Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3207

Efficient Energy-Constrained Distribution of Context in Mobile Systems

Florian Berg

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dipl.-Inf. Stefan Föll
begonnen am:	20. Juni 2011
beendet am:	20. Dezember 2011
CR-Klassifikation:	G.1.6, C.2.2

Abstract

The recent improvements in smartphones nowadays offer a widespread application of sensor-based services. Each mobile phone is equipped with several sensors like a GPS module, a gyroscope, or a high-resolution camera. As a result of this sensor integration, a whole new way of usage is opened up for the end-user, like a location-based search or people-centric sensing. The main drawback related to a smartphone is an overall high energy consumption, combined with a limited energy capacity. Due to this fact, a continuous and fine grained sensing of the user's context is not possible, as it utilizes at least one acceleration sensor. Furthermore, the captured data is transmitted via a (mobile) communication infrastructure to post the context on the Internet. Both drain the battery very quickly. For that reason, an efficient energy-constrained distribution is required to minimize the update occurrence of a producer, while simultaneously maximizing the accuracy of a consumer. The primary issues to be addressed include a modeling of user behavior as well as a determination of optimal points in time for an update. Therefore, a probabilistic approach is used to forecast the user's context pattern. The prediction is based upon a Markov chain and enables the extraction of meaningful information. The proper times for an update are determined with the help of a constrained optimization problem. Different methods from mathematical optimization are applied like linear and nonlinear programming or a constrained Markov decision process, which obtain an update policy. For a better comparison of the weaknesses and strengths related to the developed methods, dynamic programming is used to achieve the optimal points in time for an update. The evaluation upon a real trace shows that an accuracy gain of more than 30% is achieved by sending the equal amount of messages.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

> We're going to make some history together today.
>
> *(Steve Jobs)*

The introduction of Apple's iPhone 1 at Macworld 2007 started off with the keynote speech by Steven "Steve" Jobs who began with "We're going to make some history together today." [APP]. This presentation of the first high-end mobile phone (smartphone) marked the beginning of significant changes in everyday life. The main differences of a smartphone compared to a former "common" mobile phone are the usage of an expandable operating system combined with highly-developed computing power and connectivity. These enable the smartphone to act as a daily aid and companion of many people. Indeed, there were a lot of business companies like Nokia[1] or RIM[2] selling successfully mobile phones for years. But the major use of their devices was to write a Short Message Service (SMS) or to call another person. However, great technical improvements to mobile network and devices, the flexibility and handiness of a smartphone, and a wide range of applications computerized the human life. Through this, the popularity of smartphones raised vastly and altered the use of personal computers or laptops over the last four years.

1.1 Motivation

The technology improvements since the iPhone introduction in 2007 equip the today's smartphone with powerful monitoring and networking capacities, to sense, share, and query personalized information of the physical world in real-time. Here, the most common sensors are a Global Positioning System (GPS) to identify its own position in the world, an accelerometer or gyroscope to determine the three dimensional position, a compass, and a camera. The communication modules are well-known from the prior mobile phones like Global System for Mobile Communications (GSM), General Packet Radio Service (GPRS), or Bluetooth, on the one hand and the wireless communication standards from the laptop like Wireless Fidelity (WiFi), on the other hand. Therefore, the sensors embedded on this device extract meaningful characteristics of the user and environment, upon which "mobile" applications provide rich contextual information and adaptation. This constitutes a main feature related to the great popularity of a smartphone, because everybody with programming skills can implement an application, which processes the raw sensor data to detect the user's environmental conditions like traffic conditions or the user's activity, such as "Driving". The appropriate input data is not only limited to the information sensed on the own device, but also includes the data of quite a lot of other devices over the Internet.

[1] http://www.nokia.de/
[2] http://www.rim.com/

Hence, the context can automatically be recognized and transmitted via the mobile infrastructure for various mobile applications and services, such as social networks, personal health care systems, or blogs. For example, the sensing units of a smartphone extract the meaningful properties of the end-user and surroundings in real-time, which are mapped and rendered to a physical human activity in a virtual world displayed by an avatar.

Through the mobile characteristic, a major issue related to smartphones becomes the energy consumption as each hardware part, like the above-mentioned sensors and modules, spends energy from the limited battery capacity to fulfill its assigned tasks. Mainly as the most applications utilize more than one unit, which drains the battery very quickly. For example, a location-aware guidance, which entails a continuously detection and distribution of real-time user localization (e.g. "Home", "University", or "Library"), uses at least the GPS sensor and one communication module. The GPS sensor on a Nokia N95 phone consumes alone basically about 400mW. With a sensor read and data write operation the energy consumption increases about 250mW on average. So the device lifetime is just nearly 10 hours by spending the whole battery of N95 for the localization (without any distribution) [CGS+10]. Obviously, some basic functions of a mobile phone like a call or a SMS use energy additionally and should also run in cases of emergency. Correspondingly, all applications running on the mobile device have to consider energy as a finite resource involving energy-efficient approaches at design. Moreover, the financial aspect causes a further issue for an energy-efficient approach, because the operating company of a web service does not require the additional capacity for the messages that are saved and therefore not sent. Note that the traffic generated by a continuously and fine-grained tracking of the user's context is not to be taken likely and can limit a widespread deployment. Consequently, energy efficient approaches play a decisive role and become increasingly popular in research interests.

Since the different applications running on a smartphone have to share the same energy source, only a small fraction of the total battery capacity is available for an application, called the energy budget. This restricts the continuous functioning and sending of all messages, whereby an application must select the best points in time for an operation considering the limited energy. The definition of "best" together with "operation" belongs to the application functionality and can not be answered universally. In general, it serves as an optimization criterion together with the restricted energy.

1.2 Purpose of Study

In connection with this thesis, the question of an efficient energy-constrained distribution of context in mobile systems is answered based on the popular producer/consumer model to guarantee a widespread adaption of the proposed update techniques. For that reason, the term "context" is not specified definitely to constitute for each application the required information like an activity and/or a location. The basic concept can best be explained with an example: In a social network, the user's mobile phone (producer) publishes automatically the current context, whereupon friends (consumer) follow these posts to be up-to-date in their social environment. The energy conservations on the producer's smartphone are associated with the reduction of updates at the expense of the consumer's accuracy. As a consequence, the trade-off between the context accuracy and the energy consumption is required in order to achieve an efficient update method.

So that the problem-solving approach can update the contextual information each time, which has the biggest impact on the accuracy, it has to predict the future behavior of the end-user. The prediction is based upon context patterns, which arise from analyzing the passed through context sequence. For example, an energy-efficient transmission of personal context regards the corresponding dwell time to put the sensor applied in the non-operating state and save energy. Once the client enters into a context with a long duration of stay (e.g. a lecture at the university), the update of the corresponding context must only occur at the beginning and advances the accuracy for a long time. However, sequences of fast context changes need a very high energy consumption to accomplish a certain accuracy. To reduce the energy consumption and simultaneously maintain a desired accuracy, the aim is to identify the possible points in time for a context update with the highest accuracy gain. For this purpose, the user behavior is modeled with a probabilistic approach to forecast the manner of the end-user. Thereby, the above mentioned dwell time of a context is characterized (e.g. the activity "Attending a Lecture" is sensed successively ten times) and the context with the most future impact on the accuracy is identified. In this regard, the user behavior is modeled with the aid of a Markov chain and forms the basis for the determination of the (optimal) points in time.

The second part of the problem-solving approach establishes a mathematical description of the related problem to have a framework for simplification and optimization. Here, a constrained optimization problem provides the best approach and includes techniques for solving. The applied optimization methods have a big influence on the quality of the energy savings and the corresponding accuracy gain, because of the different advantages and disadvantages. In this context, three approaches are developed and discussed, which decide immediately at the context recognition to make an update or not. Two of them belong to mathematical optimization, namely linear and nonlinear programming. The third, a constrained Markov decision process, is evolved, which constitutes a control process and is widely used to solve optimization problems. Furthermore, another subtopic of mathematical optimization, namely dynamic programming, is applied to detect the optimal moments for an update and functions as a comparison of the suitability related to the three optimization techniques. Dynamic programming is applied after the simulation with the "a priori" knowledge of the whole passed through context sequence and achieves the optimal solution.

Summarizing, this thesis reveals the interesting question of an efficient distribution related to the maximal achievable accuracy under an imposed energy budget and vice versa. The design, implementation and evaluation of several techniques to optimize the information uploading process on mobile phones are presented and evaluated. Therefore, the energy-accuracy trade-off is quantized for two scenarios based on a simulated and a real trace.

1.3 Outline

The remainder of this thesis is organized as follows: The subsequent chapter describes the underlying system model with the basic components and interactions giving background information. Therefore, the relevant part of the world is abstracted and characterized with different views, like the infrastructure view, to identify the mathematical relation and formulation. Chapter 3 classifies the thesis in the field of the intensive research topic "energy-efficient methods for mobile phones" and summarizes related approaches.

The defined mathematical formulation forms the basis for the proposed problem-solving approach in Chapter 4, which constitutes the main part of the thesis. First, an overview of the required steps is presented including the user behavior model to forecast the accurate points in time for an update. Further, the above mentioned optimization techniques are presented to determine the probabilistic values of an update policy.

Since each technique is attended by assets and/or drawbacks, Chapter 6 evaluates the developed optimization methods based on a prototypical implementation, which is introduced in Chapter 5. The quantification of the trade-off between the utilization of the energy budget and the achieved context accuracy is compared and discussed on two scenarios, evolving in the best approach for the constrained optimization problem.

To conclude the thesis, Chapter 7 summarizes the whole work and points out the limitations. Finally, an outlook of future work is presented.

2 Background

From the introduction of Apple's iPhone 1 in 2007 until June 2011, the company sold about 130 million smartphones with their operating system iOS. A few competitors, like the most successful one HTC (with the operating system Android from Google), has stimulated business. The first-generation smartphones are notable for their new multi-touch screen combined with a virtual keyboard for manual input. Another distinction from a previous "common" mobile phone is the focus on multimedia and the Internet. Therefore, many technical changes to the phone itself and improvements to the mobile Internet infrastructure have taken place. On the one hand, the memory capacity, the electrical energy storage, and the Internet speed have increased steadily, as well as the number of embedded sensors. For instance, today's smartphones are equipped with a GPS sensor for positioning, a gyroscope for a three-dimensional detection of position, a high-resolution camera to observe graphically the surroundings, etc. Additionally, modules like WiFi, Bluetooth, or High Speed Downlink Packet Access (HSDPA)/High Speed Uplink Packet Access (HSUPA)[1] enable fast wireless communication [APP].

This variety of technical hardware opens up capabilities for many new applications. With the aid of sensors, the user's real world situation is observed, upon which environmental settings are executed automatically. The user can choose a location- and activity-specific preference for his/her mobile phone settings, like the display brightness or the call signal volume. However, the possibilities are not limited to the device itself. The mobile communication modules allow one to check the Internet as well as to post something on it. This can involve the user posting automatically his/her actual position, for example, or uploading photos to an on-line community. However, an accurate interpretation of the raw sensor values combined with the capabilities of the Internet open up many application areas for a smartphone, which are much more complex. Based on the monitoring of the directly-measurable acceleration, the user's activity like "Running", "Attending a Lecture", or "Meeting with Friends" is sensed. Nevertheless, despite the usefulness, the wealth of sensors and applications, the main drawback is the overall energy consumption that is still limited. The techniques for activity recognition, the sensors themselves, or the wireless communication, drain the battery very quickly in addition to the basic functions of a simple mobile phone. Without recharging a smartphone, the energy capacity for continuous sensing and transferring of information is limited to hours. Therefore, a corresponding energy management together with energy-efficient approaches are becoming increasingly important

Associated with this thesis, the focus is on an efficient distribution of contextual information from a mobile phone within a certain energy limit, as for instance the addition of contextual user information to a social network or virtual world. In order to address the finite resource energy, it must be considered at design pattern to provide a general reusable solution. Thus, the following chapter describes the functions of and the connections between the components involved (producer/consumer model), the

[1] Theoretical downlink speed up to 13.98 Mbit/s and uplink speed up to 5.8 Mbit/s

underlying (technical) infrastructure, and the meaning of the rather general context model. Furthermore, the definition of the problem with the mathematical description follows, which serves as a basis for the problem-solving approach developed in Chapter 4.

2.1 System Model

The difficulty of restricted energy limits to the bare essentials related to the applied algorithms and protocols. Thus, a good strategy for the system design is the use of a system model to concentrate on the basic components and their interactions. The purpose is to provide all relevant assumptions, generalizations, and abstractions for the problem-solving approach. Especially as informational systems only refer to a partial and abstracted section of the real world. Therefore, a transformation from reality to digital form occurs, which constitutes an abstract, simplified but consistent overview of the basic system with various views and corresponding descriptions. The abstraction to only the essential reflection of reality admits the concentration on the important details and prevents irrelevancy. Upon this, a model is developed with the essential components and functions, which provide a basis for the operating principles and the informational system [HA04]. Each following subsection describes the model from different perspectives, namely an **application view**, an **infrastructure view** and a **data flow diagram**.

2.1.1 Application View

The **application view** explains the components involved from the user's standpoint and constitutes a structured-system analysis. Hence, Figure 2.1 illustrates the underlying system with the elemental components, namely a *producer*, a *consumer*, an *interconnection*, and a *context recognition*.

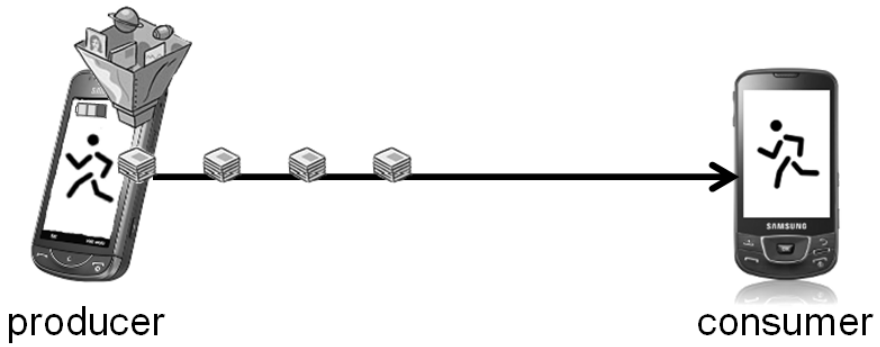


Figure 2.1: The application view of the system model

A main functionality of the system depicts the *context recognition*, which is illustrated in the figure as a funnel. Figuratively, it takes the producer's "real" contexts, like the activity "Playing Soccer" or the location "University", at discrete times as input and converts them in digital form. For this, the raw sensor values are read and perhaps interpreted with additional information resulting in the

current context. For example, additional facts are the previous sensed context, producer's vicinity, calendar entry, or persons to whom the producer is closest. However, Khan et al. [KLLK10] presents a detection function for 15 user's activities with an accuracy of 97.9 %, which only relies on the values from a single triaxial accelerometer. To fix the attention linked to this thesis on the efficient energy-constrained distribution of context, the recognition algorithm applied is assumed perfectly and observes the true user context at every discrete point in time. Moreover, as the proposed upload techniques are independent from the underlying algorithm of context observation, the problem-solving approach does not take any consideration towards the energy consumption of the recognition. Both methods consider a stand-alone problem, which can be deployed together to achieve even better results. The techniques for an energy-efficient and reliable context recognition is an actual field of research, which is elaborated by [LCB+05] or [WKZA10] for example.

For reasons of abstraction and mathematical formulation, the applied context model consists of discrete states, which represent the user's contextual information at a certain point of time. This is also true for the context description related to humans, which rather characterize their activities discrete instead of continuous. For example, nobody says "I am half walking and half running". In some cases, continuous information like location occurs, which can be transformed into discrete space through an equidistant sampling. Subsequently, the context is assumed to be discrete, such as the above mentioned human activities "Running", "Standing", or "Going", and sensed with a discrete temporal characteristic. Note that a distribution protocol related to a practical application only transmits a few bytes to represent the corresponding context (e.g. location or activity) in order to save energy.

The recognition algorithm runs on the *producer*, which is exemplified as the left smartphone in Figure 2.1. It is a mobile device equipped with sensors and a restricted energy capacity. With this quantity all the different applications running on the phone, such as the main function call or new apps[2], must handle, before it is reloaded. Moreover, the installed applications shall not reduce the battery too rapidly, so that after one or two hours the device must be recharged. A common situation of single-sided energy consumption is the urgent use of the mobile phone, every time the battery is empty. Therefore, the "energy management" gets very important related to the application development. Since the main focus of this thesis is on the context distribution and the concomitant energy consumption, the available energy capacity of the producer is only consumed for message broadcasting. It is not influenced by the usage related to the standard functionality of the device as well as the context recognition. Furthermore, the detection software is assumed to always deliver the right context, so that the producer is in each point in time in an well-defined state, namely the last recognized context.

However, the limitations of the *consumer* are not so extensive, because the main functionality of the second device constitutes the usage of a communication endpoint, which is interested in the producer's contextual updates. For example, a social network affords the capability to follow the activity changes of an end-user (the producer) throughout the day. Therefore, a user's friend (the consumer) registers and receives the corresponding context publications. Hence, the smartphone of the producer recognizes the own current activity and publishes the context to the other (mobile) device, at which the consumer stays up-to-date about its social environment. The consumer can be a stationary computer or a mobile

[2] An app is a software application, which is specifically developed for a smartphone or tablet computer. It is distributed trough a platform like the Apple App Store or the Android Market (cf. http://www.apple.com/itunes/ and https://market.android.com/).

device being at every point of time in a defined state, which corresponds to the last updated state of the producer. The major difference between the producer and consumer is the unlimited energy to concentrate on the development of an efficient distribution of the producer. The energy consumption related to the consumer is not optimized and further observed.

Both devices are linked with a robust and reliable *interconnection*, which serves as an ideal message transportation between the producer and consumer. On the one hand, the communication channel guarantees, that the connectivity between both endpoints is always-on, that every message reaches its destination complete, uncorrupted, and in the right order, etc. On the other hand, the problem-solving approach is based on perfect properties like real-time or constant transmission costs, which are also fulfilled. Furthermore, the application view (compare Figure 2.1) assumes a logical one-to-one connection between the producer and consumer, which enables the specification of single communication properties, such as a distinct energy budget and/or accuracy.

2.1.2 Infrastructure View

The **infrastructure view** of the system model gives an overview of the interconnection (compare Figure 2.2), which is only schematically illustrated in Figure 2.1 to limit it to the essential information. The following subsection has a closer look at the message distribution together with the involved parts to avoid a "real" one-to-one connection between the producer and consumer.

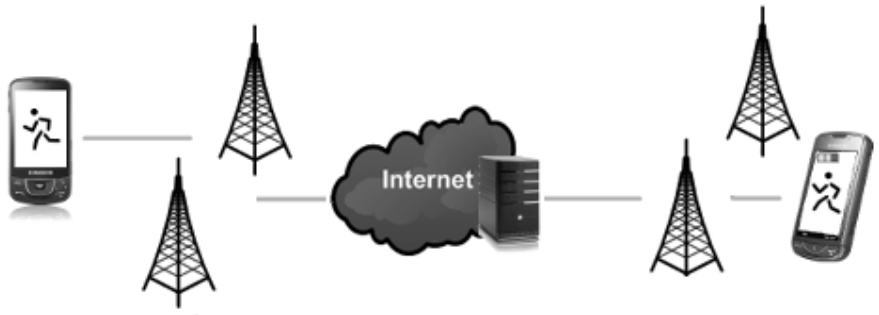


Figure 2.2: The infrastructure view of the system model

For this reason, there is a (back-end) server between the two mobile devices, which receives the messages from the producer and sends them to the consumer. Figure 2.2 displays this circumstance and depicts the location of the server in the Internet, whereas the producer and consumer are connected with it over the mobile Internet. The associated energy cost for a message distribution refers only to the section between the producer and server and is assumed to be always constant. Regardless of whether the underlying physical communication channel is based on the infrastructure of a cellular network operator (e.g. GSM or UMTS[3]), the usage of WiFi or a cabled communication. Afterwards, the server distributes the messages to the mobile or stationary device of the consumer over a wireless or cabled connection, whereby the required energy consumption is neglected. Summarizing, the server

[3]Universal Mobile Telecommunications System (UMTS)

is a distribution interstage between the producer and consumer with some additional functions, for example a login or administration of users. The communication, the cost, or each server functionality are not further analyzed in detail, because it is beyond the scope of this study to describe, for instance, the handover of a mobile transceiver with an ongoing continuous communication moving from cell to cell. Mishra provides in [Mis04] a good overview related to cellular networks.

2.1.3 Data Flow Diagram

As the main concern of this thesis is the development of an approach for an energy-efficient distribution of context, a description of the data flow across the system boundaries follows. On that account, the **data flow diagram** in Figure 2.3 outlines the operations, the passed through stages and the overall idea of the approach. Moreover, it provides a decomposition into sub-systems and the transaction from the elementary input to the desirable output. Therefore, the whole data treatment and flow are visualized, whereby an elliptical shape represents a process and a square corresponds to data. The view contains each single step from reading the raw sensor values to the consumer's receiving of the sensed context. The intermediate steps are a context recognition based on the raw sensor values, a decision maker, which determines either to update the current context or not, and the message transmission.

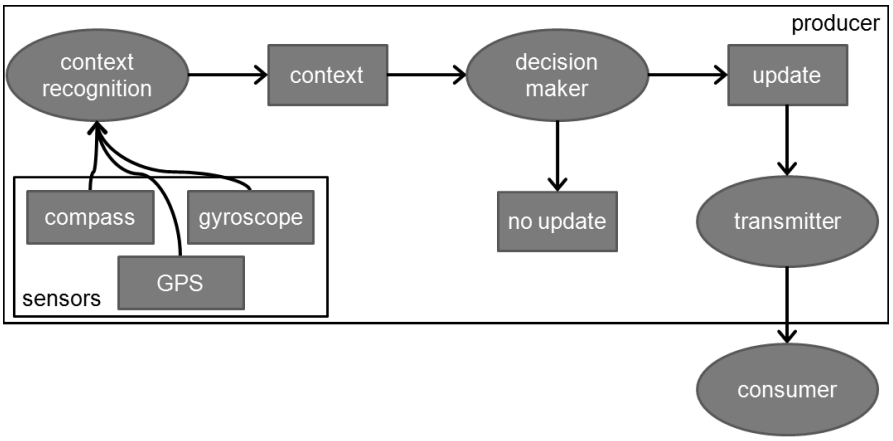


Figure 2.3: The data flow diagram of the system model

The *sensors* of a smartphone interact with the outside and capture the real world at discrete time. Upon this, the *context recognition* transforms the obtainable information to a corresponding context, which serves as the input data for the *decision maker*. Considering the imposed energy budget and the required accuracy, it determines with the help of the current context and perhaps further information, whether to send a message or not. The available information at the decision point are limited to the passed through sequence as well as the current context, whereupon the determination takes place and influences the achievable accuracy. If an update is triggered, which decreases the available energy budget about the update cost, a message with the current context is sent to the consumer and refreshes its current state. Instead of that, the energy budget is unaltered and the consumer remains in the last updated context of the producer, which can still rise the accuracy. Hence, the decision maker has to choose the possible updates with care, since it is possible that the accuracy gain of a transmission is not as high as the

increase without an update. As a result, the main impact on the two optimization criteria, the energy consumption together with the achieved accuracy, is due to the decision maker, which determines the points in time for an update and therefore assigns the consumer's context.

2.1.4 Summary

Each view described typifies the essential characteristics and components of the system model and contains all information for a better understanding. The application view describes the difficulties on a superior level with a strong abstraction, whereas the infrastructure view deals with the elementary components and interactions. The data flow diagram, however, illustrates the system from the data viewpoint and completes the comprehensive description. As the main topic of this thesis is the efficient energy-constrained distribution of context in a mobile system, several optimization methods are considered and compared related to the associated update costs as well as the impact on the accuracy. The transmission of the monitored context takes place on a mobile device, called producer, to another (mobile/stationary) device, called consumer. The producer has the following main properties:

- finite energy capacity (battery), that is drained by an update
- context recognition software, which provides in each point in time the correct context
- decision maker: determine whether to update or not
- discrete context model and discrete time behavior
- connection to a consumer (logical one-to-one communication)

The consumer is interested in the producer's contextual information receiving the corresponding updates and is characterized by:

- infinite energy capacity: no optimization
- communication endpoint
- wants to stay in as many equal contexts as the producer (high accuracy)

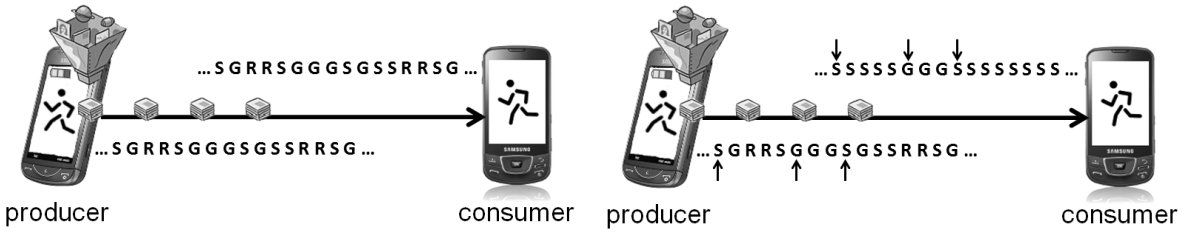
The interconnection between the producer and consumer has the subsequent properties:

- constant cost for an update
- reliable and robust
- no loss of messages or connectivity

The made assumptions in this section limits the field of application, wherefore Chapter 7.2 and 7.3 have a closer look on the limitations and future work. For instance, one producer can publish its context to many consumers with different required accuracies. Moreover, a producer itself is interested in the contextual information of another producer, wherefore not only the energy cost for an update, but also the receiving costs must be considered. Nevertheless, upon this system model a problem-solving approach for a wide range of energy-efficient context distribution is developed in Chapter 4.

2.2 Problem Definition

The main parts related to the context distribution are the producer's limited energy and the consumer's required accuracy. Assuming that the producer has an unlimited energy, it sends every context change to the consumer. The continuous updating entails, that the consumer knows in each point of time the producer's current context and consequently has the maximal achievable accuracy. This circumstance is illustrated in Figure 2.4a, whereat the context sequence *SGRRSGGGSGSSRRSG* with $S =$ "Standing", $G =$ "Going" and $R =$ "Running" is recognized from the producer (corresponding to the sequence below the connection arrow) and updated to the consumer (according to the sequence above).



