbf{c}}_{i+\Delta t} \ \ldots \ \hat{\mathbf{c}}_{i+p\Delta t}]^T)$ has to be fetched from the server and sent to the smartphone. Similar to the application of traffic speed estimation, the extrapolation matrix can be either pre-computed on the server or re-computed on the smartphone; leading to the already described D-NA and D-AD scenarios, respectively.

D. CX method for travel time prediction

In the following we extend the idea of compressed prediction to additional practical applications. More precisely, we rely on the compressed prediction method for inferring the future traffic conditions along several potential routes. Next we use the speed predictions to predict the travel times along these routes.

Similar to (5), we predict the traffic speed of the link j and horizon k as follows:

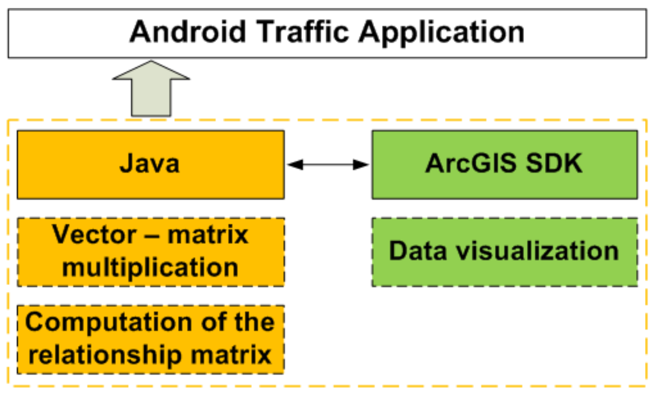
$$\hat{\alpha}_j^{i+k\Delta t} = \hat{\mathbf{c}}^{i+k\Delta t}\mathbf{x}_j. \tag{7}$$

By fetching the matrix $\hat{\mathbf{C}}$ from the server and multiplying each row of the matrix $\hat{\mathbf{C}}$ with the corresponding columns of $\mathbf{X}$, we can compute the expected traffic conditions at different time instances (see (7)). We estimate the prediction horizon (k) when the driver will be traveling through the link s_j as follows [24]:

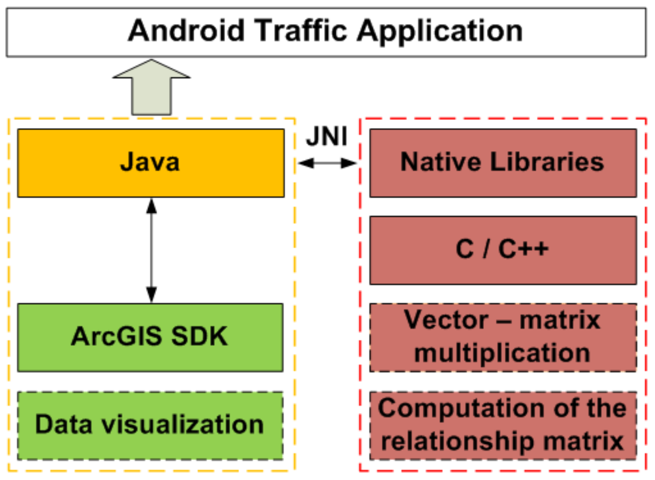
$$k_{s_j} = \left\lfloor \frac{1}{\Delta t}\left(T_o + \sum_{i=1}^{j-1} \frac{l_i}{\hat{v}_{i,k}}\right) \right\rfloor + 1, \tag{8}$$

where k_{s_j} is an integer $([1, 2, \ \ldots \ p])$. l_i is the length of the link s_i. $\hat{v}_{i,k}$ is the predicted speed for the link s_i and prediction horizon k $(k \ \leqslant k_{s_j})$. The variable T_o represent the time offset between the current time and trip starting time. We assume that the server performs route selections by means of a state-of-the-art routing algorithm, and sends the ordered list of L road segments, along the selected route, to the smartphone. As an alternative, the lists of road segments (of commonly used routes) may be pre-stored in the smartphone memory. For each link in the list, we use (8) to compute the prediction horizon k. Then, we apply the computed k in (7) to obtain a link travel time. Hence, the complexity of the travel time computations for the selected route involves L vector-vector multiplications (where each vector has c elements) and straightforward addition of link travel times.

We use the smartphone's resources to predict the travel time and inform the driver about the most likely traffic conditions of the particular link at time when he is expected to travel across this link. This information can help the drivers in deciding the route, mode of transportation and the departure time. As part of future work, we will investigate the option to (partially) perform routing on the smartphone.


