TOWARDS ENERGY EFFICIENT MOBILE SENSING

by

Yi Wang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

May 2011

Copyright 2011 Yi Wang
Dedication

To my beloved parents, Wang Yongning and Cai Qingyun.
Acknowledgements

This thesis is certainly not a singular effort. It would not have been completed without the building blocks from those who guided, and collaborated with me in achieving this milestone.

Foremost, I would like to express my sincere appreciation to my advisers, Prof. Bhaskar Krishnamachari and Prof. Murali Annavaram. Bhaskar has been providing continuous motivation ever since the first day I was a Ph.D. student. Under Bhaskar’s guidance, I not only developed a comprehensive set of advanced research skills, but also learned what a Ph.D. truly means. I want to thank Bhaskar, for realizing my potentials, for training me in every necessary aspect to help me grow up to be an outstanding researcher, and for sharing his experiences that are outside the campus life. Along with Bhaskar, I also owe my deepest gratitude to my co-adviser, Prof. Murali Annavaram. Murali’s support and encouragement have been accompanying me throughout the last three years of my Ph.D. program. Murali helped me develop rigorous research habits that are extremely important for an outstanding researcher. Besides, Murali provided me a unique opportunity to practice in an industrial research lab, which significantly broadened my views and made me involved in a different research community for the first time. It is my great honor to be a student of Bhaskar and Murali. Their immense knowledge, hard working and enthusiasm in research have been, and will always be inspiring me throughout my lifetime.
I would like to give my special thanks to my collaborator, Jialiu Lin, a Ph.D. student from Carnegie Mellon University. Without her contribution, the majority of Chapter 3 of this thesis would not have been accomplished. In addition, I’m really grateful to Prof. Qing Zhao for her insightful comments and valuable contributions to my research subjects. Without her guidance, the majority of Chapter 4 of this thesis could not have been accomplished.

My sincere thanks also goes to Dr. Quinn Jacobson and Dr. Fan Bai, who were my intern managers at Nokia Research Center and General Motor R&D labs respectively. They not only broadened my perspective on how research is conducted with practical aspects in the industry, but also fought with me together on writing two extremely challenging research papers.

I have to thank all my dissertation committee members, Prof. Bhaskar Krishnamachari, Prof. Murali Annavaram, Prof. Urbashi Mitra, Prof. Rahul Jain, and Prof. Gaurav Sukhatme, for their intriguing questions and valuable feedback in order for me to complete this thesis.

Besides the individuals mentioned above, I would like to thank all my collaborators, including Akram Ahmed, Dr. Joon Ahn, Dr. Amitabha Ghosh, Prof. Jason Hong, Dr. Shyam Kapadia, Prof. Konstantinos Psounis, Prof. Norman Sadeh, and Prof. Thomas Valente. I could never have completed my Ph.D. unless working together with such outstanding researchers.

The members from Autonomous Networks Research Group deserve my heartily thanks as well. It was this group of amazing people that mentored me, accompanied me, and fought with me together day and night. I will never forget the time we spent together.

Lastly, I want to thank my beloved fiancee, Rui, who always gives me support unconditionally.
# Table of Contents

Dedication .......................................................... ii  
Acknowledgements ................................................... iii  
List of Tables ........................................................... viii  
List of Figures ........................................................... ix  
Abstract ................................................................. xii  

Chapter 1: Introduction .............................................. 1  

Chapter 2: Background .............................................. 6  
2.1 User context sensing and applications in mobile devices .......... 6  
2.2 Energy efficiency research in mobile sensing ....................... 9  
2.3 Information processing in detection oriented systems ............ 12  
2.3.1 The Markovian case ........................................ 12  
2.3.1.1 Hidden Markov models .................................. 12  
2.3.1.2 The forward-backward algorithm ..................... 14  
2.3.1.3 The Viterbi algorithm .................................. 16  
2.3.2 The semi-Markovian case .................................... 18  
2.3.2.1 Semi-Markov processes ................................. 18  
2.3.2.2 State estimation in hidden semi-Markov models ....... 19  
2.3.3 Differences from traditional models .......................... 21  
2.4 Constrained stochastic optimization .............................. 22  
2.4.1 CMDP definition ........................................... 23  
2.4.2 Classes of policies .......................................... 24  
2.4.3 Cost criteria ................................................ 25  
2.4.4 Solving the CMDP ........................................... 26  

Chapter 3: The framework design for an Energy Efficient Mobile Sensing System .............................................. 28  
3.1 Overview ......................................................... 28  
3.2 Design methodology ............................................ 29  
3.2.1 Sensor management module .................................. 30  
3.2.1.1 State and sensor relationship: the XML-format user state descriptor .............................................. 31
3.2.1.2 Sensor duty cycles ............................................. 32
3.2.1.3 Generalization of the EEMSS framework ..................... 34
3.2.2 Classification module ........................................... 35
   3.2.2.1 GPS sensing and mode of travel classification ............. 36
   3.2.2.2 WiFi scanning and usage ................................... 36
   3.2.2.3 Real-time motion classification based on accelerometer sensing 37
   3.2.2.4 Real-time background sound recognition ..................... 39
3.3 Performance evaluation ........................................... 41
   3.3.1 Verifying EEMSS capabilities ................................. 42
   3.3.2 State recognition accuracy ................................... 43
   3.3.3 Device lifetime .................................................. 43
3.4 Summary ............................................................ 46

Chapter 4: User state estimation and sensing policy optimization: the Markovian case 52
  4.1 Overview ............................................................ 52
  4.2 Preliminaries ....................................................... 53
     4.2.1 Model and assumptions ....................................... 53
     4.2.2 User state estimation with missing observations .......... 54
     4.2.3 Expected energy consumption and expected user state estimation error 55
     4.2.4 The optimization problem ................................... 56
  4.3 The Markov-optimal Policy ....................................... 57
     4.3.1 The CMDP formulation ....................................... 57
     4.3.2 Finding the Markov-optimal policy $u^*_m$ .................... 59
     4.3.3 A case study: obtaining $u^*_m$ for sample Markov chains .. 63
  4.4 Performance comparison on Markov models: Markov-optimal $u^*_m$ vs. uniform sampling $\overline{u}$ ..................................................... 68
     4.4.1 Deriving the expected estimation error for uniform sampling . 68
     4.4.2 Comparing $u^*_m$ with $\overline{u}$ ................................ 70
  4.5 Policy comparison on real user state traces ....................... 72
  4.6 Summary ............................................................ 79

Chapter 5: User state estimation and sensing policy optimization: the semi-Markovian case 81
  5.1 Overview ............................................................ 81
  5.2 Semi-Markovian state estimation for missing observations with sojourn time truncation ................................................................. 82
     5.2.1 Preliminaries .................................................... 82
     5.2.2 The forward-backward variables: definition and initialization ... 83
     5.2.3 State estimation ............................................... 87
     5.2.4 The forward-backward variables: an alternative ............... 90
     5.2.5 State estimation performance on simulated two-state transition processes 92
     5.2.6 State estimation performance on real traces ................... 98
        5.2.6.1 Heavy-tailed state durations: device network connectivity .. 98
        5.2.6.2 Arbitrary state durations: temperature .................. 101
  5.3 Optimal sensor scheduling with energy constraint ................. 102
5.3.1 Obtaining the semi-Markov optimal policy $u_s^*$ .......................... 103
5.3.2 Performance evaluation of $u_s^*$ on simulated processes ............... 105
5.3.3 Performance of $u_s^*$ on heavy-tailed real user state traces .......... 106
5.4 A novel client-server based system framework for energy efficient mobile sensing 109
5.4.1 System design and implementation .......................................... 109
5.4.2 System performance evaluation ................................................. 112
5.5 Summary .................................................................................. 114

Chapter 6: Conclusions and future work ........................................ 115

References ................................................................................. 118
List of Tables

3.1 Power and current consumption summary for different sensors on Nokia N95.  33
3.2 Sensor duty cycles used in current EEMSS implementation  33
3.3 The computation time and sensor energy cost per sample corresponding to table 3.2.  34
3.4 Standard deviation range of accelerometer magnitude readings for different user activities  38
3.5 Classification results based on standard deviation of accelerometer magnitude values. The first column represents the ground truth while the first row represents the classification results based on accelerometer readings.  39
3.6 Percentage of sound clips classified as “speech” for different $SR_{thres}$ values.  40
3.7 EEMSS confusion matrix of recognizing “Walking”, “Vehicle” and “At some place”. The first column represents the ground truth while the first row represents the recognition results.  44
4.1 For each data trace, the theoretical gain (obtained based on the Markovian assumption) and the experimental gain (obtained by policy application to real traces) are displayed.  78
5.1 Four different distribution types and their specific parameters utilized in the simulation.  94
5.2 Average and maximum percentage gain of semi-Markovian estimation with M1, M2, and M3 over Markovian estimation under different estimation window sizes ranging from 1 to 500.  98
5.3 Smartphone battery lifetime and state estimation error comparison among all experiments.  113
List of Figures

2.1 The system states \((N = 3)\), outputs \((M = 4)\), transition probabilities, and emission probabilities. ......................................................... 14

2.2 The process is sampled at discontiguous time slots, leading to multiple estimation intervals. ................................................................. 22

3.1 System architecture of EEMSS implementation on Nokia N95. ................. 31

3.2 The format of XML based state descriptor. ........................................... 47

3.3 The state transition implication of figure 3.2. ........................................ 47

3.4 Decision tree based background sound classification algorithm. .................... 48

3.5 Shows the frequency domain features after applying FFT to different types of audio signals. ................................................................. 48

3.6 Shows the frequency histogram plots after applying SSCH to different types of audio signals shown in figure 3.5. ......................................... 49

3.7 Figure shows the time, location as well as background sound condition detected by EEMSS, for two sample users. (Top: a CMU user. Bottom: a USC user.) .... 50

3.8 Average N95 lifetime comparison of running EEMSS, CenceMe and the case where all sensors are powered. .............................................. 50

3.9 Power usage at a glance. ................................................................. 51

4.1 Illustration of the Markovian user state sequence, sampled at certain time slots. 54

4.2 An illustration of the average idle intervals in \(u_n^*\) for different energy budgets. Sub-graphs are positioned according to the six transition matrices given before. \(I_1, I_2,\) and \(I_3\) are the average optimal idle interval lengths when state 1, 2, and 3 is observed, respectively. ......................................................... 64
4.3 Policy comparison in terms of $E[R]$ for different energy budgets.

4.4 Given different state transition probabilities, this figure shows the average improvement over different energy budgets, by applying optimal policy as compared to periodic sampling.

4.5 The gain of the optimal policy over uniform sampling (averaged over different energy budgets) vs. the standard deviation of state entropies, in case of $\{2, 3, 4, 5\}$-state Markov chains.

4.6 Markov chains showing two-state transition probabilities for: (1) Stable vs. Moving (2) Not in contact with other users vs. In contact with other users (3) Not in cellular coverage vs. In cellular coverage (4) Not in WiFi coverage vs. In WiFi coverage and (5) No Bluetooth connectivity vs. Bluetooth connectivity.

4.7 Are the traces strictly Markovian? Figure compares the cdf of state duration in real data (solid curve) to the corresponding geometric distribution (dashed curve) generated with success probability indicated in figure 4.6.

4.8 Top graph shows the user motion trace (a solid vertical line means “Moving” and a blank space means “Stable”) and the sampling points of both policies. Bottom graph shows the aggregated state estimation error at each sampling point for both policies.

5.1 The semi-Markov process is sampled at discontiguous time slots, leading to multiple estimation intervals.

5.2 Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state distributions are deterministic (left) and geometric (right).

5.3 Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state distributions are binomial. Right figure shows convergence at very large window sizes.

5.4 Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state sojourn durations follow Zipf’s Law distributions, with different tail sizes according to the definitions in table 5.1.

5.5 State duration pmfs for Bluetooth connectivity state trace and the comparison of both estimation algorithms.

5.6 State duration pmfs for WiFi state trace and the comparison of both estimation algorithms.
5.7 State duration pmfs for temperature connectivity state trace and the comparison of both estimation algorithms. ................................................................. 102

5.8 Performance of $u_m^*$ vs. $u_s^*$, on binomially and Zipf’s Law distributed state processes. Figure shows the estimation error w.r.t. energy budget (in log scale). . . . 105

5.9 Performance of $u_m^*$ vs. $u_s^*$, and the genie-aided estimation error on CoSphere network connectivity traces. Figure shows the estimation error w.r.t. energy budget (in log scale). ......................... 107

5.10 System modules and operation flow. .................................................. 110
Abstract

Mobile device based urban sensing, participatory sensing, and user activity recognition can provide rich contextual information for mobile applications such as social networking and location-based services. However, as the sensors on mobile devices consume significant amount of energy, the major bottleneck that restricts the continuous functioning of these mobile applications is the limited battery capacity on mobile devices.

In this thesis, we first present a novel design framework for an Energy Efficient Mobile Sensing System (EEMSS). EEMSS uses hierarchical sensor management strategy to recognize user states as well as to detect state transitions. By powering only a minimum set of sensors and managing sensors hierarchically, EEMSS significantly improves device battery life. We present the design, implementation, and evaluation of EEMSS that automatically recognizes a set of users’ daily activities in real time using sensors on an off-the-shelf high-end smart phone. Evaluation of EEMSS with 10 users over one week shows that our approach increases the device battery life by more than 75% while maintaining both high accuracy and low latency in identifying transitions between end-user activities.

We then propose a computationally efficient algorithm to obtain the optimal sensor sampling policy under the assumption that the user state transition is Markovian. This Markov-optimal policy minimizes user state estimation error while satisfying a given energy consumption budget.
The Markov-optimal policy is compared to uniform periodic sensing and performance improvement is obtained on both simulated and real user state traces, with approximately 20% average gain on empirically collected user data that pertains to user motion, inter-user contact status, and network connectivity.

Finally, we formulate the user state sensing problem as the intermittent sampling of a semi-Markov process, a model that provides more general and flexible capturing of realistic data. We propose (a) a semi-Markovian state estimation mechanism that selects the most likely user state while observations are missing, and (b) a semi-Markov optimal sensing policy $u^*_s$ which minimizes the expected state estimation error while maintaining a given energy budget. Their performance is shown to significantly outperform Markovian algorithms on simulated two-state processes and real user state traces. In addition, we propose a novel client-server based energy efficient mobile sensing system architecture that automatically learns user dynamics and computes user-specific optimal sensing policy for mobile devices. A system prototype aiming to recognize basic human activity is implemented on Nokia N95 smartphones and desktop computers. We demonstrate the performance benchmark of $u^*_s$ through a set of experiments, and show that it improves the estimation accuracy by 27.8% and 48.6% over Markov-optimal policy and uniform sampling, respectively.
Chapter 1

Introduction

As the number of transistors in unit area doubles every 18 months following Moore’s law, mobile devices such as smartphones are continually integrating more sophisticated functionalities. For example, current generation mobile devices integrate a wide range of sensing and networking features such as GPS, microphone, camera, light sensor, orientation sensor, accelerometer, WiFi, Bluetooth, 3G chips, and so on. These sensors and transceivers collectively enable us to obtain and share user’s ambient context, both unobtrusively and in real time.

At the same time, mobile devices have become more powerful and user-friendly clients that support mobile applications and web browsing, through which users are able to conveniently receive desired information and services. In addition, user contextual information can be transferred via the Internet to back-end servers with stronger computation and storage capabilities that are able to deliver desired personalized services back to user, such as location based advertising [7], traffic information reporting [40], social networks [27], and health monitoring [5].

Throughout this thesis, we use the term user state as an important way to represent user context that evolves over time. User state may contain one or a combination of user features including activity, location, and proximity information such as background sound and number
of peer devices. As driven by the improving demand of user satisfaction, a growing number of mobile applications and services rely on accurate and automatic detection of user state. For example, it would be much more convenient if our phones can automatically and correctly adjust the ring tone profile to appropriate volume and mode according to the surroundings and the events in which the users are participating.

While incorporating user’s contextual information brings mobile application personalization to new levels of sophistication, a major impediment to collecting and determining user context is the limited battery capacity on mobile phones. Since the integrated sensors can consume significant amount of energy, continuously collecting context information through sensor activation will make even the most compelling mobile application less than desirable to users. For instance, our experiments show that a fully charged battery on Nokia N95 device would be completely drained within six hours if its GPS is operated continuously [90]. As excessive energy consumption may become a major obstacle to the broader acceptance of context-aware mobile applications or services, energy efficient mobile sensing is critical in order to maintain the lifetime of mobile device battery and to continue to take advantage of mobile applications and services.

The first major problem associated with traditional mobile sensing systems is that all the sensors on mobile devices are activated continuously to collect user data regardless of user status. For example, the GPS would be turned on continuously no matter where the user is located, yet the ideal case is to shut it down automatically when the user is indoor since GPS cannot communicate with satellites at indoor positions in general. In order to address this problem and provide a more efficient system architecture, we first present in this thesis the design, implementation, and evaluation of EEMSS [90], an Energy Efficient Mobile Sensing System that incorporates a hierarchical sensor management scheme to recognize user states pertaining to a comprehensive set of daily
activities. EEMSS significantly extends the mobile device battery lifetime by powering a minimum set of sensors at any given time and triggering sensors hierarchically. The core component of EEMSS is a sensor management scheme which defines user states and state transition rules by an XML styled state descriptor, which is taken as an input to the system functional block to turn sensors on and off based on the user’s current condition. In addition, user motion and background sound classification algorithms that guarantee real-time system performance have been designed and implemented on EEMSS. A user study has been conducted on two university campuses and the results show that EEMSS is able to detect user states with 92.56% accuracy on average and improves the mobile device battery lifetime by over 75%, compared to a start-of-the-art system (CenceMe [60, 61]).

The second major problem in traditional mobile sensing systems (including the EEMSS design) is that sensors adopt fixed and uniform duty cycles whenever activated, which is not adaptive to user behaviors. In general, some user states once entered may stay unchanged for a long while, and if the sensor is sampled too frequently within that state duration, an excessive amount of energy will be consumed. On the other hand, if the user is frequently transiting among different states, it is preferred to sample the sensor more frequently to capture his context, as otherwise it becomes more difficult to estimate the information between two sensor observations.

In order to obtain an intelligent sensing policy which is able to assign different frequencies at different user states, two fundamental problems need to be investigated: first, state estimation algorithms needs to be designed to provide the “best guess” of user state when the sensor stays idle; second, it is necessary to obtain the optimal sensor sampling policy that leads to the minimum state estimation error while satisfying the energy budget constraint.
We first investigate these two problems under the assumption that user state transitions follow a Markov process. We propose state estimation algorithm for missing observations, and formulate the optimal sensor duty cycle selection problem as an infinite-horizon Constrained Markov Decision Process (CMDP). The expected average cost of the CMDP is considered as the cost criteria to be minimized, with the sensor energy consumption serving as the constraint. Solving such CMDP yields a globally optimal sensor sampling policy \( u^*_m \) which guarantees the minimum expected user state estimation error while satisfying the given energy budget for Markovian user state transitions (we thus name this policy “Markov-optimal”). The Markov-optimal policy is stationary and randomized (shown in [3, 29, 30]) and can be obtained by solving the corresponding Linear Programming (LP) of the CMDP formulation, which requires only polynomial run-time for any number of user state inputs. We show that the Markov-optimal policy always leads to a non-negative gain over uniform sampling and the gain is positively correlated to the standard deviation of state entropies. We then apply both the Markov-optimal policy and uniform sampling to real user context data traces and show that the \( u^*_m \) leads to approximately 20% improvement over uniform periodic sampling.

To generalize the analytical framework, we relax the assumption such that instead of assuming geometrically distributed state durations, we consider discrete time semi-Markov processes which allow arbitrary state sojourn time distributions and provide more general and realistic models than Markov processes\(^1\). We propose a forward-backward state estimation mechanism for semi-Markov processes, and provide a linear programming based approximate sensing policy optimization framework, which is able to efficiently schedule sensor samplings in order to minimize state estimation error given an energy constraint. The semi-Markov estimation mechanism

\(^1\) Note that a discrete time Markov process is a special case of discrete time semi-Markov process where the sojourn times are geometrically distributed.
is evaluated on both simulated state sequences and real user state traces and we demonstrate that it outperforms Markov estimation if the past sojourn time is selected using the correct method. Both the semi-Markov estimation mechanism and the optimization framework can be applied to data traces with any types of state sojourn time distribution, whose probability mass function (pmf) parameters are not necessarily known \textit{a priori} and can be obtained through the learning of historic data. The approximate semi-Markov optimal policy $u^*_s$ is examined on simulated as well as real state traces and we show that it obtains performance improvement over the Markov-optimal policy $u^*_m$. It is also implemented in accelerometer based activity monitoring system and we demonstrate that the mobile device battery lifetime can be extended to one week from two days by implementing $u^*_s$, while the user state estimation error is maintained below 3%.

