ANOMALY DETECTION FROM PERSONAL USAGE PATTERNS IN WEB APPLICATIONS

GÜRKAN VURAL

DECEMBER 2006

ANOMALY DETECTION FROM PERSONAL USAGE PATTERNS IN WEB APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜRKAN VURAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

DECEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe KİPER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Dr. Onur Tolga ŞEHİTOĞLU	Dr. Meltem Turhan YÖNDEM
Co-Supervisor	Supervisor

Examining Committee Members

Prof. Dr. Faruk POLAT	(METU, CENG)	
Dr. Meltem Turhan YÖNDEM	(METU, CENG)	
Dr. Onur Tolga ŞEHİTOĞLU	(METU, CENG)	
Assoc. Prof. Dr. İ. Hakkı TOROSLU	(METU, CENG)	
Assist. Prof. Dr. Bilge SAY	(METU, COGS)	

Name, Lastname	:	Gürkan VURAL
Signature	:	

Abstract

ANOMALY DETECTION FROM
PERSONAL USAGE PATTERNS IN WEB APPLICATIONS

Vural, Gürkan
M.Sc., Department of Computer Engineering
Supervisor: Dr. Meltem Turhan Yöndem
Co-Supervisor: Dr. Onur Tolga Şehitoğlu

December 2006, 78 pages

The anomaly detection task is to recognize the presence of an unusual (and potentially hazardous) state within the behaviors or activities of a computer user, system, or network with respect to some model of normal behavior which may be either hard-coded or learned from observation. An anomaly detection agent faces many learning problems including learning from streams of temporal data, learning from instances of a single class, and adaptation to a dynamically changing concept. The domain is complicated by considerations of the trusted insider problem (recognizing the difference between innocuous and malicious behavior changes on the part of a trusted user).

This study introduces the anomaly detection in web applications and formulates it as a machine learning task on temporal sequence data. In this study the goal is to develop a model or profile of normal working state of web application user and to detect anomalous conditions as deviations from the expected behavior patterns. We focus, here, on learning models of normality at the user behavioral level, as observed through a web application. In this study we introduce some sensors intended to function as a focus of attention unit at the lowest level of a classification hierarchy using Finite State Markov Chains and Hidden Markov Models and discuss the success of these sensors.

Keywords: Anomaly Detection, Web Application Security, Finite State Markov Chains, Hidden Markov Models, Temporal Sequence Learning.

Öz

WEB UYGULAMALARINDA KİŞİSEL KULLANIM ÖRÜNTÜLERİNDEN ANORMALLİK TESPİTİ

Vural, Gürkan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Meltem Turhan Yöndem
Ortak Tez Yöneticisi: Dr. Onur Tolga Şehitoğlu

Aralık 2006, 78 sayfa

Anormallik tespiti, bilgisayar kullanıcısı, sistemi veya ağına ait davranış ve hareketler içerisinde sabit olarak kodlanmış veya gözlemlerle öğrenilmiş normal davranış modellerini baz alarak, nadir görülen (ve potansiyel olarak riskli) davranış ve hareketlerin varlığının algılanmasıdır. Anormallik tespit ajanı, geçici veri dizilerinden öğrenmeyi, bir tek sınıfın örneklemelerinden öğrenmeyi ve dinamik olarak değişen bir ortama adaptasyonu içeren bir çok öğrenme problemiyle karşılaşır. Bu alan, güvenlik kontrolünden geçmiş kötü niyetli bir kullanıcının değişimleri (güvenilirliği tespit edilmiş kullanıcının zararsız ve zararlı değişimleri arasındaki ayrımının tespit edilmesi) göz önüne alındığında karmaşıktır.

Bu çalışma web uygulamalarında anormallik tespitini tanımlar ve geçici veri dizisi üzerinden çıkarım yapma problemi olarak formüle eder. Bu çalışmadaki hedef, web uygulaması kullanıcısının normal kullanım durumuna dayalı bir model veya profil oluşturmak ve beklenen davranış kalıplarından sapmalar olarak kabul edilen anormal durumları tespit etmektir. Burada, web uygulamaları aracılığıyla gözlemlenen, kullanıcı davranışı açısından normallik modelleri üzerinde durulmaktadır. Bu çalışmada, Sonlu Durum Markov Zincirleri ve Gizli Markov Modeller vasıtasıyla en düşük sınıflandırma hiyerarşisi düzeyinde işlev görmesi istenen bazi algılayıcılar sunuyor ve bu algılayıcıların başarımlarını ele alıyoruz.

Anahtar Kelimeler: Anormallik Tespiti, Web Uygulama Güvenliği, Sonlu Durum Markov Zincirleri, Gizli Markov Modeller, Zamansal Dizi Öğrenme.

To my love, Berivan,
and to my mother and father.

Acknowledgments

The author wishes to express his deepest gratitude to:

- His supervisor, Dr. Meltem Turhan YÖNDEM and co-supervisor, Dr. Onur Tolga ŞEHİTOĞLU for their guidance, advice, criticism, encouragements, and insight,
- His partner in life, Ülker Berivan ÖZBAYRAM, for her love and endless confidence,
- His parents and brother for their love and encouragements,
- His moral brother, Aydın BOZDEMİR for his support and encouragements,
- His colleagues for their support and encouragements,
- His company, Central Bank of the Republic of Turkey, for its support of academic researches,
- His psychologist, Dr. Gül Hınçal, for her guidance and encouragements,

throughout the research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

Today while sitting on our seats we can do a wide range of our tasks using computers. Most of these tasks can be achieved by using web applications. Hence providing a secure web environment has become a high priority for companies as e-businesses have increased the amount and the sensitivity of information that can be accessed through the Internet. Although the developers and maintainers of web applications are trying to take all the precautions that satisfies the principal of 'defense in depth', web applications stay hopeless against masqueraders, who can prove his identity to the system with a password stolen by using social engineering or other methods. Thus, most of the web applications ask unnecessary annoying security questions even while you are doing a usual task which makes these applications less user friendly. This evidently indicates that intelligent security systems are needed with the ability to adjust security level according to anomalous conditions as deviations from expected behavior patterns.

From a different point of view, human's current actions and responses reflect the unique demands of the current situation, but they are also shaped out of previously learned behaviors and skills [1]. This ability to act by recycling previously gained knowledge whenever possible while acquiring new knowledge or skills whenever necessary is one of the greatest strengths of human intelligence and is one of the major contributing factors to human adaptability. Therefore automated modeling of human behaviors is useful in the computer security domain of anomaly detection [2]. In the user modeling facet of the anomaly detection domain, the task is to develop a model or profile of the normal working state of a user and to detect anomalous conditions as deviations from this model or profile which is the actual needed thing for contemporary web applications.

We take a personal assistant view of this domain, in which the task of the anomaly detection sensor is to augment the security of a private individual's account by monitoring usage activity for suspicious incidents that do not conform to known behavior patterns of the account owner. Under this view, behavioral data are assumed to be private and available only to the valid user's assistant. Thus, training data are single class representing only the behaviors of the profiled user. The following chapter will introduce these concepts in more depth.

1.2 Goals of this Study

The general goal of anomaly detection is to model the state of a computer system, network, or user and to detect later misuse in terms of deviations from the known patterns. In this study we focus on a personal anomaly detection agent, which learns the user's behaviors in order to help protect his or her account from unauthorized access. An adaptive learning model is a promising approach for profiling user behavior for such anomaly detection systems. This domain is a particularly interesting learning context as it presents a number of difficulties to an adaptive learning system: [1]

- The system must be both adaptable to change on the part of a valid user, yet resistant to intruders masquerading as authorized users or attempting to train the system away from the valid user's profile.
- For reasons of privacy and practicality, it is often necessary to train such a system with data from only valid interactions of a single user and we thus have the task of inducing a concept from examples of only a single class.
- The anomaly detection domain presents us with a stream of discrete events (commands, system calls, user requests, etc) from which we wish to induce a concept. This task of learning from temporal sequence data has its own issues that bear investigation such as learning on non-metric spaces and learning relations within feature vectors.

Although the main goal of the anomaly detection domain is to produce a system capable of distinguishing all intrusive anomalies from all normal behaviors, it is not necessary to fully achieve this target to make substantial contribution in this domain

[3]. Our goal is not to produce a complete, stand-alone anomaly detection system that would catch all impostors or to solve all security issues. Rather we are operating within the context of two operational principles as stated in [3] , one drawn from the computer security community and the other from the machine learning community:

- All security mechanisms are susceptible to some form of compromise, and substantially increasing the strength or reliability of any given mechanism may require disproportionate effort or cost. Thus, it is desirable to construct a layered network of imperfect but relatively cheap defenses. We can have high confidence in the resulting system because of the redundancy of defenses even if we have only moderate confidence in each individual mechanism. This principal is known as defense in depth.
- It is well known in the machine learning literature that appropriate combination of a number of weak classifiers can yield a highly accurate global classifier. Although various forms of voting have been shown to be effective in many cases, a natural alternative in data-intensive domains is multi-layer or hierarchical classification. In such tiered decision systems, the function of the lowest layer of classifiers is to provide focus of attention, that is, to reduce the enormous data inputs to a manageable load for the more computationally intensive upper layers of the hierarchy which are responsible for such tasks as feature identification and final classification.

Thus, we do not aspire to provide a complete security solution but an incremental benefit. The sensors we describe in this study is intended to function as a focus of attention unit at the lowest level of a classification hierarchy; final decisions of hazard level and notification of security officers is the responsibility of a higher level agent with access to distilled results from many data sources. Nor do we intend to consider all of the thousands of potentially relevant data sources;

The objective, therefore, for these sensors is threefold as stated in [3]:

- To provide a reasonable level of accuracy and data reduction to a higher level classifier in the decision hierarchy.
- To provide results in a timely manner so that security actions can be taken promptly.
- And to run efficiently, imposing little resource burden on the classification hierarchy and the computer system as a whole.

In terms of accuracy, we focus on ability to discriminate impostors, recalling that a false alarm on the valid user can potentially be discarded by a higher-level decision maker but that an impostor missed by this sensor will never be available to higher-level classifiers.