Android Traffic Application
Java
ArcGIS SDK
Vector – matrix multiplication
Data visualization
Computation of the relationship matrix

(a) SDK scenario


Android Traffic Application
Java
JNI
Native Libraries
C / C++
ArcGIS SDK
Vector – matrix multiplication
Data visualization
Computation of the relationship matrix

(b) NDK scenario

Fig. 3: Major components of the traffic app for SDK (left) and NDK (right) scenarios. Computation part is coded using either the SDK (orange) or NDK programming platform (red). Graphics part is coded with the help of the GIS SDK module (green).

IV. System Infrastructure

In the following section, we explain the different modes of operations and development platforms that we use to evaluate the computational performance of smartphone devices for the applications of traffic speed estimation and prediction, and travel time prediction.

A. *Mode of operations*

Fig. 2 shows the list of operations that are performed on the server and smartphone, for centralized and decentralized modes of operations. In the centralized (or traditional) approach the server performs all the required computations; i.e., collecting, processing and sending the traffic data to the smartphone. The smartphone is not involved in any calculations in this scenario, instead it only handles the visualization. Specifically, the smartphone overlays the obtained data on the top of underlying road map (see Fig. 4).

In the decentralized operation mode, the server only collects the traffic data for a subset of the road segments in the network and performs predictions for these roads by means of a state-of-the-art prediction algorithm. Next the server sends the collected and predicted data to the smartphone, which uses its own resources to perform the network extrapolation, estimate the travel time along the requested route, and visualize the computed data. We recall that relationship matrix $\mathbf{X}$ can be either computed in advance for the predefined subset of the links and stored in the phone memory (see D-NA approach in Fig. 2) or computed online with the help of computational resources of the smartphone and historical traffic matrix $\mathbf{A}$, stored in the phone memory (see D-AD approach in Fig. 2).

In this study, we evaluate the execution time and memory requirement of the traffic app for the different modes of operations and ITS applications. Although any Android platform can be used to build the app, we rely on the commonly used SDK and NDK approaches which we briefly explain in the following.

B. *Android platforms*

We have developed an app that estimates or predicts the traffic speed by means of the CX method, and then overlays the traffic information on the top of the street map. The proposed smartphone app has two components. The first component conducts the vector-matrix operations on the traffic data. The second component performs visualization, where road segments are colored according to the inferred or predicted traffic speed (see Fig. 4). We have implemented apps in two environments: SDK and NDK. In Fig. 3, we depict the data flow in both implementations. In the former, we use the traditional SDK development platform to perform the computations and visualization operations, with the help of Java computation libraries and the ArcGIS module (see Fig. 3a). In the latter, we use the Java Native Interface (JNI) to call the native applications and libraries in Java code. We use these applications and libraries, written in C/C++, to perform vector-matrix operations. Once the computations have been performed, JNI sends the data to the Java environment where the visualization tasks are performed using the ArcGIS module (see Fig. 3b).

V. Experimental Setup

In this study, we consider the nationwide traffic network in Singapore that contains highways and arterial roads. The variable of interest is the average traffic speed, i.e., the mean speed of all vehicles which traverse a road segment during the given sampling interval of 5 minutes. The Land Transportation Authority provided us experimental data for a period of three months (August - October 2011). We selected 10,000 highway and arterial links that have less than 5% of missing values. We imputed the missing data by means of the Low Dimensional CP Weighted OPTimization (LDCP-WOPT) imputation method as it can deal with large data set [25], [26]. The relative imputation error is negligible ($\sim 2\%$) since only a few percent of data ($\sim 3\%$ on average) is missing [25].

We use this data set for the following two purposes: (i) to evaluate the performance of column based (CX) low-dimensional models for the applications of traffic speed estimation and prediction, and travel time prediction; (ii) to compute the execution time of the proposed app for these applications and different modes of operations. Since the size of the network significantly impacts the required execution

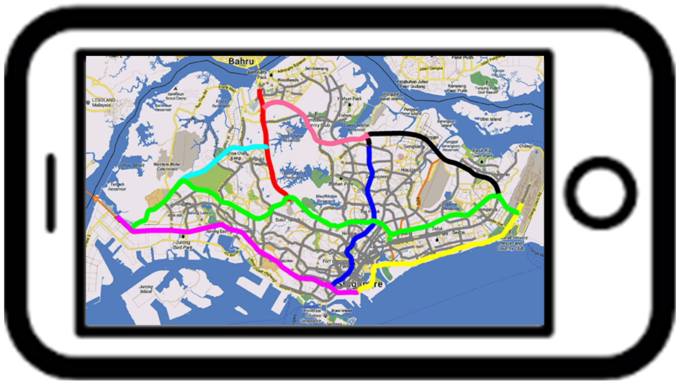


Fig. 4: City-scale traffic network of Singapore with 10,000 links of the arterial (grey) and highway (other colors) categories. The highway network contains 2,156 road segments, divided into 8 highways. Each highway is marked with different color and used as a testing route for travel time calculations.

time, we investigate the smartphone performance for three different networks: (i) highway traffic network that consists of 2,156 links (see Fig.4); (ii) traffic network consisting of 5,000 road segments; (iii) traffic network containing 10,000 road segments (see Fig.4).

