VCAMS: Viterbi-Based Context Aware Mobile Sensing to Trade-Off Energy and Delay

基于维特比环境感知的能量权衡与延迟的移动感知

Sirine Taleb, *Student Member, IEEE*, Hazem Hajj, *Senior Member, IEEE*, and Zaher Dawy, *Senior Member, IEEE*

Abstract—Monitoring context depends on continuous collection of raw data from sensors which are either embedded in smart mobile devices or worn by the user. However, continuous sensing constitutes a major source of energy consumption; on the other hand, lowering the sensing rate may lead to missing the detection of critical contextual events. In this paper, we propose VCAMS: a Viterbi-based Context Aware Mobile Sensing mechanism that adaptively finds an optimized sensing schedule to decide when to trigger the sensors for data collection while trading off the sensing energy and the delay to detect a state change. The sensing schedule is adaptive from two aspects: 1) the decision rules are learned from the user's past behavior, and 2) these rules are updated over real time whenever there is a significant change in the user's behavior. VCAMS is validated using multiple experiments, which include evaluation of model success when considering binary and multi-user states. We also implemented VCAMS on an Android-based device to estimate its computational costs under realistic operational conditions. Test results show that our proposed strategy provides better trade-off than previous state-of-the-art methods under comparable conditions. Furthermore, the method provides 78 percent energy saving when compared to continuous sensing.

Index Terms—Mobile sensing, energy efficient sensing, sensor selection, context-awareness, Viterbi algorithm

1 Introduction

Smart mobile devices have evolved into multi-purpose devices combining their traditional communication capabilities with advanced computing and sensing capabilities. Cisco's annual report stated that by the end of 2019, there will be nearly 1.5 mobile devices per capita, and more than 578 million wearable devices will be in use [1]. These devices have become a vital component for users who are driven by desire for self reflection and self improvement to understand themselves more and self-organize their daily lives [2]. A recent smartphone can be viewed as a gateway between the user and the huge world of knowledge and social interactions. It has computer capabilities with a wide range of applications being developed in several domains. On Google Play [3] alone, there are over 1.3 million Android applications, capturing several domains in context-aware computing including healthcare, navigation and personal monitoring. Such proliferation and popularity have given rise to context-aware computing as a branch of mobile computing in which applications detect and exploit contextual information such as locations, health conditions, and activities. [4], [5].

These new user-targeted applications have their unique characterization in the area of mobile computing. Several applications require near real-time response for detecting context changes especially for medical applications. For example, fall detection of elderly people requires the fastest possible detection of any change in the body's posture to avoid risks [6]. Such applications rely on continuous recognition of the user's current state and fast detection of any critical context change. This constant monitoring requires continuous sensing of embedded and external sensors to avoid delays in detecting critical context change. Many of these applications require the use of embedded sensors on the phone to collect necessary data. Certain other applications require the use of wearable devices and wearable sensors to collect data. Unfortunately, sensors' usage constitutes a major source for energy consumption that imposes heavy workloads on smartphones. Thus, extensive sensing can drain the battery quickly, which becomes a primary source of users' dissatisfaction in recent smartphones. Smartphone sensing for context detection is being investigated by a fair amount of researchers, and is gaining further attention as new context-aware applications are being developed to support user's health and life style.

Fortunately, most applications do not need to detect the states of the context continuously, rather they require the detection of critical changes in context. For example, the purpose behind monitoring signals from an electrocardiogram (ECG) sensor of a patient with heart disease is to detect risky heart activity and alert in case of emergency [7]. In such cases, continuous sensing can be substituted by efficient dynamic mobile sensing strategies that allow the smart mobile device to intelligently interact with external sensors and embedded sensors while trading off resources' energy consumption and application delay targets. Therefore, a system is required to select a sensing schedule that optimizes when sensors need to be triggered.

There have been some approaches to solve this problem; however, those approaches focus on energy and accuracy within one state of the context rather than the transitions between states. To monitor a certain state accurately while optimizing energy expenditure, some approaches propose

• *The authors are with the Department of Electrical and Computer Engineering, American University of Beirut, Beirut 1107 2020, Lebanon.
E-mail: sht06@mail.aub.edu, {hh63, zd03}@aub.edu.lb.*

*Manuscript received 18 May 2016; revised 21 Nov. 2016; accepted 7 May 2017. Date of publication 19 May 2017; date of current version 1 Dec. 2017.
(Corresponding author: Hazem Hajj.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2017.2706687*

sensor selection [8], [9] while others propose sensor sampling to collect data [10], [11]. Those approaches are application-specific and they assume the knowledge of the true context state of the user.

This paper proposes Viterbi-based Context Aware Mobile Sensing (VCAMS) mechanism to optimize the trade-off between delay in sensing context change and energy consumption via deciding when to trigger the sensors for data collection based on the user's behavior. VCAMS is context aware system that depends on the situational information about the user and her surrounding. It can be used for applications that anticipate real-time contexts such as location, activity, and health conditions. Each context can have several states; for example, the location of the user might be at home, at work, or in mall. For a given user in a specific state, the objective of the system is to dynamically provide the time instants at which a sensor needs to be triggered, also called the sensing schedule for the specific user and the particular state. The system has two modes: learning mode and execution mode. In the learning mode, the sensing schedules are derived using a Vietrbi algorithm based on historical data of the user model that captures how much time the user spends in particular states. Viterbi is chosen for its low computational complexity. The user model is continuously updated when the system recognizes actual changes of state. The learning mode is executed once to initialize the system, and on occasions while the system runs and after a user's behavior changes significantly from its initial conditions. In the execution mode, the already learned sensing schedules are used to decide on sensing triggers for a particular state.

The main contributions of this paper include: 1) A formulation for Viterbi implementation with the definition of new customized rewards to learn sensing schedules for real-time decisions on when sensing should be triggered. The energy and delay reward metrics are formulated to best represent the utility associated with each transition between trellis nodes in a Viterbi-based algorithm. The formulation includes a unified model that captures both the user's and the phone's states; and 2) A strategy that triggers learning mode in real-time to update the sensing schedule only when critical changes are captured, thus avoiding unnecessary computations. To evaluate the performance of VCAMS, we conducted simulation experiments on one state derived using a context simulator. To assess the computational complexity under realistic operational conditions, we implemented VCAMS on an Android-based smartphone. In addition, we investigated a case study using real dataset with multi-state changes to demonstrate the effectiveness of the proposed approach.

The rest of this paper is organized as follows. Section 2 summarizes existing related work. Section 3 presents the proposed system model. Section 4 presents the details of the proposed Viterbi method and studies its computational complexity. Sections 5 and 6 present the results obtained by simulations and the case study respectively. Finally, Section 7 concludes the paper.

2 Related Work

Smartphone sensors constitute a major source for energy consumption; therefore, several approaches have been considered in the literature to capture contextual information from sensors while minimizing the energy consumed, and surveys have been published to discuss the existing work [12], [13], [14]. Application-specific frameworks have been proposed in various domains including healthcare, location detection, and activity recognition. Most prior work in the field can be categorized into one of the following: sensor selection and hierarchical sensing mechanisms, stochastic approaches such as Markov Decision Processes, or adaptive duty cycles and sampling frequencies. These are further detailed below.

2.1 Sensor Selection and Hierarchical Sensing

There has been extensive research to select a minimum required number of sensors. In [8], the authors present an Energy Efficient Mobile Sensing System (EEMSS) which is a context-aware monitoring system that explores hierarchical sensor management by powering only a minimum set of embedded sensors. It combines sensor readings from an accelerometer, WiFi, GPS, and a microphone to detect the user's state. However, fixed duty cycles are assigned to active sensors. In our previous study [15], we use energy, accuracy, and context metrics to propose a sensor selection procedure. The algorithm chooses the sensor that guarantees the least acceptable accuracy while consuming the minimum energy possible. We apply our algorithm to location-based services. Wang et al. [16] trade-off energy efficiency and classification performance in two sensor schemes: off-node where the accelerometer acts as a data collector and sends its raw data to base station and on-node where it acts as a context classifier.

In [9], the authors present SeeMon which uses hierarchical sensor mechanism. It focuses on detecting accurate context by monitoring the feature data; thus avoiding extra computational energy needed to recognize the exact context. The solution does not provide answers for all types of contexts. Similarly, in [17], the authors propose the term suppression which uses less power consuming location sensors instead of GPS (which is energy hungry) when the user's location is static. In [18], sensors are selected based on their accuracies in recognizing user's activity during training the system. They incrementally select sensors using five body-mounted accelerometers to accurately estimate the human activity. In contrast to the above existing works, our work focuses on the sensing behavior of each selected sensor.

2.2 Stochastic Approaches

Recently, stochastic principles have been introduced to select appropriate sensors and their duty cycles. Markov Decision Process (MDP) frameworks have been used to schedule sensors based on the current state of the user [11], [19]. This statistical approach uses transition probabilities from one state to another to determine the optimal sensor sampling policy assuming a Markovian user state process and knowing the true system state. In addition, Thiagarajan et al. [20] use a Hidden Markov Model (HMM) to estimate the user's location. The authors develop an energy-efficient location tracking system which uses regularity in location patterns to avoid activating sensors when a user follows previously observed patterns. In [21], the authors propose an inhomogeneous framework based on Hidden Markov Model. Their model either recognizes or estimates the user's states when power optimization is considered. This framework focuses on Human Activity Recognition (HAR) and examines the trade-off between accuracy in contextual inference and required energy consumption caused by processing. Jigsaw [10] is another similar system which balances

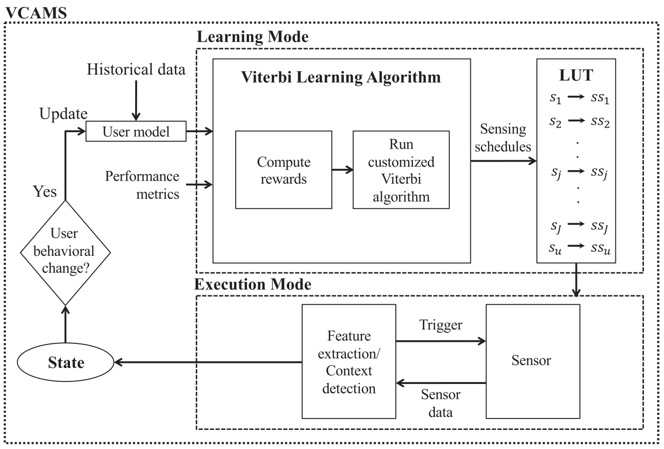


Fig. 1. The general work flow of the proposed method.

the performance needs of an application and resource demands by learning an adaptive sampling schedule using a Markov Decision Process. It uses sensor specific pipelines that have been designed to cope with individual challenges experienced by each sensor. It uses learning techniques and drives the duty cycle taking into account the activity of the user, energy budget, and duration over which the application needs to operate. However, Markov models assume predetermined systems which know in advance the true user's state which might turn out to be a wrong assumption. In addition, user state transitions in real data traces are not strictly Markovian [11]. These methods focus on one state; on the other hand, our approach aims at detecting the contextual change from one state to another.

2.3 Adaptive Duty Cycles and Sampling Frequencies

Other methods optimize sensors' sampling frequencies to minimize energy consumption [22]. In our previous work [23], we propose an entropy-based optimized energy-accuracy activity-dependent sensing algorithm for recognizing human physical activity. We mathematically prove that entropy is directly proportional to accuracy. In Kobe framework [24], the authors trade-off three criteria which are energy, latency and accuracy for classification. Kobe generates optimized inference pipelines depending on the training data it obtains from developers and decides accordingly whether to perform inference pipelines locally or remotely. In A3R [25], the authors adapt the sampling frequencies and the classification features according to the current activity of the user to reduce energy consumption. A3R considers a desired minimum level of classification accuracy, then it uses a heuristic algorithm to find the optimal sampling frequency which gives the highest fraction of accuracy compared to energy. AdaSense [26] also uses genetic programming to control body sensor sampling frequencies while meeting a user-specified accuracy to trade-off energy and accuracy. In [27], Rachuri et al. use a learning technique to control the sampling rate of the sensors. Their decision whether to sense or not is based on the probability of sensing. The idea is that when a sensor expends some energy in sensing and it results in capturing an event of interest then it is considered as a success, otherwise as a failure. The probability of sensing from a sensor is dynamically adjusted according to their previous successes and failures. Those approaches trade-off energy and accuracy to obtain the current contextual state, and some can also be used to detect change in state; however, they don't focus on rapidly detecting the critical transition between one state and another with minimal delay.

TABLE 1
Table of Parameters

Groups	Parameter	Description
State-based parameters	s_j	Particular state out of J total states
	T_j	Time duration spent in s_j
	$\hat{T}_j$	Time limit of s_j
	δ_j	Sampling interval between time instants
	N_j	Number of time instants
	p_j	Probability of state switch
	h_j	Probability's decaying rate
	μ_j	Mean time spent in state s_j
	σ_j	Standard deviation of spent in state s_j
	ss_j	Output sensing schedule for state s_j
Time-based parameters	t_i	Time instant
	a_i	Decision action: 1 for "sense" and 0 for "not sense"
	Δt_i	Time instants accumulated since last sense
Weighting parameters	ω	Pareto weighting factor
	α	Energy-delay weighting factor
	β	Energy-delay weighting factor
Output parameters	D	Output delay
	E	Output energy

2.4 Viterbi Approaches

Typically, Viterbi algorithm [28] is used to find the most probable path of hidden states in Hidden Markov Models used for modeling sequences. In [29], the authors use the Viterbi decoder to extract the sequence of user's locations which are hidden from sequence of observations derived from the signal measurements. In our proposed approach, we define customized reward functions in terms of optimization criteria which depend on the sensing action and the current state. The goal is then to find the optimal sensing schedule that maximizes the cumulative rewards.