As a conclusion in this study we aim to develop a model or profile of normal working state of web application user and to detect anomalous conditions as deviations from the expected behavior patterns. In other words we aim to augment the security of a private individuals account by monitoring usage activity for suspicious incidents that do not conform to known behavior patterns of the account owner. For this purpose we propose different approaches to develop sensors considering the remarks explained in this section. Moreover we discuss the success and the application dependency of these sensors on a sample web application.

Chapter 2

Issues and Related Work

2.1 Anomaly Detection

Anomaly detection as an approach for Intrusion Detection Systems was first proposed by Dorothy Denning [4] in 1987. It was suggested that by observing abnormal behavior data, anomalies can be detected. This was based on concept that abnormal data is in some way necessarily different from normal data, and if some measure of normal/abnormal data can be deduced further anomalies can be detected. In [5] anomaly detection was defined as follows: "Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive". Under this definition, the scope of anomaly detection encompasses not only violations by an outsider but also anomalies arising from violations on the part of an authorized user (the trusted insider threat) as stated in [1]. In the taxonomy of Intrusion Detection Systems like [6] and [7], anomaly detection was generally classified as an independent approach from other classical intrusion detection approaches.

Anomaly detection has historically been viewed as a difficult problem in computer science. Conventional techniques to solve this problem have not yielded any satisfactory solutions. One of these approaches is to develop a set of rules which describes all possible misuse scenarios and to employ some form of pattern matcher to activate appropriate rules. This form of detection is appealing because it can detect patterns of attack quickly as it does not depend on aggregate data that must be acquired over an extended time period such as statistics or behavioral profiles. However these security policies are decided by humans and the number of possible states that a computer system can be in is huge, complete partitioning of this space into 'abusive' and 'normal' is almost impossible. So, heuristic and machine learning approaches are often looked upon as potential solutions for this problem.

The following sections will introduce these techniques and recent works briefly.

2.1.1 Data Mining for Anomaly Detection

Data mining generally refers to the process of extracting descriptive models from large stores of data. Data mining techniques as an approach for anomaly detection was first proposed by Wenke Lee [8]. His research aimed developing a set of tools that can be applied to a variety of audit data sources to generate intrusion detection models. He considered intrusion detection as a data analysis process. The main purpose of his approach was to apply data mining techniques to extensively gathered audit data to compute models that accurately capture the actual behavior of intrusions and normal activities. Because of the huge volume of audit data, both in the amount of audit records and in the number of system features, efficient and intelligent data analysis tools were required to discover the behavior of system activities.

The recent rapid development in data mining has made available a variety of algorithms, drawn from the fields of statistics, pattern recognition, machine learning, and databases. Several types of algorithms are particularly useful for mining audit data:

- **Classification** maps a data item into one of several predefined categories. An ideal application in intrusion detection is to gather sufficient normal and abnormal audit data for a user or a program, then apply a classification algorithm to learn a classifier that can label or predict new unseen audit data as belonging to the normal class or the abnormal class. Kang et. al. used bag of system calls representation to classify normal and intrusive activities [9]. Their experimental results showed that standard machine learning techniques are surprisingly effective in misuse detection when they were used to train misuse detectors using simple bag of system calls representation.

- **Link analysis** determines relations between fields in the database records. Correlation of system features in audit data can serve as the basis for constructing normal usage profiles.

- **Sequence analysis** models sequential patterns and can discover what time-based sequences of audit events are frequently occurring together. These frequent event patterns provide guidelines for incorporating temporal statistical measures into intrusion detection models. This concept will be discussed in the Section 2.1.4.

2.1.2 Decision Trees and Rules for Anomaly Detection

Decision trees and rules are examples of disjunctive normal form (DNF) models. Decision rules are similar in characteristics to decision trees, they however also have some potential advantages brought front by being a stronger model and having better explanatory capabilities. DNF rules, unlike trees, need not be mutually exclusive. Thus, their solution space includes all tree solutions. In 1995 Weiss published the most important study in this area on Machine Learning for predicting anomalies using decision trees and rules [10].

In one of the recent works, Chan et. al. proposed an algorithm called LERAD that can learn the characterization of normal behavior in the form of logical rules [11].

2.1.3 Genetic Programming for Anomaly Detection

These techniques are based on the phenomenon of natural selection. A population of solutions, which have not been created by the programmer explicitly, are tested against each other using an evolution function. The next generation of solutions is developed from the current generation using techniques such as mutation and crossovers. The major problem in these techniques is to find an evaluation function to judge the performance of various solutions. The first anomaly detection technique using Genetic Programming was developed by Ludovic Me [12] in 1998. He applied a genetic algorithm tool called GASSATA on a command sequence.

2.1.4 Sequence Learning for Anomaly Detection

There are two major ways in which the sequence learning problem differs from many previously examined learning tasks. Temporal learning has been studied for time series data, but almost exclusively for numerical time series in which the existence of a full ordering and a distance metric allow the application of many powerful mathematical techniques. Time series of discrete elements however, do not inherently possess such distance properties so techniques such as spectral analysis, clustering, or neural networks for temporal prediction are not directly applicable. Discrete, non-metric

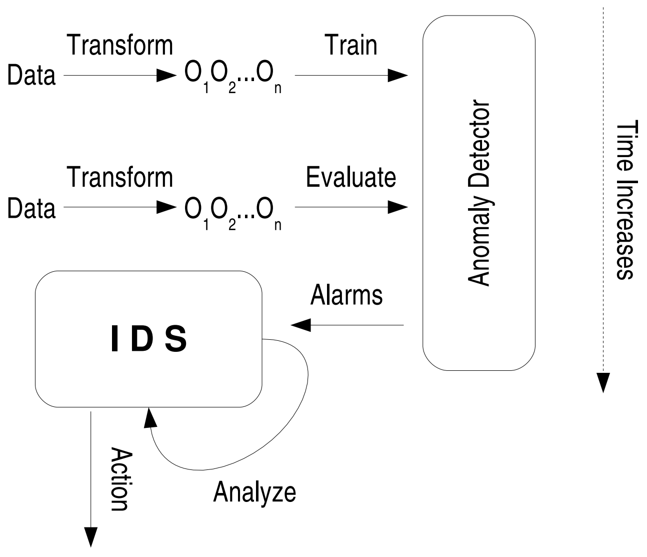


Figure 2.1: Simple model of Sequence Learning for Anomaly Detection

spaces have been studied, but mostly for the atemporal case, in which an instance is considered to be a feature vector in which element ordering is nor significant. Decision trees, summary statistics, rule learners have been used for learning on discrete valued spaces but all of these methods consider each feature independently. That is, the contribution of a single feature to the final classification depends only on its value and not on its position within the sequence, its relation to the preceding and succeeding features. Thus, a straightforward application of such techniques ignores the information contained within the temporal relations among elements.

There also exists learning methods explicitly developed to model sequence data. Methods for learning the structure of deterministic finite-state automata (DFA) have been widely studied. DFA's, however, are not well suited to modeling highly noisy domains such as human-generated computer interaction data. The simplest extension of DFA models to noisy domains are Markov Chain Models, which allow stochastic state transitions. These models have the advantage that the Maximum-Likelihood estimation for transition probabilities has a closed form. Markov Chain Models typically emit symbols deterministically (requiring a state for each symbol of the alphabet). When the alphabet is large, the dimensionality of the parameter space is high and the amount of training data required to accurately estimate low probability transitions is very large. Finally, deterministic output Markov models with unique states can only represent a single context for any given symbol.

In late 90's Lane and Brodley proposed a method that aroused great interest in this area using Hidden Markov Models [13]. Their approach was based on the following important assumptions:

- Human-Computer interaction is essentially casual in nature, which means that a particular user always reacts in the same manner when faced with a particular situation.
- A user's behavior is characteristic and differs on a per user basis.

In studies [13], [2], and [1] they worked with historical data from UNIX command usage patterns (extracted from UNIX Shell). This data was used to make profiles for individual users in the system. For any such user, the system compared his current behavior with his profile and classified it as being consistent with his past record or not. If his current behavior were found significantly different from what was normal for him, an alarm was raised indicating an anomaly. This relatively simple approach yielded reasonably good results dependent on the number of optimal hidden states which reflected the measure of the syntactic complexity present in the command line data.

In 1996 it was founded that when a vulnerable UNIX system program or server was attacked (for example, using a buffer overflow to open a root shell), the program made sequences of system calls that differed from the sequences found in normal operation [14]. In this work n-gram models (sequences of $n = 3$ to 6 calls), and matching them to sequences observed in training were used. A score was generated when a sequence observed during detection was different from those stored during training. Later on there have been different additions to typical n-gram models that solely relied on sequences of system calls [15].

In light of these studies, there have been other studies to model program behaviors with Hidden Markov Models. This approach usually yielded reasonably good results. For example, Cho and Park modeled privilege flows using Hidden Markov Models [16], [17]. Another study modeled program sequences of system calls for anomaly detection in UNIX programs [18].

2.2 Analysis of Web Applications

The World Wide Web, initially intended as a way to publish static hypertexts on the Internet, is moving toward complex applications. Static web sites are being gradually replaced by dynamic sites, where non trivial computation is performed before serving the dynamic content. Massive amounts of web data are being collected and stored everyday for different reasons, for example to detect fraud or malicious visitors, to improve the organization of web site to better serve customers, or to identify hidden patterns and new trends in consumer behavior for improving profit. Growing size of web applications make security considerations become more important while analyzing web applications [19].

A variety of data mining techniques have been used to analysis web data. Association rule extraction, collaborative filtering, clustering, dependency modeling, dynamic model extraction, and sequential pattern analysis are the most common and noticeable of these methods.