The organization of this thesis is as follows: In Chapter 2, we present relevant works in mobile sensing and the corresponding energy efficiency research, and introduce the background of Hidden Markov Models (HMM), Hidden Semi-Markov Models (HSMM), and Constrained Markov Decision Process (CMDP), which are fundamental tools utilized in this thesis. In Chapter 3, we present the design, implementation, and user study result of EEMSS. In Chapter 4 and 5, we investigate state estimation and sensing policy optimization in case of Markovian and semi-Markovian user state processes respectively, and show their evaluation results on simulated and real user state traces, as well as from real system implementations. Finally, in Chapter 6, we conclude the contributions of this thesis, and present several open problems, which indicate future research directions.
Chapter 2

Background

2.1 User context sensing and applications in mobile devices

As current generation commodity mobile devices contain a wide range of sensors (such as accelerometer, GPS, WiFi, Bluetooth, camera, etc.), and provide significantly better computation and communication capabilities than traditional wireless sensor devices, a new wave of research has been focusing on the development of people-centric mobile sensing and computing applications.

In the context of using mobile device to sense and recognize user state, it is worth noting that Schmidt et al. [74] first propose to incorporate low level sensors to mobile PDAs to demonstrate situational awareness, and Gellersen et al. [37] pointed out the idea that combining a diverse set of sensors that individually captures just a small aspect of an environment may result in a total picture that better characterizes a situation than location or vision based context.

A great number of works thereafter utilize commodity cell phones as sensing, computing or application platforms such as [1, 18, 36, 49, 60, 61, 65, 72, 78, 91]. For examples, the authors of [1] and [49] discussed the vision of the sensor network of mobile phones and its potential
usages and design challenges. The authors of [65] designed a mobile device based health monitoring system that aggregates on-body health related data aiming to provide health alert and emergency response. In [72], a mobile system has been proposed that automatically estimates the demographics of a region based on mobile device population sensing and relevant user inputs. “CenceMe” [61] is originated from “MetroSense” [26], and uses the integrated as well as external sensors to capture the user’s status such as activity, disposition, and surroundings and enables members of social networks to share their sensing presence. A CenceMe prototype has been made available on Facebook, and the implementation and evaluation of the CenceMe application has also been discussed [60]. Similarly, “Sensay” [78] is a context-aware mobile phone that uses data from a number of sources to dynamically change cell phone ring tone, alert type, as well as determine users’ “un-interruptible” states. Another example is “MicroBlog” [36], which is a participatory sensing application that allows users with smart phones to generate geo-tagged multimedia based on user location information detected by GPS and WiFi.

Motion sensors have been widely used in monitoring and recognizing human activities to provide guidance to specific tasks [4, 56, 82]. For example, in car manufacturing, a context-aware wearable computing system designed by Stiefmeier et al. [82] could support a production or maintenance worker by recognizing the worker’s actions and delivering just-in-time information about activities to be performed. A common low cost sensor used for detecting motion is the accelerometer. In [33], a software architecture “Satire” was proposed that aims monitor human activity by smart clothes, with the assistance from accelerometer-equipped sensor motes. The authors have shown that it is possible to distinguish human actions based on their unique accelerometer signatures.
Like the work in [33], with accelerometer as the main sensing source, activity recognition is usually formulated as a classification problem where the training data is collected with experimenters wearing one or more accelerometer sensors in a certain period. Different kinds of classifiers can be trained and compared in terms of the accuracy of classification [8, 22, 52, 97]. For example, more than 20 human activities including walking, watching TV, running, stretching, etc. can be recognized with fairly high accuracy [8].

Besides motion sensors, researchers from different fields have studied and used a large number of sensors including GPS, Bluetooth, WiFi detector, blood oxygen saturation sensor, accelerometer, electrocardiograph sensor, temperature sensor, light sensor, microphone, camera, etc. in projects such as urban/participatory sensing [18, 54, 63, 92], activity recognition [5, 17, 55], and health monitoring [35, 43, 94]. For example, Whitesell et al. [92] have designed and implemented a system that analyzes images from air sensors captured from mobile phones and indoor air pollution information has been extracted by comparing the data to a calibrated chart. Targeting the obesity problem in the health monitoring domain, Annavaram et al. [5] showed that by using data from multiple sensors and applying multi-modal signal processing, seemingly similar states such as sitting and lying down can be accurately discriminated, while using only a single accelerometer sensor these states can not be easily detected. Wu et al. [94] have designed “SmartCane” system which provides remote monitoring, local processing, and real-time feedback to elder patients in order to assist proper usage of canes to reduce injury and death risks.
2.2 Energy efficiency research in mobile sensing

Earlier works that explore energy management on mobile devices include [69, 76, 80, 84]. For example, Viredaz et al. [84] surveyed many fundamental but effective methods for saving energy on handheld devices in terms of improving the design and cooperation of system hardware, software as well as multiple sensing sources. Event driven power-saving method is investigated by Shih et al. to reduce system energy consumptions [76]. In their work, the authors focused on reducing the idle power, the power a device consumes in a “standby” mode, such that a device turns off the wireless network adaptor to avoid energy waste while not actively used. The device will be powered on only when there is an incoming or outgoing call or when the user needs to use the PDA for other purposes. To further explore the concept of event-driven energy management, a hierarchical power management method was used in [80]. In their demo system “Turdecken”, a mote is used to wake up the PDA, which in turn wakes up the computer by sending a request message. Since the power required by the mote is enough for holding the whole system standby, the power consumption can be saved during system idle time.

More recently, the study of energy efficiency in mobile sensing has been conducted in works such as [51], [86], [48], and [2]. Krause et al. [51] investigate the topic of trading off prediction accuracy and power consumption in mobile computing, and the authors showed that even very low sampling rate of the accelerometer can lead to competitive classification results while device energy consumption can be significantly reduced. Wang et al. characterized the trade-off between energy usage and contact missing probability in delay tolerant networks and proposed an optimization framework to compute the optimal contact probing interval as a function of the arrival rate. The hierarchical sensor management concept is explored by the “SeeMon” system [48],
which achieves energy efficiency by only performing context recognition when changes occur during the context monitoring. Energy management in personal health care system using mobile devices is studied by Aghera et al. [2], where the authors aim to address the tradeoff between energy efficiency and data uploading latency by implementing a task assignment strategy capable of selecting different levels of operation. The authors of [60] are well aware of the battery life constraint of mobile devices and apply duty cycling mechanisms to different physical sensors in order to achieve energy efficiency. However, the lack of intelligent sensor management method still reduces the device lifetime by a significant amount.

Energy efficient localization [23, 46, 50, 58, 67, 83, 98] has been extensively studied as well recently, as location is one of the most important types of user context information. For example, Constandache et al. [23] show that human can be profiled based on their mobility patterns and thus location can be predicted. Their “EnLoc” system achieves good localization accuracy with a realistic energy budget. The energy consumption for different localization techniques including GPS, WiFi, and GSM are measured on Nokia N95 device and an energy efficient localization framework named “EnLoc” has been proposed. The authors show that “EnLoc” system achieves good localization accuracy with a realistic energy budget. Park et al. [67] investigate how GPS on mobile device can be activated and duty cycled smartly using the assistance of accelerometer, cell tower, and bluetooth, which conduct user movement and velocity estimation, location blacklisting, and location synchronization respectively. Lin et al. [58] studied sensor selection in localization, by comparing the tradeoff between energy consumption and location detection accuracy among different sensors such as GPS, WiFi, and so on. The authors provide an energy optimal sensor selection at any time instance given user’s preference in location detection
accuracy. Thiagarajan et al. [83] investigate location estimation for map matching using hidden Markov models, thus providing travel time estimation in intelligent transportation systems.

Besides in mobile sensing, energy efficient monitoring and event detection has been widely studied in a much broader context such as communication, data collection, and so on. For example, Stabellini and Zander [81] propose a new energy efficient algorithm for wireless sensor networks to detect communication interference. Shin et al. [77] propose DEAMON, an energy efficient distributed sensor monitoring algorithm where sensors are controlled using a Boolean expression and energy is conserved by selectively turning on/off the nodes and limiting communication operations. Similarly, Silberstein et al. [79] investigate suppression technique (reducing the cost for reporting changes) and take advantage of temporal and spatial data correlation to reduce data reporting therefore enlarge the lifetime of sensors.

The EEMSS framework design presented in Chapter 3 is the first work that systematically investigates energy efficiency issues in mobile sensing systems, which achieves energy efficient sensor management and user state recognition automatically. In particular, we propose a hierarchical approach for managing sensors, and do so in such a way that still maintains accuracy in sensing the user’s state. Specifically, power hungry sensors are only activated whenever triggered by power efficient ones. By only duty cycling the minimum set of sensors to detect state transition and activating more expensive ones on demand to recognize new state, the device energy consumption can be significantly reduced.

However, in EEMSS design, sensors still adopt pre-determined, fixed duty cycles whenever activated, which is not adaptive to different user behaviors. This is also a universal problem in most of the existing sensing systems. We address this problem in Chapter 4 and 5, where we model the user state as a discrete time Markov process and discrete time semi-Markov process.
respectively, and investigate how sensor duty cycles can be optimized in order to minimize the expected user state estimation error, while maintaining an energy consumption budget.

2.3 Information processing in detection oriented systems

Due to the fact that Chapter 4 and 5 of this thesis investigate user state estimation in case of partially observed stochastic process, it is worth introducing the concept of Hidden Markov (semi-Markov) Models and the traditional techniques used for state estimations, such as the forward-backward algorithm [70] and the Viterbi algorithm [85]. We divide the underlying user state process model into two categories, namely, the Markov process model and the semi-Markov process model, and introduce their background respectively in the following.

2.3.1 The Markovian case

2.3.1.1 Hidden Markov models

The basic theory of hidden Markov models (HMM) was published by Baum et al. in a series of papers [10–14] in the late 1960s and early 1970s, and its implementation in speech processing has been studied by Baker [6] and Jelinek [44, 45] in the 1970s. More recently, Rabiner [70] provided an overview of the basic theory of HMMs and its applications in speech recognition.

An HMM is a discrete Markov process in which the system states cannot be directly observed. Instead, the HMM is observed through a set of outputs, where each output is generated depending on the underlying system state. An HMM is characterized by five elements:

- The set of system states is denoted as \( X = \{x_1, x_2, \ldots, x_N\} \). The state at time \( t \) is denoted as \( s_t \).
• The set of distinct observation symbols, i.e., the outputs, are denoted as \( Y = \{ y_1, y_2, \ldots, y_M \} \).

The output at time \( t \) is denoted as \( o_t \).

• The state transition probability from system state \( x_i \) to \( x_j \) is denoted as \( a_{ij} \), where

\[
a_{ij} = \text{Prob}[s_{t+1} = x_j | s_t = x_i], \quad i, j \in \{1, 2, \ldots, N\}. \tag{2.1}
\]

• The emission probability \( b_{ik} \) is the probability of detecting output \( y_k \) while the system state is \( x_i \), i.e.,

\[
b_{ik} = \text{Prob}[o_t = y_k | s_t = x_i], \quad i \in \{1, 2, \ldots, N\}, \quad k \in \{1, 2, \ldots, M\}. \tag{2.2}
\]

• The initial distribution \( \pi = \{\pi_i\} \) where

\[
\pi_i = \text{Prob}[s_1 = x_i], \quad i \in \{1, 2, \ldots, N\}. \tag{2.3}
\]

An illustration of the probability parameters of an HMM is given in Figure 2.1.

One typical problem in HMMs is to estimate the most likely underlying system state sequence given the observation sequence. In particular, given the observation sequence \( O = o_1 o_2 \cdots o_T \) and all model parameters, what is the most likely corresponding system state sequence \( S = s_1 s_2 \cdots s_T \) that generates this output sequence and is “optimal” in some sense?

Note that since there exist many different optimality criteria, the difficulty usually lies in how to define an “optimal sequence”. In this thesis, we introduce two well-known mechanisms,

\(^1\)Source: http://en.wikipedia.org/wiki/Hidden_Markov_model
Figure 2.1: The system states \( (N = 3) \), outputs \( (M = 4) \), transition probabilities, and emission probabilities.

namely, the forward-backward algorithm and the Viterbi algorithm, both attempting to uncover the hidden system state sequence yet aiming to achieve different optimization objectives.

2.3.1.2 The forward-backward algorithm

The forward-backward algorithm aims to select an estimated state sequence where the states \( s_t \) are \textit{individually most likely}, i.e., it picks the most likely state for each individual time slot.

Let \( \lambda \) denote the set of HMM parameters, we define forward variable \( \alpha_t(i) \) as:

\[
\alpha_t(i) = \text{Prob}[o_1 o_2 \cdots o_t, s_t = x_i | \lambda], \tag{2.4}
\]

which is the joint probability of the output sequence till time \( t \) and the state \( s_t \) at time \( t \), given the model parameters.

The value of \( \alpha_t(i) \) can be solved recursively:
- Initialization:

\[ \alpha_1(i) = \pi_i b_{i0}, i \in \{1, \ldots, N\} \]  
\hspace{2cm} (2.5)

- Induction:

\[ \alpha_{t+1}(i) = \left[ \sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_{jo_{t+1}}, 1 \leq t \leq T - 1, 1 \leq j \leq N. \]  
\hspace{2cm} (2.6)

- Termination:

\[ \text{Prob}[O|\lambda] = \sum_{i=1}^{N} \alpha_T(i). \]  
\hspace{2cm} (2.7)

Similarly, the backward variable \( \beta_t(i) \) is defined as:

\[ \beta_t(i) = \text{Prob}[o_{t+1}o_{t+2}\cdots o_T|s_t = x_i, \lambda], \]  
\hspace{2cm} (2.8)

i.e., the probability of the observation sequence from time \( t + 1 \) till the end, given state \( s_t \) at time \( t \) and the model parameters \( \lambda \).

The value of \( \beta_t(i) \) can be solved recursively as well:

- Initialization:

\[ \beta_T(i) = 1, i \in \{1, \ldots, N\} \]  
\hspace{2cm} (2.9)

- Induction:

\[ \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_{jo_{t+1}} \beta_{t+1}(j), t = T - 1, T - 2, \ldots, 1, 1 \leq i \leq N. \]  
\hspace{2cm} (2.10)
Now we define the variable

\[ \gamma_t(i) = \text{Prob}[s_t = x_i|O, \lambda] \]  \hfill (2.11)

i.e., the probability of being in state \(x_i\) at time \(t\) given the entire output sequence and model parameters. \(\gamma_t(i)\) can be further written in the following forward-backward form:

\[
\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\text{Prob}[O|\lambda]} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \hfill (2.12)
\]

Therefore, the most likely state \(s'_t\) at time \(t\) is selected according to the following:

\[
s'_t = \arg\max_{1 \leq i \leq N} [\gamma_t(i)], 1 \leq t \leq T
\]

\[
= \arg\max_{1 \leq i \leq N} [\alpha_t(i)\beta_t(i)], 1 \leq t \leq T \hfill (2.13)
\]

because the term \(\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)\) is a normalizing constant.

**2.3.1.3 The Viterbi algorithm**

One problem associated with the forward-backward algorithm is that the resulting estimated state sequence may not be a valid output, due to the fact that Equation 2.13 does not take into consideration the occurrence of sequences of states. Unlike the forward-backward algorithm, the Viterbi algorithm finds the best state sequence, i.e., the most likely state sequence within a given estimation interval.
Define the quantity

\[ \delta_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} \text{Prob}[s_1 s_2 \cdots s_t = x_i, o_1 o_2 \cdots o_t | \lambda] \]  

(2.14)

which denotes the highest probability along a single path for time \( t \) that accounts for all the previous observations and the state at time \( t \). The following relationship holds by induction:

\[ \delta_{t+1}(j) = \max_i \delta_t(i) a_{ij} \cdot b_{jo_{t+1}} \]  

(2.15)

In order to retrieve the most likely state sequence, we need to keep track of all the states that maximize the quantity given by Equation 2.15, for each \( t \) and \( i \). Therefore, an extra variable \( \tau_t(i) \) is introduced that stores such state information. The complete process for finding the mostly likely state sequence is as follows:

- **Initialization:**

\[ \delta_1(i) = \pi_i b_{i o_1}, i \in \{1, \ldots, N\} \]  

(2.16)

\[ \tau_1(i) = 0. \]  

(2.17)

- **Recursion:**

\[ \delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_{jo_t}, \quad 2 \leq t \leq T, 1 \leq j \leq N, \]  

(2.18)

and:

\[ \tau_t(j) = \arg\max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], \quad 2 \leq t \leq T, 1 \leq j \leq N. \]  

(2.19)
- Boundary condition:

\[ s'_T = \arg\max_{1 \leq i \leq N} [\delta_T(i)]. \tag{2.20} \]

- Constructing state sequence:

\[ s'_t = \tau_{t+1}(s'_{t+1}), t = T - 1, T - 2, ..., 1. \tag{2.21} \]

Note that the term \( \tau_t(j) \) stores the best previous state at time \( t - 1 \) given that \( j \) is selected at time \( t \), such that the overall path can be constructed by backtracking from the last time slot \( T \).

It is important to note that, since \( \delta_t(j) \) denotes a “path probability” and this method computes the most likely state sequence instead of estimating the most possible state for each individual time slots, the estimated sequence is considered to be incorrect even if it contains a single bit error.

### 2.3.2 The semi-Markovian case

#### 2.3.2.1 Semi-Markov processes

A semi-Markov process [41] is a generalization of Markov process in which instead of geometrically distributed state durations, the state duration can follow any arbitrary distribution type. This allows the Markovian assumption to be relaxed such that analysis can be applied to more general and realistic data.

Denote \( p_{ij} \) as the probability of state transition from state \( i \) to state \( j \). Similar to Markov models, the transition probabilities satisfy

\[ p_{ij} \geq 0, \forall i, j \in \{1, 2, ..., N\}, i \neq j, \tag{2.22} \]
and
\[ \sum_{j=1}^{N} p_{ij} = 1, \forall i \in \{1, 2, \ldots, N\}, \quad (2.23) \]

with no self-transitions, i.e.:
\[ p_{ii} = 0, \forall i \in \{1, 2, \ldots, N\}. \quad (2.24) \]

However, after the “next state” \( j \) has been selected, the process “holds” for a time interval, instead of making the state transition immediately. The corresponding probability mass function (pmf) of the sojourn time of state \( i \) is denoted as \( w_i(a) \) \((a \in \mathbb{N}^+))\), which is the probability that state \( i \) will last for \( a \) time slots until the next state transition. \( w_i(a) \) satisfies \( 1 \leq a \leq A_i \leq \infty \), and \( \sum_a w_i(a) = 1 \), where \( A_i \) is the maximum sojourn duration of state \( i \).

### 2.3.2.2 State estimation in hidden semi-Markov models

A hidden semi-Markov model is an extension of the HMM that allows general state duration distributions \([31]\). In the literature such as \([57, 62, 70, 71]\), HSMM is also named “HMM with variable duration”, “explicit duration HMM”, “generalized HMM”, and so on.

In this section, we introduce a forward-backward approach for system state estimation in HSMMs that is originated from \([31]\). For consistency purpose, we use the same variable definition as in section 2.3.1.

The following two forward variables are first defined:

\[
\alpha_t(i) = \text{Prob}[o_1 o_2 \cdots o_t, \text{state } i \text{ ends at } t] \\
= \sum_{a=1}^{A} \alpha_{t-a}^*(i) \cdot w_i(a) \prod_{k=t-a+1}^{t} b_{iok}, \quad (2.25)
\]
and

\[
\alpha_t^*(i) = \text{Prob}[o_1 o_2 \cdots o_t, \text{state } i \text{ starts at } t+1] \tag{2.26}
\]

\[
= \sum_{j=1}^{N} \alpha_t(j) \cdot a_{ji}. \tag{2.27}
\]

for \( t = 1, 2, \ldots, T \).

As \( \alpha_t(i) \) and \( \alpha_t^*(i) \) are solved recursively, the initial and boundary conditions are given by:

\[
\alpha_0^*(i) = \pi_i, i \in \{1, \ldots, N\} \tag{2.28}
\]

and

\[
\alpha_t^*(i) = 0, \forall t < 0, i \in \{1, \ldots, N\}. \tag{2.29}
\]

Similarly, the backward variables are defined as follows:

\[
\beta_t(i) = \text{Prob}[o_t o_{t+1} \cdots o_T | \text{state } i \text{ starts at } t] \tag{2.30}
\]

\[
= \sum_{a=1}^{A} \beta_{t+a}^*(i) \cdot w_i(a) \prod_{k=t}^{t+a-1} b_{io_k}, \tag{2.31}
\]

and

\[
\beta_t^*(i) = \text{Prob}[o_t o_{t+1} \cdots o_T | \text{state } i \text{ ends at } t-1] \tag{2.32}
\]

\[
= \sum_{j=1}^{N} \beta_t(j) \cdot a_{ij}. \tag{2.33}
\]

for \( t = T, T - 1, \ldots, 1 \).
The boundary conditions are specified as:

\[ \beta^*_{T+1}(i) = 1, \forall i \in \{1, \ldots, N\} \]  

(2.34)

and

\[ \beta^*_{t}(i) = 0, \forall t > T + 1, i \in \{1, \ldots, N\}. \]  

(2.35)

The rest is similar to section 2.3.1.2 and the variable

\[ \gamma_t(i) = \text{Prob}[s_t = x_i|O, \lambda] \]  

(2.36)

is solved for every time slot \( t \in \{1, \ldots, T\} \), and the system state at time \( t \) is estimated such that \( \gamma_t(i) \) is maximized:

\[ s'_t = \arg\max_{1 \leq i \leq N} [\gamma_t(i)], 1 \leq t \leq T. \]  

(2.37)

A detailed implementation of the forward-backward based semi-Markovian estimation can be found in Chapter 5 with some modifications in the problem model assumptions.