The proposed app aims to inform drivers about the expected travel time and update them with up to date traffic estimates during the trip. Hence, the app is not required to be frequently run and, therefore, the battery drainage should not be an issue. However, the precise evaluation of the energy consumption of an app is still a challenging task [27]. We will investigate the battery depletion of the proposed app as part of our future work. Moreover, the communication costs and collection of traffic data are not explicitly considered here. We make the reasonable assumption that traffic information will be available in (near) real-time, similarly as the PEMS online database [28].

A. Performance of low-dimensional models

We divide the data set into training and test subsets. The training subset contains the speed data of the months August and September, 2011. In case of D-AD mode of operations, we use speed data of one week (within two months), leading to the eight mutually exclusive training subsets. The training subset is used to determine the subnetwork of c links (see (2) and Section III-A), generate relationship matrices (see (1) and (4)), and train predictors. All the predictors are trained on the training data for two months. We apply support vector regression (SVR) for prediction as it is a commonly used and effective approach [15], [29]–[32]. The test subset contains the speed data of the month October 2011. We use the test subset to assess the accuracy of the applications. In case of D-AD mode of operations, the reported accuracy refers to the average performance of eight different training subsets.

We apply SVD and uniform sampling strategies to identify the c links in the network. The SVD selection strategy is deployed for the D-NA scenario where the server sends traffic data for a predefined subset of the locations to the smartphone. The uniform selection strategy is deployed for the D-AD scenario since the traffic data is obtained from arbitrary sets of links (without control of the user).

1) Traffic speed estimation: We collect traffic information at c locations and extrapolate this information through the network using the relationship matrix (see (3)), inferred from the training data set.

	Galaxy S2	Galaxy S3	Nexus 5
System Chip	Exynos 4210	Exynos 4412	Snapdragon 800
Processor	2 Cores (1.2Ghz)	4 Cores (1.4Ghz)	4 Cores (2.2Ghz)
RAM	1024 MB	1024 MB	2048 MB
Memory	8GB	16GB	16GB
Android Version	4.2	4.3	4.4.2 (kitkat)
Release	Feb. 2011	May 2012	Oct. 2013

TABLE I: Technical characteristics of the tested smartphones.

2) Speed prediction: We predict the traffic variable at c locations using a baseline (SVR) predictor and extrapolate these predictions through the network using the relationship matrix (see (6)). As benchmark, we perform prediction by means of the same SVR algorithm at every single link in the network, as opposed to only the links in the (random) subnetwork. This is our baseline method.

3) Travel time prediction: We use the predicted traffic information (for multiple prediction horizons) to assess the travel time along each direction of 8 highways in the network (see Fig. 4). As benchmark, we consider commonly used historical and instantaneous travel time methods [33].

4) Performance measure: We compute the percent root mean distortion (PRD) to evaluate the performance of the applications. The PRD quantifies the error as:

$$\mathrm{PRD}(\%) = \frac{\| \mathbf{T} - \hat{\mathbf{T}} \|_F}{\| \mathbf{T} \|_F}. \tag{9}$$

For the applications of traffic estimation and speed prediction, $\mathbf{T}$ and $\hat{\mathbf{T}}$ are matrices that contain the true and inferred speed values, respectively for the entire network and all testing time instances. In case of traffic estimation $\hat{\mathbf{T}} = \mathbf{CX}$ while for the application of the traffic prediction $\hat{\mathbf{T}} = \hat{\mathbf{C}}\mathbf{X}$. For the application of travel time prediction, $\mathbf{T}$ and $\hat{\mathbf{T}}$ are matrices that contain the true and predicted travel times, respectively for 16 test routes in the network and all testing time instances.

B. Computation time

We test the computational performance of the smartphones for the above ITS applications. We evaluate the execution time of the app for different modes of operations, development platforms, and smartphone devices using the Android emulator. The Android emulator accurately approximates the execution time of the proposed app for different smartphone devices (see Fig. 5) [9]. The execution time is measured by inserting the monitoring functions into the Android operating system [9]. Our development platforms rely on either the NDK+SDK framework or only the SDK platform (see Fig. 3). In the former, the computation components are coded in NDK framework using C/C++ libraries while the visualization part is executed through the SDK. In the latter, traffic app is built in commonly used SDK environment. Table I shows the specifications of different smartphone devices that are used in the experiments.