(a) Unlimited energy capacity

(b) Limited energy capacity

Figure 2.4: An example of context distribution with an unlimited and limited energy budget

Through the mobile characteristic of the system model, which includes a limited energy budget of the producer, the distribution of all visited contextual information is not possible. Therefore, a resource-efficient approach is applied, which combines each context update with a significant energy consumption characterizing the energy-expensive nature of wireless communication. Thus, the approach guarantees a sufficient application lifetime and broadcasts within the predefined energy budget the context with the most increasing impact on the accuracy. Considering the context sequence of the example above, the producer sends only three messages (depicted as an upwards arrow in Figure 2.4b). As a result, the consumer receives the context *S*, *G* and *S*, whereupon nine of sixteen contexts are equal (*SxxxSGGGSxSSxxSx* with "x" meaning a failure). To select the "proper" time for an update, the decision maker (compare Figure 2.3) requires a formal problem description to assign the optimal points in time. According to the energy limitation and the maximal accuracy, this is a constrained optimization problem, whereby the mathematical definition follows.

The set of all feasible states referring to the producer and consumer is the context space S with the discrete human context $s \in S$. For example, $S = \{R, G\}$ consists of the two activities *R=Running* or *G=Going*. At each discrete point in time $t \in T = \{0, 1, \ldots\}$ the producer s_t^p is in a well-defined state, which is set to the recognized context at time t. So the term "state" related to this thesis represents an user context from the space S evolving over time t. Due to the costs for an update C and the limited energy budget E, the producer can not transfer every context change and must reduce the number of updates to meet E. Therefore, the goal of the producer is to find the optimal set of context updates $\Psi \subseteq T$ to achieve the maximal accuracy considering a given E. In this context, one message transmission causes the cost of one ($C = 1$), which denotes a simplification against the real world. For instance, the immediate transmission of several messages in a short time interval (e.g. a data burst) is not so energy consumptive as the transmission within long periods. Musolesi et al. consider in

[MPF+10] that the corresponding network interface of a Nokia N95 consumes for less than 5 seconds still some energy after the transmission of 100 bytes. This period is not standardized and varies for different devices. Hence, a problem-solving approach with a temporal factor for a time-dependent cost can improve the results and constitutes a future work.

The energy budget E is defined as the ratio of the available updates for the distribution, Φ, against the entire required messages to achieve the total equality of the producer and consumer for a passed through context sequence, Π. For example, the end-user determines the overall runtime of the context distribution of $\Pi = 1000$ context changes[4] and allocates to the application $\Phi = 100$ update messages of the total energy capacity. Therefore, the energy budget E results in 0.1 as $E = \frac{\Phi}{\Pi} = \frac{100}{1000}$. To transmit the complete context sequence resulting in the maximal accuracy, the energy budget has to be set to the sum of the occurred context shifts, $E = \frac{\Phi}{\Pi} = \frac{CS}{CS} = 1.0$. The broadcast of more than CS messages expresses a energy dissipation and does not increase accuracy. Consequently, $0.0 \leq E \leq 1.0$ holds:

$$CS = \sum_t f(s_t, s_{t+1}) \tag{2.1}$$

with

$$f(s_t, s_{t+1}) = \begin{cases} 1, \text{ if } s_t \neq s_{t+1} \\ 0, \text{ else} \end{cases}$$

So the problem-solving approach has to identify the not transmitted context changes ($\Pi - \Phi$), which have no big influence on the accuracy and minimize the loss of quality at the consumer.

Just like the producer, the consumer is at each time $t \in T = \{0, 1, \ldots\}$ in a well-defined state s_t^c, which is determined as the last updated context change of the producer. Since the consumer expects its current context as the last updated until a new one is received, there is a problem, when the first update from the producer does not occur at $t = 0$. In that case, the consumer would be in an undefined state, which is by definition not allowed. In avoidance of this, it is defined, that the producer always transfers its initial context, wherefore the first point of time $t = 0$ is added to the update set Ψ, $0 \in \Psi$. In practice, it is absolutely reasonable, as for instance the producer must log in at a server to get the consumer's IP-address. This yields to $s_t^c = s_{t'}^c$ with $t' = max\{\psi \in \Psi | \psi \leq t\}$ and the formal description of the accuracy A with $0.0 \leq A \leq 1.0$, which conforms to the ratio of the equal context between the producer and consumer related to t against the entire passed through context sequence Λ:

$$A = \frac{\sum_t g(s_t^p, s_t^c)}{\Lambda} \tag{2.2}$$

with

$$g(s_t^p, s_t^c) = \begin{cases} 1, \text{ if } s_t^p = s_t^c \\ 0, \text{ else} \end{cases}$$

[4] The entire length of the passed through context sequence Λ emerges only at the application end, since the allocated energy budget is defined subject to the number of context changes.

3 Related Work

In recent years the energy limits of mobile devices have become increasingly popular for researchers, as can be seen in the publications on energy-efficient context recognition, sensing, localization, or uploading. Regarding this, the following chapter addresses the above-named assumptions of the system model as well as associated issues. The main concepts of different papers are presented and discussed.

3.1 Recognition

The fine-grained recognition of the contextual information (e.g. activity, location) is an important part of sensing applications and services on sensor-embedded mobile phones, upon which the high-level social interactions depend. [LCB+05] and [LCK+05] delve into activity recognition and have developed a very small prototype of a "wearable multi-modal sensing device" (2.53 square inches) with a classification algorithm. The sensing unit has a wide range of application areas, like (long-term) health-care monitoring, and can be worn on one's body or integrated into a mobile device. Here, the main advantage is the automated data capture, which entails lightweight, richly detailed, cheap, and low error-prone technology compared with manual methods.

The sensor platform recognizes with about 95% accuracy ten basic activities, namely sitting, standing, walking, jogging, walking up or down stairs, riding a bicycle, driving in a car, and riding an elevator down or up. The equipped sensors are audio, 3-axis acceleration, barometric pressure, temperature, humidity, compass, IR/visible light and high-frequency light. Furthermore, modules for a wired link or wireless data transmission are embedded. Not only the hardware, but also the applied algorithm, is a key role for the fine-grained context recognition. For that reason, [LCK+05] presents a hybrid approach to identify natural human activities from the raw sensor values. It relies on a generative and a discriminative technique, which are well-known from classification problems in machine learning.

The device and algorithm that have been developed, have a battery lifetime of about 12 hours and are currently being employed in an experiment at the University of Washington. The data collection recorded shows promising results related to activity recognition over a long time and reveals the capabilities of on-board sensor devices carried by humans. As smartphones are equipped with several sensors these days, the demand for an additional sensing unit becomes redundant. Moreover, the high computational power and sufficient disk space together with a fast radio communication introduce a variety of contextual applications and services. Hence, the recognition approach described must be adapted to the characteristics of a mobile phone, which involves further issues than a single activity recognition. For example, Lester et al. [LCB+05, LCK+05] do not consider the limited energy capacity of their device, which has a bearing on the acceptance and functionality of today's application on a smartphone.

3.2 Sensing

A main source of energy consumption is the several sensors on today's mobile phones, which drain the limited energy capacity very fast. To guarantee an advantageous long "continuous" context detection without recharging, there is the deactivation of sensors for a certain period to increase the device lifetime. Accompanied by the sensor's idle time, no observation happens and the current user context is estimated from the available information. In this context, [WKZA09] recommends different methods to perform the context estimation for the time slots with no sensing.

By assuming that the embedded sensors automatically and correctly recognize the user's context, the major challenge is to determine the sensing points in time and the associated idle or duty cycles. Here, the waiting time between two observations is based on the context's dwell time. For example, if the state *"Attending a Lecture"* is monitored, the associated sensor's idle period corresponds to *"*90 *minutes"*. During this time the microphone and GPS stay inactive and consume no energy. To maintain a high accuracy, the timing of observations is very important, because uncertainty about the last set of events increases. As a result, Wang et al. concentrate on the estimation of unknown context upon available data and model the user behavior as a discrete time Markov chain (DTMC). By means of the transition probabilities from the Markov chain, which implies the context average duration, assumptions of the future behavior are made. A "stationary deterministic sensor sampling policy" ([WKZA09]) allocates fixed duty cycles to the sensors. The estimation uses a well-known Hidden Markov Model (HMM) and proceeds via two different methods:

- The **first method** is based on the "Forward-Backward Algorithm" and chooses for each time t a context with the highest probability between two subsequent observation points. The corresponding overall expected estimation error is calculated, which is the "ratio of expected number of estimation errors per observation interval and expected observation interval length" ([WKZA09]).
- The **second method** is based on the "Viterbi Algorithm" and requires only a single observation, whereupon a context sequence with the highest probability is selected. The whole sequence is wrong, even if only a single context within it is incorrect. The appropriate expected estimation error is calculated.

The main difference between method one and two is the estimation goal (context or rather context sequence) as well as the required observations (two or rather one). Both methods determine the expected estimation error and the expected energy consumption. Additionally, the work gives an overview of the trade-off between the transition probability matrix and the sensor duty cycle parameters as well as the trade-off between the expected energy consumption and the expected estimation error. Furthermore, a case study with a two-state DTMC with different transition probabilities combined with different dwell times is presented and discussed.

Summarizing, the work shows the large influence on the estimation accuracy of energy consumption and delivers for each context the optimal interval lengths of sensor's idle time, which minimize the estimation error. Therefore, two different methods are developed, which estimate during the sensor's idle time the passed through context(s). The estimation is based on a discrete time Markov chain, to model the transitions of user context. The focus of Wang et al. is the energy-efficient determination of optimal sensing times associated with the user's context estimation, which has no observation data

available. In respect of this thesis, the approach can be used for the energy-efficient sensing assumption made in the system model (compare Chapter 2.1), but does not give an answer to the optimal update times for an efficient energy-constrained distribution of context.

3.3 Localization

Due to the extensive use of localization in conjunction with applications and services on a mobile phone, the energy-efficient "continuous" localization is an interesting field of research. Obviously, the Global Positioning System is required to localize the position of the mobile phone all over the earth delivering the position with an inaccuracy of just a few meters. But the main drawback is the high energy cost, which discharges a complete loaded battery on a smartphone in just a few hours. There are two more alternatives with significant differences on a mobile phone to determine its position, namely Wi-Fi or GSM, which consume less energy at the expense of the location accuracy. Constandache et al. outline in their work ([CGS+10]) the energy-accuracy characteristic for a Nokia N95, which identifies its position with GPS, Wifi, or GSM alone:

- **GPS** has the highest energy consumption combined with an almost exact localization. With a full battery, the N95 has an operating time of 10 hours determining continuously its location based on GPS, which offers an error of around 10 meters under clear-sky and outdoor conditions. Therefore, the GPS readings are assumed to be ground truth in [CGS+10].
- **WiFi** localizes the device's position with an entire battery capacity for a duration of 40 hours combined with an erroneous accuracy of nearly 40 meters. However, the region, where the positioning happens, must be initially war-driven to allocate each access point its position.
- **GSM** requires the lowest power consumption to the account of the highest accuracy error. The corresponding operating time totals with a full charged battery to 60 hours and an accuracy error of about 400 meters. The accuracy and energy consumption depend extensively on the location, because of the available cell size and number of GSM providers.

Since the limited energy is a critical bottleneck for location capturing, [CGS+10] deals with an extension of battery-life conjoined with a minimization of the accuracy error. On these grounds, the proposed approach uses an optimal combination of GPS, WiFi, and GSM readings as well as a location prediction. As a result, it determines a time schedule, which triggers at certain points in time a sensor reading to minimize the localization error. Furthermore, a profiling of the user's mobility takes place to predict effectively the location and save further energy. Two different prediction methods are compared, namely a *Simple Linear Predictor* and a *Human Mobility Pattern*. The first one is a general technique, which extrapolates the human movements linear between two previous sampled locations and is, for example, efficiently on a straight road. The other prediction opportunity captures the individual behavior and derives mobility patterns. Thereupon, a few sensor readings at critical points suffice to predict the entire activity. The method overcomes unhabitual movements with a prediction based on an analysis of large populations. Whenever a divergence occurs, the statistical values of a generated "probability map" intervenes and predicts the localization reasonably. The schedule and prediction are used as a simple on-line heuristic on a mobile phone.

To improve comparison between each combination, Constandache et al. formulate the issue as a mathematical optimization problem. With the aid of dynamic programming, the maximal achievable accuracy is calculated under a given energy budget and mobility trace. Upon this, the on-line and the optimal off-line schedule are compared to determine the timing and amount of GPS, WiFi, and GSM readings. Furthermore, the paper presents an energy-efficient localization framework, "EnLoc", and evaluates it with realistic user traces, whereby the energy-accuracy trade-off is quantified measuring the power consumption. For example, a full charged Nokia N95 consumes for one day 25% of its battery capacity for localization and achieves an accuracy of about 25 meters.

The focus of [CGS+10] are the location-based applications and services on a mobile phone, which use different techniques to determine its position. The main drawback of pure GPS utilization is the limited energy, wherefore positioning methods based on WiFi or GSM come to the fore. As compared to GPS, both consume less energy, but cause higher erroneous location readings. Additionally, a prediction method affords an improvement and forecasts the user's behavior to save further readings. So the paper answers the question of localization accuracy under a given energy budget and presents an interesting framework. However, it does not consider an energy-efficient distribution of general context, as it incorporates the semantic information of location and the temporal user behavior to determine the optimal times for an update. As a result, it constitutes an efficient approach for the specific localization problem of mobile phones, but does not purpose an energy-efficient uploading strategy.

3.4 Uploading

Beside the important challenges of sensing and localization related to the restricted resources of mobile devices, a further issue is the continuous context distribution in real-time between a sensor-embedded mobile phone and a back-end server. Not only the gathering applications, but also the connection towards another device (e.g. a back-end server or a smartphone) imposes significant challenges to the system designer.

The uploading of sensed high-level information raises farther cost and has a bearing on the energy constraint. Just as the previous energy-efficient approaches presented, the usage of an intelligent uploading strategy is an obvious extension of the phone's battery lifetime. Therefore, Musolesi et al. present an approach ([MPF+10]), which analyzes the passed through context to transfer only the essential information and reduce the data transmission. As a consequence, various techniques determine on the phone-side the context with the highest impact on accuracy, whereas the back-end server has to deal with missing information. All the results related to the optimized uploading and prediction methods are experimentally evaluated based on real traces from a continuous sensing application running on a Nokia N95 (compare CenceMe [MLF+08]).

The focus of [MPF+10] is the energy-efficient transmission of discrete contextual information for continuous sensing applications on a mobile phone, which obtains a proper trade-off between the energy consumption and accuracy. Accordingly, two different cases of connectivity between a mobile phone and a back-end server are considered, which observe the context stream without external knowledge (e.g. location-aware-activities like cooking in a kitchen). Primarily, the connectivity is always available and four various **on-line uploading techniques** on the phone-side are developed:

- A simple solution is the continuous uploading of each user context without optimization.
- A further improved method constitutes the uploading of context changes only, which has a lower energy consumption and achieves the same accuracy as the simple one.
- The next approach uploads the context only, when the change is not isolated (at least n following occurrences).
- The most complex and optimized one is a voting based technique, which analyzes the context appearance in non overlapping time windows. An update is only send, if the context with the highest frequency in the current window is different from the previous one.

The first two methods have an accuracy of 100% and get along with no optimization together with a high energy consumption. The last two strategies are associated with a certain degree of accuracy loss and a time delay caused by analyzing the following contexts.

The second consideration is an intermittent connectivity, whereby the server uses a basic method without prediction to publish the unknown or the last known context during disconnection. Thus, an **off-line prediction mechanism** on the server-side is proposed, which can be also applied for a limited number of transmissions. Since there are periods without fresh contextual information, a predictor on the back-end server forecasts the current context based on the transition probabilities. A simple and effective approach to model a transition matrix is a Markov chain, which is generated from the context changes on the mobile phone. As a result, the points in time for a matrix upload instead of a single context transmission are determined by a threshold (e.g. the Euclidean distance, the Manhattan distance, or the weighted distance), which indicates a big enough divergence between the stored matrices on the phone and on the back-end server. A combination of the described on-line and off-line strategies is used to decrease the overall energy consumption associated with an increasing accuracy.

Moreover, Musolesi et al. present the optimization gains of the uploading process integrating location information into the generic framework, which is derived from a GPS-sensor and provides particular behavior of the user at different locations. Therefore, each location is combined with a specific transition matrix to model the future sequence of context changes. The difficulties are due to the mapping of the geographical space into discrete locations, whereby a grid based structure is used with different scales. In conclusion, the framework developed achieves an operating time of a single day with a full charged battery of N95.

Recapitulating, the work reveals an interesting approach for a phone-side determination of the update times as well as a server-side estimation of unavailable context information of the user. The uploading strategies are applied in case of permanent connectivity between a mobile phone and a back-end server and analyze the context stream with simple methods. By the time the connectivity is interrupted, voluntary related to limited energy capacity or involuntary combined with mobile reception, the server forecasts the user's context based on a Markov chain. The on-line techniques in [MPF+10] indicate the optimization capability combined with analyzing the sequence of user context. Hence, this thesis deals also with an efficient energy-constrained distribution of context, but obtains further energy savings owing to the advancement of the on-line methods. The model of the user behavior is further used to predict the context stream and apply more complex optimization methods. The prediction aspect is orthogonal to the proposed problem-solving approach in Chapter 4, whereby the two of them can coexist and deliver additional energy savings.

3.5 Discussion

Due to the enormous popularity of sensor-embedded mobile phones, new fields of sensing-based applications and services arise. A lot of people carry smartphones around oneself together with the ability to capture the environment. Therefore, many researchers step up efforts on the subjects of recognition, sensing, localization, or uploading related to contextual user information. The key problem constitutes the limited energy capacity, on account of which the restricted resources get more and more important. The above presented approaches give an overview on the related topics of this thesis.

Since the proposed distribution of contextual information on mobile phones relies on the capability of correctly determined recognition, the presented works from Lester et al. and Wang et al. play an important part for the system design. Conditioned by the high energy consumption related to transmission and recognition as well as the restricted energy resource on a mobile phone, continuous processing is not possible. Therefore, both deal with the real-time capturing of user's environment and activity, whereby Lester et al. concentrate on the pure activity recognition with a developed sensing unit. Wang et al. address the efficient sensing under a constrained energy. For the proposed problem-solving approach in Chapter 4, a correct (and energy-efficient) context recognition is assumed, for which reason both approaches can be applied. The work from Constandache et al. intent upon the energy-efficient localization on a mobile phone, which is equipped with a GPS sensor. Additionally, this approach uses the potential of semantic information, which is not considered in this thesis.

Only Musolesi et al. deal with the efficient energy-constrained distribution of context and suggest a fixed schedule, which arises from the phone-side analyzes of the context stream. The proposed optimization techniques for uploading are considered more exactly among the addition of an user behavior model. Regarding this, the times for an update are adapted to the current context and the corresponding dwell times, whereby a probabilistic approach is recommended for the distribution of the sensed data. To the best knowledge of the author, this thesis is the first approach for an intelligent distribution of discrete data on a mobile phone relied on mathematical optimization and a Markovian model of user behavior.

4 Update Policy

As there are already a lot of well-studied approaches for an efficient energy-constrained recognition of contextual information, the focus of the following chapter is the decision-making process, which has a big impact on the distribution efficiency. Given the system model of Section 2.1, the two essential components on the producer's mobile phone area a *context recognition* and a *decision maker*. Assuming that the true context is sensed at each time, the determination of proper updates affects the restricted energy budget as well as the consumer's accuracy targeting the energy management problem based on a mobile phone. Therefore, Chapter 4 starts with an overview and discussion related to the basic idea of the constrained optimization problem, which comprises the modeling of user behavior, the optimization criteria, the different optimization methods, and the resulting update policy. Further, the optimization strategies are suited for a real-time continuous sensing and provide the probabilistic update values.

4.1 Overview

The context recognition component is responsible for the correct determination of the user's current context and delivers the input for the decision maker (see Figure 4.1). In this regard, the characterized activity recognition from Lester et al. ([LCK+05], [LCB+05]) or the energy-efficient approach from Wang et al. ([WKZA09]) can be used. A further input constitutes the *update policy u*, which comprises for each possible context the non-negative update probability.

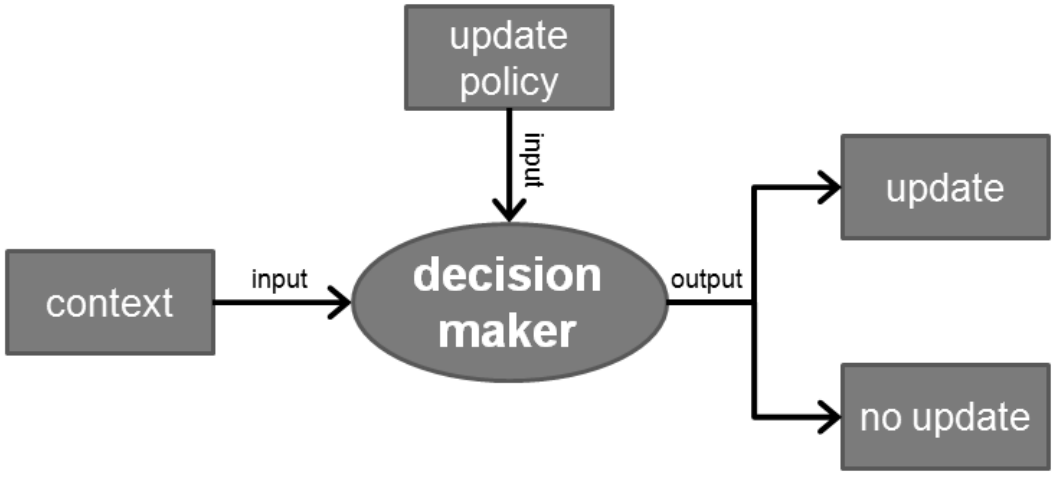


Figure 4.1: The input and output for the decision maker

The update policy is defined by a function $u : S \rightarrow [0,1]$, which reveals a context update with the probability u_s after a change to state s is observed. It is calculated on an external device (e.g. a back-end server) and is transferred to the mobile phone at the beginning of application. With the help of these two inputs the decision maker determines the output only if a context change occurs. The choice to either update the current context or not is triggered by a random update decision with a certain update probability related to each context. For example, the current detected activity on the producer is *G=Going* at time t ($G = s_t^p \in S$) and *R=Running* at time $t-1$ ($R = s_{t-1}^p \in S$). As a context change is recognized from $R \rightarrow G$, the decision maker checks the respective update probability from the update policy, which corresponds to 12.7 ($12.7 = u_G \in u$). So the decision for an update of context G follows with a chance of 12.7% and with 87.3% the decision maker does not update G.

Since the energy budget E is defined as the maximal number of context changes, which can be transferred (compare Section 2.2), the probabilistic approach selects the E contexts for transmission with the expected greatest gain on the consumer's accuracy. The expected update gain of context s_t^p at time t is calculated as the sum of equal contexts in the interval $[t, t+\varepsilon]$, whereby $t+\varepsilon$ represents the next update point, which lies (far) in the future (confer Equation (2.2)). For that reason, the expected cost of the producer $E_c[u]$ as well as the expected accuracy of the consumer $E_a[u]$ in relation to an update policy u are taken into account. $E_c[u]$ has an upper bound of the energy budget E and describes the expected cost of the solution, whereas $E_a[u]$ expresses the goodness of the update policy u. The determination of each expected update gain and thus the most qualified points in time for a transmission is based on three components, namely the *user behavior*, the *optimization criteria* and the *optimization technique*, which results in an update policy u (compare Figure 4.2). In the following, the modeling of the *user behavior* and the adaptation of the *optimization technique* are regarded intently, whereas the *optimization criteria* (the energy budget E and the accuracy A) are already described in Section 2.2.

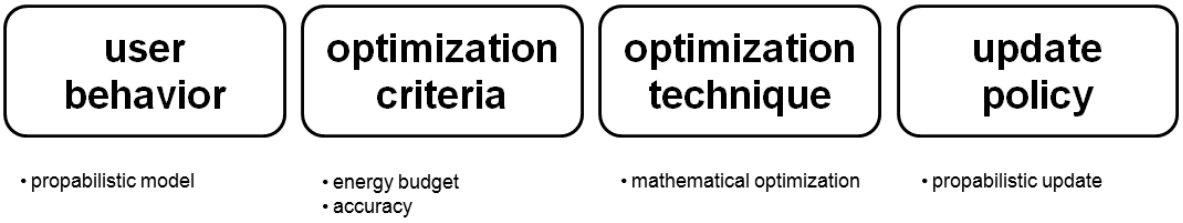


Figure 4.2: The overview of the problem-solving approach

Summarizing, the problem-solving approach should minimizes the cost criteria for the updates and maximizes the accuracy. To make assumptions about the future context changes and accordingly on $E_c[u]$ together with $E_a[u]$, the user behavior is probabilistically modeled based on analyzing past context streams. Upon the predication of the future behavior and the imposed constraints, different optimization techniques are developed to forecast the points in time for a context update related to the expected energy consumption and the expected context accuracy. As a result, the probabilistic update values for each context are determined, which serves as the second input for the decision maker.

4.2 User Behavior

The characterization related to the entire behavior of the user's context, which comprises all possible states and movements, is described by a time-varying random action, wherefore a mathematical system is required. The transition and evolution between one state and another are statistically specified by a Markov process, named after the Russian mathematician Andrey Markov. It is suited to extract significant information about the user's behavior and constitutes a well-known technique of modeling a real-world process. At each point in time the stochastic process stands exactly in one state from the random variables X_t at discrete time $t \in T$[1], which presents an observation. The successive observations $X_0, X_1, \ldots$ at times $0, 1, \ldots$ consist of the contextual information $s \in S$[2]. As a result, the variables X_t are discrete, for which reason the Markov process is referred as a Markov chain. For a better understanding, the states are mapped to a subset of natural numbers $\{0, 1, 2, \ldots, N-1\}$ [CN06].