### 2.3.3 Differences from traditional models

In Chapter 4 and 5 of this thesis, we investigate state estimation in partially observed stochastic processes, i.e., user state information needs to be estimated whenever sensor stays idle. In general, our problem model is different from the traditional HMM (HSMM) model in the following ways:
• Sensor observations are assumed to be perfect. Therefore, the output symbol is the same as the underlying system state and the emission probability satisfies:

\[ b_{ik} = \begin{cases} 
1, & \text{if } i = k, \\
0, & \text{otherwise}. 
\end{cases} \]  

(2.38)

• Sampling is carried out intermittently. Therefore, state estimations are conducted for every estimation interval along the sparsely sampled process (see illustration Figure 2.2).

![Intermittently sampled stochastic process](image)

Figure 2.2: The process is sampled at discontiguous time slots, leading to multiple estimation intervals.

• In case of semi-Markov processes, the state estimation suffers the “unknown past sojourn time” problem, i.e., unlike the traditional assumption in [31], the observed states do not necessarily need to start/finish at the beginning/end of the estimation interval. In Chapter 5 we aim to solve this problem by approximating the past sojourn time in three different ways.

### 2.4 Constrained stochastic optimization

General stochastic optimization tools have been widely used to help design better protocols in mobile operations. For example, in one of the early works, Benini et al. [16] addressed power optimization on battery-operated devices with the assistance of Markov Decision Process (MDP).
Cheung et al. [21] propose a MDP based optimization framework to optimize resource usage (e.g., battery-life), such that user-specific reward can be maximized.

In most mobile sensing systems, the major design objective is to provide as accurate user context detection as possible; however, turning on the sensor frequently to obtain user status will drain the mobile device battery quickly. In this thesis, we view this as a constrained optimization problem and investigate how sensors could be scheduled optimally such that the expected user state detection error is minimized while satisfying a given energy consumption budget.

In this section we introduce a standard optimization tool that is utilized in this thesis to solve the constrained optimization problem - the Constrained Markov Decision Process (CMDP).

CMDP [3] is a variant of Markov Decision Processes [15] (MDP) which provides a framework for constructing optimal policies for a stochastic process. CMDP considers a situation where one type of cost is to be optimized while keeping others below some pre-defined thresholds. For example, Lazar [53] used CMDP to analyze the problem of maximizing throughput subject to delay constraints in telecommunication applications. In traffic applications, Nain and Ross [66] studied the problem where different traffic types compete for the same resource. Some weighted sum of average delays of certain traffic types is to be minimized while some other traffic types need to satisfy certain delay constraints. More recent studies of CMDP include [25], where Dolgov and Durfee limit the optimal policy search to stationary deterministic policies coupled with a novel problem reduction to mixed integer programming, and the method yields an algorithm that finds computationally feasible policies (i.e., not randomized).

### 2.4.1 CMDP definition

A CMDP is defined by a tuple \( \{X, B, P, c, d\} \) where
• \( X \) is the state space containing a finite number of states.

• \( B \) is the finite set of actions. Let \( \mathcal{K} = \{(x, b) : x \in X, b \in B\} \) be the set of state-action pairs.

• \( P \) are the state transition probabilities, and \( P_{ibj} \) is defined as the probability of moving from system state \( i \) to \( j \), when action \( b \) is taken.

• \( c : \mathcal{K} \to \mathbb{R} \) is the type of cost that is to be minimized.

• \( d : \mathcal{K} \to \mathbb{R}^K \) is a vector of immediate costs related to \( K \) constraints.\(^2\)

### 2.4.2 Classes of policies

The objective of solving a CMDP is to choose a policy which specifies the actions for the controller to take at different decision epochs in order to achieve cost optimization under constraints.

To define a policy, the history at time \( t \) is first defined as a sequence of previous states and actions plus the current state:

\[
h_t = (x_1, b_1, ..., x_{t-1}, b_{t-1}, x_t) \tag{2.39}
\]

A policy \( u \) is a sequence \( u = (u_1, u_2, ...) \) such that if history \( h_t \) is observed at time \( t \), the controller will choose action \( b \) with probability \( u_t(b|h_t) \). The following classes of policies are often seen in the literature:

• \( U_M \) (Markov policies): \( u \in U_M \) if \( u_t \) only depends on \( x_t \) for any \( t \).

• \( U_S \) (stationary policies): \( U_S \) is a subset of \( U_M \). \( w \) is stationary if \( w_t \) does not depend on \( t \).

\(^2\)Note that in this thesis \( K = 1 \) since the only constraint is the sensor energy consumption budget.
• **$U_D$** (stationary deterministic policies): $U_D$ is a subset of $U_S$. $g$ is stationary deterministic if the action chosen at state $x$ is identified by the map $g : X \rightarrow B$.

• **$U_R$** (stationary randomized policies): $U_R$ is a subset of $U_S$. $r$ is stationary randomized if the action at state $x$ is drawn from the set $B$ based on a discrete probability distribution.

### 2.4.3 Cost criteria

Here we introduce two most frequently used cost criteria of CMDP, namely, discounted cost and expected average cost.

First, the discounted cost in a finite-horizon CMDP and an infinite-horizon CMDP are defined as follows:

\[
C^m_\alpha(\pi, u) = (1 - \alpha) \sum_{t=1}^{\infty} \alpha^{t-1} E_{\pi} c(X_t, B_t)
\]

\[
C_\alpha(\pi, u) = (1 - \alpha) \sum_{t=1}^{\infty} \alpha^{t-1} E_{\pi} c(X_t, B_t)
\]

where $\pi$ captures the initial distribution. Note that the discounted cost emphasizes on the near future and gives less importance to the far future.

Second, the expected average cost (under finite and infinite horizons respectively) is defined as follows:

\[
C^{ea}_m(\pi, u) = \frac{\sum_{t=1}^{\infty} E_{\pi} c(X_t, B_t)}{n}
\]

\[
C_{ea}(\pi, u) = \lim_{n \rightarrow \infty} C^{ea}_m(\pi, u)
\]

Denote $C(\pi, u)$ as one general type of cost, the constrained optimization problem (COP) is given as:
**COP:** Find a policy that minimizes $C(\pi, u)$, s.t. $D(\pi, u) \leq V$.

where both $C(\pi, u)$ and $D(\pi, u)$ stand for one type of the cost defined above.

### 2.4.4 Solving the CMDP

It has been shown by Feinberg and Shwartz [29, 30] that stationary deterministic policies are not guaranteed to be optimal due to the constraints added to the classical MDP model. In order to solve for the optimal stationary randomized policy in polynomial time, Kallenberg [47], and Heyman and Sobel [39], propose the use of LP where adding constraints with the same discount factor does not increase the complexity of the problem.

In this section we introduce the use of LP to solve the constrained optimization problems defined in the previous section with infinite horizon and expected average cost. We first need to define a very important concept, namely, the occupation measure corresponding to a policy $u$, which is the total expected time spent in different state-action pairs. It therefore specifies the probability that a particular state-action pair ever exists in the decision process. For given initial distribution $\pi$ and policy $u$, the occupation measure is defined as:

$$f_{ea}(\pi, u; x, b) = \frac{1}{t} \sum_{s=1}^{t} P^u_{\pi}(X_s = x, S_s = b), b \in B(x)$$  \hspace{1cm} (2.42)

The cost can be therefore represented as:

$$C_{ea}(\pi, u) = \sum_{y \in X} \sum_{b \in B(x)} f(y, b)c(y, b)$$  \hspace{1cm} (2.43)
Let $Q_{ea}(\pi)$ be the set of occupation measures that satisfies:

$$Q_{ea}(\pi) = \left\{ \rho(y, b), y \in X, b \in B(y) : \right.$$  

$$\sum_{y \in X} \sum_{b \in B(x)} \rho(y, b)(\delta_x(y) - P_{ybx}) = 0, x \in X$$  

$$\sum_{y \in X} \sum_{b \in B(x)} \rho(y, b) = 1$$  

$$\rho(y, b) \geq 0, \forall y, b$$

(2.44)

It has been shown in [3] that the following LP is equivalent to the COP defined previously:

**LP(\pi):** Find the infimum $C^*$ of $C(\rho)$ subject to:

$$D^k(\rho) \leq V_k, k = 1, ..., K, \text{ and } \rho \in Q_{ea}(\pi)$$

(2.45)

where $D^k$ stands for constraint of type $k$. 
Chapter 3

The framework design for an Energy Efficient Mobile Sensing System

3.1 Overview

In this chapter, we present the framework design, implementation, and evaluation of an Energy Efficient Mobile Sensing System (EEMSS)\(^1\). EEMSS manages sensors hierarchically by turning on only necessary sensors at any given time and triggering power hungry sensors on demand to achieve automatic user state detection in an energy efficient manner.

The benefits of EEMSS’s sensor management methodology are threefold. First, the state description mechanism is a flexible way to add/update user states and their relationship to the sensors. For instance, to account for emerging application needs new states and sensors may be incrementally added to the state description. Second, to achieve energy efficiency, the sensor management methodology assigns the minimum set of sensors and heuristically determines sampling lengths and intervals for these set of sensors to detect user’s state as well as transitions to new states. Lastly, our sensor management methodology can be easily extended as a middleware

\(^1\)This is based on the work that originally appeared in [90]
that manages sensor operations and provides contextual information to higher layer applications with multiple types of devices and sensors involved.

EEMSS is currently implemented and evaluated on Nokia N95 devices. In our EEMSS implementation, the state description subsystem currently defines the following states: “Walking”, “Vehicle”, “Resting”, “Home_talking”, “Home_entertaining”, “Working”, “Meeting”, “Office_loud”, “Place_quiet”, “Place_speech” and “Place_loud”. All these states are specified as a combination of built-in Nokia N95 sensor readings. The sensors used to recognize these states are accelerometer, WiFi detector, GPS, and microphone. EEMSS incorporates novel and efficient classification algorithms for real-time user motion and background sound recognition, which form the foundation of detecting user states. We have also conducted a field user study with 10 users at two different university campuses to evaluate the performance of EEMSS. Our results show that EEMSS is able to detect states with 92.56% accuracy and improves the battery lifetime by over 75%, compared to existing results. Note that although in this chapter we focus only on states that can be detected by integrated sensors on mobile devices, our sensor management methodology is general enough that one can apply our infrastructure to mobile sensing systems that involves more sensors and devices.

3.2 Design methodology

The core components of EEMSS can be viewed as a layered architecture that consist of a sensor management module, a classification module, and a sensor control interface. Some other components have also been incorporated to facilitate developer debugging and evaluation, including
real-time user state updates, logging, and user interfaces. Figure 3.1 illustrates the layered design of the system architecture and the interactions among these components: (1) System reads in the XML state descriptor which contains the sensor management scheme. (2) Management module determines the sensors to be monitored based on current user state which is specified by the sensor management scheme. (3) Management module instructs the sensor control interface to turn on/off sensors. (4) Sensor control interface operates individual sensors. (5) Sensor interface reports readings to classification module. (6) Classification module determines the user state. (7) Classification module forwards the intermediate classification result to management module. (8) The user’s state is updated and recorded in real-time. (9) The relevant information is also displayed on the smart phone screen.

The sensor control interface contains APIs that provide direct access to the sensors. Through these APIs, application can obtain the sensor readings and instruct sensors to switch on/off for any given duty cycle parameters. We describe the design of the sensor management module and the classification module in more details in the following.

### 3.2.1 Sensor management module

The sensor management module is the major control unit of the system. It first parses a user state description file (in XML format) that describes the sensor management scheme, and then controls the sensors based on the sensing criteria of each user state and state transition conditions by specifying the minimum set of sensors to be monitored under different scenarios (states). The sensor management module configures the sensors in real-time according to the intermediate classification result acquired from the classification module and informs the sensor control interface what sensors to be turned on and off in the following step.
3.2.1.1 State and sensor relationship: the XML-format user state descriptor

Sensor assignment is achieved by specifying an XML-format state descriptor as system input that contains all the states to be automatically classified as well as sensor management rules for each state. The system will parse the XML file as input and automatically generate a sensor management module that serves as the core component of EEMSS and controls sensors based on real-time system feedback. In essence, the state descriptor consists of a set of state names, sensors to be monitored, and conditions for state transitions.

Figure 3.2 illustrates the general format of a user state descriptor and the corresponding state transition process. It can be seen that a user state is defined between the “<State>” and “</State>” tags. For each state, the sensor(s) to be monitored are specified by “<Sensor>” tags.
The hierarchical sensor management is achieved by assigning new sensors based on previous sensor readings in order to detect state transition. If the state transition criteria has been satisfied, the user will be considered as entering a new state (denoted by “<NextState>” in the descriptor) and the sensor management algorithm will restart from the new state. For example, based on the sample description in figure 3.2, if the user is at “State2” and “Sensor2” returns “Sensor reading 2” which is not yet sufficient for detecting state transition, “Sensor3” will be turned on immediately to further detect the user’s status in order to identify state transition (see figure 3.3).

There are three major advantages of using XML as the format of state descriptor. First, XML is a natural language to represent states in a hierarchical fashion. Second, new state descriptors can be added and existing states can be modified with relative ease even by someone with limited programming experience. Finally, XML files are easily parsed by modern programming languages such as Java and Python thereby making the process portable and easy to implement.

### 3.2.1.2 Sensor duty cycles

In EEMSS implementation, sensor duty cycles are arbitrarily chosen based on extensive trials that measure the power consumption of each sensor and computation time based on different sample lengths of data. Table 3.1 summarizes the power consumption of different sensors on Nokia N95 device. Among these sensors, accelerometer consumes the least amount of power compared to other sensors, and fortunately accelerometer is capable of detecting any body movements with a high precision. Hence accelerometer could be first indicator of state transition with high probability if it involves user body motion. In such a case, accelerometer could be sampled periodically as triggers to invoke other sensors if necessary. On the other hand, due to the large power drain
and long initialization time, GPS is used only when it is necessary to measure the speed of user’s movement so as to discriminate between modes of travel such as riding in vehicle versus walking.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Power(W)</th>
<th>Current(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>0.0599</td>
<td>0.0150</td>
</tr>
<tr>
<td>Microphone</td>
<td>0.2786</td>
<td>0.0707</td>
</tr>
<tr>
<td>GPS</td>
<td>0.3308</td>
<td>0.0862</td>
</tr>
<tr>
<td>WiFi Scan</td>
<td>1.4260</td>
<td>0.3497</td>
</tr>
<tr>
<td>Bluetooth Scan</td>
<td>0.1954</td>
<td>0.0486</td>
</tr>
</tbody>
</table>

Table 3.1: Power and current consumption summary for different sensors on Nokia N95.

Table 3.2 summarizes the duty cycle parameters for each of the four sensors used in EEMSS.

It is important to note that a particular length of sampling duration is required for a sensor to gather sufficient data. A tradeoff in the selection of sensor idle interval is that longer sleep interval may reduce energy consumption, but the state detection error will be increased since there exist more missing observations.

Even though the duty cycle parameters have been refined through extensive empirical tests, the ones finally adopted by EEMSS (as shown in table 3.2) may not be optimal in achieving the best tradeoff between energy consumption and state detection accuracy. In Chapter 4 and Chapter 5, we will systematically investigate this problem and discuss our improved design where we model the energy efficient mobile sensing problem as the question “how a (semi-)Markov

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Duty Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>6s sensing + 10s sleeping</td>
</tr>
<tr>
<td>Microphone</td>
<td>4s sensing + 180s sleeping</td>
</tr>
<tr>
<td>GPS</td>
<td>Queries every 20s, timeout in 5 min</td>
</tr>
<tr>
<td>WiFi scan</td>
<td>Event triggered (&lt; 2s to finish)</td>
</tr>
</tbody>
</table>

Table 3.2: Sensor duty cycles used in current EEMSS implementation
process can be efficiently sampled?”, and find ways to calculate the sensor duty cycle parameters to achieve the optimal tradeoff between energy consumption and user state estimation error.

The computation time (including the time for data processing and classification) and sensor energy consumed per sample based on sensor duty cycle parameters, are summarized in table 3.3.

It can be seen that except for loud audio signal processing and classification, which takes approximately 10 seconds to complete (mainly consumed at the FFT stage), all other computations finish almost instantaneously, which enables our system to conduct real-time state recognition. The energy consumption results of each sensor are also measured and illustrated in table 3.3.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Computation Time/Sample</th>
<th>Energy(J)/Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>&lt; 0.1s</td>
<td>0.359</td>
</tr>
<tr>
<td>Microphone</td>
<td>Quiet: &lt; 0.5s, Loud/Speech: ~ 10s.</td>
<td>1.114</td>
</tr>
<tr>
<td>GPS</td>
<td>&lt; 0.1s</td>
<td>6.616</td>
</tr>
<tr>
<td>WiFi scan</td>
<td>&lt; 0.1s</td>
<td>2.852</td>
</tr>
</tbody>
</table>

Table 3.3: The computation time and sensor energy cost per sample corresponding to table 3.2.

3.2.1.3 Generalization of the EEMSS framework

Various sensors make the user’s contextual information available in multiple dimensions, from which a rich set of user states can be inferred. However, in most cases different users or higher layer applications may only be interested in identifying a small subset of states and exploit the state information for application customization. For example, a ring tone adjustment application, which can automatically adjust the cell phone alarm type, may only need to know the property of background sound in order to infer the current situation. A medical application may require the system to monitor one’s surrounding temperature, oxygen level and the user’s motion such as running and walking to give advise to patient or doctors. In a personal safety application, one
factor that one may care is whether the user is riding a vehicle or walking alone such that the mobile client is able to send warning messages to the user when he or she is detected walking in an unsafe area at late night. These are all examples of mobile sensing systems with particular needs, by which our EEMSS framework design can be potentially adopted.

New user states can also be added or updated in EEMSS since the XML based state description mechanism is highly scalable. A classification algorithm has to be associated with the new state, which takes readings from one or more sensors as input in order to detect the new state. Once the classification algorithm is defined, it needs to be incorporated to the EEMSS classification module.

### 3.2.2 Classification module

The classification module is the consumer of the sensor raw data. The classification module first processes the raw sensing data into desired format. For example, the magnitude of 3-axis accelerometer sensing data is computed, and FFT is performed on sound clips to conduct frequency domain signal analysis. The classification module returns user activity and position feature such as “moving fast”, “walking”, “home wireless access point detected” and “loud environment” by running classification algorithms on processed sensing data. The resulting user activity and position information are both considered as intermediate state which will be forwarded to the sensor management module. The sensor management module then determines whether the sensing results satisfy the sensing criteria and decides sensor assignments according to the sensor management algorithm.
3.2.2.1 GPS sensing and mode of travel classification

In our study, the primary purpose of using GPS is to detect user’s location as well as mode of travel. By combining the instantaneously velocity information and the recent distance of travel measured by comparing current position with previous ones it is possible to distinguish one’s basic mode of travel such as walking or riding a vehicle. For example, if the velocity is greater than 10 mph we consider that the user is using an automotive transport. The classifier is trained by several location tracking records of user and certain threshold values are identified and implemented into the classification algorithm.

GPS can also be used to identify when a user has entered a building or other indoor environment since a location request timeout will occur since the satellite signals are are likely to be blocked in the indoor environment. It is worth mentioning that from the system implementation point of view, obtaining instant speed as well as the location request timeout functionality are both supported by J2ME API.

3.2.2.2 WiFi scanning and usage

The MAC address of visible wireless access points around the user can be inferred by performing a WiFi scan. Since MAC address of each wireless access point is unique, it is possible to tag a particular location by the set of access points in that location. Therefore the mobile device is able to automatically identify its current location by simply detecting nearby wireless access points. For example, it is easy to tell that the user is at home if the WiFi scan result matches his or her home access point set that is pre-memorized by the device. In our current EEMSS implementation, the wireless access points feature of the user’s home and office (if applicable) will be pre-recorded for recognition purpose. While in our current implementation we pre-record
the set of access points for each of user’s well defined locations, such as home and office, there are several alternative implementations such as SkyHook [93] that provide the location information by a database table lookup of a set of access points.

WiFi scan can also be used to monitor a user’s movement range since a wireless access point normally covers an area of radius 20-30m. Hence if the user moves out of his/her recent range a WiFi scan will detect that the current set of WiFi access points are replaced by a new one. In our system implementation, if the user is detected moving continuously by accelerometer, a WiFi scan will be performed to check whether the user has left his or her recent location and if so, GPS will be turned on immediately to start sampling location information and classify the user’s mode of travel.

3.2.2.3 Real-time motion classification based on accelerometer sensing

Activity classification based on accelerometer readings has been widely studied using various machine learning tools [8, 22, 52, 97]. However, in most of the previous works one or more accelerometer sensors have to be attached to specific body joints such as knees and elbows. Several data features are then extracted from readings of multiple accelerometers in order to design sophisticated classifiers to recognize user activities. Most of these classification algorithms are both data and compute intensive and hence are unsuitable for real-time classification using current mobile phone computing capabilities.