VI. Results

In this section we analyze the performance of the low-dimensional models and execution time of smartphone app for the applications of traffic speed estimation and prediction, and travel time prediction.

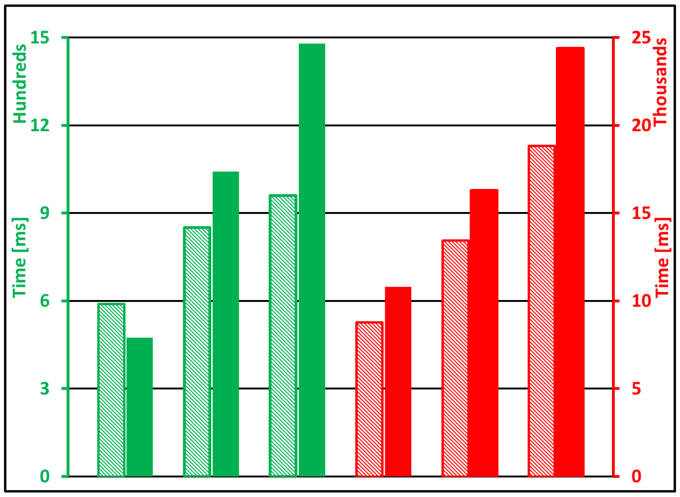


Fig. 5: Computation time of the emulator (shaded bars) and the corresponding smartphone device (solid bars) for various test cases.

Application and sampling strategy	Training subset	Compression Ratio 2	4	10
Estimation (SVD)	2 months	2.80	5.13	7.20
Estimation (uniform)	2 months	2.95	5.31	7.67
Estimation (uniform)	1 week	5.15	7.11	9.06

TABLE II: PRD error [%] of the applied low dimensional approach for applications of estimation and for two sampling strategies.

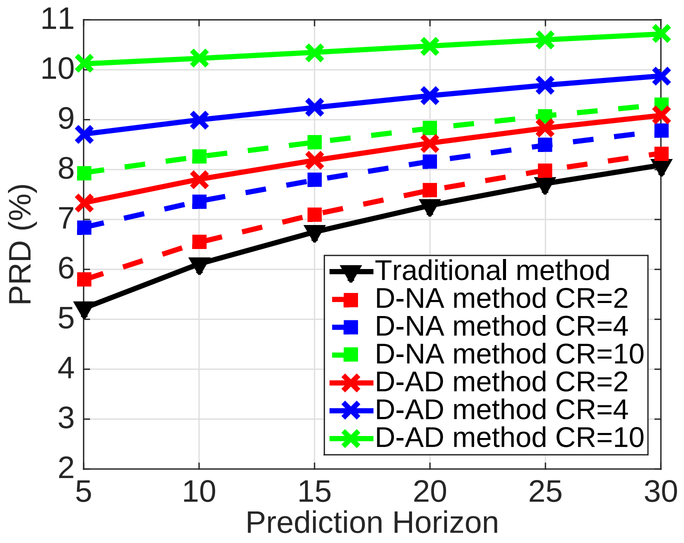


Fig. 6: The performance of compressed and traditional prediction methods. In the case of compressed prediction, the roads are sampled using the SVD and random sampling strategies which are deployed in D-NA and D-AD modes of operations, respectively. The roads are sampled from either two months of training data (D-NA mode) or one week of the training subset (D-AD mode).

CR	2	4	10
SVR	5.51	2.75	1.10
Extrapolation	0.0042	0.0025	0.0015
Total	5.52	2.76	1.10
Traditional method	10.99		
Savings	49.77%	74.89%	89.99%

TABLE III: Computation time (in seconds) for compressed and traditional prediction methods. Network extrapolation is performed on the server.

A. *Performance of the CX models for traffic applications*

We use data from the Singapore highway network to evaluate the performance of the CX-based models. We sample the road segments according to the SVD and uniform sampling methods, since these techniques are deployed in D-NA and D-AD scenarios, respectively. For each sampling strategy and compression ratio, i.e., the ratio of the number of links in the subnetwork (c) and the total number of links (n), we repeat the experiments five times and report the mean value. The standard deviation around the mean (of these five runs) is typically less than .001 due to significant homogeneity of the highway traffic network.

1) Traffic speed estimation: Table II shows the estimation accuracy of the sampling techniques for different compression ratios. As expected, the SVD-based sampling strategy outshines the uniform sampling where all the columns (roads) have the same probability of being selected. The minor difference in performance is consequence of high homogeneity of the test network that only contains the highway segments. As can be seen from Table II, there is a slight reduction in estimation performance of the D-AD mode of operations as a consequence of reduced training subset.