The suitability of the stochastic process comes from the Markov property implying the assumption that the influence of a state change is restricted. This means that future states of the system depend only on the current state and not on the preceded one. To fulfill the Markov property, the dwell time of a state must be independent of the already existing remaining time related to this state at any time t, called memoryless property. Therefore, the dwell time must be geometrically distributed. The random changes from one state to another happens instantaneously together with time-independent and are called transitions. The time-irrelevant transition property of a Markov chain is termed homogeneous. In the following, the probability for being in a state x_t at time t is attained with the aid of a homogeneous, discrete-time Markov chain (DTMC) [CN06, Ste94].

$$\text{Prob}\{X_{t+1} = x_{t+1} | X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t\} \tag{4.1}$$

$$= \text{Prob}\{X_{t+1} = x_{t+1} | X_t = x_t\} \tag{4.2}$$

Thus, the Equation (4.1) and (4.2) clarify the Markov property meaning that the future state X_{t+1} depends only on the current state X_t and is irrelevant of the history of the process $X_0, X_1, \ldots, X_{t-1}$. Equation 4.2 comprises the single-step transition probabilities from state x_t to state x_{t+1} and are further denoted by $p_{ij}(t) = \text{Prob}\{X_{t+1} = j | X_t = i\}$ for all $t = 0, 1, \ldots$. By assuming a homogeneous DTMC, the conditional probabilities $p_{ij}(t)$ are rewritten without the time parameter t as p_{ij} and satisfy $0 \leq p_{ij} \leq 1$ together with $\sum_j p_{ij} = 1$ for all $1 \leq i, j \leq n$. The transition probability p_{ij} between state i and state j for all $1 \leq i, j \leq n$ forms the transition matrix P [Ste94]:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}$$

For instant, an activity recognition component senses the current context of the user every minute with a simplified activity space $S = \{\text{Standing}, \text{Going}, \text{Running}\}$. After a long time (infinity), the user behavior follows a homogeneous, discrete-time Markov chain and satisfies the required properties. For

[1] The time set T is already defined in Section 2.2.
[2] The context set S is declared at Section 2.2 yet.

$$P = \begin{pmatrix} 0.24 & 0.00 & 0.76 \\ 0.16 & 0.72 & 0.12 \\ 0.30 & 0.70 & 0.00 \end{pmatrix}$$

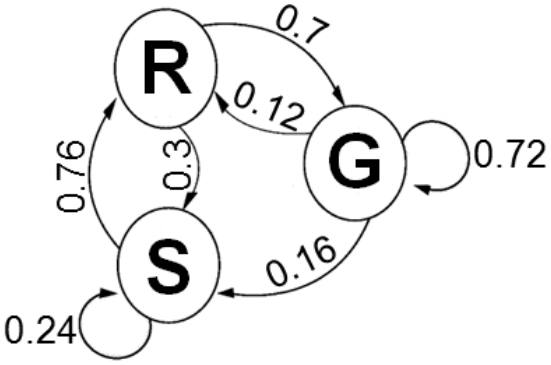


Figure 4.3: An exemplary transition matrix of a Markov chain

that reason, the three activities describe the three states of a Markov chain where state 1 equals *Standing*, state 2 represents *Going* and state 3 constitutes *Running*. Since each p_{ij} represents a conditional probability, the probability for a given *Going* to switch in the next time step to *Standing* totals at 0.16 and to *Running* at 0.12. The probability to remain in *Going* is 0.72. All probabilities for each state transition are outlined in the matrix P at the left-hand side of Figure 4.3, whereas the right-hand side depicts the corresponding transition graph.

Not only the single-step transition probability matrix at time t is an important property of a Markov chain, but also the probability related to a n-step transition $t+n$. So for a homogeneous DTMC, $p_{ij}^{(t+n)}$ evolves into

$$p_{ij}^{(n)} = \text{Prob}\{X_{t+n} = j | X_t = i\} \text{ for all } n = 0, 1, 2, \ldots$$
$$\text{with } p_{ij}^{(1)} = p_{ij}.$$

The Chapman-Kolmogorov Equation (4.3) forms a recursive formula to calculate the n-step transition probability from state i to state j. The equation regards all feasible distinct paths between the two states, whereby the probability from the first state i to an intermediate state k in l steps is multiplied by the probability from the intermediate state k to the last state j in the residual $(n-l)$ steps. The formula for calculating the probability $p_{ij}^{(n)}$ is

$$p_{ij}^{(n)} = \sum_k p_{ik}^{(l)} p_{kj}^{(n-l)} \text{ for } 0 < l < n \tag{4.3}$$

and in matrix notation

$$\begin{aligned} P^{(n)} &= P^{(l)} P^{(n-l)} \text{ for } 0 < l < n \\ &= PP \cdots PP \text{ in total n times} \\ &= PP^{(n-1)} = P^{(n-1)} P \\ &= P^{(n)} = P^n \text{ [CN06, Ste94].} \end{aligned}$$

Back to the above-mentioned example, the matrix P^2 represents the transition probabilities for the resulting activities after two minutes. For example, the probability for a change in two time steps

from the current activity *Going* in *Standing* is $P^2_{21} = 0.1896$ and to *Running* $P^2_{23} = 0.2080$. The highest probability is the transition from *Going* to *Going* with $P^2_{22} = 0.6024$. For the sake of completeness, P^∞ displays the convergence towards infinite of each transition probability.

$$P^2 = \begin{pmatrix} 0.2856 & 0.5320 & 0.1824 \\ 0.1896 & 0.6024 & 0.2080 \\ 0.1840 & 0.5040 & 0.3120 \end{pmatrix} \qquad P^\infty = \begin{pmatrix} 0.2083 & 0.5655 & 0.2262 \\ 0.2083 & 0.5655 & 0.2262 \\ 0.2083 & 0.5655 & 0.2262 \end{pmatrix}$$

For the developed optimization techniques in Section 4.3, a probability distribution is used, which is defined on the states and named stationary distribution (*sd*). Therefore, the probability is determined that the homogeneous, discrete-time Markov chain stays in state i at time t denoted by $\pi_i(t) = \text{Prob}\{X_t = i\}$. From the theorem of total probability, $\pi_i(t)$ is calculated with the initial state distribution of state k ($\pi_k(0)$) and the transition probability from the initial state k to the state i in t steps ($X_t = i | X_0 = k$),

$$\begin{aligned} \pi_i(t) &= \sum_k \text{Prob}\{X_t = i | X_0 = k\} \pi_k(0) \\ &= \sum_k p^{(t)}_{ki} \pi_k(0). \end{aligned} \tag{4.4}$$

In matrix notation, the row vector $\pi(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_n(t))$ constitutes the state probabilities at time t and the Equation (4.4) evolves to $\pi(t) = \pi(0)P^{(t)} = \pi(0)P^t$ with the initial state distribution $\pi(0)$. Moreover, for a homogeneous DTMC with a finite state space the probability distribution π fulfills for all j the conditions $\pi_j \in \mathbb{R}$, $0 < \pi_j \leq 1$, and $\sum_j \pi_j = 1$. If and only if $\pi P = \pi$, π is a stationary distribution and is calculated with $\pi_j = \sum_i \pi_i p_{ij}$ and $\sum_j \pi_j = 1$ [Ste94].

4.3 Optimization Techniques

On the basis of the probabilistic modeling of the user behavior, the future context changes are forecast to determine the optimal times for an update. Assuming that the user evolves as a Markov process, a homogeneous, discrete-time Markov chain models the user states and transitions. By the help of this discrete, stochastic process, different techniques from mathematical programming are used to maximize or minimize an objective function considering linear or nonlinear constraints. The applied subfields are *linear programming (LP)* and *nonlinear programming (NLP)*. In this context, "programming" does not denote the *programming of computer*, but rather a *schedule*. Another proposed problem-solving approach from the optimization theory is the *constrained Markov decision process (CMDP)*, which is based on a Markov decision process (MDP). LP, NLP, and CMDP have a wide range of applications in engineering or economics, for example to determine the optimal flow of resources.

Each optimization function results in a Markov-optimal update policy, which determines on-line the update probability of the current visited user state. On-line means that the decision maker determines at runtime to update the current state or not. In contrast to the on-line method, an off-line approach based on dynamic programming (DP) is developed to identify the optimal solution considering the knowledge of the passed through user state sequence. As the emerged accuracy of the dynamic programming provides the solution upper bound, two further approaches are deployed without any optimization. Therefore, *FireAll* and *Periodic* deliver a solution lower bound and serve as a comparative value for the optimization efforts of linear and nonlinear programming and constrained Markov decision process.

4.3.1 Fire All

The first method without any optimization (*FireAll*) utilizes a simple decision rule to determine the single update points in time. It sends always the current context change until the allocated energy budget is consumed. Therefore, a threshold θ is specified, which equals to the entire amount of updates, before the occurrence of the current context change. The right-hand side of Figure 4.4 displays an exemplary context sequence $\ldots SGGRSGGGSGSSRRSGSSSGSRRGSR\ldots$ with 12 transmissions (depicted as the up arrows). After the last update, θ is set to 0 and no further transmission occurs. For this reason, an accuracy gain is only achieved subsequently, if the future contexts equal the last update one. Due to the simplicity, θ constitutes a simple input for the decision maker (see the left-hand side of Figure 4.4) and is determined from the energy budget E.

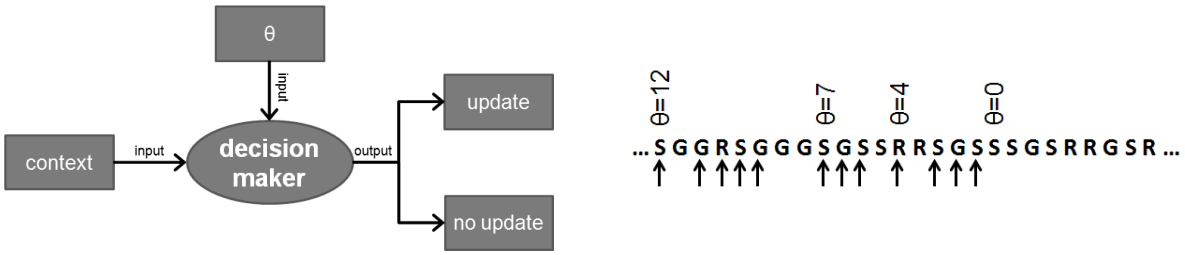


Figure 4.4: The decision maker for the method *FireAll*

For a better understanding, the decision background of *FireAll* is outlined in pseudo code at Algorithm 4.1, which is used from the decision maker either to send an update or not. Thereby, the global variable $cnt \in \mathbb{N}$ is initialized with 0 and constitutes the amount of already updated context. The function in line 1 returns a boolean variable indicating to update the current context change ($TRUE$) or not ($FALSE$) and is called every time a context change occurs. If the updates sent so far exceed θ, the function evolves always to $FALSE$.

Algorithm 4.1 The algorithm of the decision maker for *FireAll*

```
1:  function CONTEXTCHANGEOCCURRED(θ, cnt)
2:      if isFirstContextChange then
3:          cnt ← cnt + 1
4:          return TRUE
5:      end if
6:      if cnt ≤ θ then
7:          cnt ← cnt + 1
8:          return TRUE
9:      end if
10:     return FALSE
11: end function
```

The if-statement in line 2 is due to the defined property in Section 2.2 that the first occurred context change is always updated from the optimization method.

4.3.2 Periodic

As the name implies, the second method without optimization uses a periodic approach for the determination of the update times and differs from *FireAll* in the definition of threshold θ. Here, it determines the amount of context changes including the current transferred, before the next update is sent (compare the right-hand side of Figure 4.5).

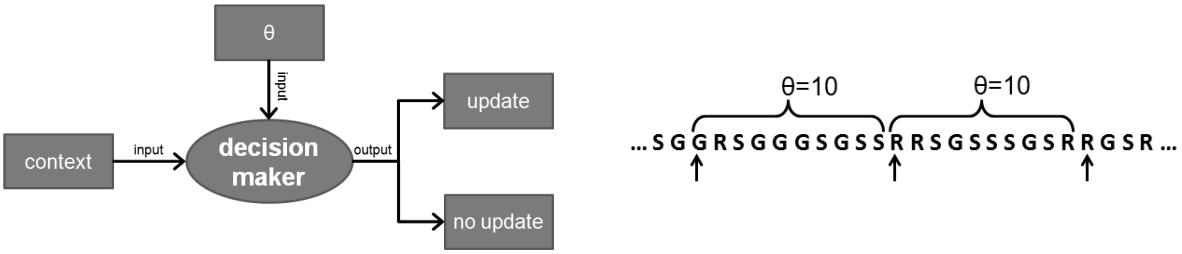


Figure 4.5: The decision maker for the method *Periodic*

In this manner, a periodic update performance is achieved without a Markov model and any mathematical optimization. The threshold θ is determined from the energy budget E, which implicates the amount of permitted context change transmissions. For example, for a given energy budget of 100 updates ($E = 0.1$) related to an overall application runtime of 1000 context shifts, the waiting time θ between two successive updates accounts for 10 context changes (compare Section 2.2):

$$\text{Given: } CS = 1000 \text{ and } E = 0.1$$
$$\text{Find: } \theta$$
$$\theta = \frac{1}{E} = \frac{1}{0.1} = 10$$

Algorithm 4.2 presents the decision background of *Periodic* and is similar to Algorithm 4.1 apart from the application of θ and $cnt, \theta \in \mathbb{R}$. Due to the fact that $0.0 \leq E \leq 1.0$, *cnt* and θ are floating-point numbers. The global variable *cnt* saves the passed through context changes since the last update. If the current waiting time exceeds the periodic one, an update is sent and *cnt* is reseted (compare line 6).

Algorithm 4.2 The algorithm of the decision maker for *Periodic*

```
1:  function CONTEXTCHANGEOCCURRED(θ, cnt)
2:      if isFirstContextChange then
3:          return TRUE
4:      end if
5:      cnt ← cnt + 1
6:      if cnt ≥ θ then
7:          cnt ← cnt − θ
8:          return TRUE
9:      end if
10:     return FALSE
11: end function
```

Notice, that the periodic approach may or may not achieve with an energy budget of 1.0 an equality of the entire context sequence between the producer and consumer. This results from the obstinate cyclic behavior, wherefore the method updates (some) same contexts.

4.3.3 Linear Programming

The first approach relying on mathematical optimization is the linear programming (LP), which is an essential tool in operations research and is used in economics as well as business (e.g. resource allocation problems in transportation or telecommunication). LP solves a number of large complex problems with maximization or minimization of a linear objective function subjecting to linear equalities and/or inequalities. It was developed in the late 1940s and belongs to applied mathematics [DT97, KB95].

The Markovian modeled user behavior combined with the linear optimization obtain an optimal update policy for the decision maker, which contains the non-negative probabilities to update the current context change (compare Figure 4.1). Therefore, a mathematical formulation for the system model outlined in Section 2.1 is required, which describes the real world activities and processes. The formal definition of linear programming is as follows [Dan98]:

Find the values of $x_1, x_2, \ldots, x_n$ that will optimize

$$f = c_1x_1 + c_2x_2 + \ldots + c_nx_n \tag{4.5}$$

subject to the constraints

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \leq (=)(\geq) & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & \leq (=)(\geq) & b_2 \\ \vdots & & & & \vdots & & & \vdots & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \leq (=)(\geq) & b_m \end{array} \tag{4.6}$$

Notice that the objective function (4.5) must be linear. Furthermore, each equality or inequality in (4.6) contains only one symbol of $\leq, =$, or $\geq$ and are a linear function of the variables $x_1, x_2, \ldots, x_n$. Consequently, the constraints span a convex polytope, over which the function (4.5) is maximized or minimized [KB95].

In matrix notation, a linear programming problem is formulated as follows:

Find a vector $x \in \mathbb{R}^n$ that will optimize

$$f = c^T x$$

subject to

$$Ax \leq (=)(\geq)b$$

with

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, c = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}, \text{ and } b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

A very important property of linear programming is the duality, which characterizes the feasibility to convert a minimization in a maximization problem and vice versa. Therefore, the negative objective function is maximized instead of minimizing it and the sign of the solution is alternated [KB95]:

$$\min \sum_{i=1}^{n} c_i x_i = -\max -\sum_{i=1}^{n} c_i x_i$$

The optimal solution, which maximizes or minimizes the objective function by satisfying all the constraints, is found in the extreme point(s) related to the set of the feasible solution(s). A trivial method considers all solutions and discards the impossible ones. Subsequently, it looks closely at the remaining solutions to find the optimal one. Therefore, George B. Dantzig developed in 1947 a fast and systematically approach to solve a linear programming problem, the simplex method. Another way to solve linear programming problems is the interior-point method [DT97, KB95].

On the basis of linear optimization, the first Markov-optimal update policy u is developed meaning that the unknown variables A, b, and c of the linear programming problem must be determined. Since the modeled user behavior is based on a probabilistic approach, the unknowns are derived from the expected cost $E_c[u]$ and the expected accuracy $E_a[u]$. At first, the expected cost $E_c[u]$ is regarded, which is composed of the summation related to the update probability after observing a context change. Therefore, the probability of the occurrence from state i, described with the stationary distribution sd_i, is multiplied by the probability of a transition from state i to state j in one time step, p_{ij}. As an update of the same state causes only costs and does not increase the accuracy, Equation (4.7) does not include the change between same states ($j \neq i$). Then, the transition probability p_{ij} is multiplied by the update probability of state j, u_j, because a transmission of the context j happens only if the decision maker fires. As a result, the final expected costs are calculated in the following way:

$$E_c[u] = \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij} u_j \tag{4.7}$$

As a consequent, the main part of the linear programming constraints, namely matrix A and vector b, are derived from Equation (4.7). This is rewritten so that each element in a $n \times 1$ matrix A indicates the probability to change in state j from any state unequal it:

$$\begin{aligned} E_c[u] &= \sum_{i=1}^{n} u_i \sum_{j=1:j\neq i}^{n} sd_j p_{ji} \\ &= xA \end{aligned}$$

with

$$A = \begin{pmatrix} \sum_{j=1:j\neq 1}^{n} sd_j p_{j1} & \sum_{j=1:j\neq 2}^{n} sd_j p_{j2} & \dots & \sum_{j=1:j\neq n}^{n} sd_j p_{jn} \end{pmatrix}$$

and

$$x = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$$

and

$$b = \begin{pmatrix} E \end{pmatrix}$$

To fulfill the restriction $u_i \in [0,1]$ for all $1 \leq i \leq n$, further constraints are added:

$$x \geq 0 \text{ and } x \leq 1$$

The expected accuracy $E_a[u]$ underlies the idea, whenever the decision maker does not send an update at the context change from state i to j, the consumer's accuracy decreases about the expected number of times the producer stays in state j. Accordingly, $E_a[u]$ represents the error probability and constitutes the objective function of the linear programming problem.

$$E_a[u] = \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}(1-u_j) \sum_{k=0}^{\infty} p_{jj}^k \tag{4.8}$$

First, Equation (4.8) computes the probability of a context change from state i to state j (compare Equation (4.7)), which is multiplied by the probability of not updating state j together with the sum of staying k-times in state j with $k \geq 0$. Therefore, the last term characterizes the loss of information and converges to $\frac{1}{1-p_{jj}}$, with the aid of a geometric series ([Pap09]):

$$\sum_{k=0}^{\infty} p_{jj}^k = \lim_{n\to\infty} \sum_{k=0}^{n} p_{jj}^k = \lim_{n\to\infty} \frac{1-p_{jj}^{n+1}}{1-p_{jj}} = \frac{1}{1-p_{jj}} \text{ with } \left|p_{jj}\right| < 1 \tag{4.9}$$

Hence, the final expected accuracy is derived by inserting Equation (4.9) into (4.8) and is rewritten in vector notation.

$$\begin{aligned} E_a[u] &= \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}(1-u_j)\frac{1}{1-p_{jj}} \\ &= c^T x \end{aligned}$$

with

$$c = \begin{pmatrix} -\sum_{j=1:j\neq 1}^{n} \frac{sd_j p_{j1}}{1-p_{11}} \\ -\sum_{j=1:j\neq 2}^{n} \frac{sd_j p_{j2}}{1-p_{22}} \\ \vdots \\ -\sum_{j=1:j\neq n}^{n} \frac{sd_j p_{jn}}{1-p_{nn}} \end{pmatrix} \text{ and } x = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$$

Note that the term $\sum_{i=1}^{n}\sum_{j=1:j\neq i}^{n} \frac{sd_j p_{ji}}{1-p_{ii}}$, which is left over from Equation (4.9) after expanding, is constant and thus not further regarded. Generally, the division of each component of an objective function by a constant factor has no influence on the decision variables of a linear programming problem.

Summarizing, the entire linear programming problem results in:

$$\begin{aligned} &\text{minimize } \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}(1-u_j)\frac{1}{1-p_{jj}} \\ &\text{subject to } \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j \leq E \\ &\text{and } 0 \leq u \leq 1 \end{aligned} \tag{4.10}$$

4.3.4 Nonlinear Programming

The adaption of the efficient energy-constrained distribution of context as a linear programming problem has an elementary error, since the recurrence to the updated state without intermediate update is not included. For instant, the decision maker updates state G at time t_1, because it predicts at the context change ($S \rightarrow G$) to stay k-times in the state G, which corresponds in the example to $k = 2$ at t_1 with a gain $g = 2$. In reality, the decision process revisits after the state sequence RS 4-times more the transferred state G (compare the underlined states in Figure 4.6), before the successive update of state R at time t_2 follows. Therefore, the total gain for the updated state G at t_1 is $g = 6$. The same applies to the transmitted state R at t_2, where the entire gain is $g = 4$ and not $g = 2$. Basically, this relies on the $\sum_{k=0}^{\infty} p_{jj}^{k}$ in Equation (4.8), which considers only the k direct following states.

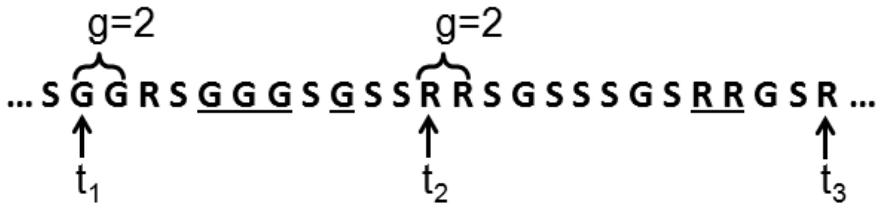


Figure 4.6: The main drawback of the linear programming approach

Additionally, the calculated expected energy cost $E_c[u]$ contains the costs related to consecutive updates of equal states (compare the transferred state R at time t_2 and t_3 in Figure 4.6), whereas in fact no further energy consumption occurs for the second transmission. This results from the fact that the Equation (4.7) sums over the occurrence of a context change together with the combined update probability disregarding the next possible update.

For these reasons, the linear programming computes a suboptimal update policy, which results in a too pessimistic accuracy under a given energy budget. To overcome the shortcoming related to the expected accuracy $E_a[u]$, the recurrence probability M is formulated, which covers the probability that a transition from state i to j happens without updating state j, M_{ij}. Therefore, the transition matrix P from the Markov chain is extended with the no-update probability from the policy u.

$$M = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ m_{21} & m_{22} & \dots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nn} \end{pmatrix}$$

where $m_{ij} = p_{ij}$ for $i = j$ else $m_{ij} = p_{ij}(1 - u_j)$. To comprise all possible sequences of context recurrences, the sum for all different path-lengths l is established with matrix multiplication. Thereby, the entry m_{ij} of the l-step transition matrix M^l characterizes the probability of starting in state i and changing in l-steps to j without an intermediate update.

$$\sum_{l=1}^{\infty} M^l = \sum_{l=1}^{\infty} M M^{l-1} = \sum_{l=1}^{\infty} M^{l-1} M$$

The summation over all M^l is rewritten with the help of the Neumann series, which constitutes the analog to the geometric series [ZF86].

$$(\mathbb{E}-A)^{-1} = \mathbb{E}+A+A^2+A^3+\ldots = \sum_{k=0}^{\infty} A^k \text{ with } A \in \mathbb{R}^{n\times n}$$

$\mathbb{E}$ means the identity matrix and the sum evolves to $R = (\mathbb{E}-M)^{-1} = \sum_l M^l$. Upon this, the objective function $E_a[u]$ reformulates to:

$$E_a[u] = \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j sd_j \frac{1}{(1-p_{jj})^2} \sum_{k=1}^{n} R_{jk} \sum_{l=1:l\neq j\&l\neq k}^{n} p_{kl}u_l$$

Obviously, the first part of $E_a[u]$ remains equal compared to the objective function of the linear programming. It calculates the probability of updating context j after the occurrence of context i and an immediate change from i to j. Afterwards, the residual part determines the accuracy gain related to j ($sd_j \frac{1}{(1-p_{jj})^2}$) for the update interval $[j,l]$, where no intermediate updates of another context happens. Figuratively, the formulation identifies a left update j and the immediate following update l. Therefore, it is true that in the space of the interval $[j,l]$ occurs no additional update (compare $\sum_{k=1}^{n} R_{jk}$), because the accuracy gain is calculated within it.

The second problematical aspect, the expected energy cost $E_c[u]$, is determined as follows:

$$E_c[u] = \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j \sum_{k=1}^{n} R_{jk} \sum_{l=1:l\neq j\&l\neq k}^{n} p_{kl}u_l \leq E$$

Here, the description of $E_c[u]$ is very similar to $E_a[u]$, which is not surprising as it counts the overall update cost probability related to the objective. In this connection, the constraint computes the probability of an update interval $[j,l]$ for each context i. Therefore, the stationary distribution of i is multiplied by the summation of the transition and update probability of context j, which is equivalent to the calculation of the linear one. However, the secondary part takes into account the probability of changing from the updated context j to k without any intermediate transmission. After this, a following transaction from the state k to l occurs, which is again updated. As a result, the update interval $[j,l]$ is achieved related to the initial context i.

As both, the objective function $E_a[u]$ and the constraint $E_c[u]$, are not anymore a linear combination of the independent, unknown variables u, another approach as the linear programming has to be used to compute a Markov-optimal update policy. Therefore, the so called nonlinear programming is utilized, which differs only from the linear programming in the feasible representation of the functional components. As the name implies, the objective function and/or (some of) the constraints are nonlinear [LY08]. Summarizing, the constrained optimization problem is

$$\begin{aligned}
&\max \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j sd_j \frac{1}{(1-p_{jj})^2} \sum_{k=1}^{n} R_{jk} \sum_{l=1:l\neq j\&l\neq k}^{n} p_{kl}u_l \\
&\text{subject to } \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j \sum_{k=1}^{n} N_{jk} \sum_{l=1:l\neq j\&l\neq k}^{n} p_{kl}u_l \leq E \\
&\text{and } 0 \leq u \leq 1
\end{aligned}$$

4.3.5 Constrained Markov Decision Process

The last on-line approach views the constrained optimization problem as a control problem, since the decision maker chooses sequentially a control action from the set $\{\text{update}, \text{no-update}\}$ based on a policy at each discrete time unit t ($t = 0, 1, \ldots$). In this manner, the problem related to the correct computation of the expected updated gain should be solved and a Markov-optimal update policy u is derived.

For this reason, a discrete-time, stochastic control process from control theory is used, which represents a generalization of a Markov chain. It is denoted as a Markov decision process (MDP) and constitutes a mathematical framework for the dynamic system. The controller (e.g. the decision maker) being at time t in state x chooses an available action a, which causes some instantaneous cost c. Due to the valid Markov property of (non-controlled) Markov chain, the transition probability from state x to y in next time step $t+1$ is fully determined by the current state x together with the chosen action a. For this reason, the transition probabilities are conditionally independent from the past. Setting the available actions (choice) in each state to one and the immediate costs (motivation) to zero, the resulting Markov decision process evolves to a simple Markov chain [Alt95].