3 System Model and Components

The general work flow of the proposed method is shown in Fig. 1 for each required context; and the parameters used in VCAMS are listed in Table 1. There are two modes in the solution. The learning mode aims at determining the sensing schedule to be used in association with a given user and a given context. This mode is shown in the upper part of Fig. 1. The inputs to the learning mode of the system are: 1) the user model which captures a transition model from historical behavior of the user such as the probabilities of transition $p_j(t_i)$ from state s_j to another at any time instant t_i, and can also include the statistics of the time duration T_j spent in a specific state s_j and 2) the performance metrics: energy and delay which are used to define rewards in the customized Viterbi algorithm. The learning part of the system generates a user-specific lookup table (LUT) capturing which sensing schedule should be used for each state. The second part of the solution reflects the execution mode shown in the lower part of Fig. 1. The execution mode of the system includes integration with a context recognition application. It takes

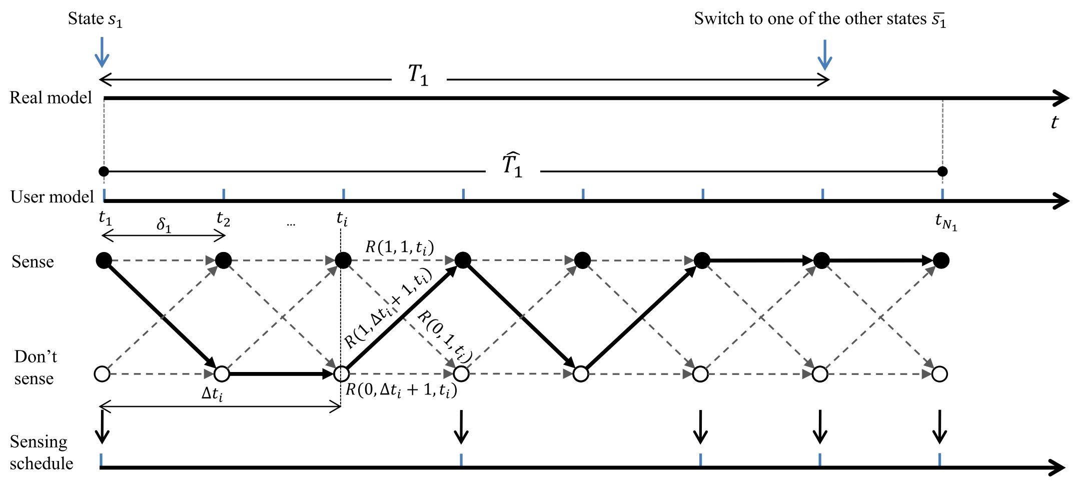


Fig. 2. A unified model for user context and smartphone sensing: Real model reflects the real state change, user model defines the derived system parameters needed for the algorithm, the Viterbi trellis shows the evolution of the algorithm, and the sensing schedule presents the instants at which sensing should be triggered.

the LUT as input, and chooses the corresponding sensing schedule for the previously detected user's state.

The proposed system starts by developing the initial user model based on the user's historical behavior. The VCAMS learning process requires two steps shown in the "Viterbi Learning Algorithm" box in the figure. The first step is to compute the expected customized rewards to estimate the gain for every combination of state and sensing decision at each time instant. This step is described more in Algorithm 1 in Section 4.3. The second step is to run the customized Viterbi algorithm to maximize those rewards and derive the optimized sensing schedules, and this step is presented in Algorithm 2 in Section 4.3. The learning of the sensing schedule is done offline, and the derived sensing schedules are saved in a LUT where each state s_j has its specific sensing schedule ss_j where $1 \leq j \leq J$. In the execution mode, the system applies these schedules in real-time by inputting the LUT to the online context recognition application which triggers the specific sensor to extract features and recognize the current state of the user. The two-step learning mode is executed on occasions when the statistics of the user's behavior change significantly based on our proposed VCAMS trigger strategy as described below in Section 4.4.

In terms of its flexibility, VCAMS is generic where it can deal with any number of activities. VCAMS can add new states if it is available in the training dataset. Futhermore, VCAMS's framework accounts for an "unknown" state. When an "unknown" state is encountered, the classification algorithm will alert VCAMS that the current state is an unknown state which does not have a specific sensing schedule. The lookup table in VCAMS accounts for an unknown state denoted by s_u. It triggers continuous sensing until a change is detected into a state which is recognized by VCAMS. When the system has collected enough training data about the "unknown" state, VCAMS can be updated since it is scalable.

3.1 The Sensing Schedule

The problem is to devise an efficient algorithm for an application to detect a critical change in the contextual state of a user. For example, the user can be using an ECG-based health application that needs to discover any sudden change in the user's state from normal heart activity to risky heart beat. As shown in Fig. 2, at each time instant t_i, we want to decide whether to trigger the sensor (action $a_i = 1$) or not (action $a_i = 0$). The real model axis in Fig. 2 reflects that the person stays in state s_1 for time duration T_1, then transits to a different state $\bar{s}_1$. Energy is impacted by the number of times sensing mechanism takes place. Energy is reduced with infrequent sensing, and delay is reduced with more frequent sensing. As a result energy and delay provide conflicting requirements on device usage. Hence, the objective is to find optimized choice of when to trigger the sensing mechanism. Triggering the sensor allows detecting whether the user has changed the state. If the state has not changed, then no sensing is needed. As a result, ideally, we do not want to sense until the exact moment of state change. Any early sensing will cause wasted device energy; on the other hand, waiting too long to sense may cause miss and delay in the detection of the state change in the context being monitored. Our goal is to detect this transition as soon as possible while minimizing the amount of consumed energy.

A unified model for user context and phone sensing trigger points is shown in Fig. 2. The user model is based on the user's historical behavior. It provides the following parameters:

- $\hat{T}_j$ is intended to estimate the time duration the user spends in context s_j. As a result, it is used to represent the time limit of state s_j before the model triggers continuous sensing. $\hat{T}_j$ is chosen based on the historical distribution of the time spent T_j in each state s_j. For example, $\hat{T}_1$ is the time limit for state s_1. If $\hat{T}_1 \leq T_1$, then no delay will be incurred in detecting state change from s_1 since we assume continuous sensing after $\hat{T}_1$; however, if $\hat{T}_1 > T_1$, there might be delay depending on the sensing decision.The choice of $\hat{T}_j$ will be further described in Section 3.3.
- $p_j(t_i)$ is the survival probability at time instant t_i that the context will stay in the same detected state s_j at this time instant. It is presented in more details in Section 3.2.
- δ_j represents the sampling interval between time instants where decisions are made to sense or not. The choice of δ_j is further described in Section 3.4.
- N_j is the number of time instants over the duration $\hat{T}_j$ at which decisions will take place.

The optimized sensing schedule is the output of the proposed Viterbi-based algorithm whose trellis is shown in Fig. 2. In the figure, $R()$ is a probabilistic reward function comprehending the rewards when state changes and the rewards when state does not change. The goal is to find the sensing schedule which consists of an optimized sequence of actions a_i at each time instant t_i by looking forward over the time duration $\hat{T}_j$ and maximizing the cumulative rewards, where the rewards comprehend the trade-off between energy and delay. This problem can be mathematically formulated to maximize the total probabilistic rewards as follows:

$$\underset{a_i; i=1\ldots N_j}{\arg\max} \sum_{i=1}^{N_j} R(a_{i+1}, \Delta t_{i+1}, t_i). \tag{1}$$

Mathematically, $R(a_{i+1}, \Delta t_{i+1}, t_i)$ represents the probabilistic reward of having action a_{i+1} as the next action at t_{i+1}. As for a_{i+1}, it represents the decision action: 1 for "sense" and 0 for "not sense". Δt_{i+1} is the time instants accumulated since the last sense was triggered till time instant t_{i+1}. Δt_{i+1} plays an important role in determining the incurred delay. Δt_{i+1} depends on the current action at time instant t_i as follows:

$$\Delta t_{i+1} = \begin{cases} 1, & \text{if at } t_i, \text{ action } a_i = 1 \\ (\Delta t_i) + 1, & \text{if at } t_i, \text{ action } a_i = 0 \end{cases}. \tag{2}$$

Δt_{i+1} resets to 1 when the action at time instant t_i is "sense"; on the other hand, Δt_{i+1} accumulates when no sensing is triggered at time instant t_i.

Each node to node link in the trellis has its own reward $R()$ which is a function of the next action a_{i+1}, the accumulated delay up to t_{i+1}, and the current time instant t_i. In addition, these reward functions need to comprehend energy and delays as will be described in Section 4.2. The trellis path that maximizes the probabilistic rewards is chosen. An example illustration is shown in bold lines on the trellis. The corresponding sensing schedule in the figure shows the optimized output sensing schedule where each impulsive arrow represents a trigger decision.

3.2 Survival Probability $p_j(t_i)$

The survival probability $p_j(t_i)$ represents the probability that the user will survive in state s_j for the current time instant t_i. In other words, it is the probability that the user will not transit to another state at this time instant. This probability is captured in the user model, and it depends on the previous historical behavior of the user and on the time limit that the user spends before transiting and changing current state. Therefore, this probability decreases with time where the highest probability lies at the instant when the state is first recognized and the lowest survival probability of the state s_j lies at the end of $\hat{T}_j$.

We used survival analysis [30], [31] to derive this probability. The duration until a state change happens is modeled as a random variable. The survival probability $p_j(t_i)$ represents the probability that no change has occurred up to the current time instant t_i. The exponential distribution is one of the most used survivor functions and it gives a good fit which matches the context more than any other distribution [30]

$$p_j(t_i) = e^{-h_j \cdot t_i}, \tag{3}$$

where h_j is the exponential decaying rate that depends on the time limit $\hat{T}_j$ of state s_j and h_j is set such that the survival probability at t_{N_j} is around 0.001 as exponential cannot reach 0.

3.3 Choice of Time Limit $\hat{T}_j$

$\hat{T}_j$ represents the time limit at which continuous sensing is triggered. It is based on the estimated time spent by the user in context s_j. From the user's past behavior, the time spent by the user in the s_jth state can be considered as a random variable T_j, with mean μ_j and a standard deviation σ_j. These time properties are learned from previous behaviors, and they are updated dynamically whenever state s_j is encountered. Typically, the choice of $\hat{T}_j$ should not fall below $(\mu_j - 3 \cdot \sigma_j)$ since most of the time, the actual time T_j is above these values. The choice of a low $\hat{T}_j$ will cause high energy consumption. On the other hand, $\hat{T}_j$ should also not exceed $(\mu_j + 3 \cdot \sigma_j)$ because delaying continuous sensing beyond $(\mu_j + 3 \cdot \sigma_j)$ would incur higher delays. Hence, the choice of $\hat{T}_j$ reflects a trade-off. Choosing $\hat{T}_j < \mu_j$ causes more energy due to continuous sensing and does not provide opportunity for the algorithm to make intelligent decisions in the more frequent cases around the mean μ_j. On the other hand, choosing $\hat{T}_j > \mu_j$ minimizes energy but leads to more delay, and also provides the algorithm the opportunity to make efficient decision in the more frequent cases around the mean μ_j. Hence, $\hat{T}_j$ is chosen to be greater than μ_j. This choice allows Viterbi-based method to minimize delay.

3.4 Choice of Sampling Interval δ_j

In the user model shown in Fig. 2, δ_j represents the sampling interval between time instants t_i and t_{i+1} at which our optimized sensing strategy checks whether to trigger sensing or not. When the model decides not to sense, the risk of delay in sensing a contextual state change increases. The time delay D_t is proportional to the sampling interval δ_j and the number of delay intervals D. D is measured as number of instants between the time instant when context change actually happened and the instant when the following sensing decision happened. Hence, the time delay D_t is measured in seconds as follows:

$$D_t = D \cdot \delta_j. \tag{4}$$

Increasing δ_j increases D_t. However, decreasing δ_j increases the frequency N_j, described in Section 3.1, at which sensing decisions need to be made; thus, more computational time and energy. As a result, the choice of δ_j provides a trade-off between energy and delay, consistent with the overall objective of the optimization model in (1). As a result, we choose to set the value of δ_j to the minimum possible $\delta_{j_{\min}}$ based on the sensor requirements. $\delta_{j_{\min}}$ is composed of the duration required to capture the raw data needed for one feature plus the duration needed to compute this feature; hence, it is sensor-dependent. For example, sensors can be divided into two categories based on their sensing behavior [22]. The first category includes sensors, such as accelerometer and microphone, that require a command from the system to turn on and start collecting samples and another command to turn it off and stop acquiring data. The second category includes sensors, such as GPS and WiFi, which are based on their own protocols to operate. These sensors need a command from the system to turn on; however, they turn into idle mode automatically by themselves after attaining the required data and finishing their tasks. In both cases, there is a minimum sampling interval $\delta_{j_{\min}}$ based on the specifications of each sensor. Furthermore, the acceptable $\delta_{j_{\min}}$ is relative to the time spent in a state; therefore, it is state-dependent. For example, the location of a user driving changes more frequently than the location of a user attending a meeting. In Section 5, we will consider the single state of a user during a meeting detected using WiFi hotspot coverage. We choose $\delta_{j_{\min}}$ based on the sensor requirements and state restrictions and delegate the energy-delay trade-off to the proposed Viterbi solution described in Section 4.