- **Association rule extraction** has been used to identify sets of items that are accessed together.
- **Collaborative filtering algorithms** have been used to find similar users based on the overlap between their requested items, and then recommend the given user items accessed by the like-minded users.
- **Clustering** has been used to group similar items or users with similar usage patterns. This approach generally has been used within other methods.
- **Dependency modeling** has been used to discover and represent dependencies among different variables such as, for instance, the effect of gender on the shopping behavior. CleverSet [20] approached the web visitor behavior modeling task using dynamic relational Bayesian modeling to infer motivation, preferences, and the probability making a purchase. Their models provided a rich vocabulary to describe the web visit in terms of customer-centric and web-centric variables.
- **Dynamic model extraction** has been used for extraction of UML model of a web application to solve problems related to traditional testing. Tonella and Ricca extracted a model from an existing web application by means of a semi-automatic procedure, requiring user involvement only in the specification of a

set of input values covering all the internal states of the application [21]. In their work, partial models could still be constructed whenever full coverage of all possible internal states was not granted by the available set of inputs.

- **Sequential pattern analysis** has been used to discover patterns such that the current history of actions is evidence to the following action. The sequential pattern analysis has been widely used for behavioral prediction. Hafri et. al. applied probabilistic exploration using Markov models and clustered these models to make possible the prediction of future states to be visited [22]. Cadez et. al. analyzed the sequences of URL categories traversed by users and made model-based clustering by learning a mixture of first-order Markov models to make possible the visualization of navigation patterns on a web site [23]. Manavoglu et. al. used maximum entropy based mixture models for generating probabilistic behavior models [24]. They first built a global behavior model for the entire population and then personalized this global model for the existing users by assigning each user individual component weights for the mixture model. Kruegel and Vigna focused on structural inference from request attributes and modeled query attributes with Hidden Markov Models to generate probabilistic regular grammars for anomaly detection of web based attacks [25]. Chen et. al. defined session comparison measurements for masquerading detection [26]. Also behavioral modeling using sequential pattern analysis were used in testing environments of web applications in the studies like [27] and [28].

Up to this section we have glanced at recent works related to this study. In the following sections we will give brief information about some concepts used in this study.

2.3 Markov Chains

A Markov chain is a discrete-state random process in which the only state that influences the next state is the current state. In other words, we have a set of states $S = \{s_1, s_2, \ldots, s_n\}$. The process starts in one of these states and moves successively from one state to another. If the chain is currently in state s_i, then it moves to state s_j, at the next step with a probability denoted by p_{ij}, and this probability does not depend upon which states the chain was in before the current state. A finite state machine is an example of a Markov chain.

In a Markov chain the following conditions are satisfied:

$$p_{ij} \geqslant 0 \, \forall (s_i, s_j) \in S^2 \tag{2.1}$$

$$\sum_{s_j \in S} p_{ij} = 1 \, \forall s_i \in S \tag{2.2}$$

The transition matrix is defined as:

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

To be more precise, let $(X_t)_{t \in \mathbb{N}_0}$ be a sequence of random variables with values in S. The sequence $(X_t)_{t \in \mathbb{N}_0}$ is called a Markov chain with discrete time, state space S, and transition matrix P, if for every $t \in \mathbb{N}_0$ the condition

$$P(X_{t+1} = s_j | X_0 = s_{i_0}, \ldots, X_t = s_{i_t}) = P(X_{t+1} = s_j | X_t = s_{i_t}) = p_{i_t j} \tag{2.3}$$

is satisfied for all $(s_{i_0}, \ldots, s_{i_t} \in S^{t+2})$, for which $P(X_0 = s_{i_0}, \ldots, X_t = s_{i_t}) > 0$.

The first identity in the Equation 2.3 is called Markov property, and the second identity assures that the transition probabilities do not vary with the time t.

Markovian systems appear extensively in physics, particularly statistical mechanics, whenever probabilities are used to represent unknown or unmodelled details of the system, if it can be assumed that the dynamics are time-invariant, and that no relevant history need be considered which is not already included in the state description. Markov chains can also be used to model various processes in queueing theory and statistics. Markov chains have also been used to analyze web navigation behavior of users. A user's web link transition on a particular website can be modeled using first order Markov chains and can be used to make predictions regarding future navigation and to personalize the web page for an individual user. Google also uses Markov chains to calculate the page rank of a web page.

2.4 Hidden Markov Models

A Hidden Markov Model (HMM) is a doubly stochastic process with an underlying stochastic process that is not observable, and can only be observed through another set of stochastic processes that produce the sequence of observed symbols [29]. HMM is a useful tool to model sequence information. This model can be thought of as a graph with N nodes called 'state' and edges representing transitions between those states. Each state node contains initial state distribution and observation probabilities at which a given symbol is to be observed. An edge maintains a transition probability with which a state transition from one state to another state is made.

Given an input sequence $O = O_1, O_2, \dots, O_T$, HMM can model this sequence with its own probability parameters using Markov process though state transition process can not be seen outside. Once a model is built, the probability with which a given sequence is generated from the model can be evaluated. A model λ is described as $\lambda = (A, B, \pi)$ using its characteristic parameters. The parameters used in HMM are as follows:

$$
\begin{aligned}
T &= \text{length of observation sequence} \\
N &= \text{number of states in the model} \\
M &= \text{number of observation symbols} \\
S &= \{S_1, S_2, \dots, S_N\}, \text{ set of states} \\
q_t &= \text{state at time t} \\
O_t &= \text{observation at time t} \\
V &= \{v_1, v_2, \dots, v_N\}, \text{ discrete set of possible symbol observations} \\
A &= \{a_{ij} | a_{ij} = P(q_{t+1} = S_j | q_t = S_i)\}, \text{ state transition probability distribution} \\
B &= \{b_i(k) | b_i(k) = P(v_k | q_t = S_i)\}, \text{ observation probability distribution} \\
\pi &= \{\pi_i | \pi_i = P(q_1 = S_i)\}, \text{ initial state distribution}
\end{aligned}
$$

The HMM gets its name from two defining properties. First, it assumes that the observation at time t was generated by some process whose state q_t is hidden from the observer. Second, it assumes that the state of this hidden process satisfies the Markov property, that is, given the value of q_{t-1}, the current q_t is independent of all the states prior to $t - 1$. In other words, the state at some time encapsulates all we need to know about the history of the process in order to predict the future of the process. The outputs also satisfy a Markov property with respect to the states, that is, given q_t, O_t is independent of the states and observations at all other time indices.

There are three fundamental problems associated with the practical implementation of HMMs as prediction or estimation models:

Observation Probabilities: Given a sequence of observations, $O = O_1, O_2, \dots, O_T$, and a model, λ, calculate the probability of observing that sequence of observations under the model, $P(O|\lambda)$.

State Sequence Selection: Given a sequence of observations, $O = O_1, O_2, \ldots, O_T$, and a model, λ, calculate the sequence of states, $Q = q_1, q_2, \ldots, q_T$, most likely (under some optimality criterion) to have generated O.

Model Training: Given a sequence of observations, $O = O_1, O_2, \ldots, O_T$, select the parameter set, $\lambda = (A, B, \pi)$, that maximizes $P(O|\lambda)$.

Previously mentioned approaches which use HMMs for anomaly detection are generally interested in calculating observation probabilities and training model. Algorithms to solve these problems and other detailed information about HMMs can be found in the Appendix A.

Rabiner widely used HMMs in speech recognition and defined most of the concepts explained in his studies [30], [31], and [32]. Besides speech recognition and anomaly detection HMMs have been widely used in optical character recognition, machine translation, prediction of protein-coding regions in genome sequences, modeling families of related DNA or protein sequences and in many other domains.

Chapter 3

Anomaly Detection in Web Applications

Up to this section we have investigated recent works and issues related to this study. In this section we will focus on the solution of anomaly detection problem in web applications using personal usage patterns and present both differences and similarities with other approaches to anomaly detection in other domains.

3.1 Anomaly Detection from Usage Patterns

We propose that anomalies can be detected from personal usage patterns based on the following assumptions:

- Demands of web users from an application do not change rapidly and frequently.
- Current actions of web users and responses reflect the unique demands of the current situation, but they are also shaped out of previously learned behaviors and skills.

By means of these assumptions anomalies can be detected as deviations from the expected behavior patterns in a web application. In other words we propose that the security of a private individual's account can be augmented by monitoring usage activity for suspicious incidents that do not conform to known behavior patterns of the account owner.

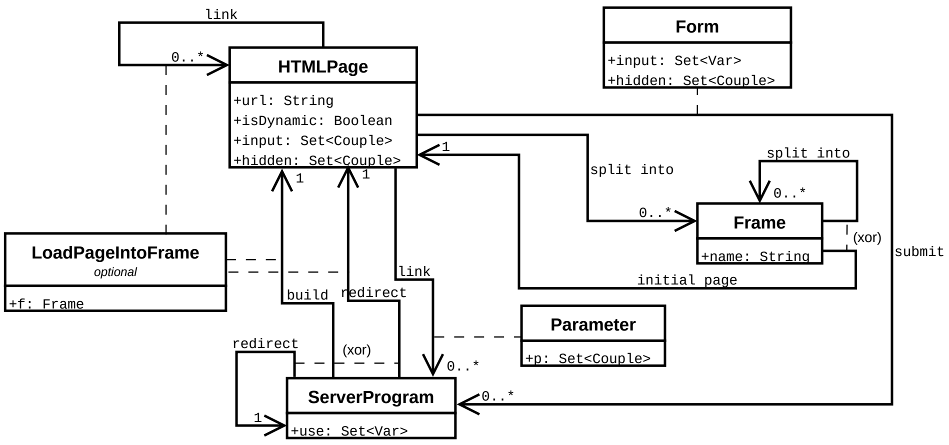


Figure 3.1: Meta model of a generic Web application structure.

3.2 Observations through Web Application

The Figure 3.1 shows the meta model used to describe a generic Web application [21]. As a summary of this model stated in [21], the central entity in a web application is named the HTMLPage. An HTML page contains the information to be displayed to the user, and the navigation links toward other pages. Organization and interaction facilities (e.g. frames and forms) are also included in an HTML page. Navigation from page to page is modeled by the auto-association of class HTMLPage named link. Web pages can be static or dynamic. While the content of a static web page is fixed, the content of a dynamic page is computed at run time by the server and may depend on the information provided by the user through input fields. The class ServerProgram models the executable that runs on the server side and generates a dynamic HTML output. When the content of a dynamic page depends on the values of a set of input variables, the attribute use of class ServerProgram contains them. A server side program can be executed by traversing a link from an HTML page whose target is the server executable and whose attributes include a set of parameters, represented as $\langle name, value \rangle$ or by submitting a form. The server program can either redirect the request to another server program, build an output, dynamic HTML page, or simply redirect to a static HTML page. The latter two cases can be distinguished only because the resulting HTML page is respectively static or dynamic. When a server program builds a dynamic page, the input and hidden variable values that have been provided to it are stored in the attributes input and hidden of the resulting page, as sets of couples $\langle name, value \rangle$.