Formally, a MDP is a 4-tuple $\text{MDP} = (X, A, P, c)$ with

- the set of states $X = \{x_1, x_2, \ldots\}$
- the set of actions $A = \{a_1, a_2, \ldots\}$ with the state-action pairs $K = \{(x, a) : x \in X, a \in A(x)\}$, which correspond to the available actions at state x
- the transformation $P : X \times A \times X \to \mathbb{R}$ with $x, y \in X$ and $a \in A$ is a transition probability P_{xay}, which describes the probability to alter from state x to state y choosing action a:

$$P_{xay} = \text{Prob}(X_{t+1} = x | X_t = y, a_t = a)$$

- the immediate cost function $c : K \to \mathbb{R}$, which describes the instantaneous costs and is minimized

As can be seen from the definition, the Markov decision process optimizes only against a single utility, whereas the optimization criteria related to the efficient energy-constrained distribution of context include two objectives. Accordingly, the decision maker maximizes the accuracy subjected to the energy constraint, wherefore a MDP is extended about a restriction resulting in a constrained Markov decision process (CMDP). Formally, a CMDP is defined as a 5-tuple $\text{CMDP} = (X, A, P, c, d)$, which distinguishes from a MDP in the added immediate cost function $d : K \to \mathbb{R}^K$. d^K represents a K-dimensional vector related to the constraints V_k with $k = 1, \ldots, K$.

The constrained optimization problem (COP) evolves to

$$\text{COP: } \min C \text{ subject to } D^k \leq V_k \text{ with } k = 1, \ldots, K,$$

whereby Altman describes in [Alt95] a solution approach based on linear programming, which attains the best $\rho(y, a)$ with $y \in X$ and $a \in A$ satisfying the energy constraint. The $\rho(y, a)$ constitutes the occupation measure, which denotes the probability related to the existence of the state-action pair (y, a)

in the decision process. The COP is equivalent to the following linear programming and returns a stationary randomized policy:

$$\min_{\rho} \sum_{y \in X} \sum_{a \in A} c(y,a)\rho(y,a)$$

subject to

$$\sum_{y \in X} \sum_{a \in A} \rho(y,a)(\delta_x(y) - \alpha P_{yax}) = 0, \qquad \forall x \in X \tag{4.11}$$

$$\sum_{y \in X} \sum_{a \in A} \rho(y,a) = 1 \tag{4.12}$$

$$\rho(y,a) \geq 0, \qquad \forall y \in X, a \in A \tag{4.13}$$

$$\sum_{y \in X} \sum_{a \in A} d^k(y,a)\rho(y,a) \leq V_k, \qquad k = 1, \ldots, K$$

$\delta_x(y)$ with $x, y \in X$ is the Dirac probability measure on x:

$$\delta_x(y) = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{else} \end{cases}$$

The Constraints (4.12) and (4.13) guarantees that $\rho(y,a)$ is a probability measure and Constraint (4.11) that for any state the outgoing and incoming rate are equal. Since the decision maker determines the action $a \in A$ on the given state $y \in X$, the according probability corresponds to

$$u_y = \frac{\rho(y,a)}{\rho_y} \text{ with } \rho_y = \sum_{a \in A} \rho(y,a), \forall y \in X.$$

As the main problem of LP (compare Equation (4.10)) is the difficulty to identify the sum of returning contexts after an update occurred, a third update policy u is derived with a constrained Markov decision process. The underlying idea is the memorizing of the last updated context to extract more information for the decision maker whether to update the current context change or not. Therefore, a CMDP is adapted to the efficient energy-constrained distribution problem in the following way:

The finite state space $S = \{s_1, s_2, \ldots, s_N\}$ from Section 2.2 is transformed to the finite set $X = \{\{s_1,s_1\},\{s_1,s_2\},\ldots,\{s_1,s_N\},\{s_2,s_1\},\{s_2,s_2\},\ldots,\{s_N,s_N\}\}$. Each state $x \in X$ is composed of two inner states, for example $x = \{s_2, s_4\}$ with $x_1 = s_2$ and $x_2 = s_4$, which describe on the one hand the last updated context (x_1) and on the other hand the current visited (x_2). Furthermore, the finite set $A = \{\text{update}, \text{no-update}\}$ denotes the possible actions of the decision maker. The probabilities P_{xay} with $x, y \in X$ and $a \in A$ are defined with the transition matrix P from the Markov chain as follows:

$$P_{xay} = \begin{cases} P_{xy}, & \text{if } a = \text{update and } x_2 = y_1 \\ P_{xy}, & \text{elseif } a = \text{no-update and } x_1 = y_1 \\ 0, & \text{else} \end{cases}$$

The first case describes the update of the current visited inner state x_2 and the change to the next state y_2, wherefore the last updated context of the following state y must be the same ($y_1 = x_2$). Since the

chosen action does not influence the transition probability, P_{xay} is equivalent to P_{xy}, if the condition is realized after an update. Otherwise ($y_1 \neq x_2$), an impossible transition from state x to y with the action *update* occurred and the corresponding probability is set to 0. The second case constitutes the no-update decision and the necessary requirement that the last updated context does not change after a transition ($x_1 = y_1$). If it is true, the probability is set again as P_{xy} or else to 0.

The cost function $c(y,a)$ describes the expected Markovian loss of context related to state $y \in X$ and action $a \in A$. Therefore, the result of $c(y,a)$ is set to 1, if the decision process is in state y with $y_1 \neq y_2$ and action *no-update*. This corresponds to a loss of information between the producer and consumer. In all remaining cases ($a =$ update or $x_1 = x_2$), the accuracy increases, because of the equality related to the last updated context and the current visited.

$$c(y,a) = \begin{cases} 1, & \text{if } a = \text{no-update and } y_1 \neq y_2 \\ 0, & \text{else} \end{cases}$$

The constrained cost function d^K with the constraints V_k, $k = 1,\ldots,K$ represents the energy constraint. For that reason, the required dimension K is 1 with $V_1 = E$ and d^1 as the expected energy cost, which ensures that the expected energy consumption is less than the available energy budget E. Each time the decision maker sends an update, the energy costs increases by 1:

$$d(y,a) = \begin{cases} 0, & \text{if } a = \text{no-update} \\ 1, & \text{else} \end{cases}$$

Figure 4.7 displays the state graph of a Markov decision process, which consists of two contexts $\{G,S\}$ (e.g. *G=Going* and *S=Standing*) and a finite action set $A = \{d,n\}$ (e.g. *d=update* and *n=no-update*). The cost functions c and d are assumed from the above-mentioned CMDP adaption. For reasons of clarity, the transition probabilities P_{xay} are not outlined in the graph.

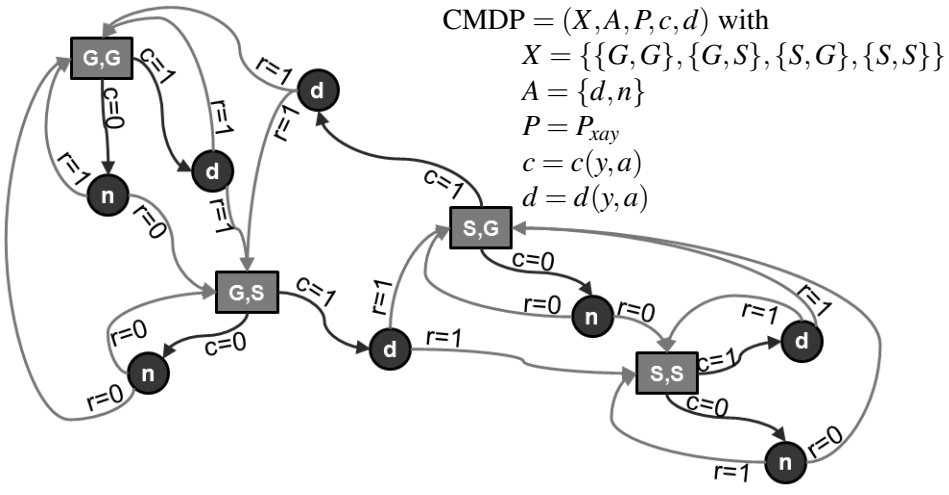


Figure 4.7: An exemplary constrained Markov decision process

4.3.6 Dynamic Programming

For a better comparison of the generated Markov-optimal update policies, a off-line approach is developed to achieve the maximal possible accuracy under a given energy budget, together with the optimal moments in time for an update. Off-line means that the entire passed through context sequence of the producer s^p is available and the method searches for the points in time $UP = \{t_1, t_2, \ldots\}$ with the maximal achievable state consistence (accuracy). This serves as a reference value for the attained accuracy from linear programming, nonlinear programming, and constrained Markov decision process. Therefore, the finite, deterministic and discrete dynamic system is analyzed via dynamic programming, which is an efficient tool to find the optimal solution of a (constrained) control problem.

The dynamic programming method solves optimization problems with many possible solutions and relies on mathematical induction. The aim is to determine one or more optimum, which means to find the minimum or maximum in a set of feasible solutions. The underlying idea is to simplify a complex problem by breaking it down into simpler disjoint sub problems, at which each problem occurs several times. The method takes advantage of repeatedly using the stored solution with a constant lookup time and avoids computationally intensive recomputing. The set of sub problems is bounded. This property is called *overlapping subproblems*. The other key characteristic is *optimal substructures* and describes the property that an optimal solution for a problem is composed from the optimal solutions of sub problems. Thus, each sub problem is solved independently of each other. The technique of dynamic programming is based on **The Principle of Optimality** from Richard Bellmann in the 1940s:

> *An optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.* [Bel57]

Therefore, the associated solution of each sub problem is stored and reused at reappearance, which is called "memoization". The word *memoization* is written correctly and comes from the term "memo", which means recording a value and look it up later. Accordingly, the effectiveness of the method relies on the appearance and the number of the saved sub problems, which is typically polynomial. If the two basic key ingredients are fulfilled, dynamic programming is a viable method for solving an optimization problem, wherefore the mathematical description follows [Bat00, CLRS09].

The underlying dynamic process consists of different stages $x_0, x_1, \ldots, x_{N_1-1}$ with the initial state vector x_0. Each state vector x_k with $k = 0, 1, \ldots, N_1 - 1$ holds the required information about the current situation (process), for example wealth. A transformation function W changes the state from x_k to a new state x_{k+1} and depends on the control variables d_{k_t}, wherefore x_{k+1} evolves to:

$$x_{k+1} = W(x_k, d_{k_t}) \text{ with } k = 0, 1, \ldots, N_1 - 1 \text{ and } t = 0, 1, \ldots, N_2 - 1.$$

The possible vectors d_{k_t} rely only on the current state x_k with $d_{k_t} \in \Phi(x_k)$ and $N_2 = |\Phi(x_k)|$. The choice of a d_{k_t} from the set $\Phi(x_k)$ at the current stage x_k is called a decision, which is independent from the past. A policy function describes the relation between the control and state variables. The optimal policy and accordingly the corresponding d_{k_t} should maximize or minimize the value of the objective (e.g. cost), which is written as a recursive state function and is called the value or return function. Here,

the value function is described as a maximization (max), but matches also a minimization by replacing the max with a min.

$$f_{N_1}(x_0) = \max_{d_0} [F(x_0, d_0) + f_{N_1-1}(T(x_0, d_0))], \text{ with } N_1 \geq 1$$
$$f_0(x_{N_1-1}) = \max_{d_{N_1-1}} F(x_{N_1-1}, d_{N_1-1})$$

with $F(x_k, d_k)$ is equal to the gain in stage k. Thus, f_{N_1} = "the sum of a N_1-stage process subject to the states x_k and the optimal decisions d_k with $k = 0, 1, \ldots, N_1 - 1$" [BK65]. The relationship between stage N_1 and $N_1 - 1$ is called the *Bellmann Equation*. The solution of a dynamic programming algorithm are the unique sequence of the value functions $f_k(x_k)$ and the policy functions $d_k(x_k)$. The $f_k(x_k)$ determines the total return g (gain), on which the accuracy is calculated. Remember that the accuracy is defined by the ratio of the total gain and the number of passed through contexts. The $d_k(x_k)$ constitutes the single decisions, which corresponds in this thesis to *update* or *no-update* [Bel57, BK65].

Equations (4.14) constitute an example with two user states, *Standing* and *Going*. The system is sampled at ten points in time (T) and returns the producer's state sequence s^p. UP_1, UP_2 and UP_3 are update sequences with one, two and three messages to send. Accordingly, the achieved gain is six, three and six. Only UP_1 is optimal, because of the transmission at point 0, which attains the maximal achievable gain six with one message. This is equal to the maximal occurrence of a state in s^p and is, in this case, identical with state *Standing*. However, UP_2 and UP_3 are suboptimal, due to the fact that the maximal gain with two messages is six ($UP_2 = \{0,4\}$) and with three messages eight ($UP_3 = \{0,4,8\}$). Furthermore, the update sequence UP_2 would by definition not be applicable in the problem-solving approach, because of property that the first context is always transferred (compare Section 2.2):

(4.14)
$$\begin{aligned} s^p &= \{S,G,S,S,G,G,S,G,S,S\} \\ S &= \{\underline{S}tanding, \underline{G}oing, \underline{U}ndefined\} \\ T &= \{0,1,2,3,4,5,6,7,8,9\} \end{aligned}$$

$$\begin{aligned} &UP_1 = \{0\} && s^c = \{S,S,S,S,S,S,S,S,S,S\} && g_1 = 6 \\ &\vdots \\ &UP_2 = \{3,7\} && s^c = \{U,U,U,S,S,S,S,M,M,M\} && g_2 = 3 \\ &\vdots \\ &UP_3 = \{0,2,9\} && s^c = \{S,S,S,S,S,S,S,S,S,S\} && g_3 = 6 \\ &\vdots \end{aligned}$$

A naive and simple algorithm solves the dynamic programming problem by checking all possibilities in a brute-force manner, which is not very effective. It suffers under the recursive calls over and over again, which calculates the solution of the same problem repeatedly. For example, exists 120 feasible update sequences for UP_3:

$$UP_3^1 = \{0,1,2\}, UP_3^2 = \{0,1,3\}, \ldots, UP_3^8 = \{0,1,9\}, UP_3^9 = \{0,2,3\}, \ldots,$$
$$UP_3^{15} = \{0,2,9\}, \ldots, UP_3^{37} = \{1,2,3\}, \ldots, UP_3^{43} = \{1,2,9\}, \ldots, UP_3^{120} = \{7,8,9\}$$

The count (and calculation) of all possibilities of this exhaustive search is done recursively by the function below. Denote the number of messages to be send by mes and the cardinality of s^p by N_{s^p}, whereas the index k conforms to the lower bound of the update interval. So the first function call happens with $k = 0$ and is reseted in every step:

$$\Delta(N_{s^p}, \text{mes}, k) = \begin{cases} \sum_{i=k}^{N_{s^p}-\text{mes}} 1, & \text{if mes} = 1 \\ \sum_{i=k}^{N_{s^p}-\text{mes}} \Delta(N_{s^p}, \text{mes}-1, i+1), & \text{else} \end{cases}$$

For instance,

$$\Delta(10,3,0) = \sum_{i=0}^{N_{s^p}-3} \sum_{j=i+1}^{N_{s^p}-2} \sum_{k=j+1}^{N_{s^p}-1} 1 = \sum_{i=0}^{7} \sum_{j=i+1}^{8} \sum_{k=j+1}^{9} 1 = 120.$$

Algorithm 4.3 represents an implementation of the brute-force method with the starting function *OptimalUpdatePolicy*. s^p, mes and k denote the sequence of states, the total number of messages to send and the starting index of the update interval. It calculates the optimal total gain (g) and the optimal update sequence (idx). The function in line 7 and 14, which is outlined in Algorithm 4.4, yields the maximal appearance of a state in the interval $[lb, ub]$ of s^p as well as the corresponding index. Therefore, it counts for each state in the interval its appearance between the lower and upper bound.

Algorithm 4.3 An algorithm for a brute-force search of the dynamic programming solution: Part I

```
 1: function OPTIMALUPDATEPOLICY(s^p, mes, k)
 2:     g, idx ← −1
 3:     if mes > 1 then
 4:         for i = k, ..., (N_{s^p} − mes) do
 5:             cnt ← 0
 6:             g_R, idx_R ←OPTIMALUPDATEPOLICY(s^p, mes − 1, i + 1)
 7:             g_S, idx_S ←CALCULATEMAXSTATEFREQUENCY(s^p, k, i)
 8:             if g_R + g_S > g then
 9:                 g ← g_R + g_S
10:                 idx ← idx_R + idx_S
11:             end if
12:         end for
13:     else
14:         g, idx ←CALCULATEMAXSTATEFREQUENCY(s^p, k, N_{s^p} − 1)
15:     end if
16:     return g, idx
17: end function
```

The big O notation delivers an upper limit of an algorithm growing rate. In this case, the function *OptimalUpdatePolicy* of the Algorithm 4.3 is bounded by $O(N_{s^p}^{\text{mes}+1})$, whereas the function *CalculateMaxStateFrequency* of Algorithm 4.4 is bounded by $O(N_{s^p}^2)$. So each time the amount of feasible update messages increases by one, the running time of the Algorithm 4.3 rises N_{s^p}-times.

Algorithm 4.4 An algorithm for a brute-force search of the dynamic programming solution: Part II

```
1:  function CALCULATEMAXSTATEFREQUENCY(s^p, lb, ub)
2:      g, idx ← −1
3:      for i = lb, ..., ub do
4:          cnt ← 0
5:          for j = i, ..., ub do
6:              if s_i^p = s_j^p then
7:                  cnt ← cnt + 1
8:              end if
9:          end for
10:         if cnt > g then
11:             g ← cnt
12:             idx ← i
13:         end if
14:     end for
15:     return g, idx
16: end function
```

As a consequence of the recursive calculation, the brute force method does much more work than necessary. For example, the call of function *CalculateMaxStateFrequency* in line 7 and 14 can be saved, if the according values g and idx are still available. Therefore, the values must be stored at first occurrence (memoization). But this is not the only and primary optimization of Algorithm 4.3 as the problem of getting the optimal update sequence has the properties of *overlapping subproblems* and *optimal substructure*. Based on these considerations, an efficient algorithm is formulated by dynamic programming, which is developed in four steps [CLRS09]:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Steps $1-3$ return the value of and Step 4 the optimal solution itself. Based on this outline, the dynamic programming method finds the solution of an energy constrained optimal update sequence. The structure of an optimal solution is as follows: For a state sequence with length N_{s^p} and mes update messages, the algorithm has to find the mes not overlapping intervals with the highest state occurrence. The sum over the occurrences returns the maximal gain, whereas the indexes of each updated state yield the optimal update sequence. The following represents the development and description of an efficient approach with the help of dynamic programming.

In contrast to the brute-force method, the necessary properties *optimal substructures* and *overlapping subproblems* must be defined to constitute the optimal update sequence by means of dynamic programming. Assuming for the explanation of *optimal substructures* that an update sequence is allowed to send mes messages, UP_{mes}. The optimal solution to UP_{mes} consists recursively of the optimal solutions to the related subproblems $UP_{\text{mes}-1}, UP_{\text{mes}-2}, \ldots, UP_2$ and UP_1, which are independent instances

and can be solved isolated in a bottom-up approach. Therefore, the dynamic programming algorithm starts to find the optimal solution for $UP_1[i]$ with $i = 0,1,\ldots,N_{p^s}-1$, which contains the maximal occurrence of a state in the appropriate interval $[i,N_{p^s}-1]$. $UP_2[j]$ with $j = 0,1,\ldots,N_{p^s}-2$ consists of the two update intervals $[0,j]$ and $[j+1,N_{p^s}-1]$. The method calculates only for the interval $[0,j]$ the most frequent state, because the values for $UP_1[j+1]$ are already stored previously. The addition related to the maximal gain of the two intervals amounts to the values of UP_2, for example $UP_2[5] =$ MaximalOccurrence$([0,5])+$MaximalOccurrence$([6,9])=$MaximalOccurrence$([0,5])+UP[6]$. The other values of $UP_3,UP_4,\ldots,UP_{\text{mes}}$ are estimated in the same way.

The other property, *overlapping subproblems*, becomes important during the calculation of the most frequent state in an interval $[lb,ub]$ at each step UP_k with $k = 1,2,\ldots,$mes. As a consequence, for each interval $[i,j]$ with $i = 0,1,\ldots,N_{p^s}-1$ and $j = 0,1,\ldots,N_{p^s}-1$ the according value of the maximal occurrence of a state will be stored in an array before the dynamic programming algorithm starts (compare Table 4.1). During the estimation of the optimal update policy, the algorithm only has to look up the corresponding value with a constant time and does not compute them over and over again.

	0	1	…	$N_{s^p}-1$
0	g_{00},idx_{00}	—	…	—
1	g_{01},idx_{01}	g_{11},idx_{11}	…	—
2	g_{02},idx_{02}	g_{12},idx_{12}	…	—
⋮	⋮	⋮	⋱	⋮
$N_{s^p}-2$	$g_{0N_{s^p}-2},idx_{0N_{s^p}-2}$	$g_{1N_{s^p}-2},idx_{1N_{s^p}-2}$	…	—
$N_{s^p}-1$	$g_{0N_{s^p}-1},idx_{0N_{s^p}-1}$	$g_{1N_{s^p}-1},idx_{1N_{s^p}-1}$	…	$g_{N_{s^p}-1N_{s^p}-1},idx_{N_{s^p}-1N_{s^p}-1}$

Table 4.1: The overlapping subproblems of dynamic programming

As the dynamic programming method reaches all possible combinations of update sequences on the calculation of UP_{mes}, the optimal one is also contained and found. Thus, it delivers the maximal corresponding revenue and the update sequence of the energy constrained problem.

The following equations present the mathematical formulation of the problem. $f_i(b_l)$ represents the maximal gain, which can be reached with i messages in the interval $[b_l,N_{p^s}]$ of the state sequence. As the dynamic programming method computes the values recursively in a bottom-up manner, it starts with $f_1(0),f_1(1),\ldots,f_1(N_{p^s}-1)$. Based hereon, the method calculates $f_2(i_2)$, which depends on $f_1(i_1)$, ..., until $f_m(i_m)$, which depends on $f_{m-1}(i_{m-1})$ with $i_1 = 0,1,\ldots,n-1$, $i_2 = 0,1,\ldots,n-2$, ..., $i_m = 0,1,\ldots,n-m$. $f_{\text{mes}}(i_{\text{mes}})$ equals the maximal utility with the first updated state at index i_{mes}:

$$
\begin{aligned}
f_1(b_l) &= \max_{i=b_l}^{N_{p^s}} \max_{j=i}^{N_{p^s}} U(i,j) \\
f_2(b_l) &= \max_{i=b_l}^{N_{p^s}-1} \max_{j=i}^{N_{p^s}-1} U(i,j) + f_1(j+1) \\
&\vdots \\
f_{\text{mes}}(0) &= \max_{i=0}^{N_{p^s}-\text{mes}+1} \max_{j=i}^{N_{p^s}-\text{mes}+1} U(i,j) + f_{\text{mes}-1}(j+1)
\end{aligned}
\tag{4.15}
$$

with

$$U(i,j) = \max_{k=i}^{j} \sum_{l=i}^{j} g(k,l) \tag{4.16}$$

$$g(k,l) = \begin{cases} 1, & \text{if } p_k^s = p_l^s \\ 0, & \text{else} \end{cases}$$

Equation (4.15) is called the *Bellmann Equation* and holds for mes ≥ 2 with known function $f_{N_{p^s}-1}$. The utility function, Equation (4.16), represents the maximal update gain in the interval with lower bound i and upper bound j with $1 \leq U(i,j) \leq j-i+1$.

Algorithm 4.5 The algorithm for the dynamic programming solution: Part I

```
1:  maxGains[][][] ← 0
2:  function DynamicProgramming(s^p, mes)
3:      intervalGains[][] ← 0
4:      for i = 0,...,N_{s^p} do
5:          for j = i,...,N_{s^p} do
6:              if i = j then
7:                  intervalGains[i][j] ← 1
8:              else
9:                  intervalGains[i][j] ← intervalGains[i][j−1]
10:                 if s_i^p = s_j^p then
11:                     intervalGains[i][j] ← intervalGains[i][j]+1
12:                 end if
13:             end if
14:         end for
15:         for j = i,...,N_{s^p} do
16:             g, idx ← −1
17:             for k = i,...,j do
18:                 if intervalGains[k][j] > g then
19:                     g ← intervalGains[k][j]
20:                     idx ← k
21:                 end if
22:             end for
23:             maxGains[i][j][0] ← g
24:             maxGains[i][j][1] ← idx
25:         end for
26:     end for
27:     OptimalUpdatePolicy(s^p, mes, 0, mes−1)
28: end function
```

Based on the mathematical formulation, Algorithm 4.5 presents a pseudo code for the dynamic programming method. The starting function is *DynamicProgramming* in line 2 with the two input parameters *state sequence* s^p and *message to send* mes. First of all, it determines the global array $maxGains[i][j][k]$

with $i, j = 0, 1, \ldots, N_{p^s} - 1$ and $k = 0, 1$, which stores the maximal occurrence of a state in the interval $[i, j]$ and the according update index. Further, the algorithm calls the function *OptimalUpdatePolicy* in line 27, which calculates in a bottom-up process the $UP_1[i_1], UP_2[i_2], \ldots, UP_{\text{mes}}[i_{\text{mes}}]$. At the end, $UP_{\text{mes}}[0]$ holds the optimal update gain for the state sequence s^p with mes update messages. If the corresponding points of the optimal update gain is also needed, the UP has to save the indexes in line 4 and 17 of Algorithm 4.6.

One of the main differences to the brute-force Algorithm 4.3 is the determination of the array *maxGains*, which replaces the function *CalculateMaxStateFrequency*. Another one is that the calculation of UP_i is based on the stored values of UP_{i-1} with $i = 2, 3, \ldots, \text{mes}$ rather than the repeatedly function call *OptimalUpdatePolicy*. For that reason, the elapsed time of the Algorithm 4.5 is greatly reduced. The function *OptimalUpdatePolicy* has a big O notation of $O(\text{mes} \cdot N_{p^s}^2)$ and the function *DynamicProgramming* of $O(N_{p^s}^3)$. So the running time is transformed from exponential (brute-force) to polynomial (dynamic programming) to the disadvantage of storage space. But this is a negligible drawback of the dynamic programming algorithm, because the used storage is bounded by $O(N_{p^s}^2)$. By the help of

Algorithm 4.6 The algorithm for the dynamic programming solution: Part II

```
 1: function OPTIMALUPDATEPOLICY(s^p, mesCnt, m, k)
 2:     if m ≤ 0 then
 3:         for i = k, ..., (N_{s^p} − m) do
 4:             UP_m[i] ← maxGains[i][N_{s^p} − 1][0]
 5:         end for
 6:         OPTIMALUPDATEPOLICY(s^p, mesCnt, m + 1, k − 1)
 7:     else
 8:         if m < mesCnt then
 9:             for i = k, ..., (N_{s^p} − m) do
10:                 max, idx ← −1
11:                 for j = i, ..., (N_{s^p} − m) do
12:                     if maxGains[i][j][0] + UP_{m−1}[j + 1] > max then
13:                         max ← maxGains[i][j][0] + UP_{m−1}[j + 1]
14:                         idx ← j
15:                     end if
16:                 end for
17:                 UP_m[i] ← max
18:             end for
19:             OPTIMALUPDATEPOLICY(s^p, mesCnt, m + 1, k − 1)
20:         end if
21:     end if
22: end function
```

dynamic programming, the energy constrained problem is solved in polynomial time for a given known state sequence. In some cases, there are more than one optimal update sequence, which results from different update times with the same update gain. Therefore, the Algorithm 4.5 can be easily extended to yield all optimal sequences. As the solution of dynamic programming is used as a reference for the accuracy gain of LP, NLP, and CMDP, it is sufficient to get only one optimal update sequence.