4 The Proposed Viterbi-Based Method

This section presents the proposed learning method for the Viterbi-based algorithm with trade-off between energy and delay for context recognition using smart sensors. It describes the Viterbi trellis with its components as was shown in Fig. 2. The Viterbi algorithm is commonly used to find the most likely sequence of states that maximizes the posteriori probability of a process. In this section, we describe our method when one sensor is considered; however, it can be generalized to multiple sensors. Section 4.1 gives a background about Viterbi algorithm. Section 4.2 presents the reward functions defined in terms of energy and delay factors, and Section 4.3 describes the steps to solve the energy-delay trade-off using Viterbi-based algorithm.

4.1 Viterbi Algorithm Background

Viterbi algorithm [28] is applied as a dynamic programming algorithm for an optimization problem that needs to maximize a statistical utility function. For example, it is used as an efficient method of estimating a sequence of hidden states in Hidden Markov Models [32], [33]. Viterbi algorithm is also used to find the shortest path through a weighted graph. It is widely used in communication as a decoding algorithm for data encoded with convolutional encoding in digital data transmission [34], [35]. It aims at recognizing data errors caused by communication channels and correcting them. Although Viterbi was initially introduced in 1967 [36], it is still used in various fields as a dynamic programming algorithm which finds the most likely sequence of hidden states. It is recently being used in emerging concepts and domains ranging from communications [37] to target tracking [38], and even biomedical engineering [39], [40] .Viterbi has been shown to be optimal for estimating the state sequence of a finite state process [28]. Hence, Viterbi can be applied to any dynamic problem with finite states. In our case, the Viterbi states are the sensing decisions which are two: "sense" or "not sense". Our problem deals with real dynamic time limits where some sensing decisions might be taken over long periods of time; hence, Viterbi provides a reduction in computational complexity by using recursion and saving only the most likely path leading to each state [36].

Viterbi algorithm is described using two diagrams. The first diagram is the trellis diagram shown in Fig. 2 which represents a graph of a finite set of nodes connected using edges that define the possible transitions between nodes at discrete time intervals. Each edge has its probabilistic reward function $R()$. The nodes in the trellis represent the decision where the upper node represents "sensing" decision and the lower node represents "not sensing" decision. The probabilistic state of the user is thus not illustrated in the trellis. Hence, we need the second diagram which is the state diagram shown in Fig. 3. It defines the instantaneous rewards of the transitions between nodes based on the state of the user. The nodes in the state diagram represent the two states s_j and $\bar{s_j}$. The links between the nodes are the instantaneous rewards $r()$ that depend on whether the state has survived as s_j or transited into $\bar{s_j}$ on time instant t_{i+1}. The Viterbi algorithm uses a set of metrics (costs or rewards) to compare the costs of the various paths through the trellis and then decide the path that maximizes the problem-specific utility function. We define those metrics for our problem in Section 4.2.

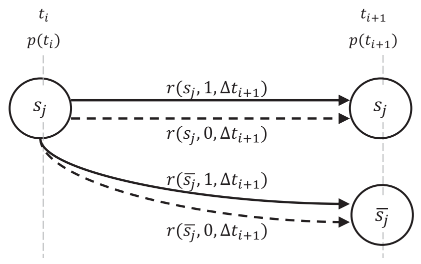

t_i
$p(t_i)$
t_{i+1}
$p(t_{i+1})$
s_j
s_j
$\bar{s_j}$
$r(s_j, 1, \Delta t_{i+1})$
$r(s_j, 0, \Delta t_{i+1})$
$r(\bar{s_j}, 1, \Delta t_{i+1})$
$r(\bar{s_j}, 0, \Delta t_{i+1})$

Fig. 3. State and reward diagram.

The general Viterbi algorithm works as follows. First, the algorithm looks at each node at time t_{i+1}, and for all the transitions from t_i that lead into that node, it chooses the path with the greatest metric, and it discards the other transitions into that node. The algorithm does the same process for all the trellis nodes starting from time t_i by moving forward to time t_{i+1} and repeating the process. When the algorithm reaches the end of the trellis ($t_i = t_{N_j}$), it evaluates the metrics for the different paths that lead to the end and outputs the path with the highest cumulative reward metric $R()$ as presented in (1).

4.2 Rewards

In dynamic programming algorithms, metrics need to be defined to represent the utility associated with each transition between nodes. The metrics used in our proposed Viterbi-based algorithm are defined as $R(a_{i+1}, \Delta t_{i+1}, t_i)$ as shown in the trellis in Fig. 2. Each edge in the figure holds a defined metric $R()$. The true state of the user is not known unless a sensing mechanism is triggered; therefore, each $R()$ metric is a probabilistic combination of two instantaneous reward functions. At each time instant t_i, there is a probability $p_j(t_i)$ of remaining in state s_j and a probability $1 - p_j(t_i)$ of transitioning into another state $\bar{s_j}$. Therefore, the metric $R()$ is to be computed with probabilities depending on the action considered at each time step. Hence, for each action (sense or don't sense) there are two instantaneous rewards $r()$: the first one $r(\bar{s_j}, a_{i+1}, \Delta t_{i+1})$ representing the reward assuming a state change has happened and the other one $r(s_j, a_{i+1}, \Delta t_{i+1})$ assuming the monitored context is still in the same state s_j. The reward $R()$ for each action a_i becomes:

$$\begin{aligned} R(a_{i+1}, \Delta t_{i+1}, t_i) &= E_s[r(s, a_{i+1}, \Delta t_{i+1})] \\ &= p_j(t_i) \cdot r(s_j, a_{i+1}, \Delta t_{i+1}) + (1 - p_j(t_i)) \cdot r(\bar{s_j}, a_{i+1}, \Delta t_{i+1}), \end{aligned} \tag{5}$$

where E_s is the expectation over states s_j and $\bar{s_j}$.

The decision of when to sense depends on the survival probability and the cumulative delay. Intuitively, as the survival probability decreases with time; that is the probability of changing the state of the monitored context increases, we should sense more frequently. In addition, as the risk of accumulated delay increases, sensing mechanism becomes more necessary. Sensing more frequently guarantees lower

TABLE 2
Energy, Delay, and Recognition Components

	state = s_j			state = $\bar{s_j}$		
	energy	delay	recognition	energy	delay	recognition
action = sense	-1	0	0	-1	$-(\alpha \cdot \Delta t_{i+1})$	$(\beta/\Delta t_{i+1})$
action = don't sense	0	0	0	0	$-(\alpha \cdot \Delta t_{i+1})$	0

delays but causes more energy consumption. Thus, energy and delay criteria are represented in rewards $r()$ to define the metrics of each transition in the Viterbi trellis. Table 2 shows how we divide the metric into three components: the energy component which represents the cost of sensing when triggered, the delay component which accumulates since the last time instant when sensing was triggered, and the recognition component which defines the reward of recognizing a state transition. Each instantaneous reward $r()$ has an energy, delay, and recognition component.

Energy and delay are measured in terms of time instants. Delay is computed as the number of time instants skipped before detecting a state transition, and energy is computed as the number of time instants at which sensing is triggered. Therefore, energy costs either 0 when no sensing is triggered or -1 when sensing is triggered. Let α and β be energy-delay weighting factors. Δt_{i+1} is used in defining the delay component since the actual delay time is not known as the true state of the user cannot be revealed unless sensing is triggered. α is used in the delay component. It varies the effect of Δt_{i+1} on sensing decision. Increasing α would increase the negative effect of incurred delay; thus, as α increases, sensing is triggered more frequently to avoid excessive delay. On the other hand, β affects the recognition component where VCAMS system gains more reward when β increases; thus, increasing β also leads to more sensing. Therefore, increasing α and β causes more sensing; thus more energy and less delay.

The choice of those weighting factors is application-dependent. For example, if the application is health-related with critical context being monitored, α and β should be chosen such that delay is minimized. Another factor that affects the choice of the weighting factors is the phone status. For example, if the phone battery is low, α and β should be chosen to minimize energy consumption; and thus sense less frequently.

4.2.1 *Pareto Optimal* (α, β)

Sensing more frequently causes more energy but decreases the delay values; hence, there is a trade-off between energy and delay. Thus, finding the best (α, β) combination is a multi-objective optimization problem which requires an optimal Pareto solution. Pareto solutions find points which are acceptable by both objectives. The optimal (α, β) combination is the one which provides a balance between the incurred delay values and the total consumed sensing energy. Several approximation methods are proposed and used to identify the most desirable Pareto points [41]. The most common approach is an approximation method based on scalarization which combines the multiple objectives into one single-objective function using weighted-sum [42]

$$\min_{\alpha,\beta} \omega \cdot \frac{E(\alpha,\beta)}{E_{\max}} + (1-\omega) \cdot \frac{D(\alpha,\beta)}{D_{\max}}, \tag{6}$$

where $0 < \omega < 1$ is the weighting factor, E is the energy function, D is the delay. $E_{\max}$ is the maximum energy and $D_{\max}$ is the maximum delay, and are used to normalize the energy and delay as shown in (6). The set of solutions, called non-dominated solutions, which provide a balance between the different objectives, can be generated by varying the weight ω in (6) [43]. Since α and β are varied to balance the different objectives, ω can be set arbitrarily. We choose to set ω to 0.5; thus giving equal weight for both energy and delay components in (6). It is worth noting that $E_{\max}$ is proportional to the time limit $\hat{T_j}$ at which decisions take place. In addition, the maximum delay $D_{\max}$ that can occur is $\hat{T_j}$ which represents the case when sensing schedule decides to sense only at $\hat{T_j}$. Therefore, if we change the time limit for the same choices of Pareto optimal α and β, we will keep the same equal proportion of energy and delay. By normalizing the energy and delay values, we avoid re-computing the Pareto optimal (α, β) combination again each time a state is encountered in real-time.

4.2.2 *Instantaneous Rewards*

As shown in Table 2, there are four combinations of actions and states; therefore, there are four instantaneous rewards $r()$ as shown in Fig. 3. The dashed lines in the figure represent the rewards when the action is not to sense. The instantaneous reward $r()$ depends on the current state of the user, the decision taken whether to sense or not at each instantaneous time instant, and the time elapsed since the last sensing mechanism Δt_{i+1}. For each state and action combination, the corresponding reward is obtained by summing up the corresponding energy, delay, and recognition components as follows:

1) When the state is still in the same state s_j and the action is to sense $a_i = 1$ at the corresponding time instant, there is an energy cost
$$r(s_j, 1, \Delta t_{i+1}) = -1. \tag{7}$$
2) When the state is still in the same state s_j and the action is not to sense $a_i = 0$, there are neither energy nor delay rewards
$$r(s_j, 0, \Delta t_{i+1}) = 0. \tag{8}$$
3) When the state has changed to $\bar{s_j}$ and the action is to sense $a_i = 1$, there is an energy cost and a negative delay cost which is directly proportional to α and Δt_{i+1}. In addition, there is a positive recognition reward for the correct decision but this reward is inversely proportional to Δt_{i+1}. As β increases, delay is given a higher weight thus emphasizing more sensing
$$r(\bar{s_j}, 1, \Delta t_{i+1}) = -1 - (\alpha \cdot \Delta t_{i+1}) + \beta/\Delta t_{i+1}. \tag{9}$$
4) When the state has changed to $\bar{s_j}$ and still the action is not to sense $a_i = 0$, there is negative delay
$$r(\bar{s_j}, 0, \Delta t_{i+1}) = -(\alpha \cdot \Delta t_{i+1}). \tag{10}$$

4.3 Viterbi-Based Algorithm for Optimized Sensing Schedule

This section presents the Viterbi learning algorithms that output the sensing schedule. Our proposed algorithm is adaptive

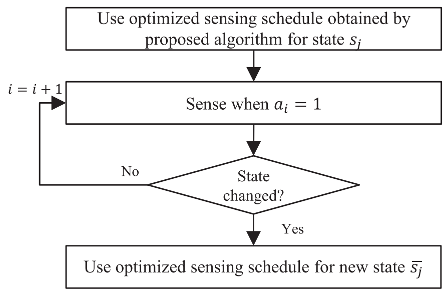

Use optimized sensing schedule obtained by proposed algorithm for state s_j
$i = i + 1$
Sense when $a_i = 1$
No
State changed?
Yes
Use optimized sensing schedule for new state $\bar{s_j}$

Fig. 4. Viterbi-based algorithm resets when detecting new state.

to the state under consideration, where the optimized sensing schedule depends on the state being monitored. Each state s_j has its own time statistics such as the time limit spent in this state is $\hat{T}_j$. If at any time instant sensing reveals a change in state, a new optimized sensing schedule is started with new state related rewards as shown in the Fig. 4.

Algorithm 1 describes how the time statistics for state s_j are computed and shows the steps to compute the rewards $R()$ which are the inputs for the Viterbi-based algorithm described later in Algorithm 2. To derive the sensing schedule, Algorithm 1 takes as input the whole collected historical data of the user model that captures how much time the user spends in particular contexts. Another input to the algorithm is the energy-delay weighting factors α and β. The first step in Algorithm 1 is to compute the mean μ_j and the standard deviation σ_j from the historical data (line 1). Then, the time limit $\hat{T}_j$ is chosen to be greater than the mean μ_j according to Section 3.3 while δ_j is chosen as the minimum $\delta_{j_{\min}}$ based on Section 3.4 (lines 2 and 3). The number of time instants N_j at which sensing decision should take place is derived as $N_j = \hat{T}_j/\delta_j$ (line 4). Survival probabilities are computed based on (3), and instantaneous rewards and computed based on Table 2 (lines 5 and 6). Finally, the rewards $R()$ are derived using $p_j(t_i)$ and $r()$ as described in (5).