A frame is in fact a rectangular area in the currently displayed page where navigation can take place independently. Moreover, the different frames into which a page is decomposed can interact with each other, since a link in a page loaded into a frame can force the loading of another page into a different frame. Organization into frames is represented by the association split into, whose target is a set of Frame entities. Today's DHTML technology allows dynamic loading of not all of but some of the parts in an HTML page. Hence, a frame can be considered as a virtual partition of an HTML page in order to cover this new technology.

In HTML user input is gathered by exploiting a Form and is passed to a server program, which processes it, through a submit link. A web page can include any number of forms. Each form is characterized by the input variables that are provided by the user through it. Additional hidden variables are exploited to record the state of the interaction. They allow transmitting pairs of the type $\langle name, value\rangle$ from page to page. Typically, the constant value they are assigned needs be preserved during the interactive session for successive usage. Since the HTTP protocol is stateless, this is the basic mechanism used to record the interaction state (a variant is represented by the cookies).

In a web application, the same server program may behave differently, according to the interaction state. To clarify this situation it is convenient to classify server programs into two categories as stated in [21]:

- Server programs with state-independent behavior.
- Server programs with state-dependent behavior.

Servers programs in the first category always exploit the same mechanism to produce the output and generate a dynamic page whose structure and links are fixed. The behavior of these server program is the same in every interaction state. On the contrary, server programs in the second category behave differently when executed under different conditions. A server program may, for example, two completely different computations and consequently different output pages according to the value of a hidden flag recording a previous user selection.

In presence of server programs with state-dependent behavior, the paths in the model can still be interpreted as navigation sessions, provided that server program and related output page are replicated for all possible variants. The resulting model is called an explicit state model. Tonella and Ricca suggested page merging heuristic criteria to simplify this explicit state model [21] for dynamic model extraction described in the Section 3.3.3. In this study we call each class of HTMLPage of the explicit state model as a 'state' of web application. Moreover we call each class of interaction, action request, between a HTMLPage and the ServerProgram as an 'observation'. Besides one can always produce an exact action request that is listed below manually, an observation can occur because of the following reasons:

- When user clicks a link or submits a form in a web page.
- When user enters a url to the browser to direct access to a web page.
- When user uses the back-forward buttons of the browser to repeat an action request in the history.
- When the web application produces DHTML requests in the background.

Based on the following facts we propose that web users' actions observed through a web application can be seen as observations generated by a Markov chain which satisfies the Markov property with hidden states:

- Based on the nature of web applications an action request observed through a web application is independent of the states and observations at all other time indices, since HTTP is a stateless protocol. Each request is executed independently, without any knowledge of the requests that came before it.
- The request through a web application at time t was generated by some process whose state q_t is hidden from the server side.

The first fact should not be confused with the result of the request, because the response may depend on previous actions. Moreover, the latter fact should not be confused with the internal states of the web application, because as stated before one can always produce an action request manually, which means web user's requests do not have to depend on the result of the last request. Hence the state at time t of the process that generates the action requests is hidden.

3.3 The Proposed Approaches

In this section we will intoduce some approaches as solution candidates of the problem described in this study. These approaches mainly depend on Markov chains and hidden Markov models. The former proposed approach make use of Markov chains to model normal working state of the web application user. This approach can be seen as one of the simplest representation of the problem with some native disadvantages. The latter approaches utilize hidden Markov models. These approaches are inspired from Lane's studies with Brodley to detect anomalies from Unix command usage patterns. The first approach is the application of original study on Unix command usage patterns to web application usage patterns. The last approach aims to give solutions to problems of the original approach.

3.3.1 Markov Chain Model

At first glance one can expect that modeling actions of web applications as a Markov chain constitutes a good approach. Because we can easily collect the statistical information of how many times an action is observed after a specific action, which can be used to calculate the transition probabilities.

Nevertheless Markov chain models typically emit symbols deterministically which means $|\Sigma|^2$ total transition probabilities to be learned for an alphabet of size $|\Sigma|$ are required and each state is emitting only a single symbol. Hence, they can only represent a single context for any given symbol. In the anomaly detection domain symbols can have multiple contexts. For example, in a Unix shell environment the command *vi* can be employed for editing both source code and conference papers [2]. Analogically requests in a web application can have multiple contexts. For example, in a music sharing application, the request for play in the main page can mean that the user is willing to play popular choices brought by the application, which has a different context from a regular play request resulting from a search.

3.3.2 Hidden Markov Model

By analogy with above facts in the Section 3.2, requests of web users can be considered as commands in a Unix shell environment. Reasoning by this analogy we can approach to this problem with the approach that was used to detect anomalies from Unix command usage patterns by Lane [1]. That is, we can train a hidden Markov model with history of each user and use Forward-Backward algorithm to calculate the probability of new sequences observed.

As stated before, Lane and Brodley's studies [13], [2], and [1] yielded reasonably good results dependent on the number of optimal hidden states which reflected the

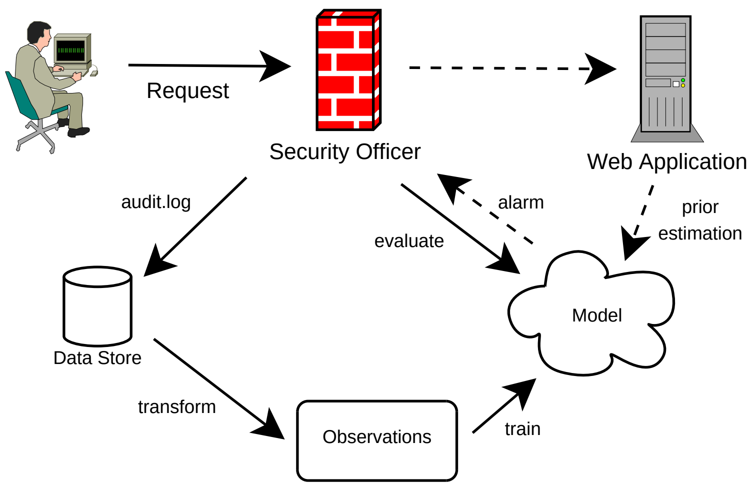


Figure 3.2: Overview of the proposed approach with prior estimation

measure of the syntactic complexity present in the command line data. Considering the fact that the view layer of the web application generally shapes the requests, in general HMMs with the number of hidden states about the number of nodes in the model of the application will be enough to represent the syntactic complexity of a web user.

3.3.3 Hidden Markov Model With State Transitions

Lane and Brodley suffered from not only the number of optimal hidden states but also the lack of prior estimation of transition probabilities of HMMs. Nevertheless anomaly detection in web applications with the similar approach can go beyond this problem. One can use the structure of the web application which can be extracted by semi-automatic [21] or manual methods to estimate the number of optimal hidden states and the initial transition probabilities. Because we know that most of the time, web users' requests will follow the view layer of the web application. With this prior information HMMs are expected to be avoided getting stuck in a poor local maximum and slow convergence problems.

For dynamic model extraction Tonella and Ricca suggested three page merging heuristic criteria to simplify the explicit state model [21]:

- Dynamic and static pages that are identical according to a character-by-character comparison are considered the same in the model.

- Dynamic pages that have identical structure, but different texts, according to a comparison of the syntax trees of the pages are considered the same in the model.
- Dynamic pages that have similar structure, according to a similarity metric, such as the tree edit distance, computed on the syntax trees of the pages are considered the same in the model.

Our approach will follow these conventions to produce a finite prior model of a web application. These techniques require user involvement in the specification of a set of input values covering all internal states of the application and can produce partial models which are still of interest whenever full coverage of all possible internal states is not granted by the available set of inputs. Hence adding some additional nodes would be reasonable, because the extracted model can be partial or the user may not follow the view layer.

In summary, when an action is requested from the application, a high level security officer will collect the audit logs. These collected logs will be labeled as observations. Using these labeled observations and prior explicit state model information from the web application, a behavioral model is trained and evaluated against newly observed actions informing the security officer about anomalities when necessary. The Figure 3.2 illustrates this approach.

Finally, different from the other approaches using HMMs for behavioral prediction in Web applications, our approaches are completely personal view and deal with task-centric observations (not deal with the semantic category of the requested data). However some requests can be clustered into sub-categories to have better personalization in behavioral usage. For example, a search request can be labeled with search criteria tag in order to learn search habits.

3.4 Data Collection

To learn characteristic patterns of actions, our approach uses the sequence (an ordered, set of temporally adjacent actions in a user session) as the fundamental unit of comparison. These sequences can be collected from audit logs or directly from the application. As expected the latter case is easier way for this job. However usually it is not possible to gain access to the application layer. Therefore understanding the user behavior require discovering the valuable information within audit logs. This involves several phases, details of which are out of the scope of this study:

- **Data cleaning and preprocessing**, where typically noise is removed, log files are broken into sessions and users are identified.
- **Data transformation**, where useful features are selected to represent the data and dimension reduction techniques are used to reduce the size of the data.

3.5 Prerequisites for The Proposed Approaches

In consequence of dealing only with requests, our approaches do not have any requirements related to data collection as stated before. However, in order to apply our approaches to a web application, the web application should satisfy the following single assumption:

- The structure of the web application should have enough number of depths and branching factors to be able to produce distinguishable models of usage patterns.

Moreover in the training phase the following conditions should be satisfied to get reasonable results from our approaches:

- The model of the application covering most of the internal states should be extracted.
- The profile of a user should be trained with enough number of valid observations covering the usage behavior from a single user.