4.4 Further Approaches

The efficient energy-constrained distribution of context in mobile systems implies not only the trade-off between a given energy budget and the optimal accuracy, but also the reverse case. For this reason, the following section answers the question of the required energy budget for a specified accuracy on the one hand and considers weighted contextual information on the producer-side, on the other hand. In both cases, the above mentioned approaches are reformulated to match these altered problems. At first, the question of the required energy budget is characterized with the aid of LP and DP. Subsequently, the weighted context problem is described on LP.

Generally, the problem searches for the maximal achievable accuracy under a given energy budget.

maximize the accuracy function
subject to the energy budget

As the reverse case seeks for the minimal required energy budget under a given accuracy, the objective function and the constraint from the original problem are easily transposed:

minimize the energy budget
subject to the accuracy function

On this account, the transcribed linear programming is stated as follows:

$$\begin{aligned} &\text{minimize } \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j \\ &\text{subject to } \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}(1-u_j)\frac{1}{1-p_{jj}} \geq A \\ &\text{and } 0 \leq u \leq 1 \end{aligned}$$

Now, the objective function is formed of the energy budget, whereas the required accuracy constitutes the constraint. The LP is thereby optimizing the required energy consumption related to a predefined accuracy. For example, a consumer allocates the producer to provide an entire accuracy of 80%, wherefore the corresponding points in time for an update must be determined. The similarity of this method results from the reversion of the objective and constraint related to the distribution problem. To determine the minimal required energy budget, the dynamic programming approach is also used in this case. The Algorithm 4.5 is extended simply with an if-statement to check whether the desired accuracy is achieved with the current amount of updates or not. Therefore, the starting function *DynamicProgramming* is called with the parameter s^p and *mes*, whereat *mes* conforms to the maximal number of updates ($mes = CS$, compare Equation (2.1)). Function *OptimalUpdatePolicy* proofs on the basis of the the current computed $UP_m[i]$ with $i = 0, 1, \ldots, N_{s^p} - m$, if the specified accuracy is fulfilled and the algorithm concludes. Otherwise it proceeds with the recursive function call in line 19.

The second extension is the context weight resulting in a varied update behavior, wherefore each context is associated with a weight w_k with $k = 1, 2, \ldots, n$. The weights characterize the importance for an update of a context, which are defined before the application starts. For example, a consumer determines

the importance of the context, because he/she is only interested in certain activities like "producer misses the bus". It represents a hierarchy with very important states, which have to be updated always at occurrence, and rather uninteresting states, which should be updated scarcely. Therefore, the objective function of Equation (4.10) is modified:

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{n} sd_i w_i \sum_{j=1:j\neq i}^{n} p_{ij}(1-u_j)\frac{1}{1-p_{jj}} \\ \text{subject to } & \sum_{i=1}^{n} sd_i \sum_{j=1:j\neq i}^{n} p_{ij}u_j \leq E \\ & \text{and } 0 \leq u \leq 1 \end{aligned}$$

In the same way, the other methods can be easily extended to perform the reverse or weighted case.

4.5 Summary

This chapter introduced the developed solution for the efficient energy-constrained distribution of contextual information in mobile systems, that satisfies the identified requirements in Chapter 2. Summarizing, a context recognition software detects at discrete times t the producer's current context (e.g. activity). A consumer is interested in the contextual information as to why a distribution happens. Due to the mobile characteristics of the producer, the energy consumption related to the message transmission is restricted by an energy budget E. The energy constraint E identifies the overall number of update messages and is too low to transfer every context change. For this reason, a decision maker determines on-line the optimal times for an update to achieve the maximal possible accuracy A. Since the decision is taken directly after the occurrence of a context change, it needs to be assisted to find the optimal points in time for an update in the dynamic process. Therefore, the developed probabilistic approach identifies an update policy u, which constitutes the update probability for each context.

To make general statements of the discrete, stochastic process, the producer's behavior is modeled as a Markov chain. Subsequently, the user behavior is analyzed to identify significant producer characteristics, which relies on different subfields of mathematical optimization. The proposed optimization techniques, namely linear programming, nonlinear programming, and constrained Markov decision process, establish diverse update policies which each delivers different accuracies under a given energy budget. Based on the variety, a fourth off-line approach is developed to compare the on-line results of the constrained optimization problem. The dynamic programming gets the entire passed through context sequence of the producer and calculates the maximal achievable accuracy under the specified energy budget. Moreover, the optimal times for an update are attained to analyze the on-line distribution.

Since the main focus of attention is on the problem to find the best accuracy subjected to a predefined energy constraint, Section 4.4 addresses the question of further approaches. In respect to this, the reversed case as well as a weighted approach are very interesting.

5 Implementation

The proposed problem-solving approach is theoretically presented and discussed in Chapter 4. The biggest impact on the determination of the update times has the individual optimization techniques, which calculate different Markov-optimal policies. Therefore, the advised optimization methods are not only evaluated in a theoretical manner. For this, a framework is implemented to simulate and evaluate the designed approaches against different scenarios. Each scenario rely on a dataset, which abstracts the real world information to relevant attributes of the test environment. The underlying dataset, as well as the implemented simulator, are introduced and presented shortly. After this overview, the used metrics and the parameters for the evaluation are outlined. The analysis enables the measurement of the method quality, which is displayed and discussed in Chapter 6.

5.1 CenseMe

The evaluation of the designed model is based on a dataset from the project CenseMe, which is developed as a social network. The basic concept of the project is the sensing extension of the well-known functionality related to a common social network (like Facebook[1], which are, for example, messages, groups, or photos. The sensing capability enables the users to share automatically their current activity and environment with their friends over the Internet, whereby the people-centric sensing application CenseMe is installed on a sensor-enabled mobile phone. The software together with the available sensors (e.g. compass, GPS, or accelerometer) provides a context recognition, whereupon the sensing, classification and presentation of the user's context is performed [MLF+08].

With the aid of the CenseMe application running on an off-the-shelf sensor-equipped mobile phone, a user study is accomplished, which delivers the corresponding dataset for the evaluation of the proposed system model in this thesis. Therefore, 12 undergraduate and 7 graduate students, 1 research assistant, 1 staff engineer, and 1 professor from the campus town at Dartmouth College carry a Nokia N95 and execute continuously the CenceMe software over a time period of three weeks. This user study is the first one applying a personal sensing application to many people, which are continuously sensed for a long time. The presented results rest upon the university scenario of students and staff, which stay at a narrowed geographical area like small communities or cities. However, to the best knowledge of the author, the dataset is the only public available recording, which represents human activities and movements. The according dataset can be downloaded from the project page [CEN].

[1] http://www.facebook.com/

The framework learns the required user behavior from the dataset of the CenseMe project [MPF+08] and generates the Markov chain, which delivers the prediction model of the producer. Below, the main characteristics and the statistical properties related to the stream of data are outlined:

- The dataset comprises four different real traces with the sensed states *Sitting (0)*, *Running (1)*, *Walking (2)*, and *Standing (3)*. Consecutively, the evaluation is based on an extract of a real trace.
- The stationary distribution of each state is $sd_0 = 0.53$, $sd_1 = 0.02$, $sd_2 = 0.21$, and $sd_3 = 0.24$ illustrating the circumstance that the main activity of the sensed end-user is *Sitting*. This state dominates the occurrence frequency of the entire trace with a probability greater than 0.5, which means that more than every second state corresponds to *Sitting*.
- The self-transition probability characterizes the chance to stay within one time step in the same state and equals for each state $p_{00} = 0.89$, $p_{11} = 0.68$, $p_{22} = 0.65$, and $p_{33} = 0.57$. The high values related to each state (p_{ii} with $i = 0,1,2,3$) compared to the low corresponding leaving probabilities (p_{ij} with $i, j = 0,1,2,3$) are owed to the human characteristic that once the end-user enters a state it remains in it for a long time. In conjunction with the high stationary distribution sd_0, the dominating factor of state 0 is also acknowledged, especially with an accordant remaining probability of nearly 0.9.
- The transition probability matrix P characterizes the different feasibilities to shift from one state to another in one time step. For a better understanding, Figure 5.1 depicts the state transition diagram as well as the matrix P.

$$P = \begin{pmatrix} 0.89 & 0.00 & 0.02 & 0.09 \\ 0.00 & 0.68 & 0.28 & 0.04 \\ 0.05 & 0.02 & 0.65 & 0.28 \\ 0.19 & 0.00 & 0.24 & 0.57 \end{pmatrix}$$

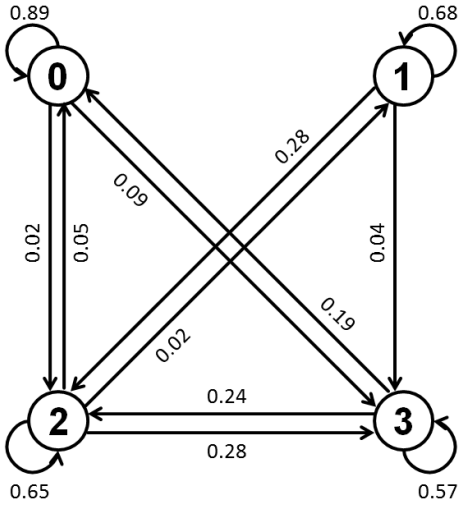


Figure 5.1: The transition probability matrix of the CenseMe dataset

- The number of state changes *CS* defined in Equation (2.1) denotes another important attribute, since it constitutes the maximal amount of updates achieving the maximal accuracy
- The cardinality of state sequence N_{sp} represents the entire visited states of the simulation

All these characteristics have a big influence on the following evaluation and are discussed as well as accounted for in the results based on the prototypical implementation.

5.2 Implementation Details

A prototypical implementation of the proposed system model is realized to measure the quality of each method and compare them among each other. Hence, an overview of some implementation details as well as the main components follows.

The major part of the framework is implemented with the object-oriented programming language JAVA[2], which comprises the following five classes[3]:

- **Import**: This class constitutes the interface between the framework and the CenseMe dataset, wherefore it includes a routine to import the context sequence. Upon this information the user behavior (Markov chain) is learned and generated.
- **Producer**: This elementary class implements the decision maker and delivers at each discrete time the current context, which is attained either in a probabilistic manner from the Markov chain or with the real passed trough context sequence from the dataset. Moreover, it determines probabilistically the instant of time for a context update with the help of the update policy.
- **Server**: The Server displays the interconnection between the producer and consumer. Therefore, it implements the functionality of a reliable and robust connection, which avoids the true one-to-one communication and transmits each message in the right order, uncorrupted, etc.
- **Consumer**: This class is the counterpart of the Producer and wants to keep up-to-date of the producer's contextual information with a feasible high accuracy. It "receives" the updated context and counts the number of misses, hits, and messages. Hence, it provides the basic data for the evaluation as it measures the performance of each optimization method.
- **Simulation**: The basic class Simulation serves as an entry point for the framework and contains all the required parameters and classes. It initializes and starts the simulation several times to compute the arithmetical mean due to the probabilistic approach.

The second part of the framework forms the generation of the update policies, which are transferred from a back-end server to the producer before the simulation starts (compare Section 4.1). As the solution of a linear and nonlinear programming requires special software, the "Optimization Toolbox" from MATLAB[4] is used rather than an own implementation of the simplex algorithm of Dantzig or an interior point method. It facilitates a multiplicity of algorithms for standard and large-scale optimization, constrained and unconstrained optimization, or continuous and discrete problems.

For this reason, an interface between Java and MATLAB is required and implemented to provide the producer class with the calculated update policy. As the exchange of data comprises only the update policy, the interface between them is kept simply by a string transmission.

[2]For further information: http://www.oracle.com/technetwork/java/index.html

[3]Notice, the name of a class begins with a capital letter (e.g. *Producer*), whereas a lowercase stands for the system component (e.g. *producer*), such as *Producer* $\neq$ *producer*.

[4]For further information: http://www.mathworks.com

5.3 Methodology

For the evaluation, the most significant measured values for the comparison of the different update policies are the attained relative *hits*, relative *misses*, and relative *messages* per run. The consumer records these outcomes, which each corresponds to the defined requirements and is relative to the number of total context changes (*CS*). The *hits* along with the affiliated *misses* reflect the accuracy, whereas the *messages* symbolizes the energy budget. As the problem-solving approach is probabilistic, the simulation is performed several times to determine the arithmetical mean of the three values. The arithmetical mean Σ is calculated from the sample space $\{\sigma_1, \sigma_2, \ldots, \sigma_{cyc}\}$

$$\Sigma = \frac{1}{cyc} \sum_{i=1}^{cyc} \sigma_i$$

For example, the relative *hits* Γ_Σ computes with the measured values of each single run $\{\gamma_1, \gamma_2, \ldots, \gamma_{cyc}\}$ as follows:

$$\Gamma_\Sigma = \frac{1}{cyc} \sum_{i=1}^{cyc} \frac{\gamma_i}{CS}$$

with γ_i, $i = 1, 2, \ldots, cyc$ corresponding to the reached context consistency between the producer and consumer (compare Equation (2.2)). The arithmetical mean of the relative *messages* is computed in the same manner. However, the *misses* results from the hits by subtraction (*misses*$= 1 - $*hits*), why the evaluation of the different optimization techniques is only based on the relative hits and messages.

6 Evaluation

The purpose of evaluation is to depict the pros and cons of the approaches developed, whereby the results of the different Markov-optimal update policies are shown and discussed on a realistic trace. First of all, each single update policy of the linear programming, the nonlinear programming, and the constrained Markov decision process is evaluated alone compared to a lower and upper bound to present the strengths and weaknesses. Therefore, Section 6.1 describes the selection and outcomes of the bounds. Afterwards, all the results from the different policies are put together for a better comparison and discussion, wherefore each single evaluation is based upon the same passed through context sequence. At this, two miscellaneous scenarios from the real user trace are applied to characterize the particular performances. For both, the user behavior is learned and generated from a subset of the CenseMe dataset, which delivers for each scenario the Markov model. The difference is the determination of the next user context:

- **Scenario 1**: The next context is determined with the aid of the achieved transition probabilities from the Markov chain.
- **Scenario 2**: The next context is determined from a subset of the CenseMe dataset and represents a real chronology.

Due to the probabilistic approach, a parameter is used for the simulation, which constitutes a watchdog for the decision maker. On the one hand, the watchdog admits only to send as many updates as allowed to fulfill the energy budget. This case is labeled by the subscript R. On the other hand, the decision maker has a rather casual attitude to the energy limitation and sends as many updates as the probabilistic policy wants. This circumstance is called non-restricted. As the user behavior is generated from a real trace, it can still happen that the run through context sequence conforms better or worse to the transition probabilities and stationary distributions characterized of the Markov chain. To identify this, the "restricted" and "non-restricted" case is displayed. So the simulation analyzes the satisfaction of the identified requirements and ambitions, whereby the best optimization technique emerges.

In detail, the parameter *cyc* represents the quantity of simulation runs and is set to 50 to calculate the average of the probabilistic approach. The cardinality of the state sequence amounts to 5000 for both scenarios. The number of state changes is 1212 in scenario 1 and 1197 in 2 respectively. As a result, an energy budget of 1.0 corresponds to 1212 and 1197 messages, whereas the accuracy of 1.0 equals to 5000 contexts (compare Chapter 2). To get an overview of the quality related to each optimization technique, the different results are compared for an energy budget of 0.0 to 1.0 along with the achieved accuracy. Hence, the x-axis (axis of abscissas) related to each diagram of the evaluation persists of the relative energy budget E, whereas the y-axis (axis of ordinates) illustrates the relative hits or messages. Every section contains a figure for the results of scenario 1 and 2, which is sub-divided into 4 figures. These illustrate the energy-accuracy trade-off, the energy-update trade-off, and the accuracy benefit in relation to the lower bound and to the upper bound.

6.1 Bounds

In order to show the benefits and drawbacks of the developed problem solving approach, a lower and upper bound are presented for the simulation results. The bounds serve as a comparative value for the proposed methods and illustrates the value of effort.

The simplest and clearest lower bound related to an energy-constrained distribution is the consequent update of context changes until the allocated energy budget is consumed, called *FireAll* (compare Subsection 4.3.1). This method does not require any optimization and expects only the amount of allowed updates for input. Generally, it achieves in the first half of the sequence an accuracy of 1.0, whereas the latter part relies extensively on the last updated context. If the final transferred context has a great appearance in the remaining sequence, the accuracy error is rather low, and vice versa. This circumstance occurs in scenario 1 with an energy budget of 0.4 and an accuracy of nearly 0.7. It constitutes a rise of about 0.22 against the previous energy budget and even an increase of 0.07 compared to the direct successor (see Figure 6.1). At the beginning, *FireAll* attains an accuracy of about 0.34 and grows nearly linear until all context changes are updated achieving an accuracy of 1.0.

Another obvious method for the energy-constrained distribution without optimization is a periodic approach, which updates each context after a constant waiting period, called *Periodic* (compare Subsection 4.3.2). It also requires only the amount of allowed updates for input and transfers periodically the current context. *Periodic* starts with an accuracy of 0.4 at the energy budget of 0.1 and increases linear, too. The difference is that the gradient of *Periodic* is lower than *FireAll*. Moreover, *Periodic* scores only an accuracy of 0.7 at the end, whereas *FireAll* achieves 1.0. This is due to the fact that *Periodic* is based on a temporal behavior rather than the context changes. As *Periodic* attains a higher accuracy than *FireAll* at the beginning, the intersection is at the energy budget of 0.32. Afterwards, *Periodic* achieves a lower accuracy until the end. Both methods can be used as a lower bound for scenario 1, wherefore it is important to await the outcomes of scenario 2.

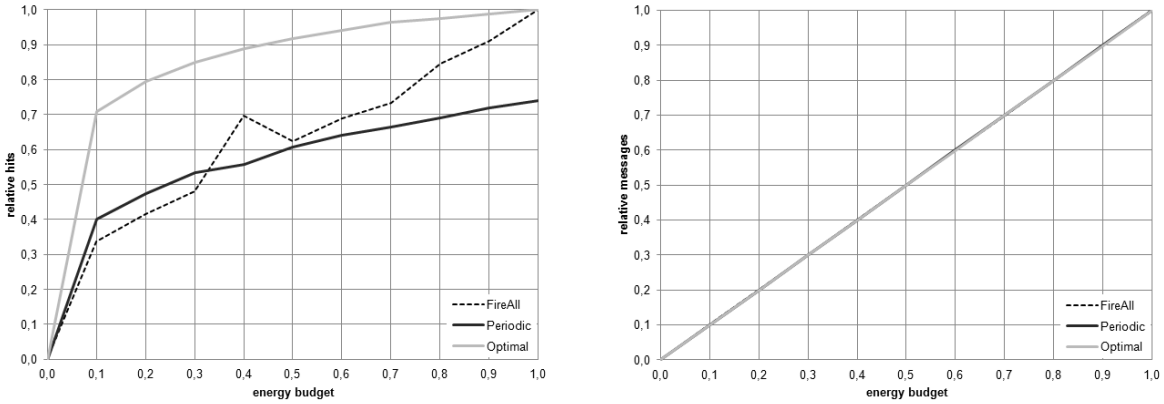


Figure 6.1: *FireAll*, *Periodic*, and *Optimal*: the simulation results related to scenario 1

The choice of an adequate upper bound is easier. The dynamic programming, which is described in Subsection 4.3.6, computes the optimal accuracy related to an energy budget. Therefore, it is called *Optimal* and starts with an accuracy of 0 as the allowed amount of updates is also 0. In the next

step, the accuracy increases quickly to 0.7. After that, the curve of *Optimal* grows also nearly linear with a constant gap (0.3) compared to *Periodic* and attains an accuracy of 1.0 at the end. The high accuracy gain with a low amount of updates relies on the one-sided stationary distribution and self transition probability of the real trace. The great occurrence probability related to context *Sitting* (compare Section 5.1) enables this benefit, which is acknowledged by the achievements of *Periodic*. The probability to hit the context with the highest appearance in each update interval is big enough (0.53) so that even a not-optimized approach transmit the right one. Summarizing, the left-hand side of Figure 6.1 illustrates the area, in which the values of the different techniques should stay at the best. For the sake of completeness, the right-hand side displays the consumed messages, which are equal for all three methods and stay within the limits.

In scenario 2, the outcomes related to the techniques without any optimization differ highly, which is illustrated by the big gap between *FireAll* and *Periodic* in Figure 6.2. At first, *FireAll* achieves only an accuracy of 0.27, whereas *Periodic* has 0.67. This deviation amounts to 0.4 and remains the same until an energy budget of 0.3. Afterwards, *FireAll* recovers slowly and decreases the gap to 0.3 until an energy budget of 0.6. With an increasing energy budget, this decline is reduced explosively and the intersection of *FireAll* and *Periodic* is at 0.9. The good performance of *Periodic* is due to the above mentioned characteristics of the real trace. Only with an energy budget greater than 0.9, *FireAll* achieves a higher accuracy than *Periodic*. Nevertheless, *FireAll* is not usable as a lower bound, because it would have a too low comparative value related to each energy budget. For this reason, *Periodic* is selected as the lower bound in the following evaluation. Only for the sake of completeness, each figure contains in the subsequent sections the outcomes related to *FireAll*.

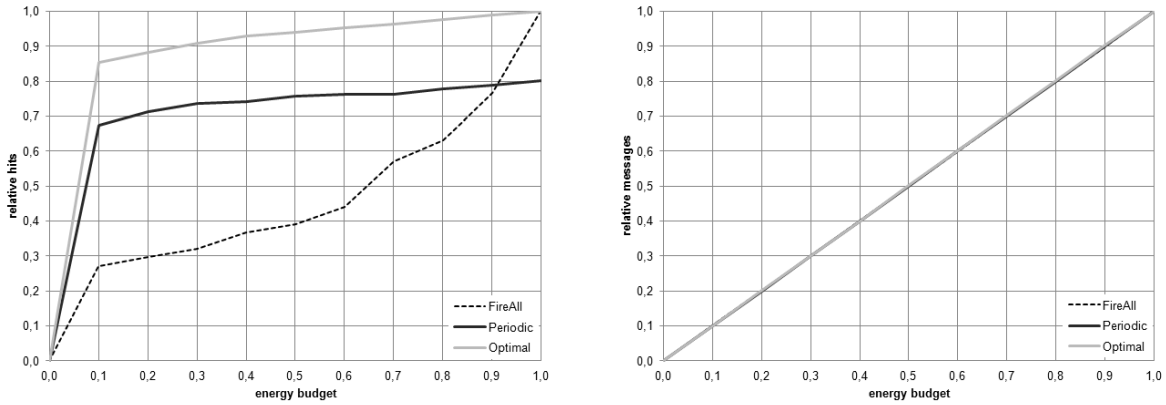


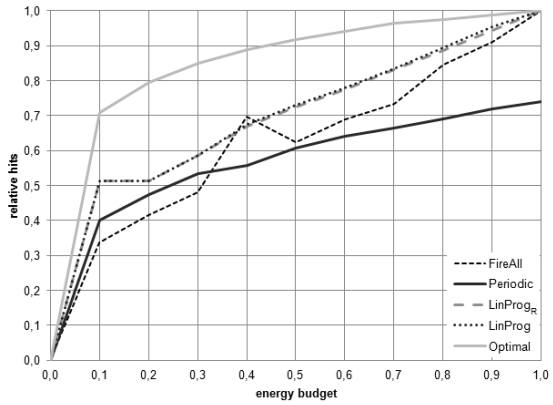
Figure 6.2: *FireAll*, *Periodic*, and *Optimal*: the simulation results related to scenario 2

Especially as the outcomes of *Optimal* and *Periodic* are similar to the simulated scenario. Only the effect related to the one-sided stationary distribution is more significant at an energy budget of 0.1, which causes an accuracy gain of 0.67 with *Periodic* and 0.85 with *Optimal*. Thereafter, the gap of 0.18 remains nearly constant from energy budget to energy budget. At the end, *Periodic* achieves only an accuracy of 0.8. However, *Optimal* attains the entire achievable accuracy of 1.0 by updating all of the context changes. Hence, the left-hand side of Figure 6.2 shows these circumstances and marks the limits for the developed optimization techniques in the following sections. The right-hand side presents the amounts of used messages, which stay for all three methods within the limits.

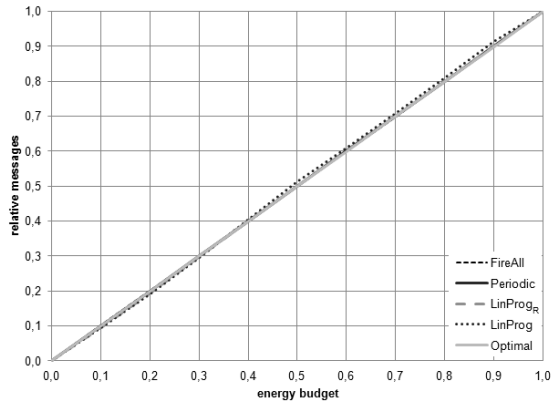
6.2 Linear Programming

The simulation results from scenario 1 and 2 of the update policy generated from linear programming are visualized, whereat the Figure 6.3 and 6.4 contain the outcomes of the periodic method (*Periodic*), the "restricted" linear programming ($LinProg_R$), the "non-restricted" linear programming (*LinProg*), and the optimal method (*Optimal*).