Algorithm 2 presents the pseudocode to learn the optimized sensing schedule by searching for the best Viterbi path that chooses the best action a_i at each time instant t_i. The general Viterbi algorithm works as follows. At each step $t_i = t_1$ to t_{N_j-1} (line 1), it computes the cost of the different possible paths and keeps the path with the highest reward for each node. There exists a path from each node at t_i to reach each node at t_{i+1}. Let A be the total number of nodes at each time instant which is $A = 2$ in our case. For each node decision at t_{i+1} representing either $a_{i+1} = 0$ or $a_{i+1} = 1$ (line 2) there are two paths: one originates from $a_i = 0$ having $R(a_{i+1}, \Delta t_i + 1, t_i)$ and the other path originates from node $a_i = 1$ having $R(a_{i+1}, 1, t_i)$. For each node decision a_{i+1}, Viterbi algorithm keeps only one path leading to this node which is the path with the highest cumulative metric $R()$ (line 3). This process is repeated for all time instants until reaching t_{N_j} where the algorithm backtracks back to the starting point to find the best path with the highest cumulative metric (line 6).

4.4 Triggering of VCAMS Learning Mode

Whenever a new sensing schedule is needed, the system computes the expected rewards using a collected historical dataset for the particular user (Algorithm 1) and runs the Viterbi algorithm to maximize those rewards (Algorithm 2). However, user's behavior may change over time; therefore, the sensing schedule must adapt to the new data available in real-time by running again the needed learning algorithms and deriving new sensing schedules. To reduce computational overhead, the system should trigger the learning mode for a new sensing schedule only when significant behavioral changes happen. We assume that a behavior has a measurable time length that characterizes it; therefore, a significant behavioral change happens when the time span spent in the state changes significantly from its typical value.

Algorithm 1. Computing the Time Statistics and the Rewards for State s_j

```
Input:
- Historical data of statistics in state $s_j$
- Energy-delay weighting factors $\alpha$ and $\beta$
Output:
The rewards $R(a_{i+1}, \Delta t_{i+1}, t_i)$
1: Compute the mean $\mu_j$ and the standard deviation $\sigma_j$ for the
   time duration $T_j$ spent in state $s_j$
2: Choose time limit $\hat{T}_j > \mu_j$ based on Section 3.3
3: Choose sampling interval $\delta_j = \delta_{j_{\min}}$ based on Section 3.4
4: Derive the number of time instants $N_j = \hat{T}_j/\delta_j$
5: Compute the survival probability $p_j(t_i)$ for $t_i = t_1$ to $t_{N_j}$
6: Compute the instantaneous rewards $r()$ for all time instants
   based on Table 2
7: Derive the rewards $R()$ using (5)
```

Algorithm 2. Choosing the Optimized Sensing Schedule a_i

```
Input:
- Rewards $R(a_{i+1}, \Delta t_{i+1}, t_i)$ (computed in Algorithm 2)
- Number of time instants $N_j$
Output:
Best action $a_i$ for time instants $t_i = t_1$ to $t_{N_j}$
1: for each $t_i = t_1$ to $t_{N_j-1}$ do
2:     for each of the $A$ nodes at $t_{i+1}$ representing $a_{i+1} = 0$ to 1 do
3:         Compare the values $R(a_{i+1}, 1, t_i)$ and $R(a_{i+1}, \Delta t_i + 1, t_i)$
           and keep the path to $a_{i+1}$ which maximizes the cumulative reward $R()$ up to time $t_{i+1}$
4:     end for
5: end for
6: Choose the path with the highest cumulative reward at
   $t_i = t_{N_j}$ to solve (1): $\arg\max_{a_i; i=1\ldots N_j} \sum_{i=1}^{N_j} R(a_{i+1}, \Delta t_{i+1}, t_i)$
```

Mathematically, to track the user's behavior and quantitatively explain what significant behavioral change means, the system tracks the statistics associated with state s_j which include the mean value μ_j and the standard deviation σ_j, and constantly updates them as new measurements are captured in real-time. To quantify this relative change from the typical norm, we measure the relative difference between the new time spent and the typical average time spent μ_j. If this normalized difference is more than 1, we assume that there is a significant change in the behavior. Let's denote the new captured time duration spent in s_j as v_j. Behavioral changes are assessed with every new measurement v_j by comparing it to the previously tracked historical average μ_j, and assessing the ration $\kappa = \frac{|\mu_j - v_j|}{\sigma_j}$. When the deviation from the average is less than σ_j, the previously computed sensing schedule for this state is not changed since v_j would still be within the bound of the time limit $\hat{T}_j = \mu_j + \sigma_j$. Behavioral

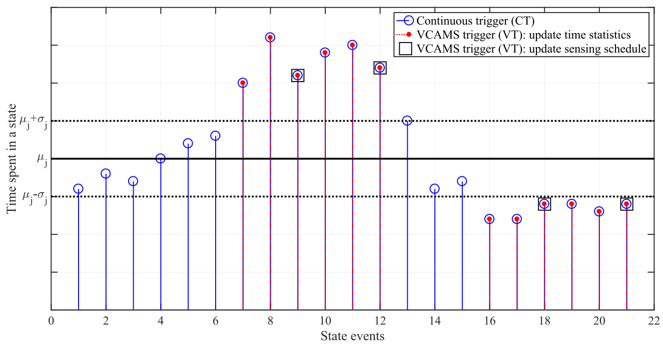


Fig. 5. Strategy to decide when to trigger VCAMS's learning mode computations.

change is flagged as having significantly changed if $\kappa > 1$. When the system detects that a state s_j has been flagged three times, it triggers the learning mode to derive a new sensing schedule ss_j. Therefore, the rules that decide when to trigger the learning mode form our proposed VCAMS adaptive trigger strategy (VT).

The rules for triggering VCAMS's learning mode are as follows:

- $\kappa \leq 1$ (no change in behavior):
  - Update the time statistics.
- $\kappa > 1$ (change in behavior):
  - Update the time statistics.
  - Derive new sensing schedule after the behavior is flagged as having significantly changed for three times.

Fig. 5 shows a simulation of measurements v_j, and indications of when learning modes are triggered. Without our proposed rules (VT), the default mode is continuous triggering (CT) which triggers learning mode with every new measurement. The x-axis represents the count of state events that increments each time this specific state is encountered. The y-axis represents the time measurement v_j spent by the user in the particular state. The horizontal line in the middle of Fig. 5 shows the mean value μ_j while the dotted lines illustrate the $\kappa = 1$ threshold of relative differences. The instants at which VT updates the time statistics are illustrated by the red dots whereas the black squares indicate the time instants at which VT triggers VCAMS's learning mode to update the sensing schedule. The blue circles indicate the time instants at which CT triggers the learning mode. The figure illustrates the savings in overhead, where 12 of the 22 events are triggered for updates and only 4 events are triggered for learning.

4.5 Computational Complexity Analysis

We now comment on the overall computational complexity of the proposed solution. We distinguish between the learning mode and the execution mode. The learning mode runs offline once to derive Pareto parameters for the multi-objective function and initialize the sensing schedules from historical collected dataset, and it runs as needed with behavioral changes to derive new sensing schedules. In the online execution mode, the system selects based on the current user's state the sensing schedule from the LUT; therefore, there are minimal computations, and thus the complexity is low.

Pareto optimal parameters for the multi-objective function are derived once offline by repeating Viterbi P times where P is the number of possible combinations for the parameters α and β. While α and β are continuous variables between 0 and 1, a finite search grid can be used without loss of accuracy. In our experiments, we varied α and β between 0 to 1 in steps of 0.1 (i.e., 11 values of α combined with 11 values of β, leading to $P = 121$ different combinations). The other parts of the learning mode (Algorithms 1 and 2) run offline using the derived Pareto points to initialize the system and derive the sensing schedules using a Viterbi algorithm based on historical data of the user model.

The feature extraction and context detection steps shown in the execution mode in Fig. 1 depend on the classification algorithm which is being used. In general, there is a trade-off between the computational complexity and the accuracy of classification algorithms. To improve the performance, we can use an ensemble of classifiers rather than one classifier; however, ensemble classification comes at the price of more computational energy and complexity. VCAMS is generic and any classification algorithm can be used along with VCAMS depending on the application and the triggered sensor. During execution mode, the system checks continuously whether VCAMS's learning mode should be triggered based on the rules set in VCAMS trigger strategy. When learning mode is triggered online, the time statistics are updated as described in Algorithm 1. These time statistics are algebraic aggregation functions; therefore, they can be defined as a scalar function of distributive computations [44], [45]. The computational complexity of calculating each of the mean and the standard deviation is low of order $\mathcal{O}(1)$ with updates.

When VT decides to update the sensing schedule online, Algorithm 2 is triggered where the optimized sensing schedule is derived by running Viterbi-based approach. A trellis diagram shown in Fig. 2 is used to represent the optimal path selection problem. Let us denote "A" as the number of distinct possible actions at each node. There are A paths that reach each node in the trellis. Viterbi algorithm [28], [36] reduces the computational complexity by using the recursion where only the node with the highest cumulative reward survives. The time complexity of the Viterbi algorithm is then $\mathcal{O}(N_j \cdot A^2)$ where N_j is the number of total time instants for state s_j, which has been shown to be smaller when compared to alternative searching algorithms [46], [47].

5 Evaluation Using a Context Simulator

Several experiments were conducted with simulation to test the efficiency of our proposed method. The optimized VCAMS sensing schedule was benchmarked against continuous sensing and other state-of-the-art methods to show energy and delay enhancements. We did two sets of experiments. In the first set of experiments, we focused on examining performance with two scenarios with one state change. For one scenario, we used a simulator monitoring one state assessed energy and delay. The second scenario was an implementation on an Android device to reflect real usage and estimate the computational costs. In the second set of experiments covered in Section 6, we used real data benchmark with multiple states in real-life situations. Furthermore, we conducted analysis for the different parameters used in the algorithms. Different energy-delay weighting factors (α, β) combinations were studied to investigate their effect on the trade-off between energy consumption and delay. In addition, the effects of the system parameters such as sampling interval δ_j and time limit $\hat{T}_j$ were investigated.

5.1 Simulation Experiments' Setup

We used the Siafu platform [48], which is an open-source context simulator developed in Java. We used the office scenario which is modeled based upon the typical behavior of an employee where agents get to work in the morning, work at their desks and attend meetings. To illustrate the method, we chose the state "$s_j =$ AtMeeting" which represents meeting attendance detected using the range of WiFi hotspots. We divided the data into two parts. The first part was used as historical data for initial learning to develop the user model where we obtain the time limit $\hat{T}_j$ for each state/activity s_j and the probability distribution of the time spent in each state and develop the initial sensing schedule for each state. The second part of the data was used for testing in the execution mode to validate the efficiency of the proposed algorithm in detecting state transitions. The time statistics derived from historical data were mean time $\mu_j = 100$ min and standard deviation $\sigma_j = 30$ min. The average time of a single WiFi scan time is 10 sec; however, running a WiFi scan every 10 sec is too restrictive; therefore, we chose to set the value of $\delta_j = 1$ min based on the sensor requirements in addition to state and application restrictions as discussed in Section 3.4. As for the testing data, we ran the simulator to generate 10,000 values of the actual time T_j spent in the meeting state. Energy and delay were extracted for each of these 10,000 runs using evaluation methods detailed in next Section 5.2, and then the average values of energy and delay for these 10,000 runs were computed. For the phone implementation of VCAMS, we selected an Android-based HTC Desire which is equipped with Octa-core 1.7 GHz processor.

5.2 Sensing Energy and Detection Delay Evaluation

The objective of VCAMS is to trade-off two factors: sensing energy and delay. Hence, we should evaluate these two criteria.

5.2.1 *Sensing Energy in Terms of Sensing Schedule*

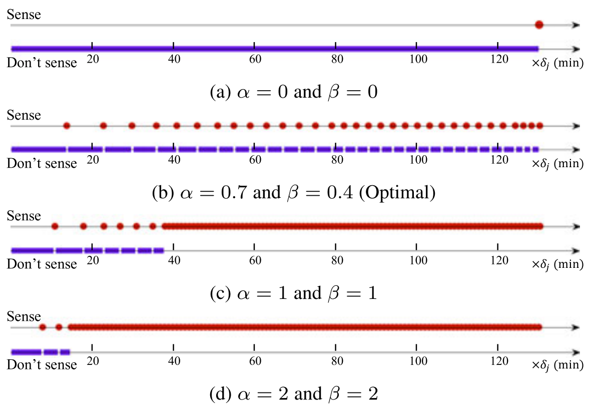
Given the optimized sensing schedule which decides when to trigger sensing, define k to be the number of times the sensor is triggered from the time instant the system recognizes being in state s_j till it detects the transition into state $\bar{s_j}$. Each sensor has its own power characteristics which lead to different energy consumptions per trigger. Hence, rather than specifying the sensor-specific Joules consumed each time the sensor is triggered, we evaluated the general energy E as the number of time instants at which sensing is triggered. E can be used for all sensors. We evaluated E as follows:

$$E = k. \tag{11}$$

5.2.2 *Delay Intervals in Terms of Sensing Schedule*

As for the delay, it is the number of time instants between the instant when context change actually happened and the instant when the following sensing decision happened. Assume t_u to be the real transition time instant and t_v be the subsequent time instant when sensing occurred and state transition is detected. Hence, the delay D can be computed as

$$D = t_v - t_u. \tag{12}$$



(a) $\alpha = 0$ and $\beta = 0$

(b) $\alpha = 0.7$ and $\beta = 0.4$ (Optimal)

(c) $\alpha = 1$ and $\beta = 1$

(d) $\alpha = 2$ and $\beta = 2$

Fig. 6. VCAMS sensing schedule for different (α, β) combinations.