The first item is required, because the prior information can affect the convergence speed of hidden Markov models initialized with state transitions of the web application, that is the number of observations to train. The latter is trying to make a point of the quality of the observations. From the point of a user's view, initial sessions in a web application can be seen as the learning phase of the application. Hence HMMs should be trained taking into account that observations from this learning phase can produce incorrect behaviors.

3.6 Sequence Labeling for Anomaly Detection

We use similar techniques for sequence labeling from Lane and Brodley's studies [2]. We construct a single model, λ, to model the observed behavioral patterns of the valid user. The likelihoods of incoming data sequences are evaluated with respect to λ and those judged insufficiently likely via a threshold test are labeled as anomalous. The value of this 'minimum acceptable likelihood' is denoted t_{min} in their studies. A feature of the anomaly detection domain is the threat of 'replay attacks' [1]. However in web applications observations are usually too similar to historical behaviors. Hence we will not introduce an upper threshold to flag these attacks.The threshold t_{min} is chosen from the lower r quantiles of the non-parametric distribution of observation likelihoods on an independent, 'parameter selection', subset of the training data. The parameter r corresponds to an 'acceptable' false alarm rate and its selection is a site-specific issue related to security policy.

We wish to be able to label arbitrary subsequences of the observed data. For models with Hidden Markov Model approach we can run the forward-backward likelihood estimation algorithm between every possible pair of subsequence start, s, and termination, t time steps. However this turns out to be computationally expensive. Hence we consider all fixed-length subsequences ($t - s = l$ for some fixed l), denoted as 'window size'. Similarly for Markov chains we consider the probability of all fixed-length subsequences calculated with Bayes' formula.

[1]A replay attack is one in which an attacker monitors a system and records information such as user commands. These commands can then later be replayed back to the system literally (or with the inclusion of a very few hostile actions). Because the vast majority of the data was, in fact, originally generated by the valid user, it will appear perfectly normal to the detection sensors unless some check is made for occurrences which are too similar to past behavior.

Chapter 4

Experimental Results

4.1 Test Environment

Considering the fact that the structure of the web application should satisfy some criteria stated in the Section 3.5, choosing the right environment for testing is important. The following remarks have been taken into consideration while constituting the environment:

- It is preferable that in the chosen application, how the requested data to be reached should depend on user preferences and habits. In such a case it is expected that distinguishable models of usage patterns will be provided.
- It is not required but preferable that the application grant access to the source code. Therefore action requests can be identified in the application layer without the need of data cleaning, preprocessing and transformation.
- It is not possible to collect the data from a simulated environment. User preferences are important, hence users should use the application without being under pressure, that is, the data collection phase can take up long time. It is important to choose an application that users are willing to use.

Taking into account above remarks we have decided to use a web based music sharing application to test the proposed approach. The chosen application is an open source PHP-based tool called *ampache*[1] for managing, updating and playing various music files via a web interface. It allows the users to save public and private playlists, browse music files by album and artist, download song, album and playlist, random

[1]It can be reached from *http://www.ampache.org/*.

play on full song lists, albums, artists and genre, search on almost every field, choose per user theme, view user statistics of song, album, artist and genre played and many more features. We have installed the application in a LAN environment, where can be used frequently in weekdays, with a music catalog of 3 GB mp3 files and a dozen of users from different social profiles.

4.2 Structural Model Extraction of the Application

We have analyzed the application manually to extract the structural model. Using the principles Tonella and Ricca suggested [21], we have identified all possible action types and states that the application owned. After the identification phase, we have split some action types into sub-categories for better personalization in behavioral usage. We have labeled each search request with search criteria tag in order to learn search habits. For example, in a financial web application one can label a payment action with amount scale. Below is the labeled action types:

Table 4.1: Possible identified observations.

Observation	**Description**
album_browse	Browse albums.
album_detail	Show album detail.
album_match	Show albums matching criteria.
album_show_all	Show all albums.
artist_browse	Browse artists.
artist_detail	Show artist detail.
artist_match	Show artists matching criteria.
artist_show_all	Show all artists.
artist_show_all_songs	Show all songs of the artist.
direct_link	Direct music file request to listen.
download	Download music file.
download_album	Download album.
download_playlist	Download playlist.
flag_view	Show flag view.
flag_song	Flag song file.
login	Login request.

Continued on Next Page. . .

Table 4.1 – Continued

Observation	Description
logout	Logout request.
main	Show main page.
play_album	Play album.
play_artist	Play artist.
playlist_add_to	Add song to playlist.
playlist_remove_from	Remove song from playlist.
playlist_browse	Browse playlists.
playlist_create	Create playlist.
playlist_delete	Delete playlist.
playlist_delete_cancel	Cancel playlist delete request.
playlist_delete_view	Show playlist delete view.
playlist_detail	Show playlist detail.
playlist_update	Update playlist preferences.
playlist_set_track	Set track numbers of playlist.
playlist_view_edit	Show playlist edit view.
playlist_view_new	Show create playlist view.
play_playlist	Play playlist.
play_popular	Play popular songs.
play_random	Play random songs.
play_random_album	Play random album.
play_random_artist	Play random artist.
play_random_playlist	Play random playlist.
play_selected	Play selected songs.
play_song	Play song.
play_uploaded	Play uploaded song.
play_your_popular	Play user popular songs.
preferences_update	Update user preferences.
preferences_view	Show preferences view.
search_by_album	Search by album name.
search_by_artist	Search by artist name.
search_by_song_filename	Search by song filename.
search_by_song_title	Search by song title.

Continued on Next Page…

Table 4.1 – Continued

Observation	Description
search_by_genre	Search by genre.
search_view	Show search view.
stats_view	Show statistics view.
unknown	Unidentified action request.
upload	Upload new song.
upload_view	Show upload view.
user_change_password	Change user password.
user_update_profile	Update user profile.
user_clear_stats	Clear user statistics.
user_profile_view	Show user profile view.

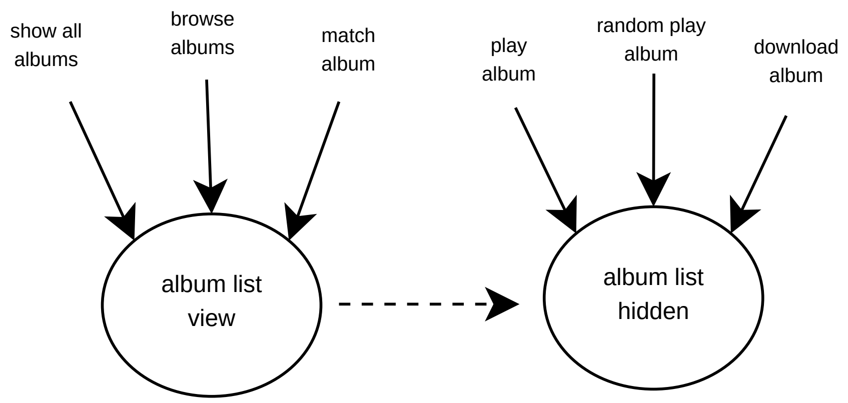


Figure 4.1: Sample view and its generated view.

Different from the regular web applications, this application has some actions which are not handled directly by browsers or do not change the current view, like downloading binary content and playing stream. We have handled these actions like additional views generated from original view. Nodes representing these additional views does not allow transition to other nodes representing additional generated views. These nodes are only attainable from nodes of original views which generated these views. For example, show all albums, browse albums, and match album observations lead up to a view listing albums. In this view user can play, or download album which does not change the current view. So we represent these actions like generated views with special transition probabilities. This situation is illustrated in the Figure 4.1.

The following table illustrates the states in the application without generated views:

Table 4.2: Extracted HTML page classes.

State	**Observations Emitted**
album	*album_browse*, *album_match*, *album_show_all*
album_detail	*album_detail*
artist	*artist_browse*, *artist_match*, *artist_show_all*
artist_detail	*artist_detail*
flag_view	*flag_view*, *flag_song*
logout	*logout*
main	*login*, *main*
playlist	*playlist_browse*, *playlist_create*
playlist_detail	*playlist_detail*, *playlist_add_to*, *playlist_remove_from*, *playlist_set_track*
playlist_view_delete	*playlist_delete*
playlist_view_edit	*playlist_view_edit*, *playlist_update*
playlist_view_new	*playlist_view_new*
preferences_view	*preferences_view*, *preferences_update*
search_view	*search_view*, *search_by_song_filename*, *search_by_artist*, *search_by_album*, *search_by_genre*, *search_by_song_title*
stats_view	*stats_view*
upload_view	*upload_view*, *upload*
user_profile_view	*user_profile_view*, *user_clear_stats*, *user_update_profile*, *user_change_password*

The following table illustrates the generated views in the application:

Table 4.3: Generated views.

Generated From	Observations Emitted
album	*play_album*, *play_random_album*, *download_album*
album_detail	*play_album*, *play_selected*, *play_song*, *download_album*, *play_random_album*, *direct_link*, *download*
artist	*play_artist*, *play_random_artist*
artist_detail	*download_album*, *play_album*, *play_artist*, *play_selected*, *play_random_artist*
main	*play_song*, *play_popular*, *play_random*
playlist	*play_random_playlist*, *download_playlist*
playlist_detail	*play_selected*, *play_song*, *play_playlist*, *direct_link*, *play_random_playlist*
search_view	*play_album*, *play_artist*, *play_selected*, *play_song*, *play_random_album*, *play_random_artist*, *direct_link*, *download*, *download_album*
stats_view	*play_your_popular*
upload_view	*play_uploaded*

4.3 Data Collection

We have deployed the application in a web server with a intrusion detection and prevention engine called *ModSecurity*[2]. While collecting audit data from this module, we have modified the controller layer of the application to identify observations directly. We have recorded the observations in user history as broken into sessions.

[2]Detailed information can be obtained from *http://www.modsecurity.org/*.

4.4 Implementation

We have implemented the proposed HMM approaches using a HMM library in Java called *Jahmm*[3]. For ordinary HMMs we have initiated a random initialized HMM for each user. On the other side, for HMMs with prior estimations model, we have initiated a HMM for each user with prior estimation from structural model. For each node in the structural model a state in the HMM has created increasing the probability of the observations emitted by this node. Transition probabilities of these states have been adjusted as discrete uniform distribution except for the generated nodes. The transition probabilities related to generated nodes have been adjusted taking into account the remarks in the Section 4.2. Additional random initialized states have been added to capture unknown states.