In EEMSS design, mobile phone is the only source of accelerometer readings. We only make the assumption that the mobile phone is carried by the user at all times without any placement restrictions. Hence, it becomes extremely difficult to perform detailed level motion classification using a single accelerometer. We use only the standard deviation of accelerometer magnitude as
Table 3.4: Standard deviation range of accelerometer magnitude readings for different user activities

<table>
<thead>
<tr>
<th>Mode</th>
<th>STDV Range of Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Still</td>
<td>0 - 1.0334</td>
</tr>
<tr>
<td>Walking</td>
<td>9.2616 - 21.3776</td>
</tr>
<tr>
<td>Running</td>
<td>35.3768 - 52.3076</td>
</tr>
<tr>
<td>Vehicle</td>
<td>4.0204 - 12.6627</td>
</tr>
</tbody>
</table>

one of the defining features independent of phone placement in order to conduct real-time motion classification, recognizing four human motion modes namely “Still”, “Walking”, “Running”, and “Vehicle”.

We have collected accelerometer data in 53 different experiments distributed in two weeks in order to train the classifier. The lengths of experiment vary from several minutes to hours. Within each empirical interval, the person tags the ground truth of his activity information for analysis and comparison purposes. The standard deviation for different activities within each empirical interval is computed off-line. Table 3.4 shows the range of standard deviation distribution based on different data sets collected.

It can be found out that there exist certain standard deviation threshold values that could well separate stable, walking, running, and vehicle mode with high accuracy. In order to verify this observation, we have implemented a real-time motion classification algorithm on N95 mobile phone that compares the standard deviation of accelerometer magnitude values with the thresholds in order to distinguish the user’s motion. The device is carried by the user without explicit requirement of where the phone should be placed. 26 experiments have been conducted each containing a combination of different user motions. The standard deviation of accelerometer magnitude is computed every 6 seconds, and right after which the user’s motion is being classified. Table 3.5 shows the classification results in percentage of recognition accuracy. It can be seen that the algorithm works very well for extreme conditions such as stable and running. Furthermore, even
Table 3.5: Classification results based on standard deviation of accelerometer magnitude values. The first column represents the ground truth while the first row represents the classification results based on accelerometer readings.

<table>
<thead>
<tr>
<th></th>
<th>Still</th>
<th>Vehicle</th>
<th>Walking</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>Still</td>
<td>99.44%</td>
<td>0.56%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vehicle</td>
<td>8.81%</td>
<td>73.86%</td>
<td>16.29%</td>
<td>1.04%</td>
</tr>
<tr>
<td>Walking</td>
<td>1.18%</td>
<td>10.62%</td>
<td>88.20%</td>
<td>0</td>
</tr>
<tr>
<td>Running</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
</tbody>
</table>

though the classifier tends to be confused with walking and vehicle mode due to feature overlap, the accuracy is still well maintained above 70%.

3.2.2.4 Real-time background sound recognition

This subsection describes the algorithm used for background sound classification. These algorithms were coded in Java and run on N95 to classify sound clips recorded using N95. The device records a real time audio clip using microphone sensor and the recorded sound clip will go through two classification steps (figure 3.4). First, by measuring the energy level of the audio signal, the mobile client is able to identify if the environment is silent or loud. Note that the energy $E$ of a time domain signal $x(n)$ is defined by $E = \sum_n |x(n)|^2$. Next, if the environment is considered loud, both time and frequency domains of the audio signal are further examined in order to recognize the existence of speech. Specifically, speech signals usually have higher silence ratio (SR) [59] (SR is the ratio between the amount of silent time and the total amount of the audio data) and significant amount of low frequency components. If speech is not detected, the background environment will simply be considered as “loud” or “noisy”. No further classification algorithm is conducted in EEMSS to distinguish audio signals such as music, noise and other types of sound due to their vast variety of the signal features compared to speech.
Table 3.6: Percentage of sound clips classified as “speech” for different $SR_{\text{thres}}$ values.

<table>
<thead>
<tr>
<th>$SR_{\text{thres}}$</th>
<th>Speech</th>
<th>Music</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>95.31%</td>
<td>18.00%</td>
<td>26.07%</td>
</tr>
<tr>
<td>0.7</td>
<td>91.14%</td>
<td>16.00%</td>
<td>18.17%</td>
</tr>
<tr>
<td>0.8</td>
<td>70.91%</td>
<td>10.00%</td>
<td>11.66%</td>
</tr>
<tr>
<td>0.9</td>
<td>32.64%</td>
<td>8.00%</td>
<td>8.59%</td>
</tr>
</tbody>
</table>

SR is computed by picking a suitable threshold and then measuring the total amount of time domain signal whose amplitude is below the threshold value. The Fast Fourier Transform (FFT) has been implemented such that the mobile device is also able to conduct frequency domain analysis to the sound signal in real time. Figure 3.5 shows the frequency domain features of four types of audio clips, including a male’s speech, a female’s speech, a noise clip and a music clip. It can be seen clearly that as compared to others, speech signals have significantly more weight on low frequency spectrum from 300Hz to 600Hz. In order to accomplish speech detection in real time, we have implemented the SSCH (Subband Spectral Centroid Histogram) algorithm [32] on mobile devices. Specifically, SSCH passes the power spectrum of the recorded sound clip to a set of highly overlapping bandpass filters and then computes the spectral centroid \(^2\) on each subband and finally constructs a histogram of the subband spectral centroid values. The peak of SSCH is then compared with speech peak frequency thresholds (300Hz - 600Hz) for speech detection purpose. Figure 3.6 illustrates the outputs of applying SSCH algorithm to the same sound clips used in figure 3.5. It can be found out clearly that the histogram peaks closely follow the frequency peaks in the original power spectrum.

The classification algorithm is trained and examined on the same data set including 1085 speech clips, 86 music clips and 336 noise clips, all with 4 seconds length which are recorded by Nokia N95 devices. We investigate the effect of different SR thresholds (denoted by $SR_{\text{thres}}$)

\(^2\)The spectral centroid $C$ of a signal is defined as the weighted average of the frequency components with magnitudes as the weights: $C = \frac{\sum_f f X(f)}{\sum_f X(f)}$
on classification accuracy. The results of speech detection percentage are shown in table 3.6. It can be seen that as SR threshold increases, the number of false positive results are reduced with sacrifice of speech detection accuracy. We choose $SR_{thres} = 0.7$ in EEMSS implementation which provides more than 90% of detection accuracy and less than 20% false positive results. The above classification results also show that a 4-second sample of audio clip is long enough for the classifier to identify the background sound type. It is also important to note that the complexity of the SSCH algorithm is $O(N^2)$ and as the filter overlaps are small, the running time is empirically observed to be close to linear. Empirical results show that on average the overall processing time of a 4 seconds sound clip is lower than 10 seconds on N95 devices. In future as compute capabilities of mobile phones increase we expect the latency of such complex audio processing will be reduced significantly.

### 3.3 Performance evaluation

We have conducted a user trial in November 2008 at University of Southern California and Carnegie Mellon University. The main purpose of the user trial was to test the performance of EEMSS system in a free live setting. We recruited 10 users from both universities including undergraduate and graduate students, faculties and their family. Each participant was provided with a Nokia N95 device with the EEMSS framework pre-installed. Basic operating instructions were provided to the users at the start of the experimental trials. Each user carried the device with EEMSS running for no less than two days. Each participants was requested to fully charge the mobile battery before starting the EEMSS application. EEMSS will continue to run in the background till the battery is completely drained. Participants then fully charge the phone once
again before starting the EEMSS application. The cycle of fully charging and discharging continued till the expiration of user’s participation time in the experiments. At the end of the EEMSS evaluation period, we collected more than 260 running hours of data with more than 300 user state transitions detected.

### 3.3.1 Verifying EEMSS capabilities

EEMSS is able to characterize a user’s state by time, location and activities. Besides providing real-time user state update, EEMSS keeps tracking the user’s location by recording the Geo-coordinates of the user when he or she is moving outdoor (recall that GPS is turned on continuously and the location information is retrieved every 20 seconds in this scenario). Figure 3.7 visualizes the data collected by EEMSS on 2-D maps. They show the daily traces captured by EEMSS of two different participants from CMU and USC on two campus maps respectively. Within the time frame of these two traces, the EEMSS application keeps running and the phone has not been recharged. Both users have reported that they took buses to school, and the dashed curves which indicate “Vehicle” mode are found to match the bus routes perfectly. The solid curves indicating “Walking” state match the traces that the user is walking between home and bus station, and within university campus.

Besides monitoring location change and mode of travel of the user in real-time, EEMSS also automatically detects the surrounding condition when the user is identified to be still at some places in order to infer the user’s activities. In figure 3.7, by probing background sound, the surrounding conditions of the user can be classified as quiet, loud and containing speech. Consequently, the system is able to infer that the user is working, meeting, resting, etc. by combining the detected background condition with location information obtained by WiFi scan.
3.3.2 State recognition accuracy

We first study the state recognition accuracy of EEMSS for each user. We compared the state predicted by EEMSS with the ground truth state recorded in the user’s diary for every time step. Accuracy is defined as the number of correct predictions over the total number of predictions. It has been found out that the average recognition accuracy over all users is found to be 92.56% with a standard deviation of 2.53%.

We also examine the overall state recognition accuracy in terms of the confusion matrix of identifying “Walking”, “Vehicle” and “At some place”. Table 3.7 shows the corresponding confusion matrix. The first column represents the ground truth while the first row represents the returned states by EEMSS. It can be seen that the accuracy is very high when the user is staying at some place such as home, office, etc. compared to traveling outdoors. From this table, 12.64% of walking time and 10.59% of vehicle time is categorized as “At some place”. This is because that GPS readings are unavailable due to the device limitations which causes the location timeout and hence the system considers that the user has entered some place. However, this false conclusion can be self-corrected since the accelerometer is able to monitor the motion of the user when he or she is considered still and hence GPS will be turned back on immediately as the user keeps moving. The reason that riding a vehicle is recognized as walking is that there exist extreme conditions where the vehicle is moving very slowly and is thus being recognized as the wrong state.

3.3.3 Device lifetime

The benefit of low energy consumption of EEMSS has been verified by researchers through lab studies lasting for 12 days. During each day of the study period two researchers have each carried
Table 3.7: EEMSS confusion matrix of recognizing “Walking”, “Vehicle” and “At some place”. The first column represents the ground truth while the first row represents the recognition results.

<table>
<thead>
<tr>
<th></th>
<th>At some place</th>
<th>Walking</th>
<th>Vehicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>At some place</td>
<td>99.17%</td>
<td>0.78%</td>
<td>0.05%</td>
</tr>
<tr>
<td>Walking</td>
<td>12.64%</td>
<td>84.29%</td>
<td>3.07%</td>
</tr>
<tr>
<td>Vehicle</td>
<td>10.59%</td>
<td>15.29%</td>
<td>74.12%</td>
</tr>
</tbody>
</table>

a fully charged N95 smart phone with EEMSS application running on the background and the device lifetime has been examined. The average device lifetime result with EEMSS application running on a fully charged Nokia N95 device is 11.33 hours with regular cell phone functionalities (as can be seen in figure 3.8). Note that the device lifetime may vary due to different user behaviors. For example, if the user stays at home most of the time with little activities, the device may stay active for much longer time since only accelerometer and microphone will be periodically monitored as compared to the case where one spends most of the time traveling outdoor and the device lifetime will significantly decrease due to extensive GPS usage.

We have also examined the device lifetime by turning on GPS, accelerometer and microphone on N95 device with the same sampling frequencies as used in EEMSS, and WiFi scanning is performed every 5 minutes (Note that in EEMSS the WiFi scan is only conducted when the user is leaving or arriving at some places). The device lifetime is found to be less than 5 hours regardless of user activity since no sensor management mechanism is implemented and all the sensors will be periodically sampled until running out of battery, shown as “Without Sensor Management” in figure 3.8. In [60] it has been shown that the CenceMe application can last for around 6.22 hours with no other applications running on the phone. Although the comparison may not be comprehensive it is the only known result so far that describes the device lifetime with an mobile urban sensing application running. Note that there exist some differences between CenceMe
and EEMSS implementations. For example, CenceMe adopts an audio sampling duration of 30 seconds and the implements a 60 seconds duty cycle for the microphone, whereas in EEMSS the audio sampling duration is only 4 seconds and the microphone duty cycle is 180 seconds. On the other hand, CenceMe implements an approximately 10 minutes duty cycle for GPS whereas in EEMSS GPS will be turned on continuously when the user is moving outdoors to provide location tracking. Moreover, CenceMe contains data upload cycles and Bluetooth probing that require extra power usage which are not implemented in our system. The results of battery life durations are summarized in figure 3.8, and it can be seen that EEMSS gains more than 75% running hours compared to CenceMe and even larger amount compared to the case where a urban sensing is conducted without sensor management.

To visualize the effect of sensor energy management, figure 3.9 illustrates the energy consumption model of our system at a glance in a 20 minutes interval when the user walks from his office to a library. It can be seen that at the beginning of the test when the user is sitting in the office, only accelerometer and microphone is being sampled to detect user movement and identify background sound type. When movement is detected, WiFi scanning will be performed to determine whether the user has left the recent location. Note that multiple WiFi scans may be required until the user leaves his previous position. As the user walks towards the library, GPS is turned on in order to provide positioning information and we allow 5 minutes for GPS timeout as the user enters the library where no GPS signal can be received. Finally, a WiFi scan is performed to recognize the current location and accelerometer will be turned back on for user motion detection.
3.4 Summary

We presented the design, implementation, and evaluation of EEMSS in this chapter. The core component of EEMSS is a sensor management scheme for mobile devices that operates sensors hierarchically, by selectively turning on the minimum set of sensors to monitor user state and triggers new set of sensors if necessary to achieve state transition detection. Energy consumption can be reduced by shutting down unnecessary sensors at any particular time. Our implementation of EEMSS was on Nokia N95 devices that uses our sensor management scheme to manage built-in sensors on the N95, including GPS, WiFi detector, accelerometer and microphone in order to achieve human daily activity recognition. We also proposed and implemented novel classification algorithms for accelerometer and microphone readings that work in real-time and lead to good performance. Finally, we evaluated EEMSS with 10 users from two universities and demonstrated that EEMSS is able to provide a high level of accuracy for state recognition, acceptable state transition detection latency, as well as more than 75% gain on device lifetime compared to a state-of-the-art system.
<StateDescriptor>
  <State>
    <StateName> state1 </StateName>
    <Sensor>
      <SensorName> sensor1 </SensorName>
      <Case>
        <Condition> sensor reading 1 </Condition>
        <NextState> State2 </NextState>
      </Case>
    </Sensor>
    <Sensor>
      <SensorName> sensor2 </SensorName>
      <Case>
        <Condition> sensor reading 2 </Condition>
        <Sensor>
          <SensorName> sensor3 </SensorName>
          <Case>
            <Condition> sensor reading 3 </Condition>
            <NextState> State1 </NextState>
          </Case>
          <Case>
            <Condition> sensor reading 4 </Condition>
            <NextState> State3 </NextState>
          </Case>
        </Sensor>
      </Case>
    </Sensor>
    <Sensor>
      <SensorName> sensor4 </SensorName>
      <Case>
        <Condition> sensor reading 5 </Condition>
        <NextState> State2 </NextState>
      </Case>
    </Sensor>
  </State>
  <State>
    <StateName> state2 </StateName>
    <Sensor>
      <SensorName> sensor2 </SensorName>
      <Case>
        <Condition> sensor reading 2 </Condition>
        <Sensor>
          <SensorName> sensor3 </SensorName>
          <Case>
            <Condition> sensor reading 3 </Condition>
            <NextState> State1 </NextState>
          </Case>
          <Case>
            <Condition> sensor reading 4 </Condition>
            <NextState> State3 </NextState>
          </Case>
        </Sensor>
      </Case>
    </Sensor>
  </State>
  <State>
    <StateName> state3 </StateName>
    <Sensor>
      <SensorName> sensor3 </SensorName>
      <Case>
        <Condition> sensor reading 3 </Condition>
        <NextState> State1 </NextState>
      </Case>
      <Case>
        <Condition> sensor reading 4 </Condition>
        <NextState> State2 </NextState>
      </Case>
    </Sensor>
  </State>
</StateDescriptor>

Figure 3.2: The format of XML based state descriptor.

Figure 3.3: The state transition implication of figure 3.2.
Figure 3.4: Decision tree based background sound classification algorithm.

Figure 3.5: Shows the frequency domain features after applying FFT to different types of audio signals.
Figure 3.6: Shows the frequency histogram plots after applying SSCH to different types of audio signals shown in figure 3.5.
Figure 3.7: Figure shows the time, location as well as background sound condition detected by EEMSS, for two sample users. (Top: a CMU user. Bottom: a USC user.)

Figure 3.8: Average N95 lifetime comparison of running EEMSS, CenceMe and the case where all sensors are powered.
Figure 3.9: Power usage at a glance.
Chapter 4

User state estimation and sensing policy optimization: the Markovian case

4.1 Overview

As described in Chapter 3, in EEMSS when a sensor is activated it adopts a fixed uniform sensor duty cycle parameter that is not adaptive to user state changes. A large amount of energy is thus wasted by frequently sampling a sensor while the user state stays unchanged. This is in fact a universal problem found in most sensing and signal processing based systems. In order to solve this problem and further improve energy efficiency in mobile sensing, we analytically study the sensor\(^1\) duty cycle selection problem by proposing state estimation mechanism and designing sensing policy optimization framework for Markovian user state processes in this chapter\(^2\).

We assume that time is discretized and the user state evolves as a hidden discrete time Markov chain (DTMC). At each time slot, a sensor can either be sampled to observe user state (assuming 100% detection accuracy) with a unit energy cost associated, or stay idle to save energy.

\(^1\)The term “sensor” used here can represent both physical sensors or networking transceivers that can probe the environment by sending packets. 
\(^2\)This is based on the work that originally appeared in [89]
We first propose a state estimation method originated from the forward-backward algorithm that selects the most likely state for each time slot along an estimation interval. We then consider the following constrained optimization question: *Given a certain device energy consumption budget, what is the optimal sensor sampling policy that leads to the minimum state estimation error, while satisfying the energy budget?*

This problem of selecting optimal sampling interval lengths with energy consumption constraints such that the expected state estimation error can be minimized, is formulated as a CMDP problem. Solving such CMDP yields a stationary randomized policy which has been shown to be the optimal solution among all policies in [3]. Moreover, since the optimal solution of such CMDP can be obtained by solving its corresponding Linear Programming (LP), the computational complexity stays polynomial for any given number of user states.

### 4.2 Preliminaries

#### 4.2.1 Model and assumptions

We assume that time is discretized and the user state evolves as a N-state discrete time Markov chain (DTMC) with transition probabilities \( p_{ij}(i, j \in \{1, 2, \ldots, N\}) \) from state \( i \) to \( j \). The discrete time horizon is denoted as \( T = \{t, t = 0, 1, 2, \ldots\} \). At each time slot during the process, a sensor may be sampled, in which case user state is detected with 100% accuracy and a unit of energy consumption is associated. On the other hand, the sensor can also stay idle to save energy. This process is illustrated in figure 4.1. We define \( O \) as the set of time slots when the sensor makes observation and \( O \) is thus a subset of \( T \), as shown by figure 4.1. Ideally, if the sensor can be sampled in each time slot \( (O = T) \), the user state sequence can be obtained perfectly. However,
in general, since sensor duty cycles are required to extend mobile device lifetime, there exist durations where user state needs to be estimated using only observed data and the knowledge of the user state transition dynamics.

### 4.2.2 User state estimation with missing observations

The problem of estimating missing state information between two subsequent observations is formulated as follows: given the fact that one observation detects state $i$ at some time $t_m$ and the next observation detects state $j$ at time $t_n$ ($t_m, t_n \in T$ and $t_m < t_n$), what is the most likely state sequence that connects them? In this section, we propose a mechanism that estimates the most likely user state for each time slot between two subsequent observations.
Given the sensor detects state $i$ at time $t_m$, and detects state $j$ at time $t_n$, the probability of being in state $k$ ($k \in 1, 2, ..., N$) at time $t$ ($t_m < t < t_n$), is given by:

\[
p[s_t = k | s_{t_m} = i, s_{t_n} = j] = \frac{p[s_t = k | s_{t_m} = i] \cdot p[s_{t_n} = j | s_t = k]}{p[s_{t_n} = j | s_{t_m} = i]} = \frac{P_{ik}^{(t-t_m)} \cdot P_{kj}^{(t_n-t)}}{P_{ij}^{(t_n-t_m)}}
\]

(4.1)

where $s_t$ denotes the true user state at time $t$. In order to select the “most likely” state (denoted by $s'_t$) at time $t$, the quantity given in (4.1) needs to be maximized:

\[
s'_t = \arg\max_k \left\{ p[s_t = k | s_{t_m} = i, s_{t_n} = j] \right\} = \arg\max_k \left\{ \frac{P_{ik}^{(t-t_m)} \cdot P_{kj}^{(t_n-t)}}{P_{ij}^{(t_n-t_m)}} \right\}
\]

(4.2)

The expected estimation error for time slot $t$, denoted by $e_t$, is defined as the probability of incorrect estimation, which is given by:

\[
e_t = 1 - p[s_t = s'_t | s_{t_m} = i, s_{t_n} = j] = 1 - \max_k \left\{ \frac{P_{ik}^{(t-t_m)} \cdot P_{kj}^{(t_n-t)}}{P_{ij}^{(t_n-t_m)}} \right\}
\]

(4.3)

(4.4)

### 4.2.3 Expected energy consumption and expected user state estimation error

Throughout this chapter, we address two important performance metrics: (a) the expected energy consumption and (b) the expected user state estimation error. These two intrinsically conflicting
metrics are fundamental performance measures of mobile sensing applications, as the former characterizes the device lifetime and the latter is directly related to the quality of the application in terms of providing high user state detection accuracy. The formal definitions of the two are given as follows:

**Definition 1.** The expected energy consumption $E[C]$ of the user state sampling process is defined as

$$E[C] = \frac{1}{E[I]}$$

(4.5)

where $E[I]$ is the expected sensor idle interval, i.e., the average number of time slots the sensor waits before the next sample.