5.3 Effect of Energy-Delay Weighting Coefficients α and β

The normalized weighting factors α and β reflect the relative effect of delay with respect to energy in each of the reward functions presented in Equations (7), (8), (9), and (10). Varying these weighting factors affects the derived sensing schedule. In this experiment, we took one standard deviation above the mean to obtain the time limit $\hat{T}_j = 130$ min before continuous sensing is triggered.

Fig. 6 demonstrates VCAMS sensing schedule that was obtained for different values of α and β. The figure shows VCAMS decision at each time unit whether to sense or don't sense. For each subfigure, the upper axis represents the time instants at which sensing is triggered, and the lower axis represents the other time instants at which no sensing is triggered. Therefore, the red dots mean "sensing" time instants and the blue boxes illustrate the "no sensing" instants. Fig. 6 illustrates four specific combinations of (α, β). Figs. 6a, 6b, 6c, and 6d capture (α, β) = (0, 0), (0.7, 0.4), (1, 1), and (2, 2) respectively. We chose those combinations because they are extreme values for the weighting coefficients. It can be seen that as α and β increase, the system approaches continuous sensing. Fig. 6a shows that when $\alpha = 0$ and $\beta = 0$, VCAMS decision is to sense only once when $t = \hat{T}_j$. Therefore, the lower bound of α and β cannot go below 0. Fig. 6b presents the Pareto optimal behavior of VCAMS that will be shown in Fig. 8. The duration of idle times when no sensing is triggered decreases with time as t approaches $\hat{T}_j$. Fig. 6c shows the behavior of VCAMS when both α and β are unity. The continuous sensing behavior appears in this case. Fig. 6d shows that increasing α and β beyond 1 will lead to excessive continuous sensing behavior; thus, we restrict α and β to be less than 1 to avoid excessive energy consumption. Therefore, we varied each of α and β between 0 and 1.

Fig. 7 shows the effect on energy and delay for different values of alpha and beta. We varied each of α and β between 0 and 1 in steps of 0.1 and for each combination, we generated 10,000 values of T_j and computed the energy and delay for each. Then, the normalized average energy and delay were computed. Fig. 7 shows the effect of varying α and β on energy and delay. It can be seen that energy increases with the increase of α and β. Delay decreases with the increase of α, and it slightly decreases with the increase of β. Therefore, increasing α and β causes more sensing; thus more energy and less delay. This result is consistent with the discussion in Section 4.2.

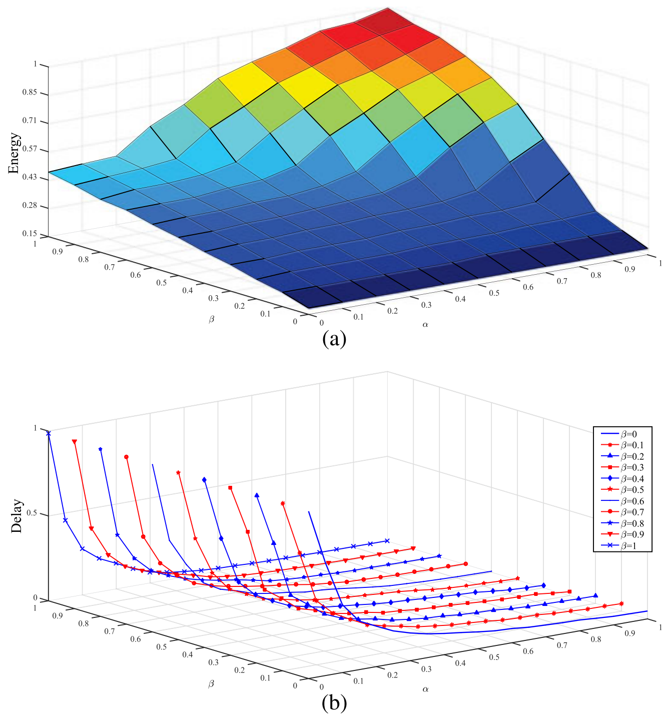


Fig. 7. Effect of different choices of α and β on energy (a) and delay (b).

To solve the trade-off between energy and delay, we searched for the best (α, β) combination which we solved by finding a Pareto optimal point. Pareto point is the one acceptable by both objectives where an improvement in energy requires a degradation in delay and vice-versa. We used the normalized averages of energy and delay by dividing each of the two factors by its corresponding maximum value. A Pareto optimal solution was derived to find the Pareto optimal α and β values as shown in Fig. 8. These values came out to be $\alpha = 0.7$ and $\beta = 0.4$. Therefore, for the remaining experiments, we set α and β to their Pareto optimal values. Fig. 8 shows the non-dominated points and the Pareto optimal point illustrated as the star point which gives a balance between the total energy consumed and the delay incurred.

5.4 Impact of False Positive in Context Classification

When the system triggers sensing, the sensor collects raw data which undergoes processing to recognize the state using a classification algorithm which is characterized by an average classification accuracy. In this section, we study the impact of false positive and false negative events on the system's performance. A false positive occurs when the system erroneously detects a context change, and a false negative occurs when the system erroneously decides that the context has not changed. We will refer to a false negative as FN and the false positive as FP. The impacts of FN and FP are shown in Fig. 9. If the system makes a FN, extra delay will be accumulated until the system correctly detects that the state has changed. During that period of time, the system will be in an erroneous state period due to the FN. On the other hand, if the system makes a FP for current state, the real state will be lost and the error may accumulate until the system gets back on the correct track. Such a false positive event causes another kind of erroneous state period of time. We use the term "undesired period" to refer to any period caused by a false detection where the system is in an erroneous state.

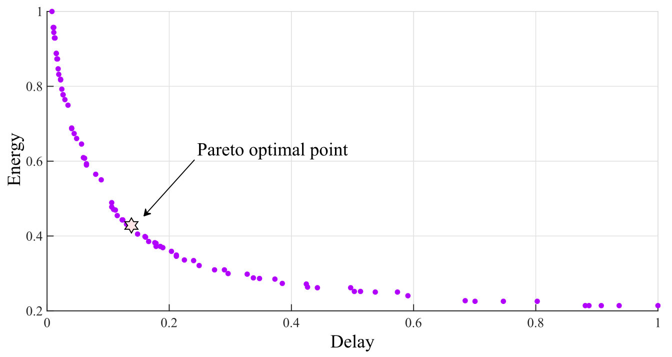


Fig. 8. Pareto effect.

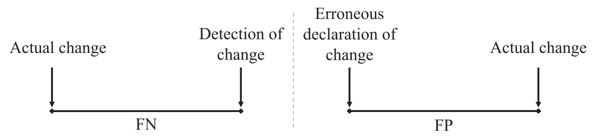


Fig. 9. False positives (FP) versus false negatives (FN).

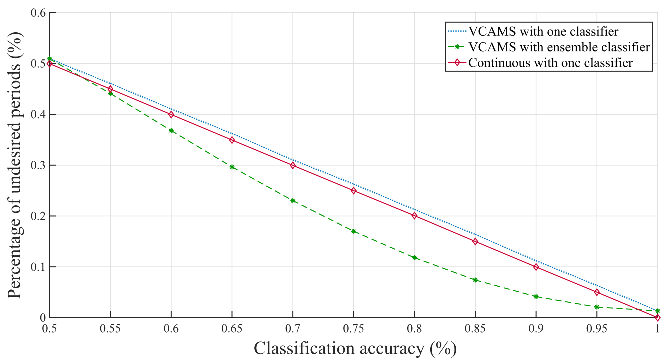


Fig. 10. Percentage of undesired periods versus the classification accuracy.

To analyze the tolerance of our proposed approach to false positives and false negatives of triggered activities, we ran VCAMS for the state "s_j = AtMeeting" and we varied the classification accuracy between 50 and 100 percent. For each classification accuracy, we ran the simulator to generate 10,000 values of the actual time spent in a meeting state. The percentage of undesired periods was extracted for each of these 10,000 runs and then the average value was computed. The results illustrated in Fig. 10 show that running VCAMS using a more accurate classification algorithm leads to lower false positives and accordingly to a lower percent of undesired periods. VCAMS system's performance is impacted linearly with the accuracy of the classifier. From the figure we can see that even when classification accuracy is at 100 percent, there is still a small percentage of undesired periods. This is related to the VCAMS sensing strategy to trade-off energy for delays. Since even if the case of 100 percent context classifier accuracy, VCAMS may still choose to delay sensing to save energy. To eliminate any delay, VCAMS can switch to continuous sensing, which is shown in the red curve, but such a strategy would have higher energy cost.

To illustrate how classification accuracy can be improved even for low accuracy classifier, we simulated VCAMS with ensemble classification as a means to enhance classification accuracy. Instead of collecting raw sensor data for one classification, we introduced a sequence of three consecutive

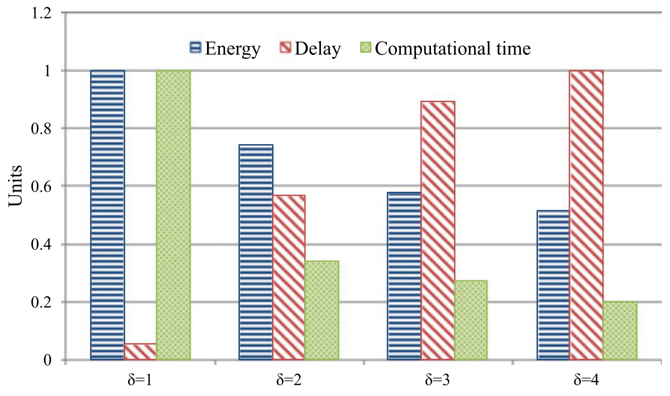


Fig. 11. Energy, delay, and computational time for different δ_js.

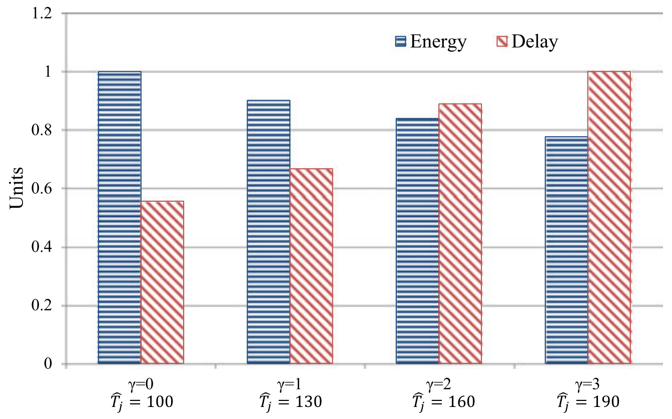


Fig. 12. Energy and delay for different choices of $\hat{T}_j$s.

classifications where the final decision uses a majority rule. When a sensing decision is triggered by VCAMS, the classifier requires three consecutive readings to feed the majority rule. The figure shows that VCAMS with ensemble classifier causes less percentage of undesired periods than that of both VCAMS with one classifier and continuous sensing. It is worth noting that while ensemble classification improves overall performance, it comes at the price of more computational energy and complexity. For the remaining experiments, we set the classification accuracy to a constant value such that we can study the trade-off between energy and delay.

5.5 Effect of Sampling Interval δ_j

We described the choice of δ_j in Section 3.4. We chose to set the value of δ_j to the minimum possible $\delta_{j_{\min}}$ based on the sensor requirements and state restrictions. $\delta_{j_{\min}}$ minimizes delay; however, it still consumes more energy than higher values of δ_j and it causes more computational time. In this experiment, we set $\alpha = 0.7$ and $\beta = 0.4$ and we tried four different values for δ_j to show its effect on energy, delay, and computational time. For each value of δ_j, we ran the experiment again for 10000 times to compute the normalized averages of energy, delay, and computational time. Their values were normalized by dividing each of energy, delay and computational time by its corresponding maximum value. Fig. 11 shows how the choice of δ affects energy, delay, and computational time values. As δ_j increases, energy and computational time decrease while delay increases. In this experiment, we coincidently noticed also that our choice of $\delta_j = 1$ optimizes the energy-delay trade-off. When δ_j increases from 1 to 4, normalized units of energy decrease from 1 to 0.51 which is a 49 percent improvement in energy's behavior while normalized units of delay increase from 0.05 to 1 which is a 95 percent degradation in delay's behavior. On the other hand, normalized units of computational time decrease from 1 to 0.2 which is almost 80 percent improvement. In our proposed approach, we are trading-off energy and delay; thus, it can be concluded that the proportional delay loss exceeds the proportional energy gain when increasing δ_j. Therefore, taking δ_j to be the minimal is justified.