2) Speed prediction: Fig. 6 shows the prediction accuracy of traditional and compressed methods, for different compression ratios and various prediction horizons. Table III shows the required computation times for the compressed and traditional methods when all experiments are run on a 2.67 GHz MacPro server on a single core and 32GB of random-access memory (RAM). The results indicate that column based (CX) low-dimensional methods provide significant reduction in computational complexity by explicitly dealing with the traffic variable for only a small subset of road segments in the network (see Table III). This reduction in the computational cost comes at the expense of a negligible increase in error (see Fig. 6). Similar to the application of traffic estimation, the degradation in prediction accuracy of the D-AD mode of operation is associated with the limited training data set.

3) Travel time prediction: Table IV shows the average accuracy of the proposed method (for different compression ratios) and benchmark methods. As can be seen from Table IV, the proposed method outperforms the historical and instantaneous models and shows that taking speed predictions into account leads to more accurate travel time estimation.

B. *Computation time of the app*

The performance of CX methods encourage us to explore the option where network extrapolation is performed on the smartphone which possesses significant computational power. In the following, we analyze the execution time of smartphone app for the applications of traffic speed estimation and prediction, and travel time prediction.

	CR=1	CR=2	CR=4	CR=10	Instant.	Hist.
PRD (%)	2.43	2.75	2.82	3.24	4.48	6.44

TABLE IV: Performance of the proposed and benchmark methods for travel time estimation.

Compression Ratio		2	4	10
Number of links		1,078	539	216
Size of the **X** matrix		**11.3 Mb**	**8.1 MB**	**3.9 Mb**
SDK	Galaxy 2	24,264	16,885	7,854
	Galaxy 3	21,348	14,567	6,939
	Nexus 5	16,861	12,047	6,240
NDK	Galaxy 2	984	612	538
	Galaxy 3	937	596	479
	Nexus 5	859	698	437

TABLE V: Required computation time [*ms*] for different programming models and smartphones to perform extrapolation and visualization tasks in the case of Singapore highway network (2,156 links).

Size of the network	Compression Ratio (CR)					
	CR = 2		CR =4		CR = 10	
5,000	3,077	**33.6 Mb**	1,312	**16.8 Mb**	795	**6.4 Mb**
10,000	6,321	**134 Mb**	2,964	**67.4 Mb**	957	**24.1 Mb**

TABLE VI: The execution time [*ms*] of the app for D-NA scenario required to perform extrapolation and data visualization.

Number of links in the network, and size of the data matrix **A**	Execution time, required to compute / perform:	Compression ratio (CR)		
		2	4	10
2,156 (**12,4 Mb**)	Extrapolation matrix	3,077	1,312	795
	D-NA scenario	859	698	437
	D-AD method	**4,378**	**1,897**	**1,211**
5,000 (**28,2 Mb**)	Extrapolation matrix	16,881	6,088	2,056
	D-NA scenario	3,077	1,312	795
	D-AD method	**19,958**	**7,400**	**2,851**
10,000 (**57,6 Mb**)	Extrapolation matrix	221, 489	27,443	9,552
	D-NA scenario	6,321	2,964	957
	D-AD method	**227,810**	**30,407**	**10,509**

TABLE VII: The execution time [*ms*] of the app for the D-AD scenario and different networks. The total execution time includes: (i) computation of the relationship matrix; (ii) network extrapolation; and (iii) data visualization.

1) Traffic speed estimation: Table V shows the execution times of different smartphone devices and Android platforms to perform network extrapolation (for different compression ratios) and display traffic information on the road map once the extrapolation is performed. The reported results refer to the highway network in Singapore and decentralized non-adaptive (D-NA) mode of operations. From Table V it can be seen that the NDK development platform requires significantly less time than the commonly used SDK. Since in both scenarios the visualization task is performed in the same manner, the significant time savings is obtained for computational tasks. Table V shows that newer Android models require less computation time to perform network extrapolation as a consequence of the increased computational resources (see Table I). Table V also shows the size of the corresponding extrapolation matrix that needs to be stored on the phone, as part of traffic app. The elements of the extrapolation matrix have type double. As can be seen from Table V memory requirement and app execution time (in the case of the NDK platform) are acceptable from the user point of view. As the NDK platform clearly outshines the SDK platform, we will only consider the NDK programming platform from now on. Similarly, we will only consider smartphone Nexus 5 in the rest of the paper.