**Definition 2.** The expected user state estimation error $E[R]$ is defined as the long term average of per-slot estimation error:

$$E[R] = \lim_{n \to \infty} \sum_{t=1}^{n} \frac{e_t}{n}$$

(4.6)

### 4.2.4 The optimization problem

In practical system operations, a device energy consumption budget is often specified in order to ensure a long enough device lifetime. We define $\xi$ (where $0 \leq \xi \leq 1$) as the energy budget which is the maximum expected energy consumption allowed. Let $u$ denote a sensor sampling policy, which specifies how sensor duty cycles should be assigned when different user states are detected, i.e., how long the sensor should stay idle after each observation. We investigate the following constrained optimization problem: **Considering a long user state evolving process that is Markovian, given a device energy consumption budget $\xi$ and user state transition matrix $P$,**
what is the optimal sensor sampling policy \( u^*_m \) such that the expected user state estimation error \( E[R] \) is minimized, and the expected energy consumption is maintained below the budget, i.e., \( E[C] \leq \xi \)?

4.3 The Markov-optimal Policy

We formulate the above constrained optimization problem as an infinite-horizon CMDP with expected average cost. Solving such CMDP yields a Markov-optimal policy \( u^*_m \) which is stationary and randomized \([3, 29, 30]\), i.e., \( u^*_m \) does not vary over time and decision is randomized over several actions. Meanwhile, the algorithm that finds the Markov-optimal policy \( u^*_m \) is computationally efficient, since \( u^*_m \) can be obtained by solving the corresponding LP, which is a polynomial time computation.

4.3.1 The CMDP formulation

We use an infinite-horizon CMDP to model the sensor duty cycle selection problem, whose major components are explained as follows:

- **Decision Epochs** \( O \). Decision is made immediately after each sensing observation. The set of decision epochs is defined as \( O = \{d, d = 1, 2, \ldots\} \). Note that this is the same \( O \) defined in section 4.2.1.

- **System State Space** \( X \). The system state at decision epoch \( d \) is the detected user state \( s_{d} \). The set of system states is denoted by \( X = \{1, 2, \ldots, N\} \).

- **Action Space** \( B \). The set of actions is denoted by \( B \). An action \( b \in B \) is the number of time slots the sensor waits until making the next observation. Note that the potential
number of actions could be infinity, i.e., \( B = \mathbb{N}^* \), the set of positive integers. However, in order to facilitate the study, we limit the set of actions to positive integers under some threshold \( b_m \). As \( \xi \) decreases, \( b_m \) needs to be increased in order to meet the energy budget constraint.

- **System Transition Probabilities** \( \mathcal{P} \). Define \( \mathcal{P}_{ibj} \) as the probability of moving from system state \( i \) to \( j \), when action \( b \) is taken. Given the user state transition probability matrix \( P \), it is easy to conclude that \( \mathcal{P}_{ibj} = P_{ij}^{(b)} \).

- **Intermediate Cost** \( c(y, b) \). The intermediate cost \( c(y, b) \) is defined as the expected aggregate state estimation error when action \( b \) is taken on state \( y \), where \( y \in X \) and \( b \in B \). \( c(y, b) \) can be calculated as:

  \[
  c(y, b) = \sum_j P_{yj}^{(b)} \cdot e_{yj}^{(b)}
  = \sum_j P_{yj}^{(b)} \cdot \sum_{t=1}^{b-1} \left[ 1 - \max_k \left( \frac{P_{yk}^{(t)} \cdot P_{kj}^{(b-t)}}{P_{yj}^{(b)}} \right) \right]
  \tag{4.7}
  \]

  where \( e_{yj}^{(b)} \) is the aggregate state estimation error for a length-\( b \) observation interval starting at state \( y \) and ending at state \( j \).

- **Expected Average Cost** \( E_u'[R] \). Under policy \( u \), the expected average cost of the infinite-horizon CMDP is defined as

  \[
  E_u'[R] = \lim_{n \to \infty} \frac{\sum_{d=1}^{n} E_u^c(X_d, B_d)}{n}
  \tag{4.8}
  \]

58
where \( n \) is the total number of decision epochs, and \( E^u \) is the corresponding expectation operator of the intermediate cost, under the policy \( u \).

- **Intermediate Sampling Interval** \( d(y, b) \). The intermediate sampling interval when action \( b \) is taken on state \( y \), satisfies \( d(y, b) = b \).

- **Expected Sampling Interval** \( E_u[I] \). Similar to \( E_u[R] \), the expected sampling interval \( E_u[I] \) is defined as:

\[
E_u[I] = \lim_{n \to \infty} \frac{\sum_{d=1}^{n} E^u d(X_d, B_d)}{n} \tag{4.9}
\]

- **Constraint** \( \xi \). The maximum allowed expected energy cost. Therefore \( E[C] \leq \xi \) or \( E[I] \geq 1/\xi \).

**4.3.2 Finding the Markov-optimal policy** \( u_m^* \)

Let \( \rho(y, b) \) denote the “occupation measure” of state \( y \) and action \( b \), i.e., the probability that such state-action pair ever exists in the decision process.
It is shown in [3, 39, 47] that the optimal policy of the CMDP with expected average cost criteria can be obtained by solving the corresponding LP. Inspired by these works, we rewrite the sensor duty cycle optimization problem defined above as the following LP (denoted as LP1):

**LP1: Find** \( \min \{ \sum_{y \in X} \sum_{b \in B} \rho(y, b)c(y, b) \} \) \hspace{1cm} (4.10)

**subject to:**

\[
\sum_{y \in X} \sum_{b \in B} \rho(y, b)(\delta_x(y) - P_{ybx}) = 0, \forall x \in X, \quad (4.11)
\]

\[
\sum_{y \in X} \sum_{b \in B} \rho(y, b) = 1, \quad (4.12)
\]

\[
\rho(y, b) \geq 0, \forall y, b, \text{ and} \quad (4.13)
\]

\[
\sum_{y \in X} \sum_{b \in B} \rho(y, b)(1 - b \cdot \xi) \leq 0, \quad (4.14)
\]

where \( \delta_x(y) \) is the delta function of \( y \) concentrated on \( x \). The constraint given in (4.11) emphasizes the property for ergodic processes and describes that the outgoing and incoming rate for a state need to be the same. The constraints (4.12) and (4.13) define \( \rho(y, b) \) as a probability measure. The inequality constraint given in (4.14) guarantees that the expected energy usage is less than the energy constraint value \( \xi \), i.e.:

\[
\frac{\mathbb{E}[\text{Overall Energy Consumed}]}{\mathbb{E}[\text{Average Sensing Interval Length}]} \leq \xi. \quad (4.15)
\]

Since the per slot energy consumption is 1 unit, the inequality (4.15) can be further written as:

\[
\frac{\sum_y \sum_b \rho(y, b)}{\sum_y \sum_b \rho(y, b) \cdot b} \leq \xi, \quad (4.16)
\]
which leads to the energy constraint in (4.14).

Solving $\text{LP1}$ yields the ideal occupation measure of each state/action pair that minimizes the quantity given in (4.10). The algorithm for selecting the Markov-optimal policy $u^*_m$ is given as follows:

**Algorithm 1** Pseudo code for selecting the Markov-optimal sampling policy $u^*_m$ and calculating the corresponding expected user state estimation error $E_{u^*_m}[R]$

1: Input: $\xi, P$
2: Output: $u^*(\xi, P), E_{u^*_m}[R]$
3: Initialize: Select a large enough $b^m \in \mathbb{N}^*$ such that the energy constraint can be sufficiently satisfied. Let $B = \{1, 2, \ldots, b^m\}, X = \{1, 2, \ldots, N\}$
4: Solve $\text{LP1}$ for the optimal occupation measure $\rho^*(y, b), \forall y \in X, \forall b \in B$
5: Calculate $\rho_y^* = \sum_{b \in B} \rho^*(y, b), \forall y \in X$
6: Constructing $u^*_m$: Assign $\frac{\rho^*(y, b)}{\rho_y^*}$ as the probability of taking action $b$ at state $y, \forall y \in X, b \in B$

7: Calculate the expected user state estimation error:

$$E_{u^*_m}[R] = \frac{E'_{u^*_m}[R]}{1/\xi} = \sum_{y \in X} \sum_{b \in B} \frac{\rho^*(y, b)c(y, b)}{1/\xi}$$  \hspace{1cm} (4.17)

In Algorithm 1, the optimal occupation measure of each state-action pair can be obtained by solving $\text{LP1}$ in step 4. Step 5 of Algorithm 1 calculates the overall probability of seeing a particular state $y$ in the decision process under the optimal policy, and this quantity $\rho_y^*$ is used later in step 6 as the normalization to the optimal occupation measure $\rho^*(y, a)$ in order to compute.
the probability of taking action $b$ whenever state $y$ is detected, in the optimal sampling policy $u^*_{m}$.

This normalization step is needed to guarantee that

$$
\sum_b \frac{\rho^*(y, b)}{\rho^*_y} = 1, \forall y \in X
$$

In equation (4.17), the optimal expected user state estimation error $E_{u^*_{m}}[R]$ can be calculated once the optimal expected average cost $E'_{u^*_{m}}[R]$ of the CMDP is obtained. Here we illustrate the relationship between $E[R]$ and $E'[R]$. Recall that $E[R]$ is defined as the long-term, average per-slot error, i.e., the ratio of incorrect estimations over total number of time slots; therefore, under policy $u$, it can be written as:

$$
E_{u}[R] = \lim_{n \to \infty} \frac{\sum_{d=1}^{n} E^{u}_c(X_d, B_d)}{\sum_{d=1}^{n} E^{u}_d(X_d, B_d)}
$$

$$
= \lim_{n \to \infty} \frac{1}{n} \cdot \frac{\sum_{d=1}^{n} E^{u}_c(X_d, B_d)}{\sum_{d=1}^{n} E^{u}_d(X_d, B_d)}
$$

where $n$ denotes the total number of decision epochs, whereas $E^{u}_c(X_d, B_d)$ and $E^{u}_d(X_d, B_d)$ denote the aggregated state estimation error and the sampling interval (action) size for the $d$th decision epoch under policy $u$, respectively. Considering the Law of Large Numbers, equation (4.20) can be further written as

$$
E_{u}[R] = \frac{E'_{u}[R]}{E_{u}[I]}
$$
where $E_u[I]$ is the expected sampling interval (action) size under policy $u$. Since the optimal policy $u^*_m$ needs to be fully utilizing the energy budget, its expected sampling interval size satisfies:

$$E_{u^*_m}[I] = 1/\xi.$$  \hfill (4.22)

It can be seen that solving LP1 to obtain the optimal occupation measures in step 4 is the most time-consuming procedure, as a result, Algorithm 1 is able to return the Markov-optimal policy $u^*_m$ in polynomial time for any number of state inputs.

### 4.3.3 A case study: obtaining $u^*_m$ for sample Markov chains

In this section, as a case study, we apply Algorithm 1 to some example Markov chains with 2 and 3 states, and find the optimal policy $u^*_m$ by solving the corresponding LP in MATLAB. We aim to illustrate the outcome of Algorithm 1 by showing the obtained sensing interval at each state when different energy budgets are specified. The six different user state transition matrices tested are:

\[
\begin{pmatrix}
0.9 & 0.1 \\
0.1 & 0.9 \\
0.1 & 0.7 & 0.2 \\
0.1 & 0.7 & 0.2 \\
0.1 & 0.7 & 0.2 \\
\end{pmatrix}
\begin{pmatrix}
0.9 & 0.1 \\
0.5 & 0.5 \\
0.4 & 0.1 & 0.5 \\
0.3 & 0.6 & 0.1 \\
0.2 & 0.2 & 0.6 \\
\end{pmatrix}
\begin{pmatrix}
0.9 & 0.1 \\
0.9 & 0.1 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{pmatrix}
\]

Different energy budget constraint values ranging from 0 to 1 are specified as different inputs to Algorithm 1.
The solution of the LP returns a randomization over all possible state-action pairs, which essentially constructs the stationary randomized optimal policy $u^*_m$. Since it is difficult to visualize the randomization of multiple actions, we compute the mean value of actions taken at each decision epoch for different user states, i.e., the expected average idle interval at each state according to the optimal policy $u^*_m$, and show the result of the average actions in figure 4.2.

![Figure 4.2: An illustration of the average idle intervals in $u^*_m$ for different energy budgets. Subgraphs are positioned according to the six transition matrices given before. $I_1$, $I_2$, and $I_3$ are the average optimal idle interval lengths when state 1, 2, and 3 is observed, respectively.](image)

When the row elements of the Markov transition matrix are identical, the Markov chain degenerates to an independently and identically distributed (i.i.d.) user state sequence, where the user state transition does not depend on any current or previous state information. In this case, the expected state estimation error $E_{u^*_m}[R]$ under the optimal policy $u^*_m$ can be quantified in terms of user state transition probability matrix $P$ and the energy budget $\xi$. The following theorem holds.
Theorem 1. For Markov chain that degenerates to an i.i.d. user state sequence, given its transition probabilities \( p_{ij} (i, j \in \{1, 2, ..., N\}) \) and the energy budget \( \xi \), the following holds for optimal sampling policy \( u_m^* \):

\[
E_{u_m^*}[R] = (1 - \xi) \cdot (1 - \max_{ij} \{p_{ij}\})
\]

i.e., \( E_{u_m^*}[R] \) is linear in \( \xi \).

Proof. For i.i.d. transition probability matrix the following holds due to identical row elements in the transition matrix:

\[
P = P^{(1)} = P^{(2)} = ... = P^{(I)}, \forall I \in \mathbb{N}^*
\]

that is, the state transition probability matrix in \( I \) steps, denoted by \( P^{(I)} \), is equal to the original state transition probability matrix \( P \), for any positive integer \( I \).

Therefore, the term \( e_t \) in equation (4.4) can be further written as:

\[
e_t = 1 - \max_k \left\{ \frac{P_{ik} \cdot P_{kj}}{P_{ij}} \right\}
\]

i.e., \( e_t \) stays constant no matter what sample interval is used. This illustrates the fact that the estimation error for each time slot with missing observation is constant. Recall that \( E[R] \) is defined as the expected per-slot estimation error, and that the optimal policy utilizes the full
energy budget with expected sensor sampling interval $1/\xi$, therefore, $E_{u_m^*}[R]$ can be expressed as:

$$E_{u_m^*}[R] = \frac{1/\xi - 1}{1/\xi} \cdot e_t = (1 - \xi) \cdot (1 - \max_{ij}\{p_{ij}\})$$

(4.27)

for optimal policy $u_m^*$. 

It can be concluded from Theorem 1 that, for Markov chains which degenerates to an i.i.d. user state sequence, there exist more than one optimal policy because any randomization of actions that leads to the optimal sampling interval results in the same expected state estimation error.

Another observation from figure 4.2 is that, whenever the two-state Markov chain is symmetric, i.e., $p_{12} = p_{21}$, the average sampling intervals for both state 1 and 2 are equal. In fact, the following theorem describing the property of optimal policy holds for all symmetric two-state Markov chain:

**Theorem 2.** For two-state symmetric Markov chain where $p_{12} = p_{21}$, there exists an optimal sampling policy that makes the same decision, i.e., selects the same randomization of actions at each decision epoch.

**Proof.** Let $S = \{s_i, i = 1, 2, 3, \ldots\}$ be the user state sequence in the original decision process, where $s_i \in \{1, 2\}, \forall i$. Define $T^*$ as the set of time slots where sensing is performed under the optimal policy $u_m^*$. We label the original process as $S^* = \{s_i^*, i = 1, 2, 3, \ldots\}$, in which $s_i^*$ is defined as follows:

$$s_i^* = \begin{cases} 
  s_i, & \text{if } i \in T^* \\
  0, & \text{if } i \notin T^*
\end{cases}$$

66
$S^*$ is a process that characterizes the sampled process based on the policy $u_m^*$, by retaining the original state sequence when observations are made, and labeling “0” for the rest time slots. Note that the numbers of consecutive zeros in all estimation intervals need not necessarily be the same, due to the fact that $u_m^*$ adopts randomized decision at each decision epoch $u_m^*$.

We then define the process $S^{**} = \{s_i^{**}, i = 1, 2, 3, \ldots\}$ where

$$s_i^{**} = \begin{cases} 
s_i^{**} = 2, & \text{if } s_i^* = 1 \\
s_i^{**} = 1, & \text{if } s_i^* = 2 \\
s_i^{**} = 0, & \text{if } s_i^* = 0
\end{cases}$$

$S^{**}$ is a state sequence containing the “flipped” state from $S^*$ for all observed time slots, and zeros otherwise.

Since $p_{12} = p_{21}$, it immediately follows that $P_{12}^{(i)} = P_{21}^{(i)} \forall i = 1, 2, 3, \ldots$. Recall equation (4.7) where the aggregated error for an observation interval with a certain length is calculated. It can be concluded that $e_{12}^{(a)} = e_{21}^{(a)}$ and $e_{22}^{(a)} = e_{11}^{(a)}$, since the following equalities hold:

$$\max_k \left\{ \frac{P_l(k^t) \cdot P_l(a-t)}{P_{12}^{(a)}} \right\} = \max_k \left\{ \frac{P_{2k}^{(t)} \cdot P_{2k}^{(a-t)}}{P_{22}^{(a)}} \right\} \quad (4.28)$$

$$\max_k \left\{ \frac{P_{lk}^{(t)} \cdot P_{lk}^{(a-t)}}{P_{12}^{(a)}} \right\} = \max_k \left\{ \frac{P_{2k}^{(t)} \cdot P_{2k}^{(a-t)}}{P_{22}^{(a)}} \right\} \quad (4.29)$$

where $k \in \{1, 2\}$. It follows that the relabeled process $S^{**}$ produces the same expected error as $S^*$, which implies that by swapping all decisions made at state 1 with all decisions made at state 2 in the original decision process, the expected state estimation error remains optimal. Therefore, the optimal policy decision at state 1 is able to be applied to state 2, and vice versa, while maintaining the optimal error. This indicates that for symmetric two-state Markov chain,
no matter what state is observed at each decision epoch, there exists an optimal policy that makes the same decision.

4.4 Performance comparison on Markov models: Markov-optimal $u^*_m$ vs. uniform sampling $\bar{u}$

Researchers have proposed many sophisticated sampling schemes in the literature such as [86] and [20], aiming at energy efficient event detection in mobile and wireless networks. However, in the context of mobile sensing for user state detection and estimation, there does not exist an established framework that aims to achieve the best tradeoff between energy consumption and state estimation error and select the best sampling intervals correspondingly. A baseline performance evaluation of the optimal policy $u^*_m$, therefore, is to compare it with a uniform policy $\bar{u}$ where the sensor simply performs periodic sampling.

4.4.1 Deriving the expected estimation error for uniform sampling

We first formally define the uniform policy $\bar{u}$ and derive its corresponding expected state estimation error $E_{\bar{u}}[R]$. Since time is discretized, a randomization of actions is required in the uniform policy such that the average idle interval length can achieve $1/\xi$, i.e., the energy budget can be fully utilized. Although the potential number of randomization choices could be infinite, we restrict the randomization to choosing from two neighboring integer values as the sampling interval size. More strictly, the uniform sampling policy $\bar{u}$ is defined as follows: after each observation,
the sensor waits for $\hat{I}$ time slots with probability $\mu$, and $\hat{I} + 1$ time slots with probability $1 - \mu$, such that the following constraint holds:

$$\mu \cdot \hat{I} + (1 - \mu) \cdot (\hat{I} + 1) = \frac{1}{\xi} \tag{4.30}$$

where $\hat{I} \in \mathbb{N}^*$ and $0 \leq \mu \leq 1$. Equation (4.30) guarantees that the energy budget $\xi$ is fully utilized by randomizing the action over two subsequent integer values. Given $\xi$, it is straightforward to obtain $\mu$ and $\hat{I}$. In particular, $\hat{I} = \lfloor \frac{1}{\xi} \rfloor$ and $\mu = \lceil \frac{1}{\xi} \rceil - \frac{1}{\xi}$ would satisfy equation (4.30).