Additionally, we have modified the model training algorithm of HMMs to avoid zero frequency problem, that is, the problem of preserving initial estimation. Even if the model did not observe some of the observations, it should not forget the initial estimation from structural model.

4.5 Testing Procedure

Because non-simulated human-level attack data has proved difficult to obtain, we have profiled our techniques on the user differentiation problem. In this formulation, data are gathered from valid system users under normal working conditions and a user profile is constructed for each. The performance of the anomaly detection sensor is evaluated with respect to the ability to correctly recognize the profiled user and discriminate the other users as anomalous. This framework simulates only a subset of the possible misuse scenarios that of a naive intruder gaining access to an unauthorized account but it allows us to evaluate the approach.

Because user behaviors change over time, the effective lifetime of a static user profile, as is employed in the work here, is limited. Thus we have constructed experiments to evaluate the detector's performance over a limited range of future activities. In order to test our proposed approaches we take an initial window of a user history and divide it into three groups: training, parameter-selection, and testing. After each testing phase we merge the sequences in the parameter-selection group with the sequences in the training group and interchange the testing group with the

[3]For more information refer *http://www.run.montefiore.ulg.ac.be/˜francois/software/jahmm/*.

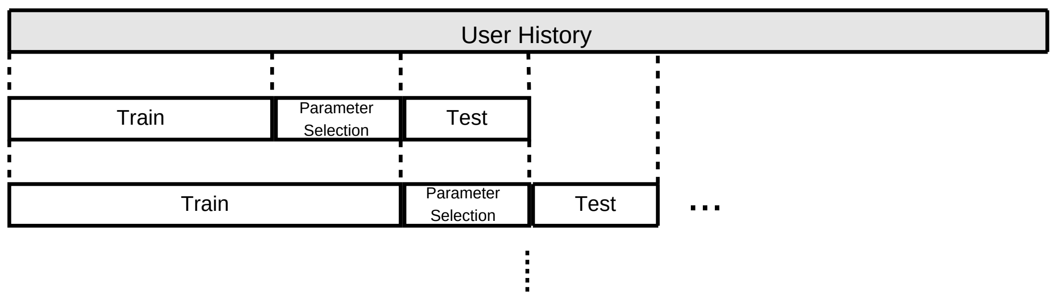


Figure 4.2: User history division for testing.

parameter-selection group taking a new testing group as illustrated in the Figure 4.2. This procedure continues until there exists no sequences to create a new testing group.

4.6 Test Results

After approximately three months of data collection period, we have chosen most active three users with more than thousand observations. We have divided chosen user histories into test sets according to the testing procedure and tested our models with observations of the other users including the users which are not so active as the chosen ones. All of these users are labeled from 1 to 12 successively. While presenting test results users are referred to as USERn where n is the number of the user. According to that our chosen users are USER4, USER7 and USER8.

In the following sections unless it is specified, the initial window of user histories contain 500 observations of training group, 200 observations of parameter-selection group and 200 observations of testing group approximately. These numbers are approximate since no user session is split while composing data sets. The thresholds are calculated according to the acceptable false-alarm rate of 0.05 in our tests. Also unless it is specified, the window size of subsequences is 20.

In the following sections we will compare our proposed approaches after presenting test results of each model with possible variations. We have used two criteria to compare our approaches:

- **Percentage Alarms:** This is the percentage of alarms in the test data that the sensor has detected. As stated in [2], the goal in the anomaly detection task is to identify potentially malicious occurrences while falsely flagging innocuous actions as rarely as possible. In this study, the rate of incorrectly flagging normal behaviors is denoted as 'false alarm rate' and the rate of failing to identify abnormal behaviors is denoted as 'the false acceptance rate'. Under the null hypothesis that all behavior is normal, these correspond to Type I and Type II errors, respectively. Moreover the converse accuracy rates are referred to as 'the true accept rate' and 'the true detect rate'. The ability to correctly accept the profiled user as normal is denoted as the true accept rate, whereas the ability to correctly detect an anomalous user is denoted as true detect rate. Hence in percentage alarm charts it is expected that a good sensor should give alarms to profiled user with low probabilities and give alarms to other users with high probabilities.

- **Mean Normal Runs:** This is the average length of the consecutive runs in the test data that the sensor have not produced any alarms. This is a measure of how quickly an anomalous or hostile situation can be detected. Hence, we wish the time to alarm to be short for hostile users so that they can be dealt with quickly and before doing much harm, but long for the valid user so that normal work is interrupted by false alarms as seldom as possible. Thus, in mean normal runs charts it is expected that a good sensor should have longer normal runs for profiled user than other users.

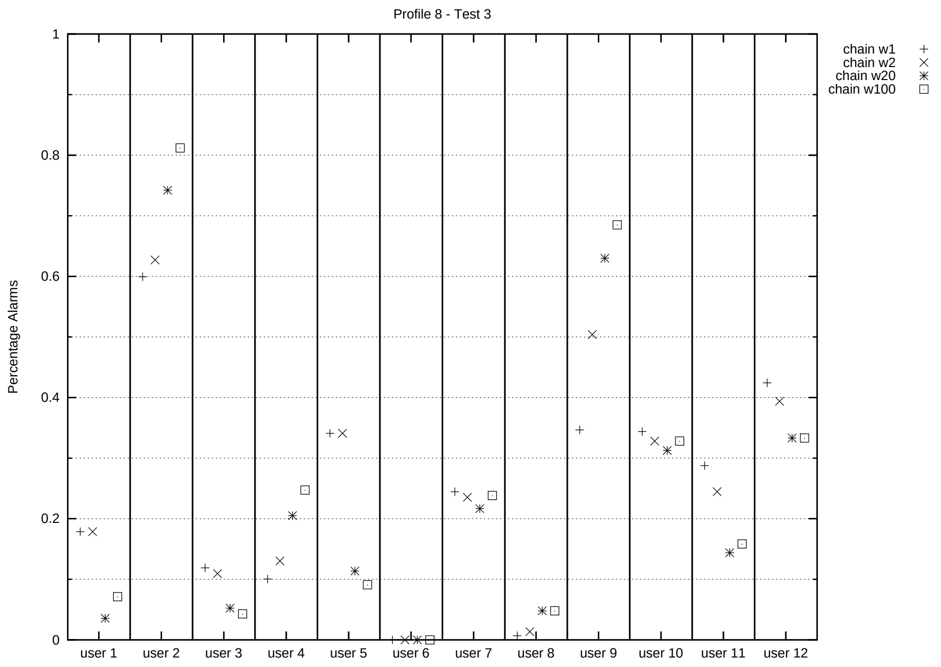
4.6.1 Markov Chain Model

As stated before to train a Markov model it is enough to collect simple statistics of transitions. After generating transition matrix from training data, we have calculated subsequences probabilities of parameterization data. We have chosen the threshold t_{min} from the lower 5% quantiles of the non-parametric distribution of subsequences probabilities of parameterization data.

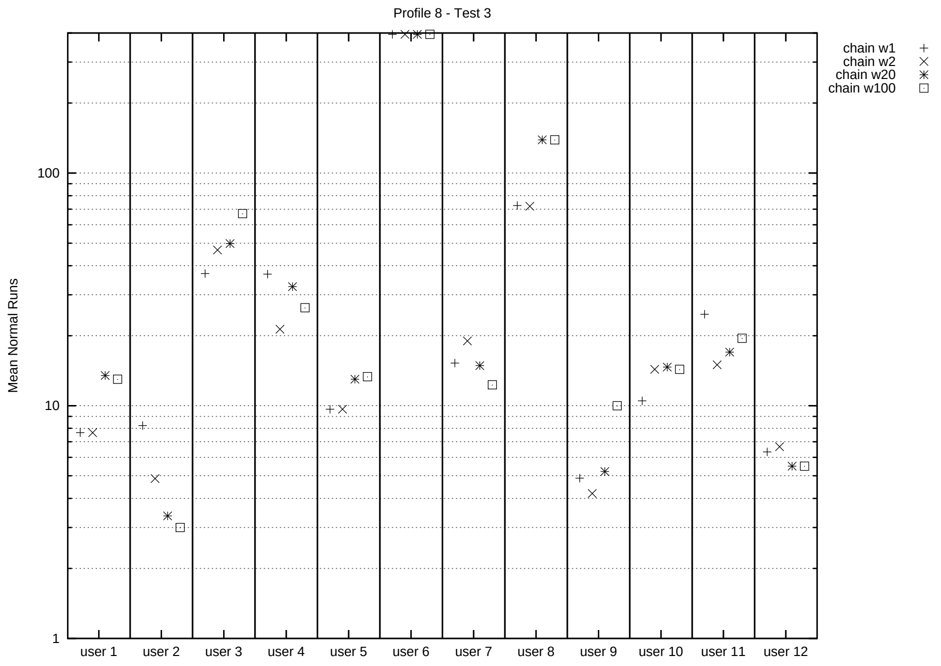
Although generating transition matrix is simple, setting initial transition numbers with zero forms a sparse transition matrix, which causes this model to give too many false alarms. To avoid zero frequency problem, we have initialized transition numbers as 1 and add each new transition to the statistics with a weight. We have tested this model with weights 1, 2, 20 and 100.

As an interpretation of test results, in some conditions Markov chain models are not able to differentiate profiled user from other users. Especially these models are not so successful for differentiating USER8 from USER6 and USER7 from USER4.

The situation for USER8 is illustrated in the Figures 4.3 and 4.4 which show the percentage alarms and the mean normal runs of these models. Generally the more observations are trained, the better results are obtained. As an explanation of these characteristics Markov chain models typically emit symbols deterministically which means $|\Sigma|^2$ total transition probabilities to be learned for an alphabet of size $|\Sigma|$ is required and each state is emitting only a single symbol as stated before. On the other hand no meaningful results are obtained by changing the weights. Other charts for these models can be found in the Appendix B.