In the following, we present the performance of the D-NA mode of operations in the case of traffic networks that contain 5,000 and 10,000 road segments (see Table VI). Here, the server sends traffic data for the subset of the links to the smartphone where data extrapolation is performed with the help of the extrapolation matrix **X**. Table VI shows the promising execution time of the D-NA mode of operations, especially for higher compression ratios. However, the memory requirement of traffic app may make it less user friendly since the large extrapolation matrix **X** has to be stored in the phone memory (see Table VI).

We now investigate decentralized adaptive (D-AD) mode of operations. Unlike in the D-NA case, the traffic data comes from the subset of locations which are only revealed in real-time and will change from one time instance to another. Consequently, the pre-defined relationship matrix can not be used here. We deploy the NDK development platform to calculate the relationship matrix **X** and perform network extrapolation in the case of three test networks. For computation of the matrix **X**, we use the historical data for the entire network, stored in the phone memory and fetched every time when a new subset of data is received. The stored historical data matrix encompasses one week of traffic data (2016 time instances) since the similar traffic patterns can be observed during the same week days [5]. Table VII shows the execution time of the traffic app for different sizes of the underlying traffic network. The computation time for the extrapolation matrix accounts for a significant portion of the total execution time, especially for the lower compression ratios and larger traffic networks (see Table VII). Hence, the D-AD mode of operations might be efficiently deployed in the following cases: (i) in small and moderate traffic networks (e.g., a few thousand links) regardless of the compression ratio; (ii) in large traffic networks in the case of higher compression ratios. In the case of large traffic networks and lower compression ratios (e.g., CR=2) the smartphone has to compute the pseudo-inverse of a large matrix which is still an intensive task for the existing smartphone resources (see Table VII). Table VII also provides the memory requirements of the historical traffic data, stored in the phone memory. The stored matrix contains the historical speed data for one week, which are stored as integer values.

2) Speed prediction: We now explore the execution time of the decentralized non-adaptive (D-NA) mode of operations for the application of compressed prediction. The predictions for c road segments and p prediction horizons are obtained from the server in the form of matrix $\hat{\mathbf{C}} \in \mathbb{R}^{p\times c}$. To obtain traffic conditions in the entire network for the k^{th} prediction horizon we multiply the k^{th} row of $\hat{\mathbf{C}}$ with the extrapolation matrix **X**, stored in the phone memory. The app execution time for the single prediction horizon is equivalent to the execution time for the application of speed estimation which we discussed earlier (see Tables V and VI). Naturally, execution time for p multiple prediction horizons will be p times larger than corresponding times given in Tables V and VI.

Similar to the application of traffic estimation, we assess the execution time of the D-AD mode of operations as the

Number of the links along the route (L)	50	100	300	700
Execution time [ms]	690	790	857	919

TABLE VIII: Required computation time [*ms*] that traffic app takes to assess the future speed for the list of links, followed by computation of travel time.

sum of two components: (i) time required to compute the relationship matrix; (ii) time required to execute corresponding D-NA scenario (see Table VII).

3) Travel time prediction: Table VIII shows the app execution time for different sizes (*L*) of the test routes in the Singapore highways network using the NDK development platform and Nexus 5. The reported results refer to the D-NA mode of operations. Similar results are obtained for other two test networks that contain 5,000 and 10,000 links. As can be seen from Table VIII, smartphones are capable of performing travel time calculations, even along the routes that contain hundreds of road segments, corresponding to several kilometers of route length.

In case of the D-AD mode of operations, we assess the execution time of the app as the sum of two components: (i) time required to compute the relationship matrix (see Table VII); (ii) time required for travel time calculations and data visualization (see Table VIII).

VII. Conclusions

In this paper, we evaluated the computational capabilities of the smartphones for three ITS applications: traffic speed estimation and prediction, and travel time prediction. Based on column based (CX) low-dimensional models, we developed a traffic app that uses speed measurements from a small number of locations in the network and delivers traffic conditions for the entire network through a vector-matrix multiplication. Furthermore, the proposed app uses generated predicted traffic data to predict the travel time along the particular route for given ordered list of segments on that route. We tested the proposed app for different development platforms, smartphone devices and different sizes of the test network. The numerical results show that column based (CX) matrix decomposition leads to accurate results and has low computational complexity, enabling real-time traffic prediction on smartphones. Results also show that the native development kit (NDK) development platform has considerably better performance than traditional software development kit (SDK) in the case of complex problems. For most of the tested scenarios, the execution time of the NDK is acceptable even for very large networks. Hence, while the "easily-programmable" SDK performs well for traditional traffic applications [1], [2], [10]–[18], the more complex operations should be coded in the NDK development platform.

We also explored the memory requirements of the app, since either the historical data or extrapolation matrix has to be stored in the phone memory. The numerical results show that in most tested scenarios, the memory requirements of the traffic app are acceptable.