We derive the expected state estimation error $E_u[R]$ for uniform sampling policy with energy budget $\xi$. First, let $p_i$ denote the steady state probability of detecting a particular state $i$ in any observation. Since the state transition follows the Markovian rule, the current state observation result depends on the previous observed state, and the number of time slots being idle:

$$p_i = \sum_j p_j \cdot [\mu \cdot P_{ji}(\hat{i}) + (1 - \mu) \cdot P_{ji}(\hat{i} + 1)] \tag{4.31}$$

Equation (4.31) accounts for all possible cases of last observation that may lead to state $i$ in the current observation. It is obvious that:

$$\sum_i p_i = 1 \tag{4.32}$$

Therefore, the results of $p_1, p_2, \ldots, p_N$ can be obtained by solving the $N + 1$ simultaneous equations given in (4.31) and (4.32).
The expected state estimation error for uniform sampling policy with energy budget $\xi$ can be expressed as the expected aggregate error per estimation interval divided by the average interval size:

$$E_{\bar{u}}[R] = \frac{\sum_{ij} p_i \cdot \{ \mu \cdot P_{ij}^{(\hat{t})} \cdot e_{ij}^{(\hat{t})} + (1 - \mu) \cdot P_{ij}^{(\hat{t}+1)} \cdot e_{ij}^{(\hat{t}+1)} \}}{1/\xi}$$

(4.33)

where $i, j \in \{1, 2, ..., N\}$, and $e_{ij}^{(t)}$ is defined similarly as in equation (4.7).

### 4.4.2 Comparing $u_m^*$ with $\bar{u}$

We first compare and visualize the performance difference between $u_m^*$ and $\bar{u}$ on two-state Markov chains in terms of their corresponding expected user state estimation error. Figure 4.3 shows the policy comparison result with respect to different energy budgets, for four simple but representative state transition probability matrices (they consist of symmetric, i.i.d., and general Markov chains). To further visualize the condition such that $u_m^*$ outperforms $\bar{u}$, we show the average policy gain in figure 4.4. Specifically, figure 4.4 shows the average Markov-optimal policy gain (denoted by $G_{u_m^* \setminus \bar{u}}$, which is the ratio of error reduction amount introduced by $u_m^*$ over the original error obtained by $\bar{u}$, averaged on all the energy budget inputs), on a comprehensive two-state matrix of user state transition probabilities ranging from 0.05 to 0.95, with interval 0.05.

It can be observed that $u_m^*$ does not outperform $\bar{u}$ on both symmetric and i.i.d. two-state Markov chains. This is illustrated by both the overlapping curves in figure 4.3 and the zero performance difference on the diagonals of the matrix in figure 4.4. In fact, Theorem 1 and 2 already indicated the fact that there exists an uniform policy which is optimal for symmetric and i.i.d. chains. In all other cases, i.e., in case of Markov chains that are neither symmetric nor i.i.d.,
$u^*_m$ always leads to a positive gain except when $\xi = 1$ (in which case the sensor is sampled in every time slot and no incorrect estimation will occur).

To further study the dependency of Markov-optimal policy gain on user state transition dynamics, we examine the state entropy of given Markov chains. Recall that the entropy of a state measures the uncertainty associated with that state, and for a first-order Markov chain, the entropy of state $i$ is defined as:

$$H_i = -\sum_j p_{ij} \cdot \log p_{ij} \quad (4.34)$$

We investigate $\{2, 3, 4, 5\}$-state Markov chains and in each case, we randomly generate 100 transition probability matrices on which $u^*_m$ and $\bar{u}$ are applied and both $G_{u^*_m, \bar{u}}$ and the standard deviation of state entropies are calculated. Figure 4.5 displays the result of the amount of policy gain and the corresponding standard deviation of state entropies, for $\{2, 3, 4, 5\}$-state Markov chains.
Figure 4.4: Given different state transition probabilities, this figure shows the average improvement over different energy budgets, by applying optimal policy as compared to periodic sampling.

chains, respectively. The positive correlation between policy gain and standard deviation of state entropies can be clearly identified. This is because as the deviation of state entropies increases, more frequent sampling is conducted at states with higher entropy to provide lower estimation error and at the same time, longer sampling interval can be assigned at low-entropy states where the state estimation error will not be greatly affected. Unlike the uniform policy, the Markov-optimal policy is able to implement different sampling frequencies at different states in order to achieve better performance.

4.5 Policy comparison on real user state traces

The “Markovian user state transition” assumption enables us to provide a tractable state estimation mechanism in case of missing observations, a way originated from the forward-backward
algorithm that estimates hidden state sequence in traditional Hidden Markov Model (HMM) problems. Moreover, by assuming that the state transition is Markovian, the Markov-optimal sensor sampling policy can be obtained by solving the corresponding CMDP in polynomial time.

There certainly are some kinds of user state transitions that have been empirically shown previously to be close to Markovian. A classic example is speech and silence patterns within a monologue [42]. To further consolidate our study and evaluate the performance of Markov-optimal policy on real user data, we collect and study data traces that pertain to user motion change, status of inter-user contact, and status of network connectivity on a broader time scale, and answer the following questions: (a) are these user state transitions strictly Markovian? and
Does the Markov-optimal policy perform well? The data collection experiments are explained as follows:\footnote{The user motion trace and contact status data set are available at ANRG website (http://anrg.usc.edu/www/downloads), and the CoSphere network connectivity data set is available at CRAWDAD website (http://crawdad.cs.dartmouth.edu/data.php).}

**User motion:** We conducted the user motion trace collection in fall 2009. We are particularly interested in the transition between “Stable” and “Moving”, two motion states that characterize the basic user activity. During the experiment, one participant carried a Nokia N95 device with a python program running in background to automatically detect his activity, using the motion classification technique introduced in [90]. Approximately 40 hours of running data distributed in five different weekdays were collected. The mobile device recorded user motion every 12 seconds, in which, 10 seconds were consumed by the accelerometer for data collection, and the classification algorithm ran at the end of each sample duration.

**Inter-user contact status:** In fall 2006, twenty-five Tmotes [64] with wireless communication capabilities were handed out to an undergraduate class at USC for a week [87]. Each student carried the device during the whole experiment and each mote automatically logged the information about other devices within communication range. Specifically, the time, duration and node IDs of each contact were stored in the memory for off-line processing. Each node transmitted a contact packet every 0.1 second and the nearby nodes receiving the contact packets recorded the contact information immediately. We view the user as in one of the following two states at any given time: “In contact” or “Not in contact”, since the existence of neighboring devices is of importance in mobile P2P networking.
Network connectivity: We study CoSphere traces [68], which contain network visibility (Cell, WiFi, and Bluetooth) records on mobile phones carried by 12 participants. The one-month experiment was conducted by researchers at Telematica Instituut in The Netherlands within February/March 2007 time frame. To adapt the data set to our own study, we identify the transition dynamics between “Connected” and “Not connected” in all three cases (Cell, WiFi, and Bluetooth) and evaluate the Markov-optimal sampling mechanism against uniform sampling technique.

All the data have been first processed into discrete-time format. We then apply a standard method proposed in [9] to find the probabilities of the user state transition. In particular, defining $n_{ij}$ as the total frequency of state transition from $i$ to $j$, the Markov transition probability matrix can be constructed, with

$$p_{ij} = \frac{n_{ij}}{n_j},$$

where $n_j = \sum_i n_{ij}$. We illustrate the resulting Markov chains for several sample participants in figure 4.6.

The Markov chains clearly show different user state transition dynamics. We first explore the existence of Markovian property in real data trace, by examining whether the time that the user spent in a state is geometrically distributed. The state duration samples from the trace data are compared to a randomly generated geometrically distributed set of data, with the state transition probabilities shown in figure 4.6 being the success probabilities. Figure 4.7 compares the cumulative distribution function (cdf) of state duration in real data, to the corresponding geometric distribution. It can be seen that the state duration in real case tends to show more concentration.

---

4The data from other participants and from different days show similar transition dynamics in the same set of experiments.
on both small and some extra-large values. This implies that the empirical user data may not strictly follow the Markovian property, as the duration of each state is not strictly geometrically distributed. In fact, in terms of inter-user contact, several previous studies such as [19, 86, 87] have shown that the inter-contact and contact time duration distribution may exhibit heavy-tails in certain time scale rather than memoryless decay.

Motivated by the fact that real data traces do not strictly show Markovian properties, in Chapter 5, we propose an analytical framework for both user state estimation and sensing policy optimization under a more generalized model - the semi-Markov process model, which provides more realistic modeling of realistic data by allowing arbitrary state duration distributions.

We compare the Markov-optimal policy derived from the transition probabilities shown in figure 4.6 and the uniform policy by applying them to the collected user data, and measure the
Figure 4.7: Are the traces strictly Markovian? Figure compares the cdf of state duration in real data (solid curve) to the corresponding geometric distribution (dashed curve) generated with success probability indicated in figure 4.6.

Table 4.1 shows the gain on expected state estimation error, by choosing the Markov-optimal policy over uniform sampling, through both numerical calculation and policy application to real state traces.

The results shown in table 4.1 indicate that even though the real user state transition is not strictly Markovian, the Markov-optimal sensor sampling policy is still able to provide significant
<table>
<thead>
<tr>
<th>Data</th>
<th>Numerical Gain</th>
<th>Experimental Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion</td>
<td>19.75%</td>
<td>18.00%</td>
</tr>
<tr>
<td>Contact</td>
<td>29.55%</td>
<td>27.86%</td>
</tr>
<tr>
<td>Cellular connectivity</td>
<td>35.31%</td>
<td>36.01%</td>
</tr>
<tr>
<td>WiFi connectivity</td>
<td>3.93%</td>
<td>4.73%</td>
</tr>
<tr>
<td>Bluetooth connectivity</td>
<td>11.16%</td>
<td>12.70%</td>
</tr>
</tbody>
</table>

Table 4.1: For each data trace, the theoretical gain (obtained based on the Markovian assumption) and the experimental gain (obtained by policy application to real traces) are displayed.

improvement over periodic sampling mechanism, and the results match the numerically obtained results quite closely.

Figure 4.8: Top graph shows the user motion trace (a solid vertical line means “Moving” and a blank space means “Stable”) and the sampling points of both policies. Bottom graph shows the aggregated state estimation error at each sampling point for both policies.

In order to better visualize the benefit of the Markov-optimal policy, we illustrate the sampling points of the Markov-optimal policy as well as uniform sampling on a selected subset of user data trace, and show the corresponding state estimation error for both cases in figure 4.8. In particular, figure 4.8 displays the motion trace (“Stable” vs. “Moving”) of a user within an approximately 1.7-hour interval. The sensor sampling points in case of both policies are marked in the top graph.
and the aggregated state estimation error at each sampling point is plotted in the bottom graph. The energy budget $\xi$ is set to be 0.04, which corresponds to a 5-minute long sampling interval on average. It can be clearly seen that, as opposed to periodic sampling, the Markov-optimal policy assigns different duty cycles at different states, and therefore is able to reduce the expected state estimation error by imposing more frequent samples when the user state transition is more uncertain (i.e., higher state entropy in Markov case). On the other hand, in order to balance the overall energy consumption, the sampling interval size is increased when state transition is more predictable (as illustrated by the second half of user trace in figure 4.8 when the user tends to be more “stabilized”).

### 4.6 Summary

In this chapter, we study the problem of energy efficient sampling of a discrete, Markovian user state process. In contrast to our prior work [88] where the best stationary deterministic policy is found by exhaustively searching through all sensing intervals, in this chapter, by formulating the constrained optimization problem as an infinite-horizon CMDP with expected average cost, the optimal sensor sampling policy for Markovian user state transitions can be obtained in a computationally efficient manner. The resulting Markov-optimal policy achieves the lowest expected user state estimation error, while maintaining the energy consumption below a given budget. The Markov-optimal policy is compared against uniform periodic sampling numerically and we find that the performance gain is positively correlated with the standard deviation of state entropies. We also apply the Markov-optimal policy to real traces and it is shown to be capable of providing
significant improvement over periodic sampling, even the real user state transitions are not strictly Markovian.
Chapter 5

User state estimation and sensing policy optimization: the semi-Markovian case

5.1 Overview

As we demonstrated in the previous chapter, strict Markovian characteristics are rarely seen in real user state traces. In this chapter, we relax the assumption such that the analysis is no longer limited to Markov processes. Instead of assuming geometrically distributed state durations, we consider discrete time semi-Markov processes, which allow arbitrary state sojourn time distributions and provide more general and realistic models than Markov processes\(^1\). We propose a state estimation mechanism for semi-Markov processes, and provide a linear programming based approximate sensing policy optimization framework, which is able to efficiently schedule sensor samplings in order to minimize state estimation error given an energy constraint. Both the semi-Markov estimation mechanism and the optimization framework can be applied to data traces with any types of state sojourn time distribution, whose probability mass function (pmf) parameters are not necessarily known \textit{a priori} and can be obtained through the learning of historic data.

\(^1\)Note that a discrete time Markov process is a special case of discrete time semi-Markov process where the sojourn times are geometrically distributed.
5.2 Semi-Markovian state estimation for missing observations with sojourn time truncation

5.2.1 Preliminaries

We assume that time is discretized into consecutive time slots and the user state evolves as a N-state discrete time semi-Markov process. Recall that semi-Markov model is similar to Markov model such that state jumps follow a Markov chain with no self-transitions, and is different such that state sojourn times could be random variables with any types of distribution. Similar to the assumption in chapter 4, sensor observations are assumed to be perfect, such that if the sensor is activated in every time slot, the state process could be obtained with 100% accuracy. However, due to energy consumption concern, the sensor needs to perform duty cycling according to certain sensing policy, which leads to a sparsely sampled state sequence with multiple intervals of missing observations (see illustrative figure 5.1). In this thesis, we use the terms “estimation interval” and “observation interval” interchangeably, both denoting the interval of time slots between two consecutive sensor observations.

Figure 5.1: The semi-Markov process is sampled at discontiguous time slots, leading to multiple estimation intervals.

A complete definition of semi-Markov process can be found in section 2.3.2.1 of Chapter 2. Let \( p_{ij} \) and \( w_i(a) \) be the state transition probability and pmf of state sojourn duration respectively.
Recall that $1 \leq a \leq A_i < \infty$, and $\sum_a w_i(a) = 1$, where $A_i$ is the maximum sojourn duration of state $i$. For practical purposes, we approximate distributions with infinite support with a sufficiently large choice of $A_i$.

Let $O_t$ represent the observed state at time $t$, and we investigate the following state estimation problem: given the state distribution parameters $p_{ij}$ and $w_i(a)$, and two state observations $O_{t_m} = o_1$ and $O_{t_n} = o_2$, what is the most likely state at any given time slot between $t_m$ and $t_n$?

It is important to note that, unlike the assumptions made in previous works [31, 96], the observed states $o_1$ and $o_2$ do not necessarily need to start at time $t_m$ and end at time $t_n$. Instead, the state estimation is conducted in real-time for any given estimation interval, where sensor observations could be made at arbitrary time slots, and the state detection may belong to any instance of the sojourn time duration.

We propose a forward-backward based state estimation mechanism in Section 5.2.2 and 5.2.3 that estimates the most likely state for each time slot within an estimation interval. In Section 5.2.4, we discuss an alternative approach, where the forward and backward variables are defined using the knowledge of the origin time, initial state distribution, and all observations made. This alternative method may lead to better performance but can only be run off-line, and is computationally untractable in most scenarios because it is hard to automatically keep track of when a state enters and finishes its sojourn time.

### 5.2.2 The forward-backward variables: definition and initialization

As the state observation $o_1$ may truncate the state sojourn time and could belong to any instance of the state duration of $o_1$, we define $t_m'$ (where $t_m - A_{o_1} < t_m' \leq t_m$) as the time at which state $o_1$ is entered, i.e., the observed state $o_1$ started its sojourn time at time $t_m'$, and lasted through time $t_m$. 
In real-time system implementation where state durations are probabilistic, the exact value of \( t'_m \) is often unavailable since the sensor makes sparse observation and may not have the knowledge of complete historic state sequence. Therefore, for a state evolving process that is divided into multiple estimation intervals, \( t'_m \) could be obtained through the following alternatives:

- **Method 1 (M1):** Use state estimation result from the previous estimation interval.

- **Method 2 (M2):** Use the expected value \( E_p \) of past sojourn time, which can be calculated as follows (for state \( i \)):

  \[
  E_p(i) = \sum_{a=1}^{A_i} \sum_{\tau=1}^{a} \frac{w_i(a) \cdot \tau}{a}.
  \]  

- **Method 3 (M3):** Simply let \( t'_m = t_m \).

It can be seen that, M1 conducts state estimation based on previously estimated results, which may cause error propagation for certain types of state distribution, i.e., the previous estimation error may accumulate and lead to undesired performance in future estimations. M2 and M3 do not suffer this problem because they approximate the history of state sojourn durations using expected and zero past sojourn time, respectively. We will study the impact of these three \( t'_m \) selection methods on simulated state sequences in section 5.2.5, under different probabilistic state sojourn time distributions.

Let \( A \) be the maximum length of state sojourn times, i.e.:

\[
A = \max_i \{A_i\}, \forall i \in \{1, 2, ..., N\}.
\]  

(5.2)
For a given time slot $t$, we define forward variables $f_t(i)$ and $f^*_t(i)$ as:

$$f_t(i) = \text{Prob}[o_1, t'_m, \text{state } i \text{ ends at } t] = \sum_{a=1}^{A} f^*_t(a(i)) \cdot w^\triangle_i(a), \quad (5.3)$$

where

$$w^\triangle_i(a) = \begin{cases} w_i(a), & \text{if } t - a \geq t_m, \\ w'_i(a) = \frac{w_i(a)}{\sum_{k=t_m-t'_m+1}^{A} w_i(k)}, & \text{else.} \end{cases} \quad (5.4)$$

and

$$f^*_t(i) = \text{Prob}[o_1, t'_m, \text{state } i \text{ starts at } t + 1] = \sum_{j=1}^{N} f_t(j) \cdot p_{ji}. \quad (5.5)$$

The reason that $w^\triangle_i(a)$ is defined differently from $w_i(a)$ (as given in equation (5.4)) under the condition that $t - a < t_m$, is that the observed state at time $t_m$ has already spent its sojourn time for $t_m - t'_m + 1$ consecutive time slots such that the rest of the sojourn time distribution needs to be conditioned upon this fact, i.e., $w'_i(a) = \text{Prob}[\text{state } i \text{ lasts for } a \text{ slots } | a \geq t_m - t'_m + 1]$. 

In order to compute $f_t(i)$, the boundary condition for $f^*_t(i)$ needs to be initialized as follows:

$$f^*_t(i) = \begin{cases} 1, & \text{if } t = t'_m - 1, \text{ and } i = o_1, \\ 0, & \text{otherwise.} \end{cases} \quad (5.6)$$
and

\[ f_{t_m}^*(i) = p_{o1i} \cdot w_{o1}(t_m - t_m' + 1), \forall i \neq o_1. \]  

(5.7)

Equation (5.6) emphasizes the fact that the detected state \( o_1 \) has started its sojourn time at time \( t_m \) whereas equation (5.7) computes the probability that state \( o_1 \) transits to another state in the first time slot of the estimation interval (i.e., \( t_m + 1 \)).

To initialize \( f_t(i) \) at time \( t_m \), the probability that observation \( o_1 \) ends exactly at \( t_m \) needs to be calculated, which satisfies:

\[
\begin{align*}
  f_{t_m}(i) & = \begin{cases} 
  w_{o1}'(t_m - t_m' + 1), & \text{if } i = o_1, \\
  0, & \forall i \neq o_1.
  \end{cases}
\end{align*}
\]

(5.8)

We further define the backward variables \( b_t(i) \) and \( b_t^*(i) \) for a given time slot \( t \):

\[
b_t(i) = \text{Prob}[o_2| \text{state } i \text{ starts at } t] = \sum_{a=1}^{A} b_{t+a}^*(i) \cdot w_i(a),
\]

(5.9)

and

\[
b_t^*(i) = \text{Prob}[o_2| \text{state } i \text{ ends at } t-1] = \sum_{j=1}^{N} b_t(j) \cdot p_{ij}.
\]

(5.10)

Since we do not assume the observed state \( o_2 \) ends at \( t_n \), the initial and boundary conditions for \( b_t^*(i) \) are therefore given by:

\[ b_{t_n}^*(o_2) = 0, \]  

(5.11)
and

\[
b_{t_n+x}^*(i) = \begin{cases} 
1, & \forall x \in \{1, 2, ..., A_{o_2}\}, \text{ if } i = o_2, \\
0, & \text{otherwise.} 
\end{cases}
\] (5.12)

Equation (5.12) describes that given the fact that state \( o_2 \) ends at any time slot after time \( t_n \), the probability of seeing \( o_2 \) at time \( t_n \) is 1, and correspondingly, the probability of seeing other states at \( t_n \) is 0.

The initial conditions for \( b_t(i) \) need to be specified as:

\[
b_{t_n}(i) = \begin{cases} 
\sum_{x=1}^{A_{o_2}} b_{t_n+x}^*(o_2) \cdot w_{o_2}(x), & \text{if } i = o_2, \\
0, & \forall i \neq o_2. 
\end{cases}
\] (5.13)

in which \( b_{t_n}(o_2) \) is the average aggregated probability that state \( o_2 \) begins at time \( t_n \).