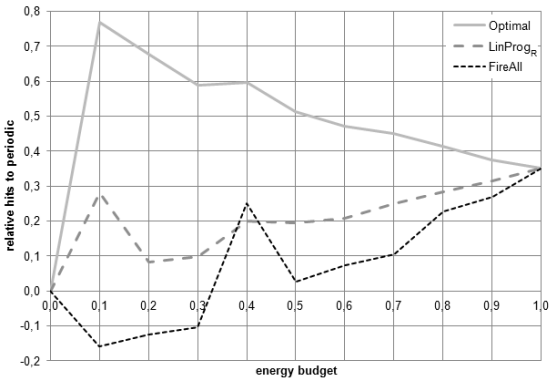
It is obvious, even at first glance of Figure 6.3a, that $LinProg_R$ as well as *LinProg* remain within the bounds and are strongly similar to each other. This is not surprising as the constrained optimization problem is limited by the energy budget, wherefore the probabilistic approach causes only in special sequences a greater energy consumption than allowed. Moreover, the update behavior of $LinProg_R$ is further optimized in such a way that the advantage related to the additional updates is minimized. For example, before the entire energy budget is consumed, the last transmitted context is the most likely one, which will appear in the future and causes further accuracy gains. At the beginning, both start with 0 updates and a corresponding accuracy of 0. Increasing the energy budget about 0.1, the



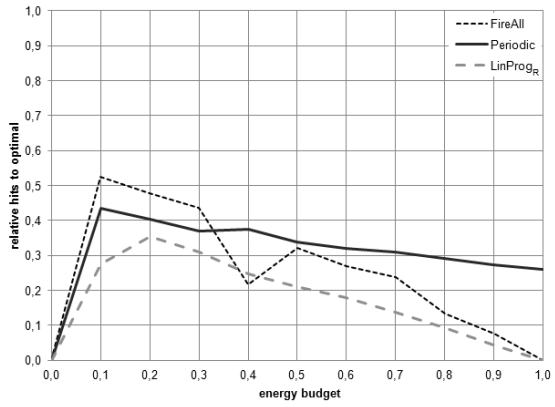
(a) relative hits



(b) relative messages



(c) relative hits to *Periodic*



(d) relative hits to *Optimal*

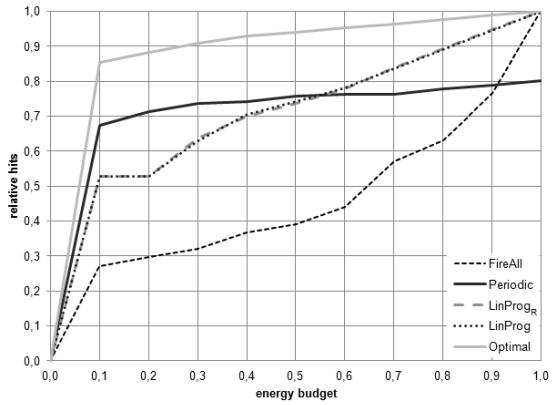
Figure 6.3: Linear programming: the simulation results related to scenario 1

accuracy takes a leap to 0.5 owing to the facts of the above mentioned stationary distribution as well as the applied optimization. With an energy budget greater than 0.1, the accuracy steps up nearly linear and ends in updating all of the context changes. However, only from an energy budget greater than 0.4, the accuracy of *LinProg* is slightly larger than the other, which affects the third decimal place. This is because the "non-restricted" case sends even more updates than allowed, which is depicted by the values of relative messages from 0.4 of Figure 6.3b. Since this thesis deals with an energy limited distribution of contextual information and the similarity of the "restricted" and "non-restricted" case, from now on there will be no further distinction in the linear scenario 1 between both.

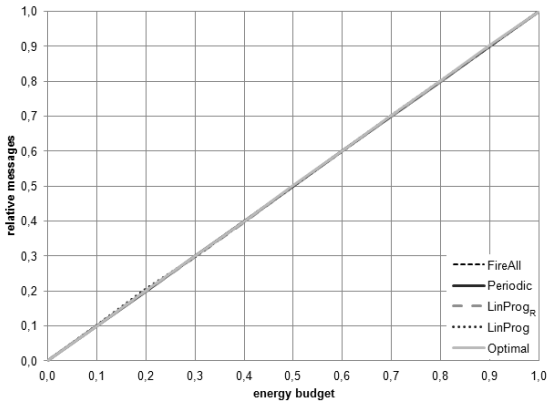
Furthermore, the simulation results show that with an energy budget of 0.1 the optimized method attains a greater accuracy of 0.1 with the equal number of updates compared to *Periodic*. This advantage continues to rise up to 0.25 until the end, which is accentuated in Figure 6.3c. It outlines the relation between the achieved accuracy of $LinProg_R$ and *Periodic* as well as *Optimal* and *Periodic*. In this connection, the benefit of the developed approach with a linear optimization is clarified as it attains a higher accuracy on average of about 20% per update. Nonetheless, the commensurate big gap between $LinProg_R$ and *Optimal* shows that there is still room for further gains, especially in view of the relation to *Optimal* (compare Figure 6.3d). This graph points up the deficit of $LinProg_R$ and *Periodic* related to *Optimal* per energy budget. For example, the $LinProg_R$ approach obtains at 0.2 its most losing in proportion to *Optimal*, which corresponds to the values 0.51 and accordingly 0.79 in Figure 6.3a. The main reason for this is that $LinProg_R$ has no further benefit of the energy budget raise from 0.1 to 0.2. The additional updates enhance only the update probability of the most likely context resulting in an increased context distribution, which does not influence the accuracy. This circumstance offers a good example of the reversed optimization described in Section 4.4. In order to achieve an accuracy of 0.5, the minimal required energy budget corresponds to 0.1.

Scenario 2 is based on the traversed real context sequence and shows in some cases equality and in others disparity (see Figure 6.4). The most obvious difference is the intersection of *Periodic* and $LinProg_R$, which implies that at the beginning the linear optimization is worse and at the end better, or vice versa. Figure 6.4a, 6.4c, and 6.4d illustrate this circumstance and depict that the method without optimization has an advantage over the linear optimized approach in the energy interval $[0.0, 0.55]$. From then on, the developed optimization comes into complete effect and raises rapidly with an accuracy gain of about 0.05 per energy budget. Comparing Figure 6.3a with 6.4a, the main reasons for this are the accuracy growth at 0.1 of *Periodic* and *Optimal*, on the one hand. Both increase from 0.4 to 0.67 and from 0.7 to 0.85 respectively. On the other hand, the approximative unchanged accuracy related to $LinProg_R$/*LinProg* of 0.51 and accordingly 0.53 is the other reason. This is mainly due to the above explained one-sided stationary distribution combined with the high self transition probability of context *Sitting*. Primary, the linear optimization differs from the periodic method as soon as the fine grained characteristics of the sequence takes up. Here, *Periodic* is not longer able to update the context with the most impact to the accuracy (compare the results of an energy budget greater than 0.55).

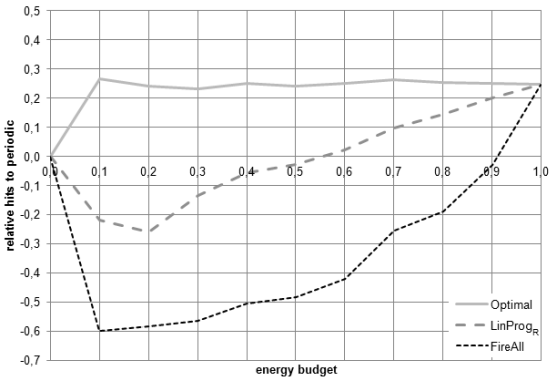
Obviously, $LinProg_R$ and *LinProg* act also similar apart from a few negligible values in this scenario (compare Figure 6.4a and 6.4b), wherefore the "restricted" and "non-restricted" methods are treated the same way. But far more interesting is the consideration of $LinProg_R$ to *Periodic* and *Optimal*. Figure 6.4c displays the deficits of $LinProg_R$ in the first half of the energy budget, since the curve should run above the x-axis. Its minimum achieves the approach at 0.2, which is also true for the relation to *Optimal*. The primary issues for this behavior are the same circumstances discussed above for the



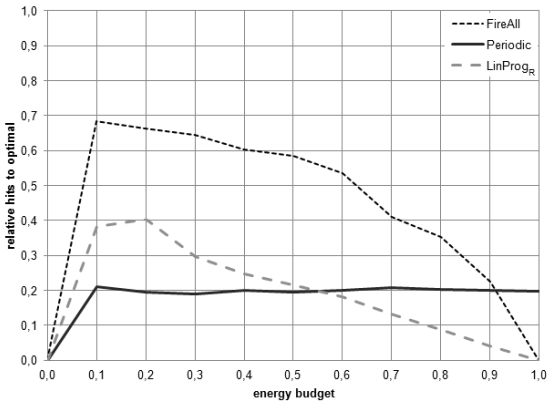
(a) relative hits



(b) relative messages



(c) relative hits to *Periodic*



(d) relative hits to *Optimal*

Figure 6.4: Linear programming: the simulation results related to scenario 2

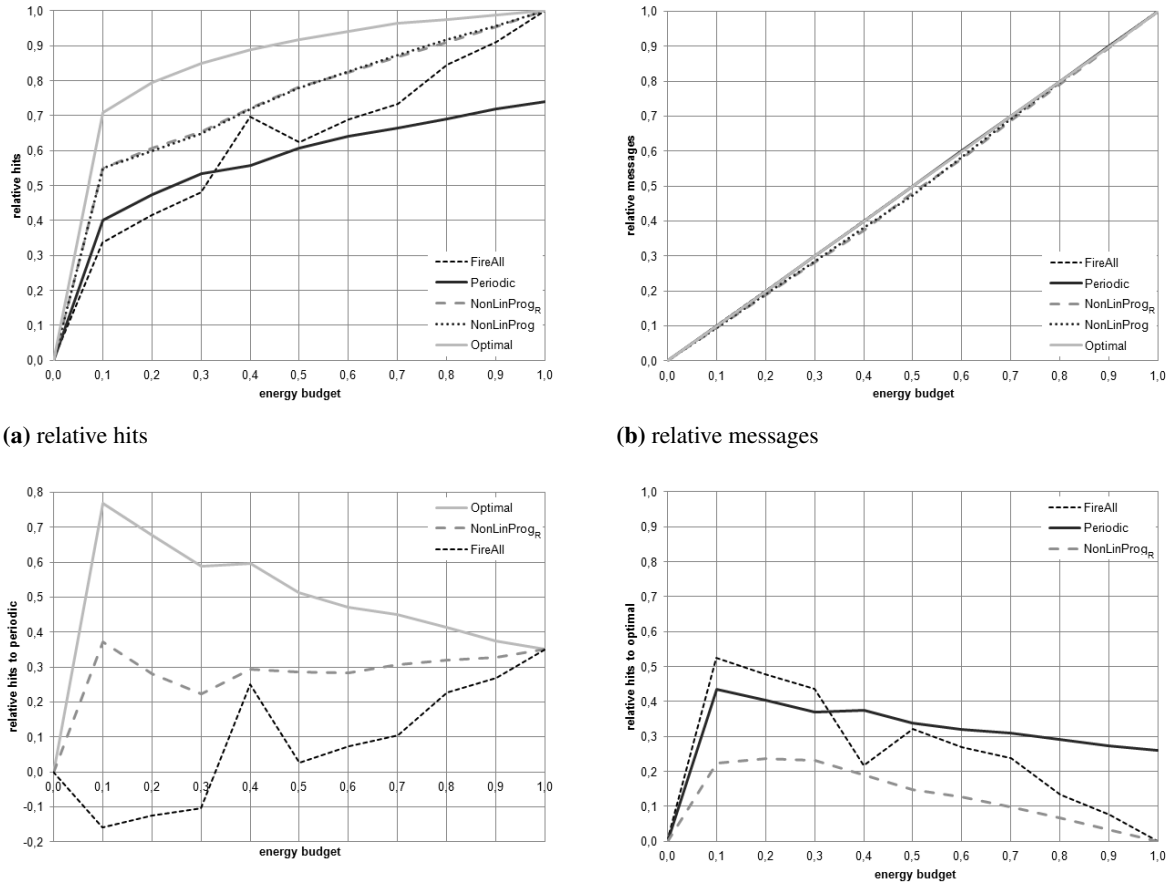
simulated scenario (compare the raising of the energy budget from 0.1 to 0.2). The only difference is that the achieved accuracy of $LinProg_R$ is about 0.4 lower than the optimal once. However, *Periodic* attains only a loss of 0.2.

Summarizing, it can be said that in both scenarios the "restricted" and "non-restricted" linear method have no significant differences and are not further distinct. Furthermore, the better results related to the simulated sample are associated with the more accurate description of the context sequence by the user behavior. This is because only the Markov chain is learned from the real trace, whereas the passed through context sequence is generated from it. Therefore, the sequence behaves Markovian and the optimization is fully exploited. Nonetheless, the real scenario demonstrates the benefits of the linear optimization, which reaches for an energy budget greater than 0.55 with the same account of updates remarkable successes. The first half of scenario 2 as well as the results from scenario 1 clarify the fact that there are further improvements of the linear method possible to achieve an even better accuracy.

6.3 Nonlinear Programming

For this reason, the nonlinear optimization is proposed, which advances the erroneous behavior of linear programming discussed in Subsection 4.3.4. The detailed comparison of the linear and nonlinear method is considered in Section 6.5, wherefore an analysis of the periodic method (*Periodic*), the "restricted" nonlinear programming ($NonLinProg_R$), the "non-restricted" nonlinear programming (*NonLinProg*), and the optimal method (*Optimal*) follows.

Figure 6.5 illustrates the results of scenario 1 in the same way as for the linear comparison and depicts in Figure 6.5a the accuracy-energy trade-off for the "restricted" and "non-restricted" case. Both start with a value of 0.55 at an energy budget of 0.1 and follow afterwards a linear curve, which reaches an accuracy of 1.0 by updating each context change. This is also clarified by the nearly horizontal line of both methods at Figure 6.5c, which illustrates the obtained accuracy in relation to *Periodic*. All along the energy budget, both outcomes stay above and below the lower and upper bound.



(a) relative hits

(b) relative messages

(c) relative hits to *Periodic*

(d) relative hits to *Optimal*

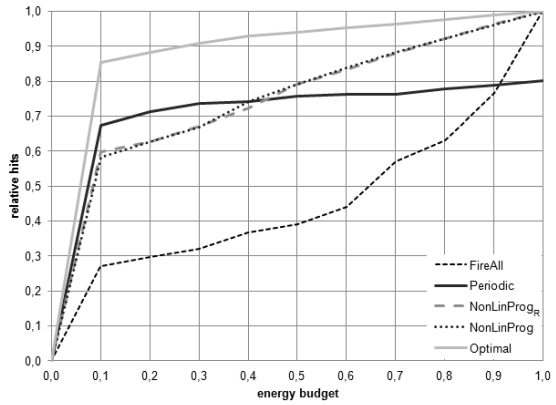
Figure 6.5: Nonlinear programming: the simulation results related to scenario 1

Especially, the analogy between *NonLinProg*$_R$ and *NonLinProg* is obvious, since the maximal deviance does not exceed 0.0065. Within the energy interval $[0.1, 0.5]$ the "restricted" method achieves an even greater accuracy than the "non-restricted" one, which in turn attains for the energy budgets of 0.6 to 0.9 a higher accuracy. During the first half of the energy capacity, the difference between *NonLinProg*$_R$ and *NonLinProg* equals to $0.0010, 0.0064, 0.0045, 0.0017, 0.0016$ and in the second half $-0.0025, -0.0059, -0.0055, -0.0038$, which is why it is impossible to identify a clear tendency. In addition, the used messages do not clarify this circumstance, which are $0.0008, -0.0033, -0.0026, -0.0056, 0.0094, -0.0040, -0.0035, -0.0048, -0.0043$. For instance, the *NonLinProg*$_R$ updates 0.0033 less context changes and gains a 0.0064 higher accuracy with an energy budget of 0.2. Typically, the "non-restricted" method should send more context changes by achieving a greater accuracy. Reasons for this variation are on the one hand the above mentioned update improvements (e.g. the last update corresponds to the most likely state). On the other hand, the passed through context sequence is well described by the Markov chain so that the extra updates of *NonLinProg* do not cause an overlarge increase. Mainly, *NonLinProg*$_R$ and *NonLinProg* do not use the entire available energy budget (see Figure 6.5b), wherewith an even greater accuracy is possible. Since both methods behave in a similar way, only the *NonLinProg*$_R$ is further considered.

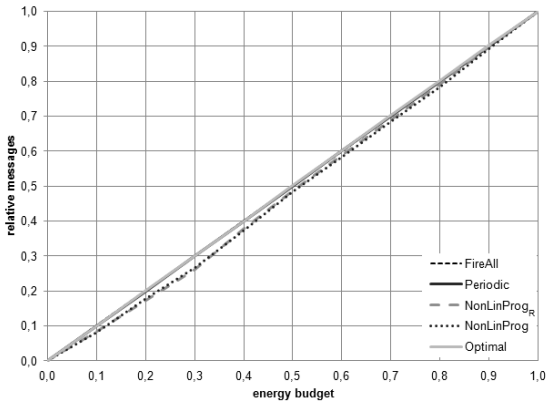
Compared with *Periodic* (Figure 6.5c) it attains with a lower (or same) amount of updates an even higher number of context equality of about 0.3 between the producer and consumer. For example, *NonLinProg*$_R$ accounts for an advantage of 0.15 with an energy budget of 0.1 (*Periodic*: 0.4, *NonLinProg*$_R$: 0.55), which remains until the end. So the proposed nonlinear optimization approach outlines an efficient improvement for energy savings in the simulated case. As *Optimal* achieves in relation to *Periodic* a maximal deviation of 0.77 with an energy budget of 0.1 and still a benefit of 0.3 at 0.4, the results of *NonLinProg*$_R$ are not in a satisfactory manner. For example, *NonLinProg*$_R$ has its maximal deficit of 0.2 at the energy budgets 0.1 to 0.4 in relation to *Optimal* (see Figure 6.5d).

At the first glance, the real trace simulation in Figure 6.6 bears resemblance to the results related to the linear optimization of scenario 2. The curves of the "restricted" and "non-restricted" nonlinear programming start with a value lower than *Periodic* (0.598 and accordingly 0.58 compared to 0.67) and have their intersections with *Periodic* at 0.43 and 0.4 respectively. Afterwards, *NonLinProg*$_R$ and *NonLinProg* achieve with a linear increase a greater accuracy, which yields to a gain of 0.2 at the end. Generally, the outcomes of both methods are very similar to the simulated scenario and contain also the up- and downturns. *NonLinProg*$_R$ attains a higher accuracy for the energy budgets $0.1, 0.3, 0.5$, and 0.9 as well as sends only at 0.2 and 0.3 less updates than *NonLinProg*. For example, the most significant point for this is at the energy budget of 0.3, where the "restricted" method has an accuracy plus by updating less messages. The amount of the deployed updates stays for both on the whole, under the permitted transmissions and has its minimum at the energy budget of 0.3 (compare Figure 6.6b). These circumstances are also based on the above mentioned complexity of the nonlinear programming and causes the variability, whereby in the following only the *NonLinProg*$_R$ is considered.

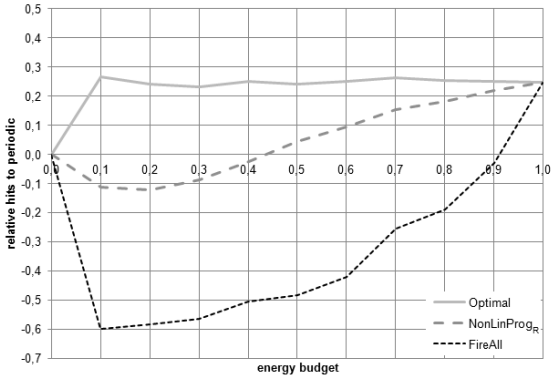
The comparison between *NonLinProg*$_R$ and *Periodic* shows that the aspired optimization does not take full advantage of the high potential already existing in it. The initial deficit related to the *Periodic* accuracy amounts to nearly 0.1 and remains even until the intersection point at 0.43 (see Figure 6.6c). Here, the characteristic of the passed through context sequence contains for an energy budget of less than 0.43 a special case, which means that a context with a high accuracy for the current update interval follows after a constant period. Afterwards, the periodic determination of the update times takes some



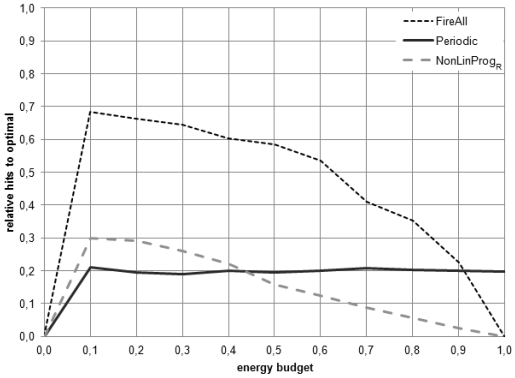
(a) relative hits



(b) relative messages



(c) relative hits to *Periodic*



(d) relative hits to *Optimal*

Figure 6.6: Nonlinear programming: the simulation results related to scenario 2

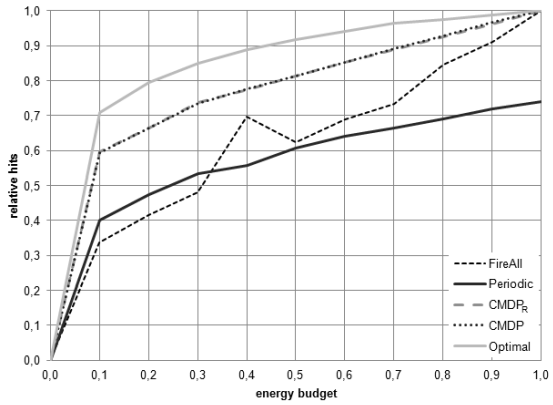
wrong decision, why the deliberate chosen points in time of $NonLinProg_R$ rise the accuracy racily. Compared to a method without optimization the nonlinear programming affords from the certain energy budget an accuracy benefit for the real trace simulation and justifies the less effort for the modeling of user behavior. Just like the linear programming, the nonlinear optimization possesses some drawbacks resulting in the accuracy misses in relation to *Optimal* (compare Figure 6.6d).

Summing up, the "restricted" and "non-restricted" case do not show any particular differences, too, which excuses a further differentiation. The better performance related to the simulated scenario relies extensively on the truthful generation of the context sequence from the Markov chain, wherefore the optimization technique presents its strength. However, the result for scenario 2 suffers only in the first half under weakness, whereas the nonlinear method has clear advantages at an energy budget of greater than 0.43. Since the main focus of this thesis is based on an "optimal" distribution of context, another approach is proposed to emerge an even better update behavior.

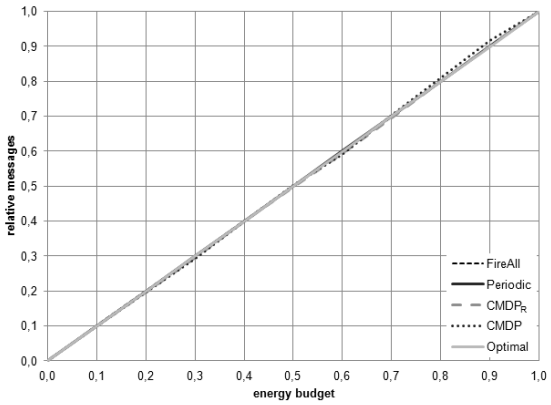
6.4 Constrained Markov Decision Process

The last developed approach for the problem utilizes a constrained Markov decision process to achieve the maximal possible accuracy between the producer and consumer, wherefore the following figures contain the lower limit (*Periodic*), the "restricted" ($CMDP_R$) and "non-restricted" (*CMDP*) constrained Markov decision process, as well as the upper limit (*Optimal*). Since the outcomes for $CMDP_R$ and *CMDP* do not differ considerably from each other in the simulated and real trace scenario (compare Figure 6.7 and 6.8), they are only illustrated for the sake of completeness. The reasons for this similarity can be translated one-to-one of the linear and nonlinear optimization, which are discussed in detail in Section 6.2 and 6.3. Therefore, this section concentrates on the results of the $CMDP_R$.

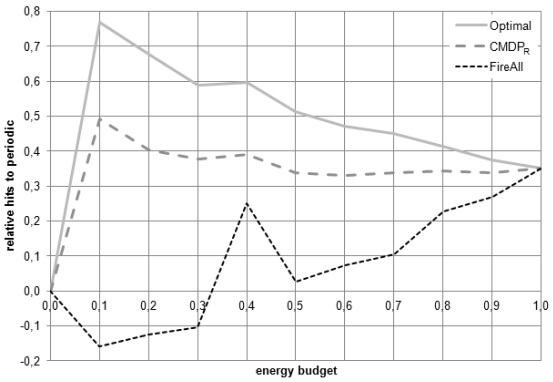
Figure 6.7 gives a familiar overview of the outputs for the simulated scenario. On the face of it, the curve in Figure 6.7a for $CMDP_R$ looks very promising due to the big distance to *Periodic* as well as the low gap to *Optimal* over the entire term. At the beginning, $CMDP_R$ reaches nearly an accuracy of 0.6, which is about 0.2 higher and only about 0.11 lower than the corresponding bounds. The spaces remain



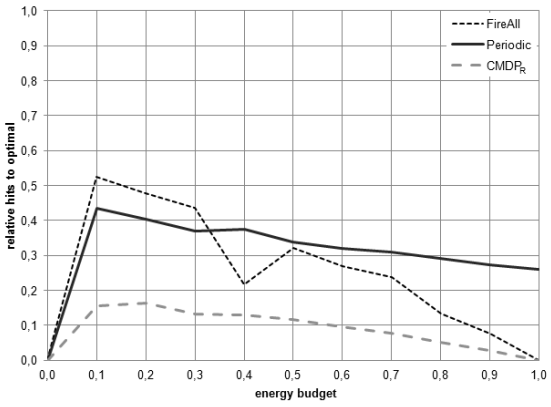
(a) relative hits



(b) relative messages



(c) relative hits to *Periodic*

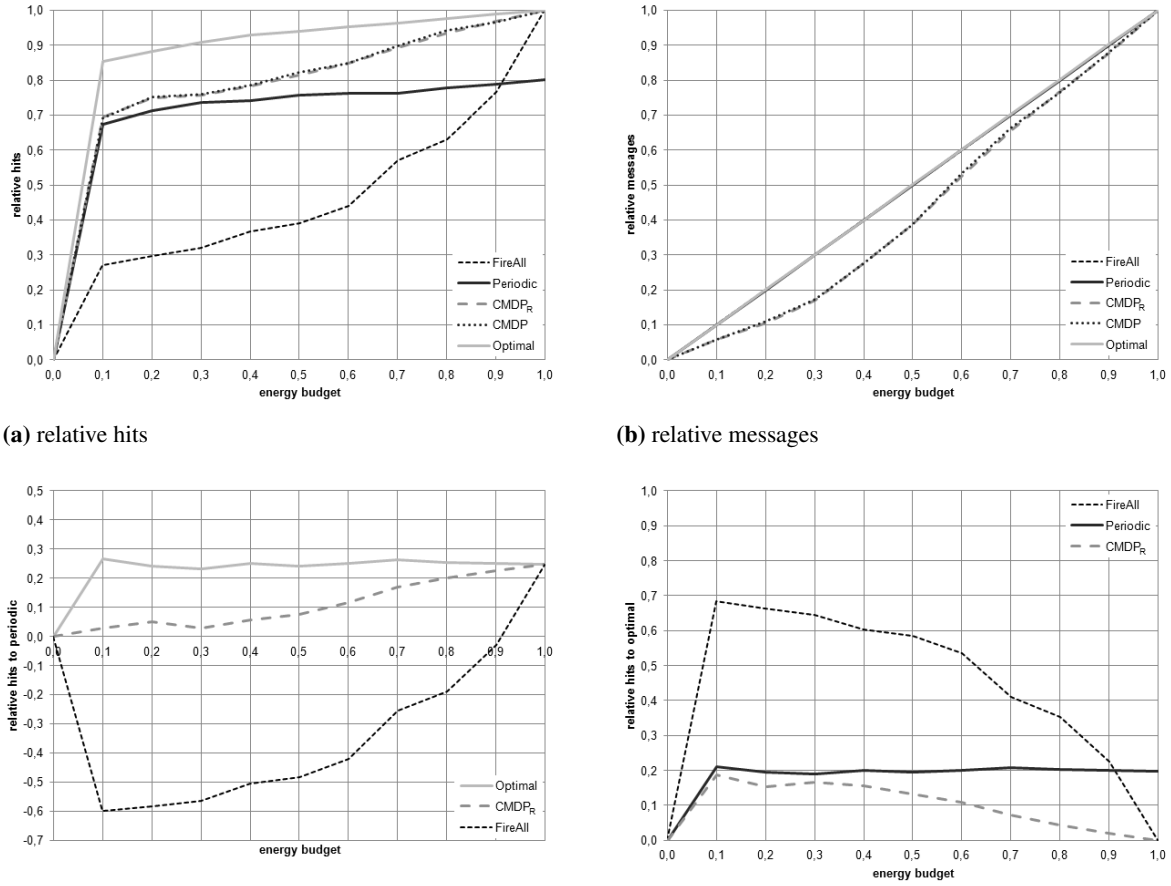


(d) relative hits to *Optimal*

Figure 6.7: Constrained Markov decision process: the simulation results related to scenario 1

approximately equal until the energy budget of 0.5, after which they decrease and accordingly increase fast. The major advantage against *Periodic* is also depicted in Figure 6.7c, which shows that $CMDP_R$ achieves an accuracy of 50% greater than *Periodic* at the energy budget of 0.1. Generally, the curves of $CMDP_R$ and *Optimal* proceed very similar in the illustration, which prove the good functionality of the optimization. The additional information related to the storage of the last updated context is a big profit and enables the constrained Markov decision process to determine accurate update times. Especially as the difference of $CMDP_R$ in comparison with *Optimal* is very low and attains only a spacing of 0.095 on average. This underscores the well-adapted mathematical formulation emerging in eminent energy savings together with high accuracy gains.