The proposed decentralized modes of operations reduce the overhead of the communication network, since only the measurements from a subset of the links are sent to the smartphone. More importantly, the decentralized adaptive mode of operations has great potential in emerging applications such as vehicle-to-vehicle communications. However, safety and security aspects of the vehicle-to-vehicle communications, for the suggested and other similar applications, have to be carefully considered and ensured [34]. In future work, we will investigate an approach where the traffic information are obtained from ad hoc probes in the network, leading to the "server-free" mode of operations. We will also provide a more detailed study of travel time estimation based on predicted traffic conditions and explore the option of performing routing on smartphones.

References

[1] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T. M. T. Do, O. Dousse, J. Eberle, and M. Miettinen, "From big smartphone data to worldwide research: The mobile data challenge," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 752–771, 2013.

[2] M. Fazeen, B. Gozick, R. Dantu, M. Bhukhiya, and M. C. González, "Safe driving using mobile phones," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 1462–1468, 2012.

[3] C. L. Schweiger, *Use and Deployment of Mobile Device Technology for Real-time Transit Information*. Transportation Research Board, 2011, vol. 91.

[4] L. M. Bergasa, D. Almería, J. Almazan, J. J. Yebes, and R. Arroyo, "Drivesafe: An app for alerting inattentive drivers and scoring driving behaviors," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 240–245.

[5] N. Mitrovic, M. T. Asif, U. Rasheed, J. Dauwels, and P. Jaillet, "CUR decomposition for compression and compressed sensing of large-scale traffic data," in *Intelligent Transportation Systems (ITSC), 2013 16th International IEEE Conference on*, oct. 2013, pp. 1475–1480.

[6] Y. Han and F. Moutarde, "Statistical traffic state analysis in large-scale transportation networks using locality-preserving non-negative matrix factorisation," *Intelligent Transport Systems, IET*, vol. 7, no. 3, pp. 283–295, 2013.

[7] N. Mitrovic, M. Asif, J. Dauwels, and P. Jaillet, "Low-dimensional models for compressed sensing and prediction of large-scale traffic data," *Intelligent Transportation Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–6, 2015.

[8] P.-E. Mazaré, O.-P. Tossavainen, A. Bayen, and D. Work, "Trade-offs between inductive loops and gps probe vehicles for travel time estimation: A mobile century case study," in *Transportation Research Board 91st Annual Meeting (TRB)*, vol. 349, 2012.

[9] R. Rogers, J. Lombardo, Z. Mednieks, and B. Meike, *Android application development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.

[10] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh, "Using mobile phone barometer for low-power transportation context detection," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. ACM, 2014, pp. 191–205.

[11] J. C. Herrera, D. B. Work, R. Herring, X. J. Ban, Q. Jacobson, and A. M. Bayen, "Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 4, pp. 568–583, 2010.

[12] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti, "Real-time urban monitoring using cell phones: A case study in rome," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 141–151, 2011.

[13] S. Amin, S. Andrews, S. Apte, J. Arnold, J. Ban, M. Benko, R. M. Bayen, B. Chiou, C. Claudel, C. Claudel *et al.*, "Mobile century using gps mobile phones as traffic sensors: A field experiment," 2008.

[14] J. Steenbruggen, M. T. Borzacchiello, P. Nijkamp, and H. Scholten, "Mobile phone data from gsm networks for traffic parameter and urban spatial pattern assessment: a review of applications and opportunities," *GeoJournal*, vol. 78, no. 2, pp. 223–243, 2013.

[15] E. J. Horvitz, J. Apacible, R. Sarin, and L. Liao, "Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service," *arXiv preprint arXiv:1207.1352*, 2012.

[16] J. Jariyasunant, B. Kerkez, R. Sengupta, S. Glaser, and A. Bayen, "Mobile transit trip planning with real-time data," 2011.

[17] S. Diewald, A. Möller, L. Roalter, and M. Kranz, "Driveassist-a v2x-based driver assistance system for android." in *Mensch & Computer Workshopband*, 2012, pp. 373–380.

[18] K. Erhardt, "Development of a navigation solution for the android based driver assistance system driveassist," 2012.

[19] R. Meier, *Professional Android 4 application development*. John Wiley & Sons, 2012.

[20] T. Peer and M. Wagner, "Embedding c++ code in android java applications," 2012.

[21] Y.-J. Kim, S.-J. Cho, K.-J. Kim, E.-H. Hwang, S.-H. Yoon, and J.-W. Jeon, "Benchmarking java application using jni and native c application on android," in *Control, Automation and Systems (ICCAS), 2012 12th International Conference on*. IEEE, 2012, pp. 284–288.