### 5.2.3 State estimation

Once the values of all the forward and backward variables have been obtained by recursion, it is feasible to estimate the state for each time slot between \( t_m \) and \( t_n \). We define a new forward variable:

\[
\phi_t(i) = \text{Prob}[o_1, t'_m, s_t = i].
\] (5.14)
\( \phi_t(i) \) holds the following property:

\[
\begin{align*}
\phi_t(i) &= \text{Prob}[o_1, t'_m, s_{t-1} = i \text{ and } s_t = i] \\
&\quad + \text{Prob}[o_1, t'_m, s_{t-1} \neq i \text{ and } s_t = i] \\
&= \text{Prob}[o_1, t'_m, s_{t-1} = i] \\
&\quad - \text{Prob}[o_1, t'_m, s_{t-1} = i \text{ and } s_t \neq i] \\
&\quad + \text{Prob}[o_1, t'_m, s_{t-1} \neq i \text{ and } s_t = i] \\
&= \phi_{t-1}(i) - f_{t-1}(i) + f^*_t(i).
\end{align*}
\] (5.15)

It can be seen that \( \phi_t(i) \) can be calculated by recursion as well. To initialize, we have the following equalities:

\[
\phi_{t_m}(i) = \begin{cases} 
1, & \text{if } i = o_1, \\
0, & \text{else}.
\end{cases}
\] (5.16)

Next, we define a new backward variable:

\[
\psi_t(i) = \text{Prob}[o_1, t'_m, o_2, s_t = i],
\] (5.17)

which is the joint probability of \( o_1, t'_m, o_2 \), as well as the unknown state at time \( t \). Following the similar relationship illustrated in equation(5.15)-(5.15), \( \psi_t(i) \) can be further written in the following recursive form:

\[
\psi_t(i) = \psi_{t+1}(i) - b_{t+1}(i) \cdot f^*_t(i) + f_t(i) \cdot b^*_t(i).
\] (5.18)
The initial condition for this backward variable $\psi_t(i)$ is given by:

$$
\psi_{tn}(i) = \text{Prob}[o_1, t'_m, o_2, s_{tn} = o_2] \\
= \text{Prob}[o_1, t'_m, s_{tn} = o_2] \\
= \phi_{tn}(i),
$$

whose value has been stored at the end of the recursion when calculating $\phi_t(i)$.

Finally, the maximum a posteriori (MAP) estimate of user state at time $t$ is given by:

$$
s'_t = \arg\max_{i \in \{1, 2, \ldots, N\}} \text{Prob}[s_t = i|o_1, t'_m, o_2] \\
= \arg\max_{i \in \{1, 2, \ldots, N\}} \frac{\text{Prob}[s_t = i, o_1, t'_m, o_2]}{\text{Prob}[o_1, t'_m, o_2]} \\
= \arg\max_{i \in \{1, 2, \ldots, N\}} \psi_t(i), \forall t = t_n - 1, \ldots, t_m + 1.
$$

The following algorithm summarizes the complete procedure of estimating user state with missing observations between two neighboring state detections for a discrete-time, semi-Markov process:

It can be seen that in Algorithm 2, solving for $f_t(i)$ and $b_t(i)$ at any given time $t$ consumes at most $A$ multiplications, whereas calculating $f^*_t(i)$ and $b^*_t(i)$ requires $N$ multiplications. Therefore, for an estimation interval with size $T$, the computational complexity of the semi-Markovian estimation algorithm is $O(TN(N + A))$. In comparison, the Markovian state estimation mechanism proposed in Chapter 4 needs $T$ matrix multiplications where the matrix size is $N \times N$. This leads to $O(TN^3)$ computational complexity without any speed-up in matrix multiplications.
Algorithm 2 An algorithm that estimates missing state information between two observations $O_{t_m} = o_1$ and $O_{t_n} = o_2$, for a discrete-time, semi-Markov process.

1: Input: $p_{ij}$, $w_n(i)$, $A_i$, and $t'_m$ (whose value depends on whether M1, M2, or M3 is used), $\forall i, j \in \{1, 2, ..., N\}$

2: Output:
   State estimation result $S' = \{s'_{t_m+1}, s'_{t_m+2}, ..., s'_{t_n-1}\}$ and the corresponding expected error sequence $e_s = \{e_{t_m+1}, ..., e_{t_n-1}\}$

3: Initialize forward variable $f_t(i)$ and $f^*_t(i)$ using equation (5.6), (5.7), and (5.8). Solve by recursion using equation (5.3) and (5.5).

4: Initialize backward variable $b_t(i)$ and $b^*_t(i)$ using equation (5.11), (5.12), and (5.13). Solve by recursion using equation (5.9) and (5.10).

5: Calculate $\phi_t(i)$, $\forall i \in \{1, ..., N\}$, and $\forall t \in [t_m, t_n]$, based on equation (5.15) and (5.16).

6: Calculate $\psi_t(i)$, $\forall i \in \{1, ..., N\}$, and $\forall t \in [t_m, t_n]$, based on equation (5.18) and (5.19).

7: State estimation:

   $s'_t = \arg\max_{i \in \{1, 2, ..., N\}} \psi_t(i)$, $\forall t = t_n - 1, ..., t_m + 1$.

8: Expected estimation error:

   $e_t = \kappa \cdot \left\{1 - \max_{i \in \{1, 2, ..., N\}} \psi_t(i)\right\}$, $\forall t = t_n - 1, ..., t_m + 1$.

   where $\kappa$ is a normalization factor.

Thus, although the complexity of Markovian estimation is not affected by the state sojourn duration $A$, as the number of states increases, our semi-Markovian approach could potentially become more computationally efficient.

5.2.4 The forward-backward variables: an alternative

An alternative way to define the forward and backward variable is to consider not only the neighboring observations but the entire observed sequence, assuming the knowledge of (a) the initial state distribution $\mu$, (b) the origin time $t_0$ that the initial state begins, and (c) the ending time $t_L$ where the last state finishes its sojourn time.
Let $O_{0\to t}$ denote the observation sequence up to the current time $t$, whereas $O_{t\to L}$ denote the rest of the observation sequence after time $t$. Then the forward and backward variables could be defined in the following way:

\begin{align*}
    f_t(i) &= \text{Prob}[\mu, t_0, O_{0\to t}, \text{state } i \text{ ends at } t] \quad (5.21) \\
    f_t^*(i) &= \text{Prob}[\mu, t_0, O_{0\to t}, \text{state } i \text{ begins at } t + 1], \quad (5.22)
\end{align*}

and

\begin{align*}
    b_t(i) &= \text{Prob}[t_L, O_{t\to L}, \text{state } i \text{ begins at } t] \quad (5.23) \\
    b_t^*(i) &= \text{Prob}[t_L, O_{t\to L}, \text{state } i \text{ ends at } t - 1]. \quad (5.24)
\end{align*}

The forward-backward variables can be initialized according to the specific inputs and state estimation can be conducted by similar recursions introduced in section 5.2.3. Since the above definitions use the exactly information of where the initial state started, it may provide better estimation accuracy as compared to Algorithm 2, which requires an approximation of the past state sojourn time. However, this mechanism only allows off-line estimation since it requires future observation results when applied. Moreover, as the length of semi-Markov process grows, it becomes harder to keep track of all observations and the computationally complexity increases correspondingly. Therefore, in this thesis, we do not provide further analysis of this approach, and we leave the algorithmization and possible optimization of this mechanism as well as its performance evaluation to future work.
5.2.5 State estimation performance on simulated two-state transition processes

In this subsection, we evaluate the performance of semi-Markovian state estimation mechanism (Algorithm 2) on two-state, simulated state transition processes and explain the observations. In particular, each state sequence is generated based on pre-specified state sojourn time distributions, and is then observed at a constant frequency, leading to consecutive estimation intervals with equal lengths. Within each estimation interval, Algorithm 2 is conducted for state estimation. Once state estimation is complete for all the estimation intervals, the overall estimated state sequence is compared to the original sequence and the state estimation error ratio $R$ (which is the total number of incorrect estimations for the entire sequence divided by the length of the state sequence) is calculated.

Recall that one of our goals is to design a real-time state estimation mechanism for sparsely sampled real user state processes. In fact, in many human related traces, state distributions have been found to exhibit heavy tails, e.g., Internet file sizes [24], human contact arrival processes [19, 86], user motion transitions [89], and so on. These types of traces are often modeled as memoryless distributions such as geometric distribution in order to reduce the complexity of study. In this section, we compare the performance of semi-Markovian state estimation to state estimation under Markovian assumption, where, no matter what distribution function does the state duration follow, the state sequence is always assumed to be a Markov process. The Markovian assumption, although not strictly valid in reality, has been widely used to model user related state traces such as speech [42, 70], mobility [38, 75], and activity [34, 73].
Under the stationary Markovian assumption, the state transition probabilities can be estimated in a standard way [9], which is to calculate the ratio of state transition frequencies:

\[ p_{ij} = \frac{n_{ij}}{n_j}, \quad (5.25) \]

where \( n_{ij} \) is the frequency of state transition from \( i \) to \( j \), and \( n_j = \sum_i n_{ij} \).

Estimating state information for missing observations under the Markovian assumption has been discussed in [89]. Here we give a brief overview of the mechanism: given that state \( i \) is detected at time \( t_m \), and state \( j \) is detected at time \( t_n \) (\( t_m, t_n \in T \) and \( t_m < t_n \)), the most likely state (denoted by \( s'_t \)) at time \( t \) is selected as

\[ s'_t = \arg\max_k \left\{ \frac{p_{ik}(t-t_m) \cdot P_{kj}(t_n-t)}{P_{ij}(t_n-t_m)} \right\}, \quad (5.26) \]

i.e., the MAP estimation for Markovian process only requires one equation in matrix multiplication form since Markovian estimation does not suffer the “past sojourn time selection” problem because it implicitly assumes that the underlying state process is memoryless.

We first conduct two sanity checks of the correctness of the semi-Markovian estimation by examining its performance on deterministic and geometric distributed state processes. We then select one simple and commonly seen distribution type, namely, binomial distribution, and generate state process and examine the performance of semi-Markovian estimation according to the distribution parameters. For the commonly seen heavy-tail distributions in real user state traces, we choose to investigate Zipf’s Law distribution, which is a discrete, power law, and heavy-tailed distribution type. The pmf parameters are summarized in table 5.1.
Deterministic
\[ \begin{align*}
  w_1(a) &= \begin{cases} 
    1, & \text{if } a = 20, \\
    0, & \text{else.}
  \end{cases} \\
  w_2(a) &= \begin{cases} 
    1, & \text{if } a = 40, \\
    0, & \text{else.}
  \end{cases}
\end{align*} \]

Geometric
\[ w_1(a) = (1 - p_1)^{a-1} \cdot p_1, \text{ where } p_1 = 0.2 \]
\[ w_2(a) = (1 - p_2)^{a-1} \cdot p_2, \text{ where } p_2 = 0.6 \]

Binomial
\[ \begin{align*}
  w_1(a) &= \binom{N_1}{a} p_1^a (1 - p_1)^{N_1-a}, \text{ where } p_1 = 0.8 \text{ and } N_1 = 100 \\
  w_2(a) &= \binom{N_2}{a} p_2^a (1 - p_2)^{N_2-a}, \text{ where } p_2 = 0.4 \text{ and } N_2 = 100
\end{align*} \]

Zipf’s Law (smaller tail)
\[ \begin{align*}
  w_1(a) &= \frac{1/a^s}{\sum_{n=1}^{N_1} 1/n^s}, \text{ where } s_1 = 0.1 \text{ and } N_1 = 100 \\
  w_1(a) &= \frac{1/a^s}{\sum_{n=1}^{N_2} 1/n^s}, \text{ where } s_2 = 0.05 \text{ and } N_2 = 100
\end{align*} \]

Zipf’s Law (larger tail)
\[ \begin{align*}
  w_1(a) &= \frac{1/a^s}{\sum_{n=1}^{N_1} 1/n^s}, \text{ where } s_1 = 0.01 \text{ and } N_1 = 100 \\
  w_1(a) &= \frac{1/a^s}{\sum_{n=1}^{N_2} 1/n^s}, \text{ where } s_2 = 0.005 \text{ and } N_2 = 100
\end{align*} \]

Table 5.1: Four different distribution types and their specific parameters utilized in the simulation.

For each state distribution instance, state estimations are conducted on 10 simulated state processes each containing 1000 state transitions. Specifically, for all non-deterministic state processes, semi-Markovian estimation mechanism (Algorithm 2) with all variations M1, M2, and M3, as well as Markovian estimation mechanism are conducted and their corresponding error ratio \( R \) are compared under different estimation window sizes ranging from 20 to 200 (we also show the result for very large window sizes in case of binomial state distribution which converges slowly). The results and observations are discussed as follows:

A. Sanity checks: deterministic and geometric distributions

The evaluation of semi-Markovian estimation on deterministic state processes serves as the first sanity check of the correctness of Algorithm 2. As can be seen from figure 5.2 (left portion) that, the semi-Markovian estimation mechanism with M1 is able to reconstruct the entire
sequence without any error, whereas applying Markovian estimation leads to undesired performance. Intuitively, when state sojourn durations are fixed, knowing how long a state has already spent its sojourn time should lead to perfect state estimations in each estimation interval.

![Figure 5.2: Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state distributions are deterministic (left) and geometric (right).](image)

The right portion of figure 5.2 shows that semi-Markovian and Markovian estimation mechanisms lead to very close results when state sojourn times are geometrically distributed\(^2\). Because Markov process is a special case of semi-Markov process, both estimation mechanisms provide very similar results on a state sequence with geometrically distributed state durations. In fact, different versions of semi-Markovian estimation (M1, M2, and M3) provide almost identical results due to the memoryless property of geometric distribution, and this observation serves as the second sanity check of the correctness of semi-Markovian estimation mechanism.

**B. Binomial**

It can be seen from figure 5.3 that semi-Markovian estimation algorithm with M1 outperforms Markovian estimation and provides the best performance, whereas M3 leads to the worst

\(^2\)As noted in section 5.2.1, although geometric distribution supports infinite set of \(a\) values, we limit it such that \(1 \leq a \leq 100\) in order to satisfy the running condition of Algorithm 2.
performance. Note the similarity between binomial distribution and deterministic distribution, where both pmfs contain peak probability and decay quickly around the concentration. As semi-Markovian estimation mechanism provides state estimations strictly based upon state distribution pmf, the “peak-like” feature greatly reduces the estimation error; therefore, knowing history state sequence information (M1) will always lead to better state estimation results. In addition, it can be found from the right portion of figure 5.3 that the semi-Markovian estimation mechanism provides better performance even under extremely large estimation window sizes, and converges eventually.

Figure 5.3: Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state distributions are binomial. Right figure shows convergence at very large window sizes.

C. Zipf’s Law

In order to provide a comprehensive performance evaluation of the semi-Markovian estimation, we study Zipf’s Law distribution with different tail sizes by varying the distribution parameters $s_1$ and $s_2$ as illustrated in table 5.1. It can be seen from figure 5.4 that semi-Markovian estimation leads to higher performance improvement under larger tail sizes. This is due to the
fact that assuming the state process to be Markovian becomes less approximate as the tail size increases, therefore, semi-Markovian estimation produces higher gain over Markovian estimation as the tail size of state distribution increases, since it estimates state information based on the exact state distribution functions.

Figure 5.4: Semi-Markovian estimation vs. Markovian estimation on simulated two-state process where state sojourn durations follow Zipf’s Law distributions, with different tail sizes according to the definitions in table 5.1.

To better understand the results, we summarize in table 5.2 the results of percentage gain over Markovian estimation provided by semi-Markovian estimations corresponding to figure 5.4. The average as well as the maximum gain of all estimation window sizes are shown for methods M1, M2, and M3. While M1 suffers the error propagation problem and M3 simply assumes zero past sojourn time, it can be seen that semi-Markovian estimation with M2 leads to the best average performance among all versions of Algorithm 2.

We also observe the fact that for probabilistic state distributions, all estimation mechanisms converge as the estimation window size becomes large enough. This is because as the state process is sampled extremely sparsely, no matter what estimation mechanism is conducted, it essentially selects a state with higher occurrence probability, i.e., steady state probability. Therefore,
<table>
<thead>
<tr>
<th></th>
<th>Smaller tail</th>
<th>Larger tail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( s_1 = 0.1 ) ( s_2 = 0.05 )</td>
<td>( s_1 = 0.01 ) ( s_2 = 0.005 )</td>
</tr>
<tr>
<td>Avg. Gain</td>
<td>Max Gain</td>
<td>Avg. Gain</td>
</tr>
<tr>
<td>M1</td>
<td>8.24%</td>
<td>18.21%</td>
</tr>
<tr>
<td>M2</td>
<td>9.08%</td>
<td>23.15%</td>
</tr>
<tr>
<td>M3</td>
<td>5.87%</td>
<td>17.26%</td>
</tr>
</tbody>
</table>

Table 5.2: Average and maximum percentage gain of semi-Markovian estimation with M1, M2, and M3 over Markovian estimation under different estimation window sizes ranging from 1 to 500.

The expected estimation error \( R \) as estimation window size goes to infinity satisfies the following relationship for an N-state semi-Markov process:

\[
R_{T \to \infty} = 1 - \max_i \left\{ \frac{M_i}{\sum_i M_i} \right\},
\]

where \( M_i \) is the mean sojourn time of state \( i \), and \( i \in \{1, \ldots, N\} \).

5.2.6 State estimation performance on real traces

To further strengthen our study, the performance of semi-Markovian estimation mechanism is evaluated on real state traces. We consider two types of data: (a) smartphone network connectivity state and (b) temperature status, in order to examine the estimation algorithm performance on both heavy-tailed state durations and arbitrary state durations that cannot be easily fitted by a certain parametric distribution type.

5.2.6.1 Heavy-tailed state durations: device network connectivity

We study traces collected in the CoSphere project \[68\]^3 conducted by researchers at Telematica Instituut in The Netherlands within February/March 2007 time frame. During the one-month

^3The data is available at CRAWDAD website (http://crawdad.cs.dartmouth.edu/data.php).
experiment, twelve participants each carried a smartphone that automatically logged its exposure to cell tower, WiFi access point, and Bluetooth devices. In our study, we pick WiFi and Bluetooth data traces and view the user state as either “Connected” or “Not connected” to infrastructure access points or peer devices, and compare the performance of semi-Markovian estimation and Markovian estimation in terms of the average state estimation error produced.

Recall that semi-Markovian state estimation mechanism requires the state distribution parameters $p_{ij}$ and $w_a(i)$ to be estimated a priori based on the collected user state traces. In particular, assuming a stationary process, a standard model parameter estimation mechanism is given by [95] and is summarized as follows:

(1) State transition probability $p_{ij}$ is estimated as the ratio between number of transitions from state $i$ to $j$ and the total number of transitions into state $j$:

$$p_{ij} = n_{ij}/n_j,$$

(5.28)

where $n_{ij}$ is the frequency of state transition from $i$ to $j$, and $n_j = \sum_i n_{ij}$. This is similar to equation (5.25) where the Markovian transition probabilities are calculated, except that state transitions only happen whenever a different state is entered. Note that for two-state semi-Markov chain, $p_{12} = p_{21} = 1$.

(2) State sojourn time distribution $w_a(i)$ is estimated as the number of times that state $i$ lasts exactly for $a$ time slots, divided by the number of times that state $i$ is visited:

$$w_a(i) = n_a(i)/n_i, \forall a \in \{1, 2, ..., A\}.$$  

(5.29)
Each user state trace is sampled periodically according to the input estimation window size, and within each observation interval, both the semi-Markovian estimation mechanism and the Markovian estimation mechanisms are executed. We compare the average state estimation error from both estimation mechanisms while varying the observation window size from 20 to 200, and demonstrate both the state duration pmfs and the error comparison result in figure 5.5 and 5.6.

Figure 5.5: State duration pmfs for Bluetooth connectivity state trace and the comparison of both estimation algorithms.

Figure 5.6: State duration pmfs for WiFi state trace and the comparison of both estimation algorithms.

It can be seen that semi-Markovian estimation outperforms Markovian estimation on both Bluetooth and WiFi connectivity traces. In particular, the semi-Markovian estimation is able
to reduce more than 40% estimation error than Markovian estimation under certain estimation window sizes. This justifies our main statement that, the state sojourn time in real user state traces tend to exhibit heavier tails than memoryless distributions, therefore, modeling the data as Markovian process and using Markovian estimation may lead to undesired state estimation performance; whereas the semi-Markov process is a more general model for real user state traces and the semi-Markovian estimation (Algorithm 2) could provide better performance by estimating state information using the exact state distribution function.

5.2.6.2 Arbitrary state durations: temperature

The reason we employ temperature data in our study is to evaluate the performance of semi-Markovian estimation on data traces that demonstrate arbitrary types of state duration distributions (as can be seen in figure 5.7). In particular, the outdoor temperature data of Los Angeles (sampled once an hour) in all Januaries within the past six years\(^4\) has been obtained and divided into “High” and “Low” states according to a pre-selected threshold. The six pieces of data are aggregated and treated as an entire state sequence on which the estimation algorithms are applied.