5.6 Effect of Time Limit $\hat{T}_j$

In Section 3.3, we chose $\hat{T}_j$ such that it is greater than the mean value μ_j to avoid excessive early continuous sensing. For experimentation, we choose:

$$\hat{T}_j = \mu_j + \gamma \cdot \sigma j, \tag{13}$$

where γ reflects the trade-off in energy versus delay. γ depends on the delay that the application is able to tolerate. In this experiment, the system operated in its optimal condition where $\alpha = 0.7$ and $\beta = 0.4$. δ_j was set to 1 according to the discussion in the previous section. To study the effects of varying γ, simulations were conducted for different values of γ. The results are presented in Fig. 12. Energy and delay values were normalized into units by dividing each by its maximum value to obtain a range of $\{0, 1\}$. Energy was examined for all the cases considered. The figure shows that energy decreases as $\hat{T}_j$ increases. Delay in recognizing a state change was also measured. Increasing $\hat{T}_j$ causes more delay since the sensor triggers around the average time are dispersed. Hence, we experimentally choose γ such that the delay is acceptable. When γ increases from 1 to 2, normalized units of energy decrease from 0.9 to 0.82 which is a 9 percent improvement in energy's behavior while normalized units of delay increase from 0.63 to 0.89 which is a 32 percent degradation in delay's behavior. Hence, increasing γ beyond 1 leads to delay loss which exceeds the slight energy gains.

5.7 Choice of Probability Distribution $p_j(t_i)$ and Its Parameters

In Section 3.2, we chose $p_j(t_i)$ to follow an exponential distribution since it gives a good fit that matches the context change. Furthermore, we set the state-based decaying hazard rate h_j shown in (3) such that the probability of survival approaches 0 when the time approaches the time limit $\hat{T}_j$. In this section, we will perform supplementary experimentation to show the impact of exponential distribution versus other more general distributions. In this experiment, the system operated in its optimal condition where $\alpha = 0.7$ and $\beta = 0.4$. To study the effect of the different survival probability distributions, we consider the general Weibull distribution [49]

$$p_j(t_i) = e^{(-h_j \cdot t_i)^{\xi}}, \tag{14}$$

where ξ is the shape parameter. Weibull distribution reduces to the exponential distribution when $\xi = 1$. We conducted simulations for different values of ξ, and the results are presented in Fig. 13. We used the normalized averages of energy and delay by dividing each of the two factors by its corresponding maximum value. The figure shows that as ξ decreases, energy decreases until it reaches the exponential distribution where $\xi = 1$ after which energy slightly increases. As for delay, it increases as ξ decreases. We chose the exponential distribution since it provides the lowest

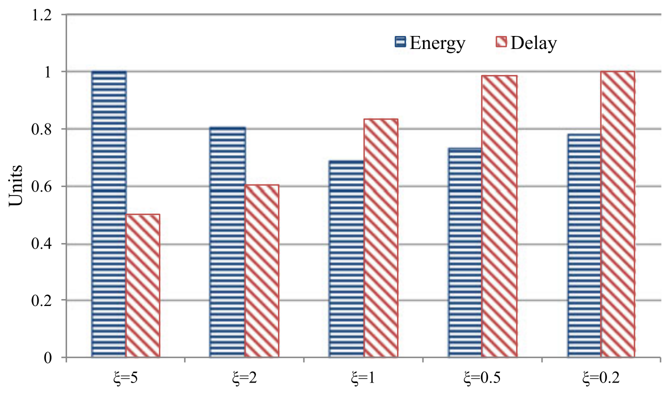


Fig. 13. Energy and delay for different choices of survival probability distributions.

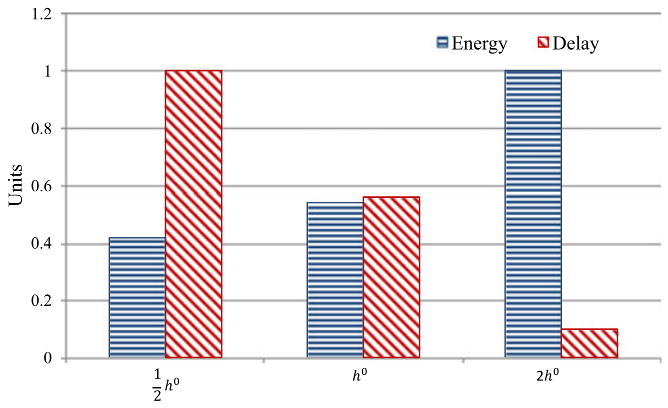


Fig. 14. Energy and delay for different choices of decaying hazard rate h_j.

energy and the trade-off between energy and delay. However, VCAMS is generic and any distribution can be used based on the data available and the context being detected.

We have also investigated the effect of changing the decaying hazard function h_j which is the instantaneous probability that an event will occur at a specific time. In exponential distribution, the decaying rate is constant over time. We consider ${h_j}^0$ to be the decaying rate that we used throughout the simulations. We conducted simulations to show the effect of faster decaying rates. We varied the decaying rate between $\frac{1}{2} \cdot {h_j}^0$ and $2 \cdot {h_j}^0$. The results are illustrated in Fig. 14. The figure shows that taking the hazard rate as ${h_j}^0$ gives the required balance since we are trading-off energy and delay.

5.8 Performance Analysis and Comparison

In this section, we first compare our work with existing similar work proposed by other researchers, then we experimentally analyze the computational complexity of our algorithm.

5.8.1 *VCAMS versus State-of-the-Art Sensing Schedules*

There are some existing sensing policies which researchers have proposed [8], [19], [22], [27] whose goal is to detect context in an energy efficient way. These proposed sensing mechanisms aim at trading-off energy and accuracy of state detection. Fig. 15 illustrates the differences in the kind of problems these different methods try to solve versus our work. While prior state-of-the-art works focus on optimizing the accuracy of detecting the correct current state, we focus

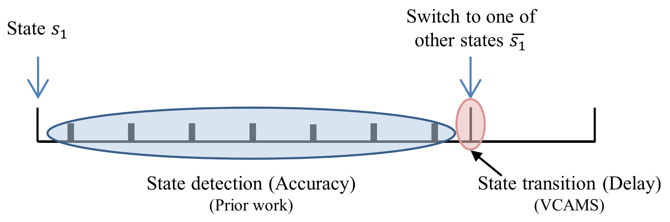

State s_1
Switch to one of other states $\bar{s_1}$
State detection (Accuracy)
(Prior work)
State transition (Delay)
(VCAMS)

Fig. 15. Illustration of VCAMS's aim versus state-of-the-art methods.

on optimizing the delay of transition detection between states. Previous methods focus on one state and recognize it accurately while minimizing the energy consumed, but can also be used to detect change in state. On the other hand, VCAMS aims at detecting with least delay the contextual change from one state to another while optimizing energy. To compare against these methods, we established continuous sensing as baseline for all. For the other methods, we reproduced their approaches with the aim of detecting state change by setting their parameters as follows:

- Wang et al. propose EEMSS [8] which uses fixed duty cycles; hence, it uses uniform periodic sensing. Their method uniformly skips a number of time units before triggering sensing. EEMSS triggers the accelerometer to sense 6 sec followed by 10 sec of sleep mode; thus, it is sensing 37.5 percent of the time.
- Lu et al. present Jigsaw [10] which is a continuous sensing engine. The authors use discrete-time Markov Decision Process to learn the optimal GPS duty cycle. There are some constants which they did not specify in their work; hence, we assumed the reward adjustment coefficient to be 2 and the penalty if the energy budget is depleted before the required duration to be -20 for comparison purposes.
- Chon et al. [19] present SmartDC which is a prediction-based scheme that porposes adaptive duty cycling to provide context while saving energy. The authors use order-1 Markov predictor to predict the time spent in a state; and they formulate the sensing decision problem as a Markov decision process. Through their experiments, they suggest setting the energy budget to 10 percent; thus, we used the same budget when reproducing their work.
- Yurur et al. [22] propose a sensing mechanism which assigns adaptive duty cycles and sampling frequencies for accelerometer to infer human activity. They propose a novel additive increase additive decrease (AIAD) approach which decreases the duty cycle hen the user's state is stable. The authors propose two methods to adjust the duty cycle; however, we compare VCAMS with the first method they propose since it outperforms the second method.
- As for Rachuri et al. [27], their system uses a learning technique to control the sampling rate of the sensors. They classify sensing actions into either success if detecting an unmissable event or failure if it is a missable event. The technique updates the probability of sensing based on the successes and failures. The probability in their work has an α_r as a weighting factor where energy and accuracy decrease as α_r increases. Through their experiments, they specify

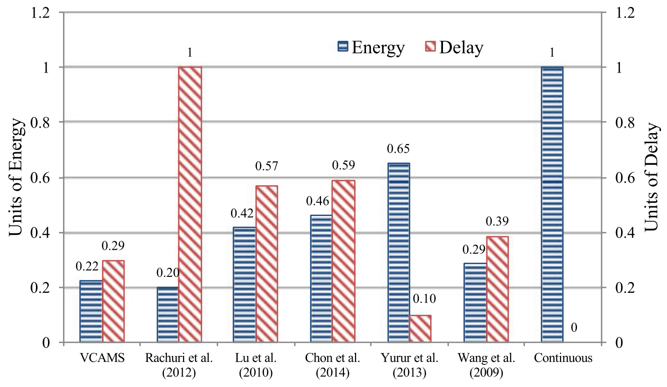


Fig. 16. Energy and delay for VCAMS and state-of-the-art sensing schedules.

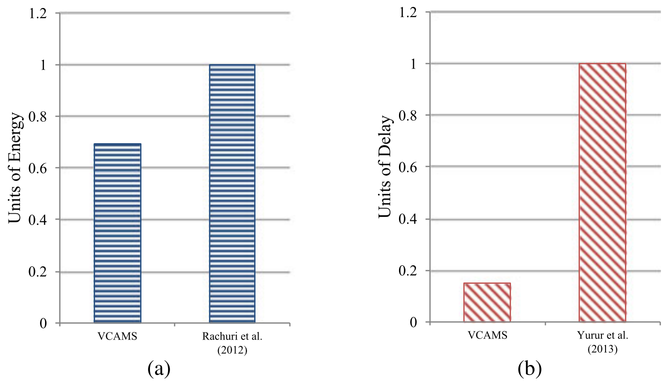


Fig. 17. Comparison between VCAMS and best energy prior work [27] for similar delay (a) and best delay prior work [22] for similar energy (b).

$\alpha_r = 0.5$; thus, we used the same value when reproducing their work.

We examined the average energy and delay values for each of the proposed methods. The simulation ran 10,000 times for each method to find the average energy and delay, then the metrics were normalized. The largest energy of all methods was used as base for energy normalization, and the largest delay of all methods was used to normalize delays and obtain values between 0 and 1. There were three types of comparisons that were conducted against VCAMS: The baseline case of continuous sensing, previous methods by considering both energy and delay, and best of previous methods when considering energy alone or delay alone. The results are summarized in Fig. 16. When comparing to continuous sensing, the delay is assumed to be zero for continuous sensing. VCAMS incurs delay, but has significant energy reduction of 78 percent.

When comparing to previous state-of-the-art methods, the results show that VCAMS gives a balance between energy and delay. It does not provide the lowest energy or the lowest delay of all methods; however, it provides the best trade-off of the two metrics (energy and delay) combined. Although Rachuri et al. (2012) system has energy less than VCAMS, it has the highest delay which is not practical for delay intolerant applications. On the other hand, the work proposed by Yurur et al. (2013) has the lowest delay, but it has higher energy since it applies continuous sensing when the time approaches $\hat{T}_j$. As for Wang et al. (2009), Lu et al. (2010) and Chon et al. (2014), they provide a trade-off between energy and delay; however, they still result in higher energy and delay than VCAMS.

To compare to best previous state-of-the-art methods considering energy or delay alone, VCAMS configuration had to be adjusted to set similar conditions. We used Rachuri et al. as reference for best energy and Yurur et al. as reference for best delay. To compare with prior method with best energy, VCAMS coefficients were set to $\alpha = 0.2$ and $\beta = 0.8$. This (α, β) combination gave similar delay performance of VCAMS compared to [27]; however, the results in Fig. 17 show that VCAMS outperforms Rachuri's system since it minimizes the amount of energy consumed by 30 percent. To compare with prior method with best delay, VCAMS coefficients were set to $\alpha = 1$ and $\beta = 1$. This combination of weighting factors gave the maximum energy consumption of VCAMS which is comparable to energy consumption of Yurur's proposed method. The results show that VCAMS provides better delay performance than Yurur's solution as shown in Fig. 17b. It's energy is comparable to Yurur's proposed method; however, it saves 75 percent of delay.

In summary, VCAMS outperforms all state-of-the-art methods when considering the combined trade-off of energy and delay. Furthermore, VCAMS can be configured (proper choice of α and β) to achieve lowest possible delay, lowest possible energy, or lowest possible trade-off combination.

5.8.2 Computational Power

To analyze the performance of our proposed approach on real systems, and to also illustrate the computational costs of VCAMS under realistic operational conditions, we implemented VCAMS on an Android-based device which is HTC Desire. The computational power consumed by the smartphone for executing the learning mode of VCAMS and extracting the optimal sensing schedule was measured by PowerTutor [50]. PowerTutor is an Android application that displays the power consumed by major system components: CPU, display, and network interfaces; and it displays the power consumed by each component separately. We ran VCAMS for the state "$s_j =$ AtMeeting". The results summarized in Table 3 show that running VCAMS to derive the optimized sensing schedule consumes almost 0.8 percent of the total CPU. In addition, the table shows that the computational power is almost 5 percent of the sensing overhead for the cheapest sensor (accelerometer) which is rather negligible, and hence the computational energy can be ignored.

6 Case Study with Real Activity Context

Several experiments were conducted to test the efficiency of VCAMS and demonstrate its effectiveness in real-life situations beyond simulation. The effects of different system parameters including sampling interval δ_j were investigated. In addition, experiments were conducted to validate the performance of VCAMS and compare it with prior state-of-the-art methods which were described in Section 5.8.