[22] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *Control Automation and Systems (ICCAS), 2010 International Conference on*. IEEE, 2010, pp. 1160–1163.

[23] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Relative-error cur matrix decompositions," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 844–881, 2008.

[24] K. Sung, M. G. Bell, M. Seong, and S. Park, "Shortest paths in a network with time-dependent flow speeds," *European Journal of Operational Research*, vol. 121, no. 1, pp. 32–39, 2000.

[25] M. T. Asif, N. Mitrovic, L. Garg, J. Dauwels, and P. Jaillet, "Low-dimensional models for missing data imputation in road networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 3527–3531.

[26] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[27] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 317–328.

[28] California Freeway Performance Measurement System, http://pems.dot.ca.gov/, accessed 09-November-2015.

[29] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 871–882, 2013.

[30] W.-C. Hong, "Traffic flow forecasting by seasonal svr with chaotic simulated annealing algorithm," *Neurocomputing*, vol. 74, no. 12, pp. 2096–2107, 2011.

[31] C. Wu, J. Ho, and D. Lee, "Travel-time prediction with support vector regression," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 276–281, 2004.

[32] Y. Zhang and Y. Liu, "Traffic forecasting using least squares support vector machines," *Transportmetrica*, vol. 5, no. 3, pp. 193–213, 2009.

[33] T. Toledo and R. Beinhaker, "Evaluation of the potential benefits of advanced traveler information systems," *Journal of Intelligent Transportation Systems*, vol. 10, no. 4, pp. 173–183, 2006.

[34] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14.

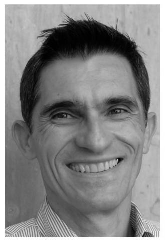
Nikola Mitrovic is currently a Ph.D. student in the School of Electrical and Electronic Engineering at Nanyang Technological University in Singapore. He received the Bachelor of Engineering degree from the University of Traffic and Transportation Engineering, Belgrade, Serbia, in 2009 and Masters degree from the Department of Civil Engineering at Florida Atlantic University, USA, in 2011. His research are simulation & data-driven traffic modeling, ITS, machine learning and transportation planning.

Aditya Narayanan completed the Bachelor of Technology from National Institute of Technology Trichy India and Master of Science in Computer control and automation from Nanyang Technological University Singapore. He is currently working as a Process control systems engineer at Micron Technology in Singapore. His research interests are control systems and optimization of large-scale systems.

Muhammad Tayyab Asif is a research staff member with IBM Research Collaboratory - Singapore. Prior to that he was a Ph.D. student in the School of Electrical and Electronic Engineering, College of Engineering, Nanyang Technological University, Singapore. He received the Bachelor of Engineering degree from the University of Engineering and Technology Lahore, Lahore, Pakistan. He also worked with Ericsson as a Design Engineer in the domain of mobile packet core networks. His research interests include sensor fusion, network optimization, and modeling of large-scale networks.

Ansar Rauf received bachelors degree in Electrical Engineering from National University of Science and Technology Pakistan in 2010. He completed his Masters degree in Computer Control and Automation from School of Electrical and Electronic Engineering at Nanyang Technological University in Singapore. He is currently working as a Software Engineer in the field of Logistics Automation. His areas of interest are software development, process control and automation.

Justin Dauwels is an Assistant Professor with School of Electrical & Electronic Engineering at the Nanyang Technological University (NTU) in Singapore. He is the Deputy Director of the ST Engineering - NTU corporate laboratory on autonomous systems. His research interests are in Bayesian statistics, iterative signal processing, and computational neuroscience. He obtained the PhD degree in electrical engineering at the Swiss Polytechnical Institute of Technology (ETH) in Zurich in December 2005. He was a postdoctoral fellow at the RIKEN Brain Science Institute (2006-2007) and a research scientist at the Massachusetts Institute of Technology (2008-2010). He has been a JSPS postdoctoral fellow (2007), a BAEF fellow (2008), a Henri-Benedictus Fellow of the King Baudouin Foundation (2008), and a JSPS invited fellow (2010,2011). His research on Intelligent Transportation Systems (ITS) has been featured by the BBC, Straits Times, and various other media outlets. His research on Alzheimer's disease is featured at a 5-year exposition at the Science Centre in Singapore. His research team has won several best paper awards at international conferences. He has filed 5 US patents related to data analytics.



Patrick Jaillet is the Dugald C. Jackson Professor in the Department of Electrical Engineering and Computer Science and a member of the Laboratory for Information and Decision Systems at MIT. He is also co-Director of the MIT Operations Research Center. His research interests include online optimization, online learning, and data-driven optimization with applications to transportation and to the internet economy. He is a Fellow of INFORMS.