It is obvious that the second scenario presents the similar characteristics as for the simulated case and provides promising results (see Figure 6.8). Firstly, the outcomes for $CMDP_R$ remain for all energy budgets in the defined bounds and send equal or less than the allowed messages (compare Figure 6.8a and 6.8b). After the rather restrained start, at which it attains still a slight greater accuracy than *Periodic*, $CMDP_R$ draws more and more benefits of the applied optimization. The difference accounts for 0.019



(a) relative hits

(b) relative messages

(c) relative hits to *Periodic*

(d) relative hits to *Optimal*

Figure 6.8: Constrained Markov decision process: the simulation results related to scenario 2

at 0.1 (*Periodic*: 0.6742, $CMDP_R$: 0.6935) and alters for the following three energy budgets up and down. The second one (0.3) marks the beginning of an accuracy rise compared to *Periodic* about 0.177 until the end. This is also clarified in Figure 6.8c, which illustrates the accuracy gain in relation to *Periodic*. At the beginning, the curve of $CMDP_R$ is very flat and distinguishes only with a maximal value of 0.05 from *Periodic*. A reason why $CMDP_R$ does not drift highly apart until an energy budget of 0.3 from *Periodic*, constitutes the minor energy consumption, which is depicted in Figure 6.8b. Its maximal divergence of 0.13 achieves the curve at 0.3 due to the deviance of the passed through sequence and the learned interval. For example, the Markov chain is generated from a (real) traversed section, which contains a context with a big impact on the accuracy. Thus, the constrained Markov decision process allocates this context a huge update probability subject to its occurrence. Since the appearance of this context is smaller in the passed through sequence, the decision maker does not spend the entire available energy budget. However, Figure 6.8d displays explicitly the advantage of $CMDP_R$ against *Periodic*, which yields clearly more profit as higher the energy budget, the greater the trajectory converges against *Optimal*.

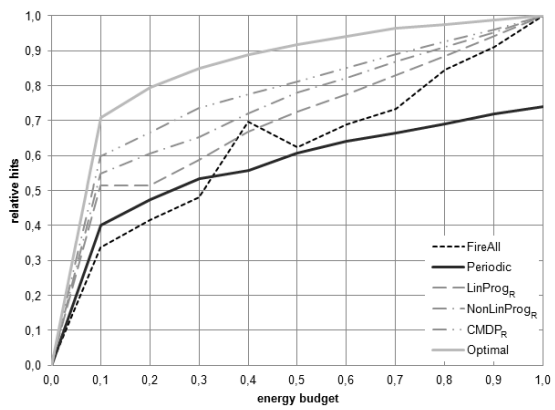
Based on the results for scenario 1 and 2, the constrained Markov decision process achieves definitely in both cases a big advantage against a method without optimization. For this reason, the application of a CMDP approach is advised for an efficient energy-constrained distribution of contextual information on a mobile phone. The achieved accuracy for the simulated case shows a great benefit for $CMDP_R$. For the real scenario, the results are not so clear until an energy budget of 0.3, but forces up gradually to attain a distinct advantage in the end.

6.5 Discussion

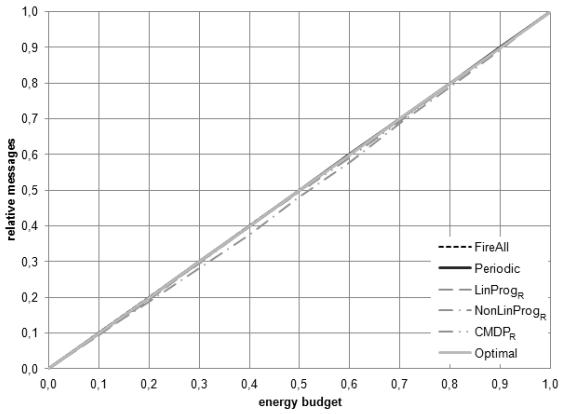
To conclude the evaluation, the subsequent section deals with the combined comparison of the above discussed optimization techniques for scenario 1 and 2. Therefore, Figure 6.9 and 6.10 contain the periodic method as the lower bound (*Periodic*), the linear programming ($LinProg_R$), the nonlinear programming ($NonLinProg_R$), the constrained Markov decision process ($CMDP_R$), and the dynamic programming as the upper bound (*Optimal*). For the sake of completeness, both figures illustrate also the results related to *FireAll* and serve as proof for the bad performance compared to the others.

For scenario 1, the outcome of *FireAll* is nearly the entire energy lower than the results of the three optimization methods (compare Figure 6.9a and 6.9d). It attains only at 0.4 a slight larger accuracy than $LinProg_R$, which follows from the luckiness that the last update of *FireAll* corresponds to the context with the highest appearance in the remaining sequence. Even in comparison with *Periodic*, *FireAll* achieves a lower accuracy until an energy budget of 0.32. Especially as the results of scenario 2 are significantly worse and approve the rightness of *Periodic* as the lower bound. At the beginning, the gap between *FireAll* and the worst optimization method ($LinProg_R$) accounts for more than 0.2, which increases during the next energy budgets to more than 0.3 (compare Figure 6.10a and 6.10d). At the end, however, the gap is still greater than 0.15 and attains only a higher accuracy than *Periodic* due to the fact that the periodic method does not achieve an accuracy of 1.0. For that reason, the *FireAll* is mentioned here only in order to explain the appliance of *Periodic*.

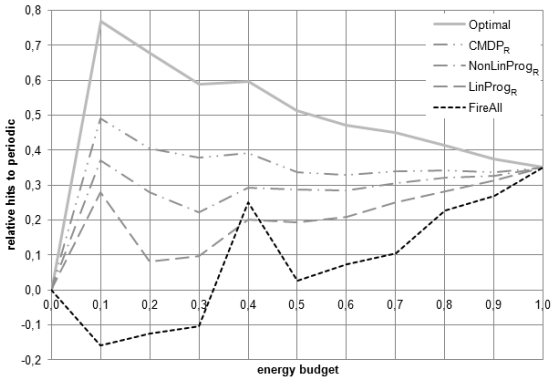
The different accuracy gains of the three optimization techniques are contrasted in Figure 6.9 for the simulated scenario, which depicts the explicit dominance of both, $NonLinProg_R$ across $LinProg_R$ and



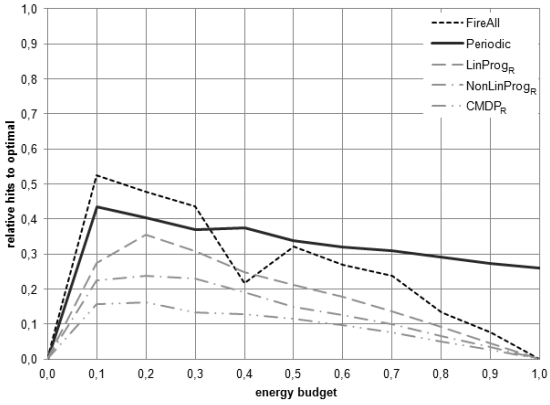
(a) relative hits



(b) relative messages



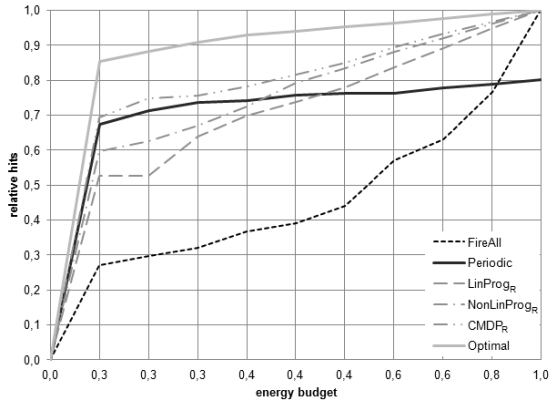
(c) relative hits to *Periodic*



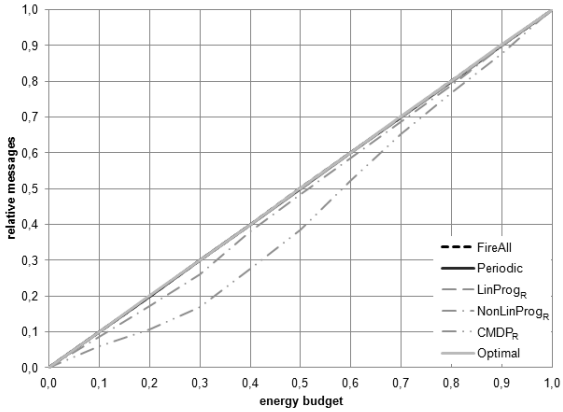
(d) relative hits to *Optimal*

Figure 6.9: Linear programming, nonlinear programming, and constrained Markov Decision Process: the simulation results related to scenario 1

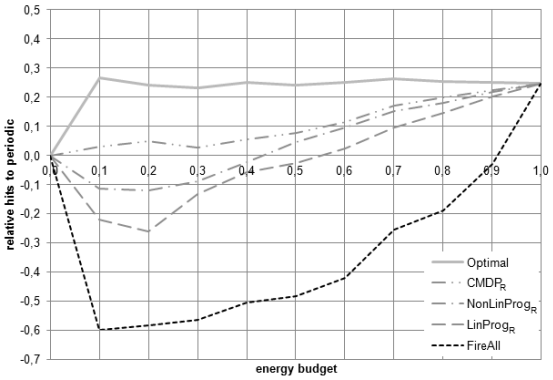
$CMDP_R$ across $NonLinProg_R$. Obviously, $NonLinProg_R$ achieves a higher accuracy than $LinProg_R$ at any time with a maximal deviation of 0.0936 at the energy budget of 0.2 and an average value of 0.0388. This is because the mathematical formulation of the linear programming does not include the accuracy gain for a transmitted context, which is revisited after a period without any updates. It only considers the equal contexts following immediately, which is why the determined update times are not so accurate (compare Subsection 4.3.4). As $NonLinProg_R$ tries for an improvement of the erroneous problem description, it attains a greater accuracy with the equal number of updates. However, the maximal accuracy difference between $CMDP_R$ and $NonLinProg_R$ is 0.0831 at 0.3 and 0.0317 on average. Only with the aid of a constrained Markov decision process it is possible to determine the appropriate points in time for a context distribution, which obtains over the entire energy budget a better accuracy compared to the non-optimized method. One important reason for this is the memorizing of the last updated context.



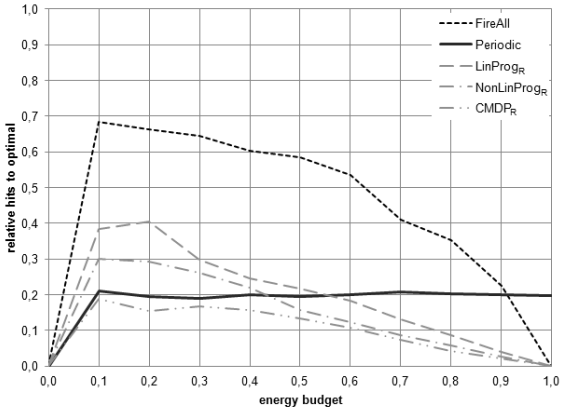
(a) relative hits



(b) relative messages



(c) relative hits to *Periodic*



(d) relative hits to *Optimal*

Figure 6.10: Linear programming, nonlinear programming, and constrained Markov Decision Process: the simulation results related to scenario 2

The results for the real trace scenario are illustrated in Figure 6.10 and constitutes a similar pattern apart from, in general, the rather worse accuracy in relation to *Periodic*. Nonetheless, the superiority of $CMDP_R$ across $NonLinProg_R$ and $NonLinProg_R$ across $LinProg_R$ hold, at which only $CMDP_R$ exceeds *Periodic* at any time. From the start, $CMDP_R$ achieves a higher accuracy than the others (except *Optimal*) with a maximal deviation of 0.0831 related to $NonLinProg_R$ (on average: 0.0317) and 0.1524 related to $LinProg_R$ (on average: 0.0705) based on the above mentioned ancillary information. In contrast, the linear and nonlinear optimization techniques suffer under the explained problems, wherefore both do not express the Markovian behavior accurately. $LinProg_R$ and $NonLinProg_R$ start with an accuracy reduction of 0.1476 and 0.0763 compared to *Periodic*, whereas $CMDP_R$ achieves a benefit of 0.0193. Combined with the late intersections, the figures demonstrate the same high functionality of a constrained Markov decision process in a real world example as for simulation. For example, the curve of $CMDP_R$ resides upside the x-axis in Figure 6.10c or below *Periodic* in Figure

6.10d. Therefore, only $CMDP_R$ is qualified for the application in a real implementation to accomplish an efficient energy-constrained distribution of context in mobile systems.

Especially, since the underlying data set presents a very significant case and contains consistently long sequences of the same context, because of the stationary distributions and self-transition probabilities described in Section 5.1. This circumstance is illustrated in Figure 6.11, which displays the absolute amount of n consecutive contexts of the same type within the passed through sequence. The dotted curve belongs to the real trace and the continuous ones for the simulated case, whereby the main difference between the two exists at the end. The maximal final five consecutive contexts of the same type sum an accuracy of 0.2286 for the real trace and only 0.05 on average for the simulated scenario, which serves as a proof for the good performance related to *Periodic* and *Optimal*. Both achieve with a small number of updates a high accuracy, whereat the probability of ten and less consecutive context of the same type increases exponentially.

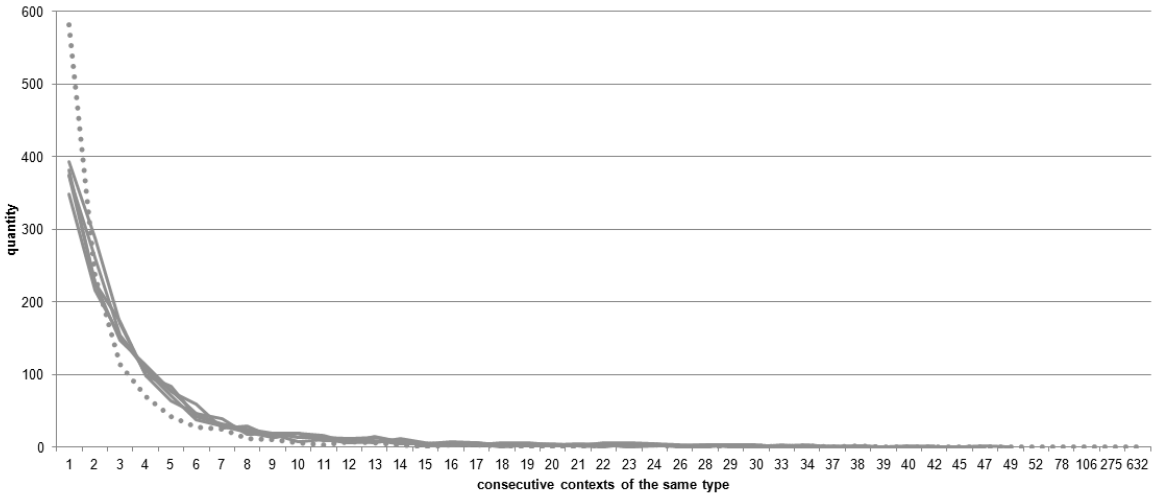


Figure 6.11: The amount of n consecutive contexts of the same type for scenario 1 and 2

For this reason, *Periodic* attains already with an energy budget of 0.1 a good accuracy of 0.4 and 0.67 in scenario 1 and 2 respectively. Even more astounding is the achievements related to *Optimal*, which gains at the beginning an accuracy of 0.7 for the simulated case and even 0.85 for the real sequence. Afterwards, the accuracy increases constantly and acquires already at an energy budget of 0.4 and 0.3 an accuracy of nearly 0.9. The residual energy capacity of 0.6 and accordingly 0.7 is required to reach the remaining accuracy of 0.1. This question of the needed energy budget in comparison with a necessitate accuracy constitutes an extension and is discussed in detail in Section 4.4.

Summarizing, the constrained Markov decision process shows the best results for the constrained optimization problem in the simulated and real scenario. It affords a straightforward approach with a good measured energy-accuracy trade-off, which is based on the following two factors. On the one hand, the learned user behavior describes in both cases accurately the passed through context sequence, on which the calculation of the corresponding context accuracy gain relies on. On the other hand, the constrained Markov decision process computes with the aid of the Markov chain a proper update

policy for the decision maker conditional upon the extra information. The auxiliary modeling of the last updated context enables the forming of a correct control process for decision-making and results in an update policy with a high accuracy gain over the entire energy budget, which is not to be under estimated.

7 Conclusions

Since the commercial launch of the first iPhone in 2007, technical advances have proceed and further innovations have changed everyday life. Equipped with a media player along with a support for multi-touch and gestures among other things, the first generation smartphones mark the alteration related to the basic application of mobile phones. Nowadays, a mobile device features a compass, an acceleration sensor, GPS, etc, which enable a location-aware search or environmental settings. With this sensor plurality not only the application range has widened, but also the energy consumption, which constitutes the main problem of an intensive sensor usage. Calls, apps, or an activity recognition drain the limited energy source very quickly, as a result the operation of the application can not be guaranteed. Considering the overall consumption of the restricted battery, energy-efficiency plays a decisive role at system design and realization. Each application must be developed and optimized against a gentle use of the energy resource to ensure a long operating life. In addition to the widely-used sensors, communication modules like WiFi, UMTS, or Bluetooth are a further main consumer of essential energy. Moreover, most of the utilized applications rely on information from or to the Internet connecting everyday commodities with the entire world. On these grounds, this thesis deals with the question of an efficient energy-constrained distribution of context in mobile systems.

7.1 Summary

Before the corresponding problem-solving approach is proposed, the thesis describes the underlying system model, which outlines the relevant components together with their interactions. In this regard, the abstracted part of the real world contains a sensor-embedded mobile phone (e.g. a smartphone), namely the producer, which runs through different contexts at discrete times. The contextual information is sensed from a context recognition software like mentioned in [WKZA09] and is assumed to be readily available as well as correct. Furthermore, no semantic information related to the producer's context is considered to provide a large field of application. In addition to the mobile phone of the producer a second device, for example also a mobile phone or a stationary computer (e.g. laptop, PC), appears, which is interested in the supplied information and wants to be up-to-date. Here, an one-to-one connection[1] between the producer and consumer is regarded consuming a constant energy cost for a message transmission from the restricted capacity. The main focus of attention is the efficient energy-constrained distribution on the producer-side.

Since different research activities address an energy-efficient approach related to mobile phones, various solutions exist in this connection, for example the activity recognition topic from Wang et al.

[1] In the strict sense, a server is needed to handle the essential functionality like the log-in, the user name resolution, etc (compare Chapter 2).

([WKZA09, WKZA10]) or the localization theme from Constandache et al. ([CGS+10]). However, all of the research topics do not consider the possible energy savings related to an efficient distribution between a producer and consumer.

Hence, a problem-solving approach for the efficient energy-constrained distribution of contextual information in mobile systems is proposed based on a probabilistic method. The underlying idea is to determine the future points in time for a context update with the most rising impact on the consumer's accuracy. The amount of transferred messages is bounded by the available energy capacity from the producer, whereas the accuracy indicates the number of contextual equality between them. To select the right points in time for an update at occurrence, the user behavior is modeled Markovian. This enables the behavioral characterization of the dynamic process to determine the accuracy gain (or deprivation) related to the update (or no update) of the current visited context. Therefore, the problem is formulated as a constrained optimization problem with the objective accuracy and an energy constraint, which results in an update policy indicating the update probability for each context. Three well-known optimization techniques are applied to derive the Markov-optimal update policies, namely linear programming, nonlinear programming and constrained Markov decision process.

In order to make realistic, verified statements, a functional prototype of the described system model is implemented to evaluate each predicated Markov-optimal update policy from the optimization functions. To compare among one another, an off-line method is applied to the passed through context sequence, namely the dynamic programming. This attains the real optimal update points in time, which achieves the maximal possible accuracy. The evaluation in Chapter 6 depicts the results for two different scenarios facing each update policy against a periodic (no optimization at all) and the optimal approach, which serves as a lower and upper bound respectively. In conclusion, all of them are compared together to identify the best optimization technique for execution in the real world. Here, both scenarios show that the update policy, which is calculated from a constrained Markov decision process, achieves the best results compared to linear and nonlinear programming. This is not surprising considering the drawbacks of LP, which are mentioned in Subsection 4.3.4 and causes the lower accuracy gain. Indeed, the nonlinear programming reaches better results, but suffers also under a too low accuracy.

As a result, the CMDP is recommended, because it attains the highest benefit in comparison with the other two methods. Moreover, on average, the results of CMDP achieve an accuracy gain of 20% for scenario 1 in contrast to the periodic method, which uses no optimization. Although the success related to scenario 2 equals just 8%, the maximal deviation is nearly 20% and at least four energy budgets obtain a greater accuracy benefit of 10%. In comparison with *FireAll*, the results are more promising and correspond to an accuracy gain of 14% for scenario 1 and even 31% for scenario 2. Besides the fact that all methods send the equal amount of updates, CMDP constitutes an attractive alternative for the efficient energy-constrained distribution of context in mobile systems.

Not only the formidable energy savings associated with only a minimal accuracy loss provides great benefits for the usage on mobile phones. A further advantage is the general keeping of "context" of the system model, which allows on the one hand an easy and wide utilization and on the other hand an even greater energy reduction adding semantic or temporal information of the context to the constrained optimization problem. For example, the contextual information "location" implies specific environment-aware transaction probabilities, which has a considerable influence on the forecast of the producer's behavior. The transaction between the user states *"Traveling by Train"* and *"Leaving the Station"* is only feasible if the producer is near a rail track or rather a railway building. The easy

combination of the described energy-efficient distribution and other energy saving approaches on mobile phones is achieved in this thesis by restricting the communication between a producer and consumer to the essential, wherewith extensions and thus further optimizations are possible.

7.2 Limitations

So the generalization in this thesis constitutes the lowest common denominator for a distribution of contextual information, which entails already significant benefits. Nonetheless, the underlying system model from Chapter 2 involves some assumptions, which have to be looked at a little more closer for a practical application. The previous chapters deal extensively with the energy problem related to the producer, assuming that the consumer has an unlimited energy capacity. In fact, the consumer can be also a mobile device with an energy limitation, whereas the consumption for each message sent or received must be considered. Furthermore, the interested consumer, which follows the context updates from a producer, provides its own contextual information to other consumers over the Internet (compare the *ProdCon*s of Figure 7.1). This leads to an incorporation of a producer and a consumer component on the mobile phone draining the same energy source. Therefore, an extension of the constrained Markov decision process could help to model the energy-efficient interaction between them. Altman [Alt95] characterizes the approach for N selfish, not cooperating controllers with different objectives, with the help of a set of coupled linear programs (providing a Nash equilibrium).