6.1 Setup

The real dataset was based on logs of Activities of Daily Living (ADL) [51]. This dataset was collected in smart homes for two users capturing the daily activities performed by each user for 35 days in their own home. The activities accompanied with their labels include sleeping (s_1), toileting (s_2), showering (s_3), having breakfast (s_4), grooming (s_5), spending spare time (s_6), leaving (s_7), and having lunch (s_8). Data was divided into two parts: 1) historical data

TABLE 3
Computational Power

Application	Power (mW)
VCAMS	28
Accelerometer	551
System	3,400

TABLE 4
Time Properties of Activities

Activity	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Mean (sec) μ_j	33,700.8	191.7	403.5	506.3	118.5	6,749.1	7,134.6	2,090.1
Standard deviation (sec) σ_j	2,310.6	331.2	267.4	141.0	188.1	5,725.8	5,360.5	551.4

which was used in the learning mode to obtain the time limit $\hat{T}_j$ of the user in each state/activity s_j and the probability distribution of the time spent in each state and 2) test data which was used in the execution mode to validate the efficiency of our proposed algorithm in detecting state transition. The derived Pareto-optimal values were $\alpha = 0.7$ and $\beta = 0.4$. As expected from Section 4.2, they turned out to be the same as the simulation case since if we change the interval for the same choices of α and β, we keep the same relative proportions of energy and delay. Therefore, there is no need to re-compute the Pareto optimal (α, β) combination again each time a state is encountered.

Table 4 represents the mean μ_j and the standard deviation σ_j of each state/activity s_j. As expected, for real life activities, the time properties differ between activities. Some last longer than others, for example sleeping's average time was 9 hours; whereas toileting has an average of 3 min. Some have less variations than others, which probably indicates routine scheduled activities versus ad-hoc activities.

6.2 Applicability of VCAMS on Real Data Traces

The statistics (μ_j and σ_j) of time spent in each state/activity depended on the behavior of the user in a specific context. To find the time limit $\hat{T}_j$ for each state s_j, we set μ_j and σ_j according to Table 4. The time limit $\hat{T}_j$ for each state s_j was set to the mean of the time spent in the activity plus one standard deviation. We chose to set the value of sampling interval $\delta_j = 20$ sec so that we use higher granularity than 1 min limited by the sensor requirements and speed of system. However, we also investigated the effect of using higher values of δ_j on dynamic real data to compare with simulation results.

The experiments aimed at determining in real-time the activity of the user, and then detecting changes in activities while optimizing delay and energy consumption. We implemented VCAMS for the 35 days data to monitor the user's activities and test the applicability of our method on dynamic real data. We ran the algorithm on 2.7 GHz Intel Core i5 processor machine. The delays in detecting transitions between states were measured by taking the difference between real transition time and time at which our system detected the transition.

Fig. 18 shows part of the time line from the results of applying VCAMS's execution mode to two activities: sleeping (s_1) and spending spare time (s_6). We focused on these two activities since they have different behaviors. Sleeping has a very small standard deviation compared to its mean, while spending spare times has a standard deviation which is very high compared to is mean time as shown in Table 4. In addition, these two states can be easily illustrated and visualized on a time line. The dot on the time line indicates the beginning of a new activity. The arrow pointing downwards indicates the true transition from s_j and the arrow pointing upwards illustrates the time at which VCAMS detects the transition from s_j. The time difference between the two arrows is the delay in detection which we are trying to minimize while optimizing energy consumed. According to the results, s_1 has delays 20 and 40 sec while s_6 has delays of 60, 100 and 0 sec. However, delay on average will not exceed 1 min for these two states as shown in Fig. 19b. The average delay for sleeping is 15.7 sec which is 0.04 percent of the mean time of s_1. Whereas, the average delay of s_6 is 47 sec which is 0.7 percent of the mean time. Both delays are considered acceptable compared to their mean times. s_6 has a higher delay since it has more variability around the mean time as reflected by the standard deviation value. Although some prior methods such as continuous sensing and [22] will cause less delays, they will cause excessive amounts of unnecessary energy.

Fig. 19 demonstrates the energy consumed, the delay incurred and the computational time spent to compute each activity. In addition, the figure shows the variation of energy, delay and computational time versus δ_j. Fig. 19a shows the number of times the sensor is triggered for the different activities which directly corresponds to the energy consumed since energy is represented in our proposed approach as the number of times sensing is triggered as described in Section 5.2. Energy increases as the mean of the activity increases. This is due to VCAMS sensing times increasing when the mean time μ_j increases. For instance, considering $\delta_j = 20$ sec, sleeping (s_1) has a mean time of 33700.8 sec and an energy consumption of 395.6 whereas toileting (s_2) has mean time of 191.7 sec and average energy of 4.7. Another observation is the considerable decrease in energy compared to continuous sensing. For example, the energy consumption for state s_1 when VCAMS is used is 395.6; whereas, the energy consumption when using continuous sensing for state s_1 would be 1,685. Therefore, VCAMS saves up to 77 percent energy. Furthermore, increasing δ_j leads to less energy consumption in all activities as the sensor in this case has fewer time units at which it is triggered for a reading. This result confirms the results of Section 5.5. Fig. 19b presents delay versus activity for two values of δ_j. Delay varies between activities, and it

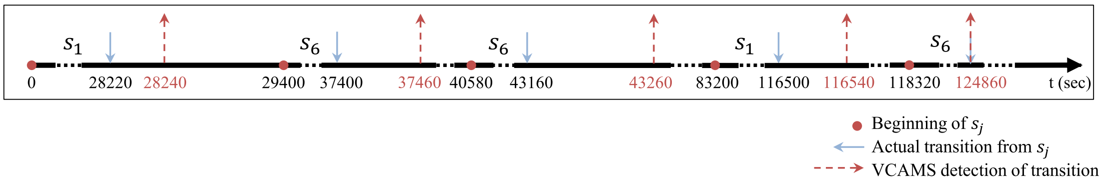


Fig. 18. Dynamic real-time illustration of applying VCAMS to detecting transitions for s_1 and s_6 activities.

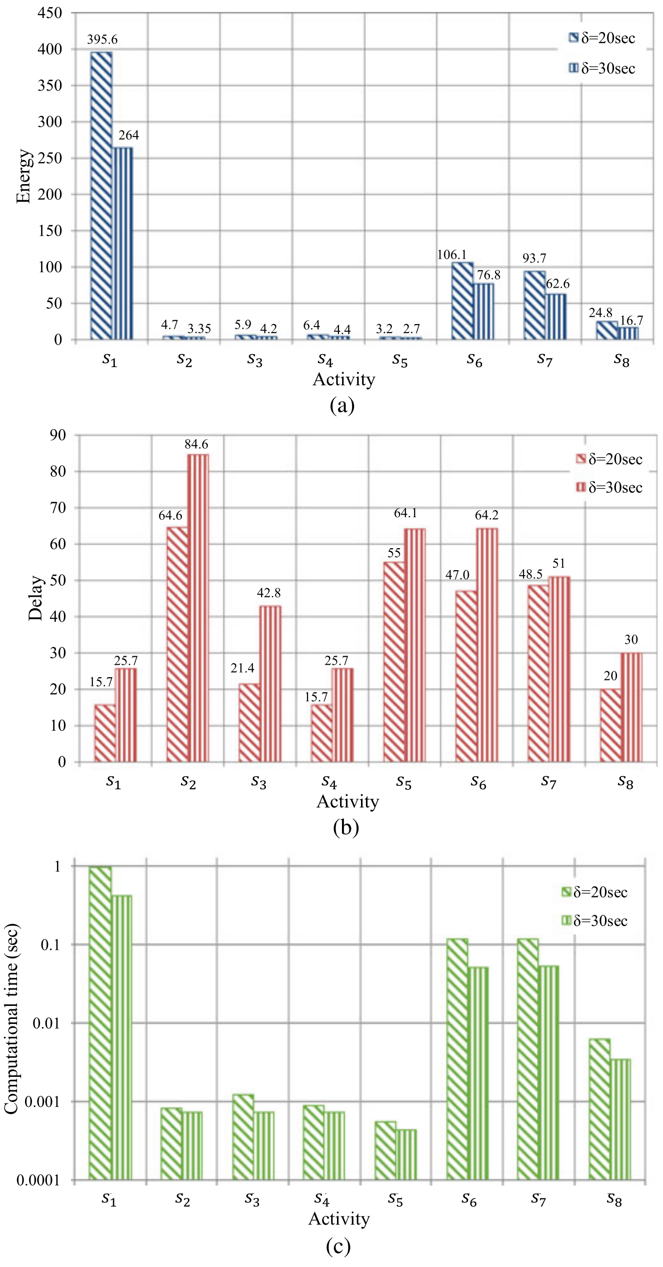


Fig. 19. Energy (a), delay (b), and computational time (c) for activities considering two different δs.

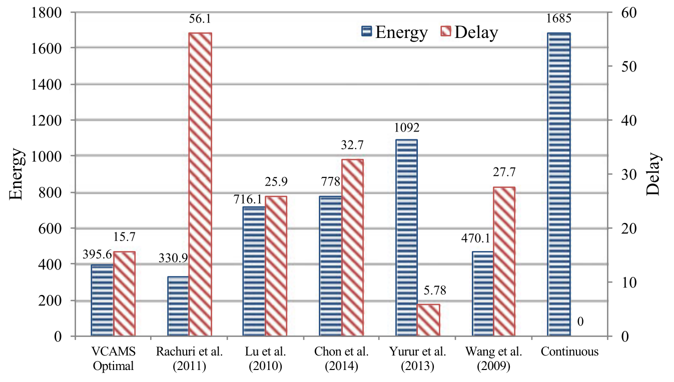


Fig. 20. Energy and delay for optimal VCAMS and prior state-of-the-art sensing schedules for sleeping activity s_1.

reaches maximum of 84.6 sec for activity s_2 which has a standard deviation higher that its mean; hence, data is more dispersed. Delay decreases when minimizing δ_j as the intervals between sensing become smaller, with higher chances of early detection of state changes. Fig. 19c presents the computational time spent in the learning mode of VCAMS for each activity separately. Computational time varies between activities, where activities with higher time limits require more computational times to obtain the best sensing schedule. In addition, increasing δ_j leads to less computational time since the number of total time instants N_j for state s_j decreases. These results validate Viterbi's complexity analysis presented in Section 4.5.

6.3 Comparison of VCAMS to Other Methods with Real Data Traces

To compare the performance of VCAMS with other existing approaches, we reran each prior state-of-the-art methods presented in Section 5.8. For fairness, we fixed $\delta_j = 20$ sec for all approaches. The overall energy and delay results for activity s_1 are presented in Fig. 20. Although we focused on one activity due to space limit, other activities have similar analysis when comparing VCAMS to other prior methods. Comparing VCAMS with continuous sensing shows almost 77 percent decrease in energy with a slight increase in delay from 0 to 15.7 sec which is negligible compared to the mean value of sleeping activity. Rachuri's approach results in 16 percent decrease in energy compared to VCAMS, but it causes 72 percent increase in delay. As for Wang et al. (2009), Lu et al. (2010) and Chon et al. (2014) approaches, they result in higher energy and delay than VCAMS. On the other hand, Yurur's approach gains on the delay side; however, it costs 63 percent more energy. In summary, when applied on real data, the results were consistent with simulations, and VCAMS outperformed other state-of-the-art methods.

6.4 Computational Complexity

To show the effectiveness of our proposed VCAMS trigger strategy described in Section 4.4, we considered different adaptive strategies and showed that VT is the best in terms of the trade-off between energy, delay and computational time. We considered the following strategies:

- WT "without any trigger": In this case, the learning mode is run offline only once to derive the sensing schedule, and no further adaptation is applied.
- CT "continuous trigger": In this case, the learning mode runs online each time a new state is encountered.
- ST "state-based trigger": For this case, the learning mode runs only during the longest state which can be obtained from analyzing the historical data. The system keeps track of all the behavioral changes encountered in all states but it does not trigger the learning mode except when the longest state is encountered. Since we are monitoring the user's activities during the day, the longest continuous state is sleeping which is s_1 during which most users charge their devices.
- VT "VCAMS proposed selective triggering": This case implements the rules proposed in Section 4.4 to dynamically decide when to trigger VCAMS's learning mode.

The results are illustrated in Fig. 21, which shows the normalized averages for sensing energy, delay, and

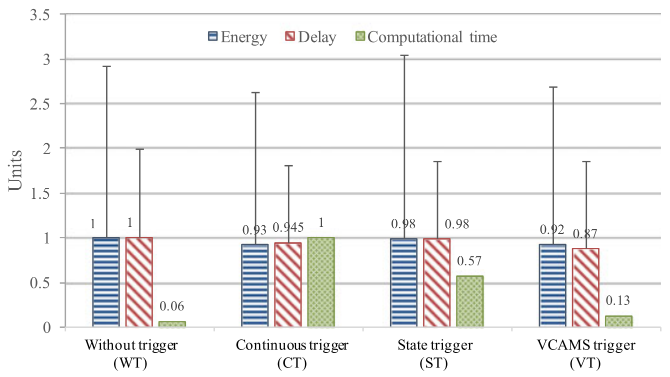


Fig. 21. VCAMS trigger strategy and other strategies.

computational time that were obtained for each of the aforementioned strategies. Our goal is to minimize energy and delay while avoiding excessive computational complexity. When comparing VT with other adaptive strategies, the results show that VT gives the best performance in terms of sensing energy consumed and delay in detecting a state change. The figure shows that VT causes less energy and delay than CT because CT adapts the sensing schedule continuously each time a state is encountered. For example, while CT consumes 0.93 units of energy, VT consumes 0.92 which is slightly less. On the other hand, VT causes 0.87 units of delay while CT causes 0.945; therefore, CT updates the sensing schedule continuously causing sometimes higher time limit $\hat{T}_j$ for state s_j; thus, causing more delay in the upcoming encounters of this state. Regarding the computational time, VT requires slightly more time than the WT strategy and much less time than ST and CT strategies. VT saves 87 percent from the computational time when compared to CT by avoiding unnecessary computations.