The state estimation result is shown in figure 5.7, from which it can be seen that although the state duration distributions tend to be “bimodal” and may not be easily characterized by a single parametric distribution type, the semi-Markovian estimation algorithm provides lower error by learning the model parameters using equation (5.28) and (5.29) and estimating missing state information using these parameters according to Algorithm 2.

\(^4\)The data is available at Weather Underground (http://www.wunderground.com/).
5.3 Optimal sensor scheduling with energy constraint

In practical system operations, a device energy consumption budget is often specified in order to ensure a long enough device lifetime. Similar to Chapter 4, we define $\xi$ (where $0 \leq \xi \leq 1$) as the budget which is the maximum expected energy consumption allowed, and let $u$ denote a sensor sampling policy, which specifies how sensor duty cycles should be assigned under different user states detections, i.e., how long the sensor should stay idle after each observation. We investigate a similar constrained optimization problem that was first studied in Chapter 4 of this thesis: given the sensor energy consumption budget $\xi$ and state distribution parameters, how should sensor duty cycles be scheduled such that the expected state estimation error can be minimized while the expected energy consumption is maintained below the budget?

Feinberg [28] demonstrated the use of LP to solve CSMDP with average rewards. In fact, LP is also the standard tool to solve the above type of constrained optimization problems under Markovian assumption [3] (which can be formulated as a CMDP), and its application on energy efficient mobile sensing can be found in [89]. In this paper, we utilize the LP framework provided
in [28, 30] with specific definition of costs (i.e., energy and error) to solve the above constrained optimization problem in order to obtain $u^*_s$, the semi-Markov optimal sensing policy, and evaluate its performance.

### 5.3.1 Obtaining the semi-Markov optimal policy $u^*_s$

Minimize

$$\sum_{y \in X} \sum_{b \in B} \rho(y, b) c_s(y, b)$$  \hspace{1cm} (5.30)

subject to:

$$\sum_{y \in X} \sum_{b \in B} \rho(y, b) (\delta_x(y) - P_{y|bx}^s) = 0, \forall x \in X,$$  \hspace{1cm} (5.31)

$$\sum_{y \in X} \sum_{b \in B} \rho(y, b) = 1,$$  \hspace{1cm} (5.32)

$$\rho(y, b) \geq 0, \forall y, a, \text{ and}$$  \hspace{1cm} (5.33)

$$\sum_{y \in X} \sum_{b \in B} \rho(y, b)(1 - b \cdot \xi) \leq 0,$$  \hspace{1cm} (5.34)

where $X$ and $B$ are the set of states and the set of action inputs, respectively.

Note that the outcome of the above constrained optimization framework is a stationary randomized policy, i.e., the optimal choice of sampling interval is randomized over several integers and the decision does not vary over time.

Unlike the original LP framework [28] which yields a policy that does not depend on past sojourn times, the above LP formulation takes into account the previous sojourn time by defining the system state transition probability $P_{y|bx}^s$ and the cost $c_s(y, b)$ based on the expected past sojourn duration (defined in M2). In particular, given the estimation interval size $b$, starting state
and ending state $x$, the interval transition probability $P_{ybx}^s$ could be calculated at step 5 of Algorithm 2, i.e.:

$$P_{ybx}^s = \phi_{ts}^n(x),$$

(5.35)

under appropriate input conditions:

$$o_1 = y, o_2 = x, t_n - t_m = b,$$

(5.36)

and

$$t_m - t'_m = \sum_{a=1}^{A} \sum_{\tau=1}^{a} w_y(a) \cdot \tau. - 1,$$

(5.37)

as it is assumed that a detected state $y$ has already spent $E_p(y)$ time slots, the expected past sojourn time.

Similarly, after step 8 of Algorithm 2, the aggregated estimation error within that estimation interval can be expressed as:

$$e_{sum} = \sum_{t=t_m}^{t_n} e_t,$$

(5.38)

Define $e_{sum}$ as the aggregated estimation error under condition (5.36), therefore, the intermediate cost $c_s(y, b)$ could be expressed by the following:

$$c_s(y, b) = \sum_{x} P_{ybx}^s \cdot e_{sum},$$

(5.39)

which is the expected, aggregated semi-Markovian state estimation error while state $y$ is detected and sensing interval size $b$ is chosen.
Once the linear programming is solved for optimal occupation measures $\rho^*(y, b)$, the semi-Markov optimal policy $u^*_s$ can be obtained such that the probability of taking action $b$ at state $y$ is equal to $\frac{\rho^*(y, b)}{\rho^*_y}$, where $\rho^*_y = \sum_{b \in B} \rho^*(y, b), \forall y \in X$.

### 5.3.2 Performance evaluation of $u^*_s$ on simulated processes

We first evaluate the performance of $u^*_s$ on simulated state processes with binomial and Zipf’s law (with larger tail) state distributions whose parameters are specified in table 5.1. In particular, for each simulated state sequence, the Markov-optimal sensing policy $u^*_m$ as well as $u^*_s$ are computed. Both policies are then applied to the state sequence and state estimations are carried out using Markovian and semi-Markovian state estimation respectively.

![Figure 5.8](image.png)

**Figure 5.8: Performance of $u^*_m$ vs. $u^*_s$, on binomially and Zipf’s Law distributed state processes.** Figure shows the estimation error w.r.t. energy budget (in log scale).

Figure 5.8 indicates that, in case of processes whose state distributions are binomial, $u^*_s$ provides significant state estimation error reduction as compared to $u^*_m$, especially under low energy budget values. When state duration distributions follow Zipf’s law, $u^*_s$ produces lower estimation error than $u^*_m$ under small energy budgets, and demonstrates similar performance as energy budget increases. This can be explained by the fact that semi-Markovian estimation mechanism
provides significant performance improvement only on a subset of estimation window sizes (as illustrated by figure 5.4), which leads to the corresponding gain by $u^*_s$ when energy budgets are stringent.

In fact, for a given state process, if the semi-Markovian estimation mechanism outperforms Markovian estimation by producing lower estimation error at certain estimation window sizes, $u^*_s$ should provide better performance than $u^*_m$ on the same data set under corresponding energy budgets, since the LP effectively searches for a combination of sensing intervals that lead to the minimum expected estimation error while not exceeding the energy budget.

5.3.3 Performance of $u^*_s$ on heavy-tailed real user state traces

The performance of $u^*_s$ is evaluated on the same set of real user state traces introduced in 5.2.6. As $u^*_m$ is optimal for Markov processes, we aim to explore whether $u^*_s$ could outperform $u^*_m$ by achieving a better trade-off between energy consumption and state estimation error, on user state traces that are not strictly Markovian. In particular, the following steps are executed for each user state trace:

- Estimate model parameters using (5.28) and (5.29).

- Compute $u^*_s$ using LP (4.10) - (4.14).

- Compute $u^*_m$ using LP1 proposed in [89].

- Apply both policies on real state process and conduct semi-Markovian estimation and Markovian estimation on missing observations, respectively.

- Obtain the overall estimation error as well as the energy usage, which is the ratio of the total number of samples over the length of state process.
Since the optimization framework is stochastic and is constrained on the expected energy usage, the true energy consumption when executing the policy on a particular trace may not exactly reflect the given energy budget. We thus show the results of estimation error vs. the actual energy usage in figure 5.9 instead of the energy budget input parameter. To better visualize the performance gain, the linear fit of the energy-error pairs under different policies are shown in figure 5.9. It can be seen that $u^*_s$ yields a better trade-off between energy consumption and state estimation error as compared to $u^*_m$, by producing 12.04% and 14.19% less estimation error on average in each respective case.

![Figure 5.9: Performance of $u^*_m$ vs. $u^*_s$, and the genie-aided estimation error on CoSphere network connectivity traces. Figure shows the estimation error w.r.t. energy budget (in log scale).](image)

For reference purpose, we also consider an idealized genie-aided mechanism that is able to find the “truly optimal” sampling policy based on complete past and future knowledge of the state sequence. In particular, if the entire state sequence and its length is known \textit{a priori}, the optimal
arrangement of a given number of sample points according to the energy budget can be selected using the following dynamic programming form:

\[
f(S, l, s, N, \mathcal{P}) = \min \left\{ f(S, l, s + 1, N, \mathcal{P}), \right. \\
\left. f(S, s, s + 1, N - 1, \mathcal{P}) + e(S, l, s, \mathcal{P}) \right\} \tag{5.40}
\]

where \( f(S, l, s, N, \mathcal{P}) \) corresponds to the minimum state estimation error for sequence \( S \), last observed at time \( l \), current decision epoch \( s \), model parameters \( \mathcal{P} \), and a total number of \( N \) available sample points, and \( e(S, l, s, \mathcal{P}) \) measures the semi-Markovian estimation error of the subsequence starting at time \( l \) and ending at time \( s \).

The recursive equation (5.40) can be solved by specifying the following boundary conditions:

\[
f(S, l, s, 0, \mathcal{P}) = e(S, s - 1, L, \mathcal{P}), \tag{5.41}
\]

where \( L \) is the last time slot index of the sequence, and

\[
f(S, l, s, N, \mathcal{P}) = e(S, l, s, \mathcal{P}), \text{ if } s = L - N. \tag{5.42}
\]

Although the genie-aided mechanism is able to find the lowest estimation error for a given state sequence, it does not provide a general sensing policy guideline because it is trace-specific and requires both future knowledge and complete state record. Nonetheless, it enables us to visualize the performance gap between the stochastic optimal policies and the trace-specific “ideal” result.
Note that the comparison under large energy budgets is not shown in figure 5.9 because \( u^*_s \) and \( u^*_m \) are found to provide comparably low estimation errors when energy budget is not stringent. In fact, this can be easily concluded from figure 5.4 and 5.8 which suggest that the estimation errors are similar under small estimation window sizes. However, real mobile sensing systems often operate under small energy budgets and deal with sparse observations, in which case the semi-Markovian estimation as well as the semi-Markov optimal policy lead to the best improvement and are therefore desirable implementation choices.

## 5.4 A novel client-server based system framework for energy efficient mobile sensing

In this section, we present the design, prototype implementation, and evaluation of a novel client-server based system architecture that conducts user behavior learning, optimal policy calculation, energy efficient sensing, and state estimation in an automated manner.

### 5.4.1 System design and implementation

The high level system design architecture is illustrated in figure 5.10. First, mobile client executes the *Behavior Learning Module*, which collects continuous user state process under full sensor activation and calculates the user state distribution parameters from recorded data trace. This process allows the mobile sensing client to be “user-specific”, i.e., different optimal sensing policies can be derived later on for users with different habits. Once enough data is collected and the dynamics of user state transitions are obtained, the state distribution parameters will be sent
to the Optimal Policy Calculation Module on the back-end server in order to compute the user-specific $u^*_s$. The optimal sensor duty cycles are transferred back to mobile client to begin energy efficient sensing, which is realized via the Energy Efficient Sensing Module. Eventually, when missing state information needs to be estimated, the intermittently sampled state trace will be sent to the State Estimation Module on the server again for state estimations, and the estimation result could then be transferred back to the client if necessary.

It can be seen from figure 5.10 that by sacrificing certain amount of networking cost, intensive and occasional computations are shifted to servers with much more powerful computational capabilities for faster and potentially more energy efficient computing. Note that an alternative system design approach, which is to perform all sensing and computations locally on the mobile device, although eliminates the need of communication, suffers several problems such as the lack of mathematical libraries and low computational capabilities on mobile devices. In this paper, we do not further investigate the trade-off between these two design methodologies and leave it to future study.

Figure 5.10: System modules and operation flow.
We implement a prototype of the system architecture shown in figure 5.10 as an activity recognition system on Nokia N95 devices (mobile client) and desktop PCs (back-end server) using Symbian C++ and Matlab, which communicate to each other via socket connections.

The mobile sensing client is designed to differentiate two basic human activity states ("Stable" and "Moving") based on accelerometer sensing. In particular, the acceleration values on x, y, and z axis of the accelerometer is continuously read for a certain duration, and the standard deviation of the magnitudes of all three axes readings is calculated. The user activity is classified into “Stable” or “Moving” based on whether the standard deviation is below or above some given threshold. Recall that because we assume perfect sensor detections, our goal is not to develop sophisticated activity classification algorithms, but to explore how to derive and implement the semi-Markov optimal policy on the mobile client and whether it is able to provide satisfying performance in real system implementations.

In our current system implementation, the user behavior learning phase is conducted for seven consecutive days, during which, the user activity trace is fully detected and logged in the mobile device file system. The symbol “0” and “1” are used to represent different user states so that the state distribution parameters can be easily calculated using equation 5.28 and 5.29. During the learning phase, the mobile device is frequently charged to maintain continuous sensing requirement.

On the server side, the policy calculation, state estimations, as well as socket connection setup are all accomplished in Matlab which provides standard LP solver. Upon receiving the user state distribution parameters, the server calculates the semi-Markov optimal sensing policy \( u_s \) and transfers it back to the client in terms of the average idle interval size under different state detections, such that at each decision epoch (time is virtually divided into discrete slots with 10
seconds length each), the mobile sensing client will determine how long to turn off the sensor until the next observation according to the policy.

### 5.4.2 System performance evaluation

As one novelty of this paper is to first implement the stochastic optimal sensing policies in real mobile sensing systems and compare them against the often-used uniform sampling, we focus on providing benchmark performance measurement of the system on the smartphone in terms of the battery lifetime as well as the state estimation error while implementing these policies. Thus, the investigation of the system energy efficiency while carrying out communications and the evaluation of how well the user behavior could be learned through the “full sensor activation” mechanism are currently beyond the scope of this paper, and we leave the potential optimization in the trade-off between communication and computation cost and the investigation of advanced learning techniques such as on-line learning and decision making to future study.

Five independent experiments have been conducted and their setups are introduced below. During each experiment, the empirical devices are carried by the participant as his personal cell phone but with no usage other than activity sensing.

**Exp-1** ($\xi = 1$): the accelerometer is continuously sampled and user activity is recognized and recorded at each time slot. This experiment sets the benchmark of the device battery lifetime while sensor is fully activated.

**Exp-2** ($\xi = 0$): this experiment provides a bottom-line comparison and measures how long the same empirical device would last without any accelerometer sensing.

**Exp-3,4,5** ($\xi = 0.05$): the accelerometer on the main empirical smartphone is operated according to the sensing intervals specified by $u^*_s$, $u^*_m$, and uniform sampling, respectively. A
second smartphone is used throughout these experiments which samples its accelerometer in every time slot to provide ground truth user state record. This second device is frequently charged to ensure long operating duration.

The user behavior learning and optimal policy calculation phases are not considered part of the empirical duration, such that the device battery lifetime is measured from the time the smartphone receives the optimal duty cycle instructions (and the battery is still fully charged) until the battery is completely drained while conducting energy efficient mobile sensing. The partially sampled user state sequences are reconstructed off-line, and are compared to the ground truth recorded by the second N95 device in order to obtain the estimation error. The empirical results are shown in table 5.3.

<table>
<thead>
<tr>
<th>ξ</th>
<th>Policy</th>
<th>Battery Lifetime</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp-1</td>
<td>1</td>
<td>Full</td>
<td>52 hours</td>
</tr>
<tr>
<td>exp-2</td>
<td>0</td>
<td>None</td>
<td>236 hours</td>
</tr>
<tr>
<td>exp-3</td>
<td>0.05</td>
<td>$u_s^*$</td>
<td>160 hours</td>
</tr>
<tr>
<td>exp-4</td>
<td>0.05</td>
<td>$u_m^*$</td>
<td>158 hours</td>
</tr>
<tr>
<td>exp-5</td>
<td>0.05</td>
<td>Uniform</td>
<td>150 hours</td>
</tr>
</tbody>
</table>

Table 5.3: Smartphone battery lifetime and state estimation error comparison among all experiments.

First, it is observed that by implementing energy efficient sensing policies with an appropriate energy budget, the smartphone lifetime could be extended significantly with tolerable estimation errors as compared to continuous sensing. The relatively low estimation error is due to the fact that the device is operated continuously to include night hours (i.e., significant idle durations), which is usually formed by a long sequence of consecutive “Stable” states and thus reduces the overall estimation error. Despite this fact, it can be seen from table 5.3 that implementing $u_s^*$ achieves the longest device battery lifetime among all three energy-saving policies and provides 27.8% and 48.6% gain on estimation accuracy over $u_m^*$ and uniform sampling respectively. This
is not only in line with our results in previous sections, but also validates that the semi-Markov optimal sensing policy is a desirable implementation choice in real systems.

5.5 Summary

Modeling real user state traces using Markovian model is often unrealistic and could lead to undesired state estimation performance, as real user state distributions tend to exhibit more unstructured distribution types.

In this chapter, we use semi-Markov process to model user state traces, and propose a semi-Markovian state estimation mechanism that can be executed in real-time in order to estimate the most likely user state while sensor observations are missing. A linear programming based constrained optimization is designed in order to find a tractable approximation to the best sensing policy for semi-Markov processes, which minimizes the expected state estimation error while maintaining an energy consumption budget. Both the state estimation and the approximate optimal policy have been evaluated on simulated as well as real user state traces and their performances have been shown to be better than Markovian estimation and Markov-optimal policy.

We also design and implement a client-server based, user-specific energy efficient mobile sensing system, and demonstrate through prototype implementation that semi-Markov optimal sensing achieves significantly better performance as compared to Markov-optimal sensing and uniform sensing.
Chapter 6

Conclusions and future work

Mobile devices such as smart phones are continually integrating more sophisticated sensing and networking capabilities. This makes mobile devices not only communication tools, but also powerful sensing and networking entities that can monitor and share user’s ambient context such as activity, location, and proximity information, both unobtrusively and in real time. The accurate detection of user context could provide information support to various kinds of mobile applications and services such as location based advertising, traffic information reporting, social networks, and health monitoring. However, due to the significant amount of sensor energy consumptions and the limited mobile battery capacity, there is a critical need for energy efficiency in most mobile sensing based applications.

In this thesis dissertation, we first developed and evaluated a novel design framework for an Energy Efficient Mobile Sensing System (EEMSS). We utilized the fact that not all the sensors on mobile smart phone need to be activated concurrently to recognize user state, and investigated how the sensors can be automatically turned on/off based upon real-time user status. By specifying sensor management scheme in a machine-understandable language (XML), hierarchical sensor management is achieved where only a minimum set of sensors is turned on at any given time,
and power hungry sensors are triggered by power efficient ones whenever necessary. We also
designed user motion and background sound classification algorithms that are light-weight, real-
time, but provide high classification accuracies. A prototype of EEMSS has been implemented
on an off-the-shelf smart phone, and the evaluation with 10 users over one week demonstrates
that the design is able to increase the device battery life by more than 75% as compared to the
state-of-the-art system, while maintaining high accuracy in recognizing user states.

We then further investigated how sensors should be duty cycled intelligently in order to
achieve the best trade-off between two important performance metrics: energy consumption and
user state estimation accuracy, under the assumption that the background user state is a Markov
process. We designed user state estimation algorithm and proposed computationally efficient
constrained optimization framework based on CMDP in order to solve for the Markov-optimal
sensing policy, which minimizes the expected user state estimation error while satisfying a given
energy budget, for Markovian user state processes. The Markov-optimal sensing policy is evalu-
ated on both simulated state processes and real user state traces, and it has been shown to outper-
form uniform sampling (which is a popular energy-saving mechanism) significantly.

Motivated by the fact that realistic data are seldom strictly Markovian, we developed a more
general analytical framework for state estimation as well as sensing policy optimization, for the
more generalized semi-Markov processes. Semi-Markov process model allows arbitrary state
duration distributions and therefore is more realistic as compared to Markov process model. We
demonstrated that the semi-Markovian state estimation algorithm is able to provide better per-
formance than Markovian estimation on different types of state duration distribution, as well as
on real data pertaining to mobile device’s network connectivity status. We then utilized linear
programming to obtain a tractable approximation to the semi-Markov optimal sensing policy,
which has also been shown to outperform the Markov-optimal sensing policy on simulated as well as real data traces. Lastly, we designed and implemented a novel client-server based mobile sensing system architecture which does user behavior learning, optimal policy calculation, energy efficient sensing, and state estimation automatically. We implemented a prototype of this system as an accelerometer based user activity recognition system on an off-the-shelf smartphone, and as a benchmark result, we demonstrate the performance benchmark of $u_s^*$ through a set of experiments, and show that it improves the estimation accuracy by $27.8\%$ and $48.6\%$ over Markov-optimal policy and uniform sampling, respectively.

Our research presented in this thesis dissertation provides guidelines for designing energy efficient mobile sensing systems in terms of how different sensors should be better cooperated and how each individual sensor could be optimally duty cycled to achieve the best trade-off between energy consumption and user state estimation accuracy. There exist many possible research directions that can succeed the content of this thesis dissertation. For example, how to effectively schedule multiple sensors that could potentially detect the same set of user states with different detection accuracies and energy costs? How to conduct policy optimization where different weights are associated with different user state detection error, i.e., some states may be more important than others? How to design an energy efficient mobile sensing system where not only sensor cost, but multiple types of energy consumption such as computation and data communication are taken into consideration? How to more efficiently learn the user behavior model in real-time instead of We believe that the design, analysis, and optimization frameworks introduced in this thesis dissertation could serve as significant building blocks for all these kinds of future study in energy efficient mobile sensing.
References