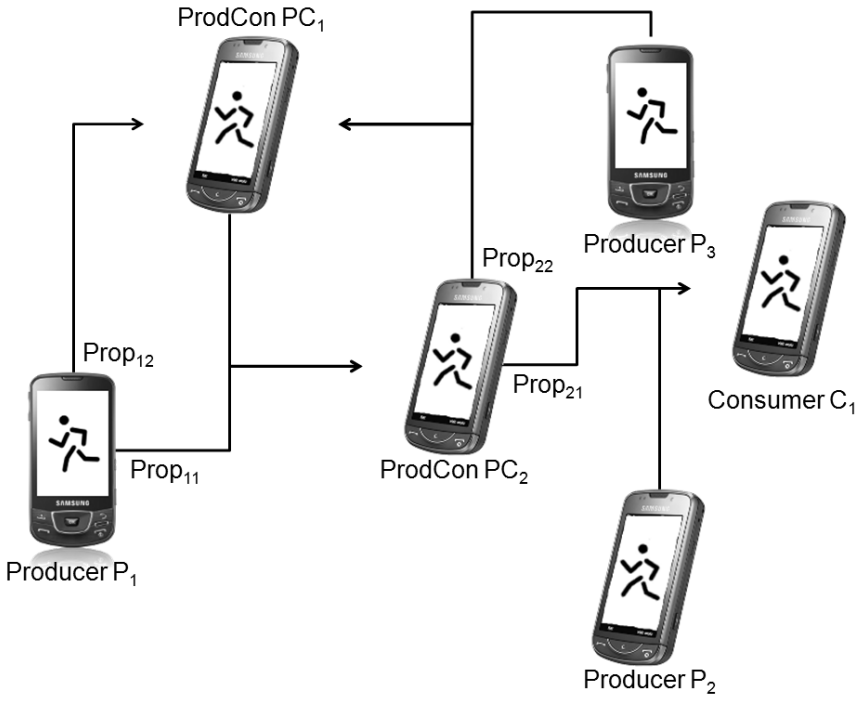


Figure 7.1: A many-to-many connection between producer(s) and consumer(s)

Continuing this train of thought, the cooperation from many producers to many consumers is required in a networked world, which are all send and receive contextual information among each other as well as are featured with individual energy capacities. For example, the mobile phone PC_1 in Figure 7.1 serves as a consumer for the producers P_1 together with P_2 and as a producer for PC_2. Moreover, every logical one-to-one connection between one producer and one consumer has different properties for the energy budget and the required accuracy such as $Prop_{21}$ to C_1 and $Prop_{22}$ to P_3. All these particular connection settings must be fulfilled to satisfy each constraint.

7.3 Outlook

As mentioned above, a remaining future work is the adding of semantical information to the user behavior and the evaluation of the effects on the accuracy. Additionally, a more detailed analysis of the user's mobility pattern results in improved optimizations as the human behavior is extensively based on (daily) recurrence. For example, an employee leaves the home on weekdays in the morning to travel to the workplace, where he/she spends the most time of the day. Depending on the job description, the daily work of the employee is partially structured by repetitive tasks (e.g. weekly meetings), whereas the way home is once again a daily event in the afternoon. In the same manner, regularly recurring occurrences are found at the weekend such as staying at home or visiting of friends. Concomitantly, a fragmentation of the overall update policy in smaller parts (e.g. an own update policy for the daily work) can advance the modeling of the user resulting in a more accurate prediction of the following user context. Especially as humans only remain in a small subset of the entire possible context set.

The ongoing improvement related to mobile phones enables another opportunity to enhance the proposed problem-solving approach. The system model in Chapter 2 assumes that the calculation of the update policy is performed on a powerful, remote back-end server and transferred to the mobile phone before the application starts. Nowadays, it is not unusual that a smartphone[2] is equipped with a Dual-Core-CPU having processor performance of 2×1.2 GHz, a 1 GB RAM, and 16 GB internal memory, as a consequence the computation of an update policy can be executed on the mobile device. In this regard the initial update policy, as well as a policy adaption, is performed on-line on the producer-side to reduce the error probability related to an inexact Markov model. For example, the user behavior is learned from a context sequence with rare changes, whereas the current passed-through succession contains the sensing of unusual frequent context shifts. Accordingly, the producer sends too many updates in a short time and drains the energy capacity too quickly. Therefore, adaptation and relearning would put things right.

[2]For exmaple, the Samsung Galaxy S II I9100, http://www.samsung.de

A Appendix

A.1 Update Policies for Scenario 1

Linear Programming	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.435, 0.000, 0.000, 0.000\}$
0.2	$\{0.870, 0.000, 0.000, 0.000\}$
0.3	$\{1.000, 1.000, 0.150, 0.000\}$
0.4	$\{1.000, 1.000, 0.485, 0.000\}$
0.5	$\{1.000, 1.000, 0.820, 0.000\}$
0.6	$\{1.000, 1.000, 1.000, 0.107\}$
0.7	$\{1.000, 1.000, 1.000, 0.338\}$
0.8	$\{1.000, 1.000, 1.000, 0.568\}$
0.9	$\{1.000, 1.000, 1.000, 0.799\}$
1.0	$\{1.000, 1.000, 1.000, 1.000\}$

Nonlinear Programming	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.751, 0.000, 0.000, 0.141\}$
0.2	$\{0.797, 0.000, 0.000, 0.342\}$
0.3	$\{0.846, 0.000, 0.000, 0.618\}$
0.4	$\{0.940, 0.000, 0.000, 0.895\}$
0.5	$\{1.000, 0.000, 0.097, 1.000\}$
0.6	$\{1.000, 0.000, 0.292, 1.000\}$
0.7	$\{1.000, 0.000, 0.492, 1.000\}$
0.8	$\{1.000, 0.000, 0.698, 1.000\}$
0.9	$\{1.000, 0.000, 0.909, 1.000\}$
1.0	$\{1.000, 1.000, 1.000, 1.000\}$

Table A.1: The update policies related to linear and nonlinear programming

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.000, 1.000, 0.033, 0.000\}$
0.2	$\{0.000, 1.000, 0.218, 0.000\}$
0.3	$\{0.000, 1.000, 1.000, 0.000\}$
0.4	$\{0.000, 1.000, 1.000, 0.106\}$
0.5	$\{0.000, 1.000, 1.000, 0.374\}$
0.6	$\{0.000, 1.000, 1.000, 0.884\}$
0.7	$\{0.000, 1.000, 1.000, 1.000\}$
0.8	$\{0.000, 1.000, 1.000, 1.000\}$
0.9	$\{0.000, 1.000, 1.000, 1.000\}$
1.0	$\{0.001, 1.000, 1.000, 1.000\}$

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{1.000, 0.000, 1.000, 0.000\}$
0.2	$\{1.000, 0.000, 1.000, 0.000\}$
0.3	$\{0.592, 0.000, 1.000, 1.000\}$
0.4	$\{0.490, 0.000, 1.000, 1.000\}$
0.5	$\{0.510, 0.000, 1.000, 1.000\}$
0.6	$\{0.501, 0.000, 1.000, 1.000\}$
0.7	$\{0.498, 0.000, 1.000, 1.000\}$
0.8	$\{0.489, 0.000, 1.000, 1.000\}$
0.9	$\{0.486, 0.000, 1.000, 1.000\}$
1.0	$\{0.353, 0.070, 1.000, 1.000\}$

Table A.2: The update policies related to state 0 and 1 of constrained Markov decision process

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{1.000, 0.000, 0.000, 0.000\}$
0.2	$\{1.000, 0.000, 0.000, 0.000\}$
0.3	$\{1.000, 0.049, 0.000, 0.000\}$
0.4	$\{1.000, 1.000, 0.000, 0.000\}$
0.5	$\{1.000, 1.000, 0.000, 0.000\}$
0.6	$\{1.000, 1.000, 0.000, 0.000\}$
0.7	$\{1.000, 1.000, 0.000, 0.113\}$
0.8	$\{1.000, 1.000, 0.000, 0.299\}$
0.9	$\{1.000, 1.000, 0.000, 0.585\}$
1.0	$\{1.000, 1.000, 0.004, 1.000\}$

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.990, 0.498, 0.002, 0.000\}$
0.2	$\{0.873, 0.271, 0.021, 0.007\}$
0.3	$\{1.000, 1.000, 1.000, 0.000\}$
0.4	$\{1.000, 1.000, 1.000, 0.000\}$
0.5	$\{1.000, 1.000, 1.000, 0.000\}$
0.6	$\{1.000, 1.000, 1.000, 0.000\}$
0.7	$\{1.000, 1.000, 1.000, 0.000\}$
0.8	$\{1.000, 1.000, 1.000, 0.000\}$
0.9	$\{1.000, 1.000, 1.000, 0.000\}$
1.0	$\{1.000, 1.000, 1.000, 0.004\}$

Table A.3: The update policies related to state 2 and 3 of constrained Markov decision process

A.2 Update Policies for Scenario 2

Linear Programming	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.430, 0.000, 0.000, 0.000\}$
0.2	$\{0.859, 0.000, 0.000, 0.000\}$
0.3	$\{1.000, 1.000, 0.137, 0.000\}$
0.4	$\{1.000, 1.000, 0.468, 0.000\}$
0.5	$\{1.000, 1.000, 0.799, 0.000\}$
0.6	$\{1.000, 1.000, 1.000, 0.090\}$
0.7	$\{1.000, 1.000, 1.000, 0.318\}$
0.8	$\{1.000, 1.000, 1.000, 0.546\}$
0.9	$\{1.000, 1.000, 1.000, 0.773\}$
1.0	$\{1.000, 1.000, 1.000, 1.000\}$

Nonlinear Programming	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.710, 0.000, 0.000, 0.141\}$
0.2	$\{0.748, 0.000, 0.000, 0.354\}$
0.3	$\{0.847, 0.000, 0.000, 0.604\}$
0.4	$\{0.933, 0.000, 0.000, 0.886\}$
0.5	$\{1.000, 0.000, 0.085, 1.000\}$
0.6	$\{1.000, 0.000, 0.277, 1.000\}$
0.7	$\{1.000, 0.000, 0.475, 1.000\}$
0.8	$\{1.000, 0.000, 0.677, 1.000\}$
0.9	$\{1.000, 0.000, 0.885, 1.000\}$
1.0	$\{1.000, 1.000, 1.000, 1.000\}$

Table A.4: The update policies related to linear and nonlinear programming

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.000, 1.000, 0.032, 0.000\}$
0.2	$\{0.000, 1.000, 0.211, 0.000\}$
0.3	$\{0.000, 1.000, 1.000, 0.000\}$
0.4	$\{0.000, 1.000, 1.000, 0.096\}$
0.5	$\{0.000, 1.000, 1.000, 0.353\}$
0.6	$\{0.000, 1.000, 1.000, 0.831\}$
0.7	$\{0.000, 1.000, 1.000, 1.000\}$
0.8	$\{0.000, 1.000, 1.000, 1.000\}$
0.9	$\{0.000, 1.000, 1.000, 1.000\}$
1.0	$\{0.000, 1.000, 1.000, 1.000\}$

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{1.000, 0.000, 1.000, 0.000\}$
0.2	$\{1.000, 0.000, 1.000, 0.000\}$
0.3	$\{0.563, 0.000, 1.000, 1.000\}$
0.4	$\{0.493, 0.000, 1.000, 1.000\}$
0.5	$\{0.520, 0.000, 1.000, 1.000\}$
0.6	$\{0.505, 0.000, 1.000, 1.000\}$
0.7	$\{0.496, 0.000, 1.000, 1.000\}$
0.8	$\{0.490, 0.000, 1.000, 1.000\}$
0.9	$\{0.487, 0.000, 1.000, 1.000\}$
1.0	$\{0.847, 0.003, 1.000, 1.000\}$

Table A.5: The update policies related to state 0 and 1 of constrained Markov decision process

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{1.000, 0.000, 0.000, 0.000\}$
0.2	$\{1.000, 0.000, 0.000, 0.000\}$
0.3	$\{1.000, 0.020, 0.000, 0.000\}$
0.4	$\{1.000, 1.00000.000, 0.000\}$
0.5	$\{1.000, 1.000, 0.000, 0.000\}$
0.6	$\{1.000, 1.000, 0.000, 0.000\}$
0.7	$\{1.000, 1.000, 0.000, 0.100\}$
0.8	$\{1.000, 1.000, 0.000, 0.277\}$
0.9	$\{1.000, 1.000, 0.000, 0.546\}$
1.0	$\{1.000, 1.000, 0.000, 1.000\}$

Constrained Markov Decision Process	
E	u
0.0	$\{0.000, 0.000, 0.000, 0.000\}$
0.1	$\{0.990, 0.508, 0.002, 0.000\}$
0.2	$\{0.880, 0.243, 0.020, 0.007\}$
0.3	$\{1.000, 1.000, 1.000, 0.000\}$
0.4	$\{1.000, 1.000, 1.000, 0.000\}$
0.5	$\{1.000, 1.000, 1.000, 0.000\}$
0.6	$\{1.000, 1.000, 1.000, 0.000\}$
0.7	$\{1.000, 1.000, 1.000, 0.000\}$
0.8	$\{1.000, 1.000, 1.000, 0.000\}$
0.9	$\{1.000, 1.000, 1.000, 0.000\}$
1.0	$\{1.000, 1.000, 1.000, 0.000\}$

Table A.6: The update policies related to state 2 and 3 of constrained Markov decision process

A.3 Accuracy Results for Scenario 1

Relative Hits									
E	*FiAl*	*Per*	LP_R	*LP*	NLP_R	*NLP*	$CMDP_R$	*CMDP*	*Opt*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.3368	0.4008	0.5132	0.5132	0.5497	0.5488	0.5978	0.5933	0.7086
0.2	0.4146	0.4742	0.5132	0.5132	0.6068	0.6005	0.6656	0.6649	0.7950
0.3	0.4794	0.5346	0.5865	0.5851	0.6533	0.6488	0.7364	0.7354	0.8490
0.4	0.6974	0.5574	0.6692	0.6742	0.7208	0.7191	0.7754	0.7766	0.8900
0.5	0.6230	0.6070	0.7246	0.7310	0.7811	0.7794	0.8120	0.8144	0.9180
0.6	0.6884	0.6408	0.7742	0.7785	0.8228	0.8253	0.8517	0.8516	0.9422
0.7	0.7340	0.6648	0.8310	0.8336	0.8682	0.8741	0.8897	0.8916	0.9634
0.8	0.8462	0.6902	0.8853	0.8946	0.9109	0.9164	0.9264	0.9287	0.9754
0.9	0.9122	0.7184	0.9436	0.9529	0.9536	0.9575	0.9611	0.9660	0.9876
1.0	0.9998	0.7406	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table A.7: The relative hits related to scenario 1

Relative Messages									
E	*FiAl*	*Per*	LP_R	*LP*	NLP_R	*NLP*	$CMDP_R$	*CMDP*	*Opt*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.0998	0.0998	0.0941	0.0939	0.0954	0.0946	0.0997	0.0987	0.0998
0.2	0.1997	0.1997	0.1927	0.1904	0.1876	0.1909	0.1967	0.1969	0.1997
0.3	0.3003	0.2995	0.2974	0.2965	0.2810	0.2837	0.2944	0.2917	0.2995
0.4	0.4002	0.3993	0.3997	0.4043	0.3748	0.3804	0.3992	0.3998	0.3993
0.5	0.5000	0.5000	0.4998	0.5116	0.4818	0.4724	0.4952	0.5002	0.4992
0.6	0.5998	0.5998	0.6007	0.6084	0.5772	0.5812	0.5922	0.5914	0.5990
0.7	0.6997	0.6997	0.7005	0.7078	0.6875	0.6909	0.6934	0.7025	0.6988
0.8	0.8003	0.7995	0.8002	0.8096	0.7903	0.7950	0.7977	0.8086	0.7987
0.9	0.9002	0.8993	0.8993	0.9142	0.8952	0.8995	0.8995	0.9170	0.8985
1.0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table A.8: The relative messages related to scenario 1

Relative Hits To Periodic						
E	*FireAll*	*Periodic*	$LinProg_R$	$NonLinProg_R$	$CMDP_R$	*Optimal*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	−0.1597	0.0000	0.2804	0.3716	0.4914	0.7680
0.2	−0.1257	0.0000	0.0822	0.2797	0.4035	0.6765
0.3	−0.1033	0.0000	0.0970	0.2221	0.3776	0.5881
0.4	0.2512	0.0000	0.2005	0.2931	0.3911	0.5967
0.5	0.0264	0.0000	0.1938	0.2868	0.3377	0.5124
0.6	0.0743	0.0000	0.2082	0.2840	0.3292	0.4703
0.7	0.1041	0.0000	0.2501	0.3060	0.3383	0.4492
0.8	0.2260	0.0000	0.2827	0.3198	0.3423	0.4132
0.9	0.2698	0.0000	0.3134	0.3274	0.3378	0.3747
1.0	0.3500	0.0000	0.3503	0.3503	0.3503	0.3503

Table A.9: The relative hits to *Periodic* related to scenario 1

Relative Hits To Optimal						
E	*FireAll*	*Periodic*	$LinProg_R$	$NonLinProg_R$	$CMDP_R$	*Optimal*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.5247	0.4344	0.2758	0.2242	0.1564	0.0000
0.2	0.4785	0.4035	0.3545	0.2367	0.1628	0.0000
0.3	0.4353	0.3703	0.3092	0.2305	0.1326	0.0000
0.4	0.2164	0.3737	0.2481	0.1902	0.1288	0.0000
0.5	0.3214	0.3388	0.2106	0.1492	0.1155	0.0000
0.6	0.2694	0.3199	0.1783	0.1267	0.0960	0.0000
0.7	0.2381	0.3099	0.1374	0.0988	0.0765	0.0000
0.8	0.1325	0.2924	0.0924	0.0661	0.0502	0.0000
0.9	0.0763	0.2726	0.0446	0.0344	0.0269	0.0000
1.0	0.0002	0.2594	0.0000	0.0000	0.0000	0.0000

Table A.10: The relative hits to *Optimal* related to scenario 1

A.4 Accuracy Results for Scenario 2

Relative Hits									
E	*FiAl*	*Per*	LP_R	*LP*	NLP_R	*NLP*	$CMDP_R$	*CMDP*	*Optl*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.2700	0.6742	0.5266	0.5266	0.5979	0.5817	0.6935	0.6923	0.8538
0.2	0.2974	0.7124	0.5266	0.5266	0.6259	0.6262	0.7483	0.7518	0.8836
0.3	0.3214	0.7362	0.6380	0.6285	0.6708	0.6680	0.7564	0.7598	0.9076
0.4	0.3676	0.7414	0.6989	0.7054	0.7233	0.7423	0.7830	0.7860	0.9282
0.5	0.3908	0.7578	0.7370	0.7414	0.7920	0.7919	0.8153	0.8211	0.9402
0.6	0.4412	0.7612	0.7783	0.7809	0.8340	0.8382	0.8493	0.8497	0.9522
0.7	0.5698	0.7634	0.8370	0.8362	0.8801	0.8819	0.8937	0.8977	0.9642
0.8	0.6314	0.7786	0.8915	0.8914	0.9202	0.9214	0.9339	0.9429	0.9762
0.9	0.7672	0.7896	0.9485	0.9445	0.9624	0.9596	0.9672	0.9652	0.9882
1.0	1.0000	0.8024	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table A.11: The relative hits related to scenario 2

Relative Messages									
E	*FiAl*	*Per*	LP_R	*LP*	NLP_R	*NLP*	$CMDP_R$	*CMDP*	*Optl*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.1003	0.0994	0.0984	0.1023	0.0851	0.0814	0.0582	0.0585	0.1003
0.2	0.1997	0.1997	0.2000	0.2057	0.1718	0.1771	0.1051	0.1095	0.2005
0.3	0.2999	0.2999	0.2984	0.2974	0.2602	0.2657	0.1701	0.1724	0.3008
0.4	0.4002	0.3993	0.3972	0.3987	0.3798	0.3738	0.2760	0.2767	0.4010
0.5	0.5004	0.4996	0.4979	0.4981	0.4851	0.4822	0.3866	0.3861	0.5013
0.6	0.5998	0.5998	0.6000	0.6012	0.5868	0.5820	0.5248	0.5320	0.6015
0.7	0.7001	0.6993	0.6986	0.7019	0.6887	0.6854	0.6550	0.6636	0.7018
0.8	0.8003	0.7995	0.7983	0.8027	0.7905	0.7831	0.7678	0.7662	0.8020
0.9	0.8997	0.8998	0.8996	0.9001	0.8981	0.8941	0.8775	0.8807	0.9023
1.0	1.0000	0.9992	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Table A.12: The relative messages related to scenario 2

Relative Hits To Periodic						
E	*FireAll*	*Periodic*	$LinProg_R$	$NonLinProg_R$	$CMDP_R$	*Optimal*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	−0.5995	0.0000	−0.2189	−0.1131	0.0286	0.2664
0.2	−0.5825	0.0000	−0.2608	−0.1214	0.0503	0.2403
0.3	−0.5634	0.0000	−0.1334	−0.0888	0.0274	0.2328
0.4	−0.5042	0.0000	−0.0573	−0.0244	0.0561	0.2520
0.5	−0.4843	0.0000	−0.0274	0.0451	0.0759	0.2407
0.6	−0.4204	0.0000	0.0225	0.0957	0.1157	0.2509
0.7	−0.2536	0.0000	0.0964	0.1529	0.1707	0.2630
0.8	−0.1891	0.0000	0.1450	0.1819	0.1994	0.2538
0.9	−0.0284	0.0000	0.2013	0.2188	0.2250	0.2515
1.0	0.2463	0.0000	0.2463	0.2463	0.2463	0.2463

Table A.13: The relative hits to *Periodic* related to scenario 2

Relative Hits To Optimal						
E	*FireAll*	*Periodic*	$LinProg_R$	$NonLinProg_R$	$CMDP_R$	*Optimal*
0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	0.6838	0.2104	0.3832	0.2997	0.1877	0.0000
0.2	0.6634	0.1938	0.4040	0.2916	0.1532	0.0000
0.3	0.6459	0.1888	0.2971	0.2609	0.1666	0.0000
0.4	0.6040	0.2012	0.2470	0.2207	0.1564	0.0000
0.5	0.5843	0.1940	0.2161	0.1577	0.1329	0.0000
0.6	0.5367	0.2006	0.1826	0.1241	0.1081	0.0000
0.7	0.4090	0.2083	0.1319	0.0872	0.0731	0.0000
0.8	0.3532	0.2024	0.0867	0.0573	0.0434	0.0000
0.9	0.2236	0.2010	0.0402	0.0261	0.0212	0.0000
1.0	0.0000	0.1976	0.0000	0.0000	0.0000	0.0000

Table A.14: The relative hits to *Optimal* related to scenario 2

Bibliography

[Alt95] E. Altman. *Constrained Markov Decision Processes (Stochastic Modeling)*. Chapman & Hall, 1995. (Cited on pages 31 and 65)

[APP] Homepage of Apple Inc. Technical and entrepreneurial information. URL http://www.apple.com. (Cited on pages 1 and 5)

[Bat00] J. Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. John Wiley & Sons, 2000. (Cited on page 34)

[Bel57] R. Bellmann. *Dynamic Programming*. Princeton University Press, 1957. (Cited on pages 34 and 35)

[BK65] R. Bellmann, R. Kalaba. *Dynamic Programming and Modern Control Theory*. Academic Press, 1965. (Cited on page 35)

[CEN] Community Resource for Archiving Wireless Data At Dartmouth. Wireless network data resource for the research community. URL http://www.crawdad.org/. (Cited on page 43)

[CGS+10] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, L. Cox. Energy-Efficient Localization via Personal Mobility Profiling. In *Mobile Computing, Applications, and Services*, volume 35 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 203–222. Springer, 2010. (Cited on pages 2, 15, 16 and 64)

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2009. (Cited on pages 34 and 37)

[CN06] W.-K. Ching, M. K. Ng. *Markov Chains: Models, Algorithms and Applications (International Series in Operations Research & Management Science)*. Springer, 1 edition, 2006. (Cited on pages 21 and 22)

[Dan98] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 10 edition, 1998. (Cited on page 26)

[DT97] G. B. Dantzig, M. N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag New York, Inc., 1997. (Cited on pages 26 and 27)

[HA04] P. Hubwieser, G. Aiglstorfer. *Fundamente der Informatik - Ablaufmodellierung, Algorithmen und Datenstrukturen.* Oldenbourg, 2004. (Cited on page 6)

[KB95] B. Kolman, R. E. Beck. *Elementary Linear Programming with Applications, Second Edition (Computer Science and Scientific Computing)*. Academic Press, 2 edition, 1995. (Cited on pages 26 and 27)

[KLLK10] A. M. Khan, Y.-K. Lee, S. Lee, T.-S. Kim. A Triaxial Accelerometer-Based Physical-Activity Recognition via Augmented-Signal Features and a Hierarchical Recognizer. *Information Technology in Biomedicine*, 14:1166–1172, 2010. (Cited on page 7)

[LCB+05] J. Lester, T. Choudhury, G. Borriello, S. Consolvo, L. James, K. Everitt, I. Smith. Sensing and Modeling Activities to Support Physical Fitness. Workshop paper in UbiComp '05 Workshop: Monitoring, Measuring and Motivating Exercise: Ubiquitous Computing to Support Physical Fitness, 2005. (Cited on pages 7, 13 and 19)

[LCK+05] J. Lester, T. Choudhury, N. Kern, G. Borriello, B. Hannaford. A Hybrid Discriminative/Generative Approach for Modeling Human Activities. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, pp. 766–772. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. (Cited on pages 13 and 19)

[LY08] D. G. Luenberger, Y. Ye. *Linear and Nonlinear Programming*. Springer, third edition, 2008. (Cited on page 30)

[Mis04] A. R. Mishra. *Fundamentals of Cellular Network Planning and Optimisation: 2G / 2.5G / 3G ... Evolution to 4G*. John Wiley & Sons, 2004. (Cited on page 9)

[MLF+08] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, A. T. Campbell. Sensing Meets Mobile Social Networks: the Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pp. 337–350. ACM, New York, NY, USA, 2008. (Cited on pages 16 and 43)

[MPF+08] M. Musolesi, M. Piraccini, K. Fodor, A. Corradi, A. Campbell. CRAWDAD data set dartmouth/cenceme (v. 2008-08-13). Downloaded from http://crawdad.cs.dartmouth.edu/meta.php?name=dartmouth/cenceme, 2008. (Cited on page 44)

[MPF+10] M. Musolesi, M. Piraccini, K. Fodor, A. Corradi, A. T. Campbell. Supporting Energy-Efficient Uploading Strategies for Continuous Sensing Applications on Mobile Phones. In P. Floréen, A. Krüger, M. Spasojevic, editors, *Proceedings of the 8th International Conference on Pervasive Computing*, pp. 355–372. Springer Berlin Heidelberg, 2010. (Cited on pages 12, 16 and 17)

[Pap09] L. Papula. *Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler*. Vieweg, 10 edition, 2009. (Cited on page 28)

[Ste94] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994. (Cited on pages 21, 22 and 23)

[WKZA09] Y. Wang, B. Krishnamachari, Q. Zhao, M. Annavaram. The Tradeoff between Energy Efficiency and User State Estimation Accuracy in Mobile Sensing. In *In Proceedings of MobiCase*. 2009. (Cited on pages 14, 19, 63 and 64)

[WKZA10] Y. Wang, B. Krishnamachari, Q. Zhao, M. Annavaram. Markov-Optimal Sensing Policy for User State Estimation in Mobile Devices. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pp. 268–278. ACM, New York, NY, USA, 2010. (Cited on pages 7 and 64)

[ZF86] R. Zurmühl, S. Falk. *Matrizen und ihre Anwendungen Teil 2: Numerische Methoden*. Springer, 5 edition, 1986. (Cited on page 30)

All links were last followed on December 20, 2011.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Florian Berg)