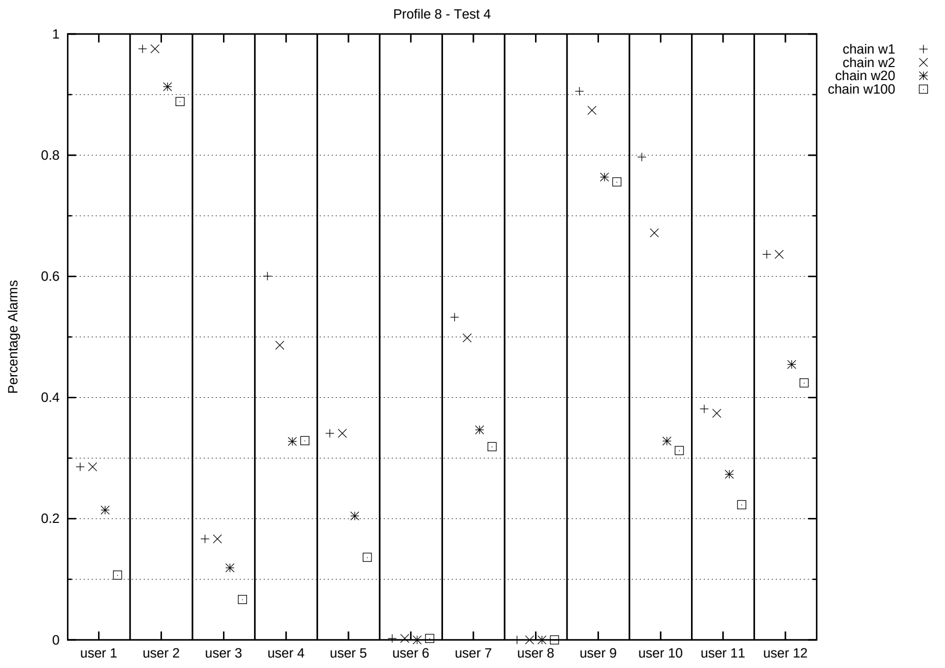


(a)

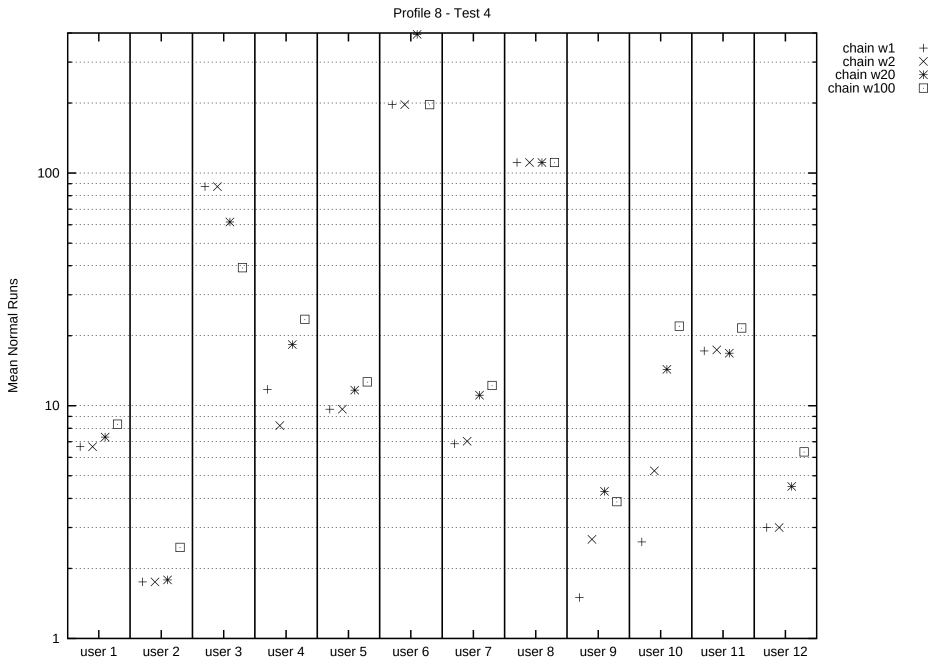


(b)

Figure 4.3: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER8's behaviors (Test Set 3 of 4).



(a)



(b)

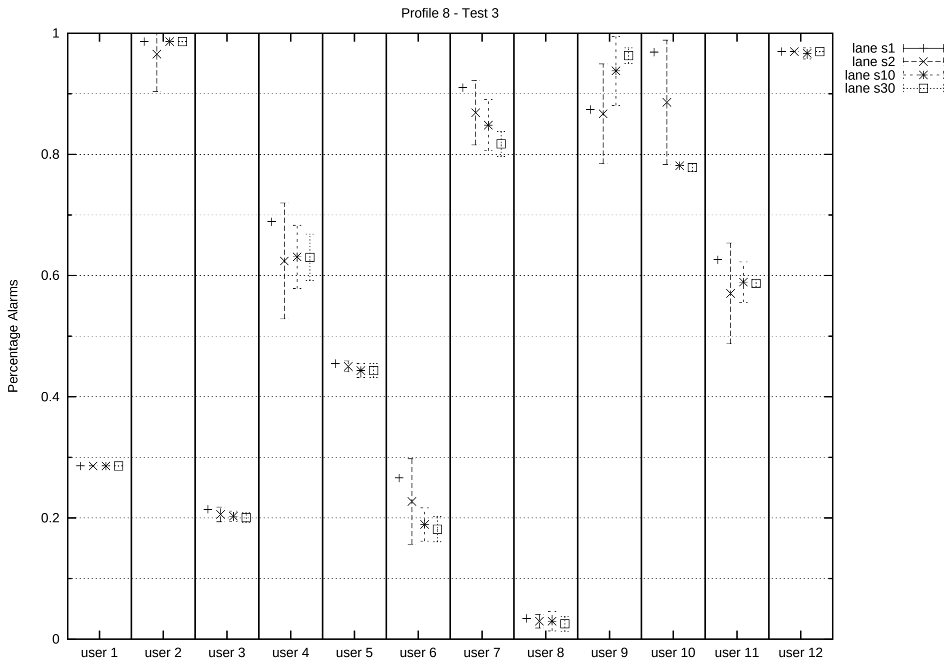
Figure 4.4: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER8's behaviors (Test Set 4 of 4).

4.6.2 Hidden Markov Model

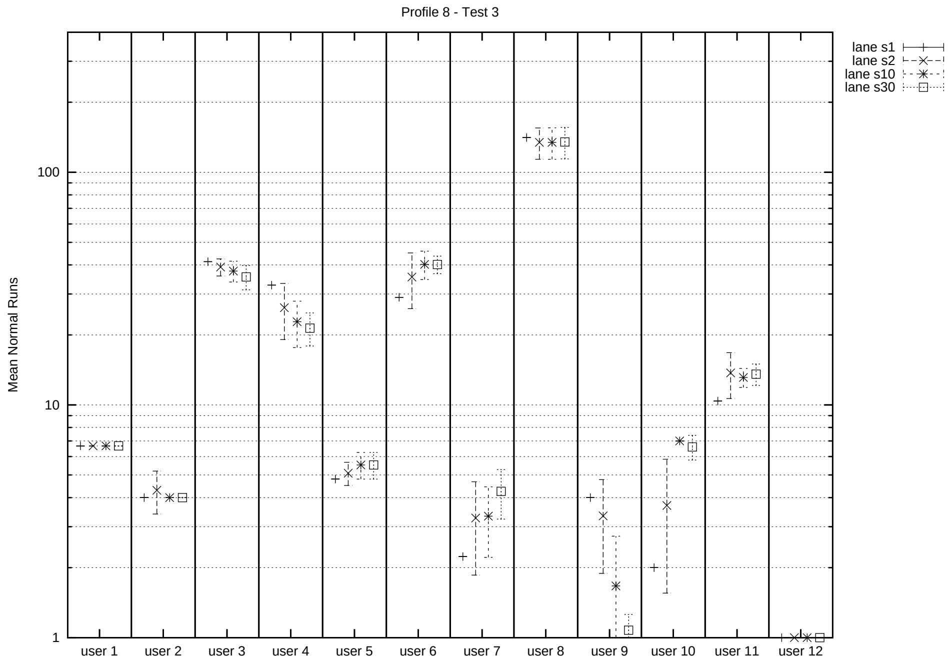
As stated before there are certain problems in the use of HMMs. First problem is the choice of the number of states. The second problem is the initial transition and observation probabilities. Hence we have tested these models with 1, 2, 10 and 30 states and 10 different random initialization of transition and observation probabilities.

We have trained our HMMs with observations from training data using Baum-Welch Algorithm. We have calculated subsequences probabilities of parameterization data using Forward-Backward Algorithm. We have chosen the threshold t_{min} from the lower 5% quantiles of the non-parametric distribution of subsequences probabilities of parameterization data.

The general performance of HMMs are reasonably good. When users with incomplex observations like USER8 and USER4 are considered, all of the HMMs behave similar to each other. The Figures 4.5 and 4.6 illustrate the situation for USER8. However when USER7 is considered, HMMs with smaller values of state number suffer from confusing USER7 with USER9 as illustrated in the Figures 4.7 and 4.8. Because USER7 has continued to use the web application while listening music which causes some complex observations. Hence the only value of state number that can correctly detect the difference between USER7 and USER9 is 30. On the whole as the state number increases, both false alarm rates and true detect performance decrease. This trend agrees with the Lane and Brodley's studies. As stated in [2], the 'improved true accept coupled with degraded true detect performance' can be viewed as an indication that bigger values of state number are subject to the 'everybody is the profiled user' difficulty with respect to smaller values of state number. General reasonable performance of HMMs with 30 states agrees with our expectation that HMMs with the number of hidden states about the number of nodes in the model of the application will be enough to represent the syntactic complexity of a web user. Because the view layer of the web application generally shapes these requests. Other charts for these models can be found in the Appendix B.