7 Conclusion

In this paper, we have presented VCAMS: a Viterbi-based Context Aware Mobile Sensing algorithm to trade-off energy consumption and delay when detecting any contextual state change. The method includes a system model for user context and phone sensor parameters. The algorithm chooses an optimized sensor scheduling which maximizes a set of rewards according to Viterbi algorithm. The system has two modes: learning mode and execution mode. In the learning mode, the sensing schedules are initialized offline using a Vietrbi algorithm. The learning mode is also triggered when VCAMS has to update the sensing schedules when significant behavioral changes are observed. In the execution mode, the already learned sensing schedules are used to decide on sensing triggers. We experimentally validated the proposed system models, and showed that our proposed strategy outperforms all state-of-the-art methods when considering the combined trade-off of energy and delay. Results showed that VCAMS saves up to 78 percent energy when compared to continuous sensing. Future work includes evaluation of the recommended method for health related applications.

References

[1] Ciscoz, "Cisco visual networking index: Global mobile data traffic forecast update," *White Paper*, Feb. 2015.
[2] R. Jain and L. Jalali, "Objective self," *IEEE MultiMedia*, vol. 21, no. 4, pp. 100–110, Oct.–Dec. 2014.
[3] Google play Wiki page. [Online]. Available: http://en.wikipedia.org/wiki/GooglePlay/
[4] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dartmouth College, Hanover, NH, USA, Tech. Rep. TR2000–381, 2000.
[5] J.-Y. Hong, E.-H. Suh, and S.-J. Kim, "Context-aware systems," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 8509–8522, May 2009.
[6] J. K. Lee, S. N. Robinovitch, and E. J. Park, "Inertial sensing-based pre-impact detection of falls involving near-fall scenarios," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 23, no. 2, pp. 258–266, Mar. 2015.
[7] X. Zhang and Y. Lian, "A 300-mV 220-nW event-driven ADC with real-time QRS detection for wearable ecg sensors," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 6, pp. 834–843, Dec. 2014.
[8] Y. Wang, et al., "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proc. 7th Int. Conf. Mobile Syst., Appl. Services*, Jun. 2009, pp. 179–192.
[9] S. Kang, J. Lee, H. Jang, Y. Lee, S. Park, and J. Song, "A scalable and energy-efficient context monitoring framework for mobile personal sensor networks," *IEEE Trans. Mobile Comput.*, vol. 9, no. 5, pp. 686–702, May 2010.
[10] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The Jigsaw continuous sensing engine for mobile phone applications," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, 2010, pp. 71–84.
[11] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram, "Markov-optimal sensing policy for user state estimation in mobile devices," in *Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2010, pp. 268–278.
[12] O. Yurur, C. Liu, Z. Sheng, V. Leung, W. Moreno, and K. Leung, "Context-awareness for mobile sensing: A survey and future directions," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 68–93, Jan.–Mar. 2016.
[13] R. Prez-Torres, C. Torres-Huitzil, and H. Galeana-Zapin, "Power management techniques in smartphone-based mobility sensing systems: A survey," *Pervasive Mobile Comput.*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119216000225
[14] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *IEEE Commun. Mag.*, vol. 48, no. 9, pp. 140–150, Sep. 2010.
[15] S. Taleb, N. Abbas, H. Hajj, and Z. Dawy, "On sensor selection in mobile devices based on energy, application accuracy, and context metrics," in *Proc. 3rd Int. Conf. Commun. Inf. Technol.*, Jun. 2013, pp. 12–16.
[16] N. Wang, G. Merrett, R. Maunder, and A. Rogers, "Energy and accuracy trade-offs in accelerometry-based activity recognition," in *Proc. 22nd Int. Conf. Comput. Commun. Netw.*, Jul. 2013, pp. 1–6.
[17] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Services*, 2010, pp. 315–330.
[18] P. Zappi, et al., "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," in *Proc. 5th Eur. Conf. Wireless Sensor Netw.*, 2008, pp. 17–33.
[19] Y. Chon, E. Talipov, H. Shin, and H. Cha, "SmartDC: Mobility prediction-based adaptive duty cycling for everyday location monitoring," *IEEE Trans. Mobile Comput.*, vol. 13, no. 3, pp. 512–525, Mar. 2014.
[20] A. Thiagarajan, et al., "VTrack: Accurate, energy-aware road traffic delay estimation using mobile phones," in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2009, pp. 85–98.
[21] O. Yurur, M. Labrador, and W. Moreno, "Adaptive and energy efficient context representation framework in mobile sensing," *IEEE Trans. Mobile Comput.*, vol. 13, no. 8, pp. 1681–1693, Aug. 2014.
[22] O. Yurur, C. Liu, X. Liu, and W. Moreno, "Adaptive sampling and duty cycling for smartphone accelerometer," in *Proc. IEEE 10th Int. Conf. Mobile Ad-Hoc Sensor Syst.*, Oct. 2013, pp. 511–518.
[23] S. Taleb, H. Hajj, and Z. Dawy, "Entropy-based optimization to trade-off energy and accuracy for activity mobile sensing," in *Proc. 4th Annu. Int. Conf. Energy Aware Comput. Syst. Appl.*, Dec. 2013, pp. 6–11.
[24] D. Chu, et al., "Balancing energy, latency and accuracy for mobile sensor data classification," in *Proc. 9th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2011, pp. 54–67.
[25] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *Proc. 16th Annu. Int. Symp. Wearable Comput.*, Jun. 2012, pp. 17–24.

[26] X. Qi, M. Keally, G. Zhou, Y. Li, and Z. Ren, "AdaSense: Adapting sampling rates for activity recognition in body sensor networks," in *Proc. 19th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2013, pp. 163–172.

[27] K. K. Rachuri, C. Mascolo, and M. Musolesi, *Energy-Accuracy Trade-Offs of Sensor Sampling in Smart Phone Based Sensing Systems*. London, U.K.: Springer, 2012, pp. 65–76.

[28] G. Forney Jr, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[29] N. Viol, J. Link, H. Wirtz, D. Rothe, and K. Wehrle, "Hidden Markov model-based 3d path-matching using raytracing-generated Wi-Fi models," in *Proc. Int. Conf. Indoor Positioning Indoor Navigat.*, Nov. 2012, pp. 1–10.

[30] Z. Ma and A. Krings, "Survival analysis approach to reliability, survivability and prognostics and health management (PHM)," in *Proc. IEEE Aerosp. Conf.*, Mar. 2008, pp. 1–20.

[31] T. Aven and U. Jensen, *Stochastic Models in Reliability*. Berlin, Germany: Springer, 1999.

[32] L. White and H. Vu, "Maximum likelihood sequence estimation for hidden reciprocal processes," *IEEE Trans. Autom. Control*, vol. 58, no. 10, pp. 2670–2674, Oct. 2013.

[33] E. Ramasso and T. Denoeux, "Making use of partial knowledge about hidden states in HMMs: An approach based on belief functions," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 2, pp. 395–405, Apr. 2014.

[34] Q. You, Y. Li, Z. Chen, and M. S. Rahman, "A simple near-optimal path selection scheme for multi-hop wireless relay networks based on Viterbi algorithm," *Trans. Emerging Telecommun. Technol.*, vol. 27, pp. 1294–1307, 2016.

[35] M. El Jourmi, H. El Ghazi, A. Bennis, and H. Ouahmane, "Performance analysis of channel coding in satellite communication based on VSAT network and MC-CDMA scheme," *WSEAS Trans. Commun.*, vol. 12, no. 5, May 2013, Art. no. 219.

[36] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.

[37] A. Desiraju, M. Torlak, and M. Saquib, "Multiple-attempt decoding of convolutional codes over rayleigh channels," *IEEE Trans. Veh. Technol.*, vol. 64, no. 8, pp. 3426–3439, Aug. 2015.

[38] Y. Chen, V. P. Jilkov, and X. R. Li, "Multilane-road target tracking using radar and image sensors," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 51, no. 1, pp. 65–80, Jan. 2015.

[39] A. Domingues, T. Paiva, and J. M. Sanches, "Hypnogram and sleep parameter computation from activity and cardiovascular data," *IEEE Trans. Biomed. Eng.*, vol. 61, no. 6, pp. 1711–1719, Jun. 2014.

[40] Z. Wang, M. Guo, and C. Zhao, "Badminton stroke recognition based on body sensor networks," *IEEE Trans. Human-Mach. Syst.*, vol. 46, no. 5, pp. 769–775, Oct. 2016.

[41] S. Ruzika and M. Wiecek, "Approximation methods in multiobjective programming," *J. Optimization Theory Appl.*, vol. 126, no. 3, pp. 473–501, 2005.

[42] M. Ehrgott, "A discussion of scalarization techniques for multiple objective integer programming," *Ann. Operations Res.*, vol. 147, no. 1, pp. 343–360, 2006.

[43] S. Gass and T. Saaty, "The computational algorithm for the parametric objective function," *Naval Res. Logistics Quart.*, vol. 2, pp. 39–45, 1955.

[44] A. Vaisman and E. Zimányi, "Data warehouse concepts," in *Data Warehouse Systems: Design and Implementation*. Berlin, Germany: Springer, 2014, pp. 53–87.

[45] F. Akdag and C. Eick, "An optimized interestingness hotspot discovery framework for large gridded spatio-temporal datasets," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2015, pp. 2010–2019.

[46] S. Chatterjee and S. Russell, "A temporally abstracted Viterbi algorithm," in *Proc. 27th Conf. Uncertainty Artif. Intell.*, 2011, pp. 96–104.

[47] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 1, pp. 76–80, Jan. 1978.

[48] M. Martin and P. Nurmi, "A generic large scale simulator for ubiquitous computing," in *Proc. 3rd Annu. Int. Conf. Mobile Ubiquitous Syst.*, Jul. 2006, pp. 1–3.

[49] A. Eyal, et al., "Survival analysis of automobile components using mutually exclusive forests," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 44, no. 2, pp. 246–253, Feb. 2014.

[50] PowerTutor: A power monitor for android-based mobile platforms. [Online]. Available: http://ziyang.eecs.umich.edu/projects/powertutor/

[51] F. Ordez, P. de Toledo, and A. Sanchis, "Activity recognition using hybrid generative/discriminative models on home environments using binary sensors," *Sensors*, vol. 13, pp. 5460–5477, 2013.

Sirine Taleb received the BE (with distinction) degree in computer and communications engineering from the American University of Beirut (AUB), Beirut, Lebanon, in 2012. She is currently working toward the PhD degree in electrical and computer engineering at AUB. Her research interests include smartphone sensing, energy efficiency, adaptive and optimal sensing, mobile computing, and context awareness. She received the CNRS-L/AUB doctoral scholarship award in 2014 and Merits Baccalaureate Full Scholarship in 2009. She is a student member of the IEEE.

Hazem Hajj received the bachelor's (with distinction) degree in electrical engineering from AUB, in 1987 and the PhD degree from the University of Wisconsin-Madison, in 1996. He is an associate professor with the American University of Beirut (AUB). For the academic year 2015-2016, he is a visiting associate professor with the University of Texas-Austin. Over the years, he has established a strong mix of both industry and academics backgrounds. He has more than 20 years of experience as a technical leader and a manager. He joined the American University of Beirut (AUB), 2008, and got promoted to associate professor in 2013. Before joining AUB, he was a principal engineer at Intel Corporation. He also received several teaching awards, including the University Teaching Excellence Award. His research interests include data mining, energy-aware computing, with special interests in deep learning, opinion mining, mobile sensing, and emotion recognition. He is a senior member of the IEEE.

Zaher Dawy received the BE degree in computer and communications engineering from the American University of Beirut (AUB), Beirut, Lebanon, in 1998, and the ME and Dr-Ing degrees in communications engineering from Munich University of Technology (TUM), Munich, Germany, in 2000 and 2004, respectively. Since 2004, he has been in the Department of Electrical and Computer Engineering, AUB, where he is currently a professor. His research and teaching interests include wireless communications, cellular technologies, context-aware mobile computing, mobile solutions for smart cities, computational biology, and biomedical engineering. He serves as editor of the *IEEE Communications Surveys and Tutorials*, the *IEEE Transactions on Communications*, the *IEEE Transactions on Wireless Communications*, and Elsevier the *Physical Communications*. He has also served as executive editor of the *Wiley Transactions on Emerging Telecommunications Technologies* from 2011 to 2014. He received Abdul Hameed Shoman Award for Young Arab Researchers in 2012, IEEE Communications Society 2011 Outstanding Young Researcher Award in Europe, Middle East, and Africa Region, AUB Teaching Excellence Award in 2008, Best Graduate Award from TUM in 2000, Youth and Knowledge Siemens Scholarship for Distinguished Students in 1999, and Distinguished Graduate Medal of Excellence from Hariri Foundation in 1998. He is a senior member of the IEEE.