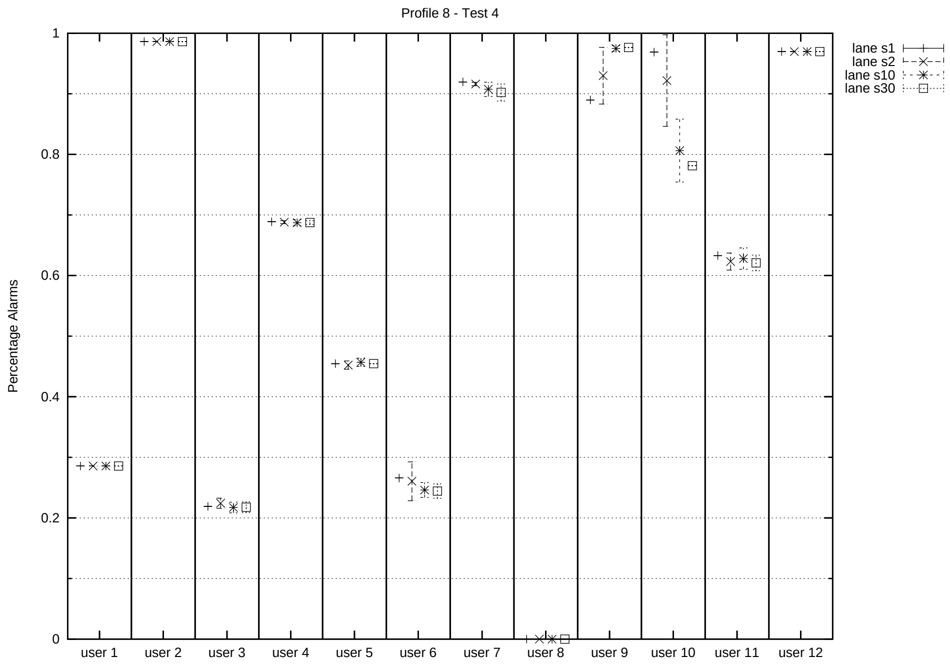


(a)

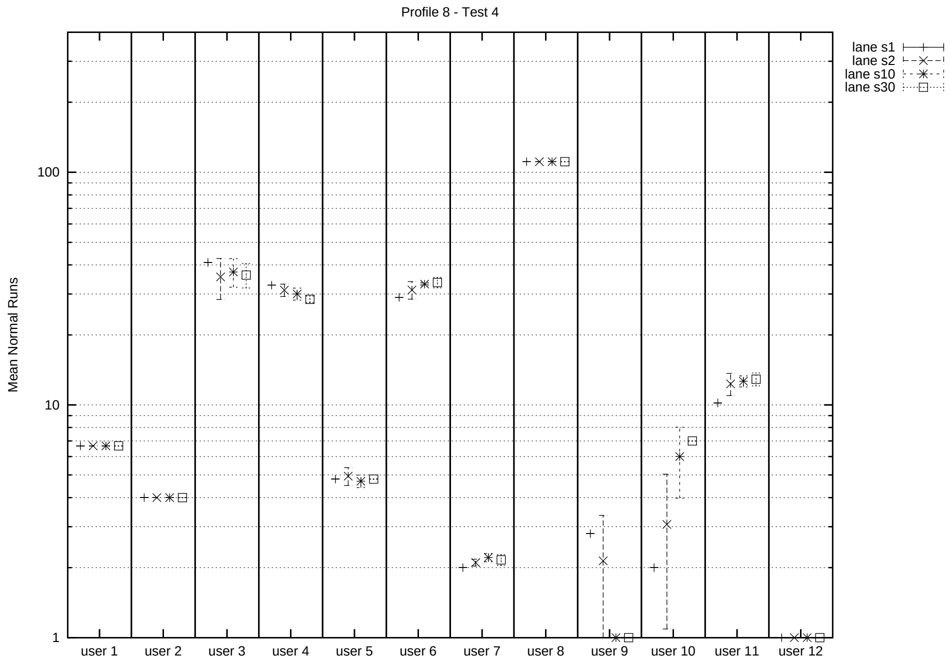


(b)

Figure 4.5: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER8's behaviors (Test Set 3 of 4).

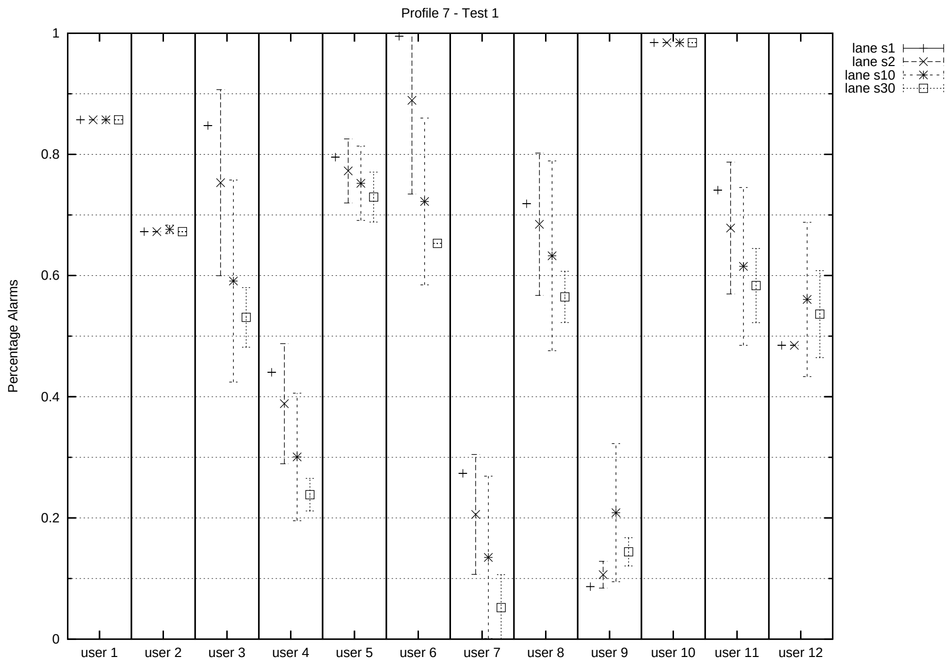


(a)

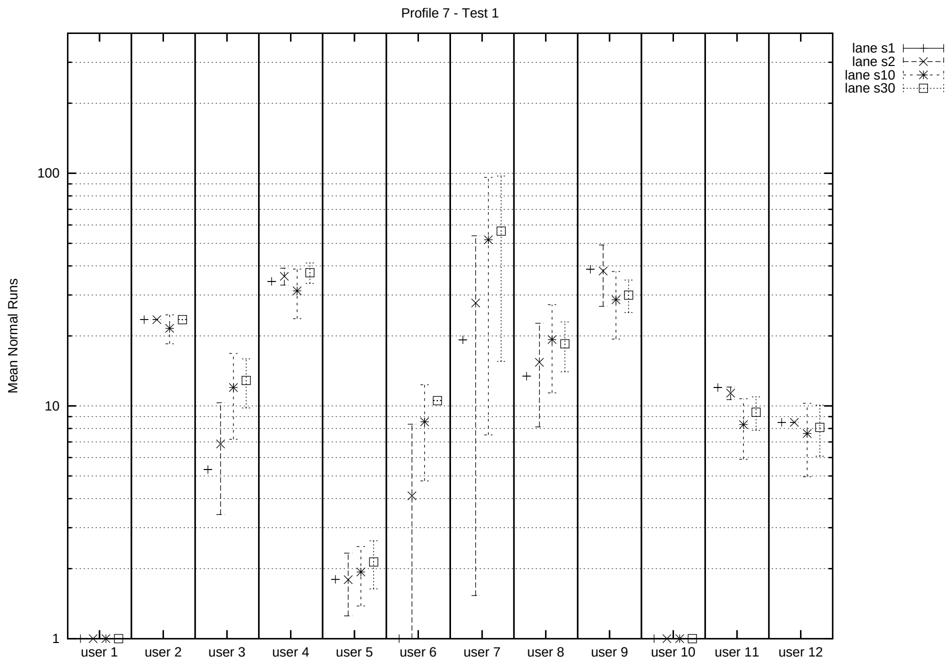


(b)

Figure 4.6: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER8's behaviors (Test Set 4 of 4).

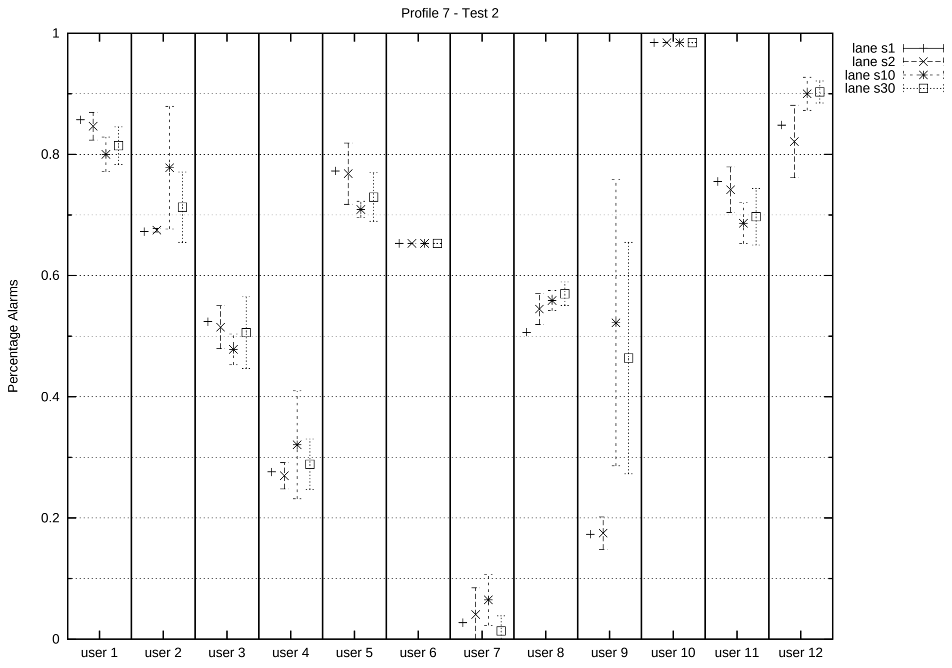


(a)

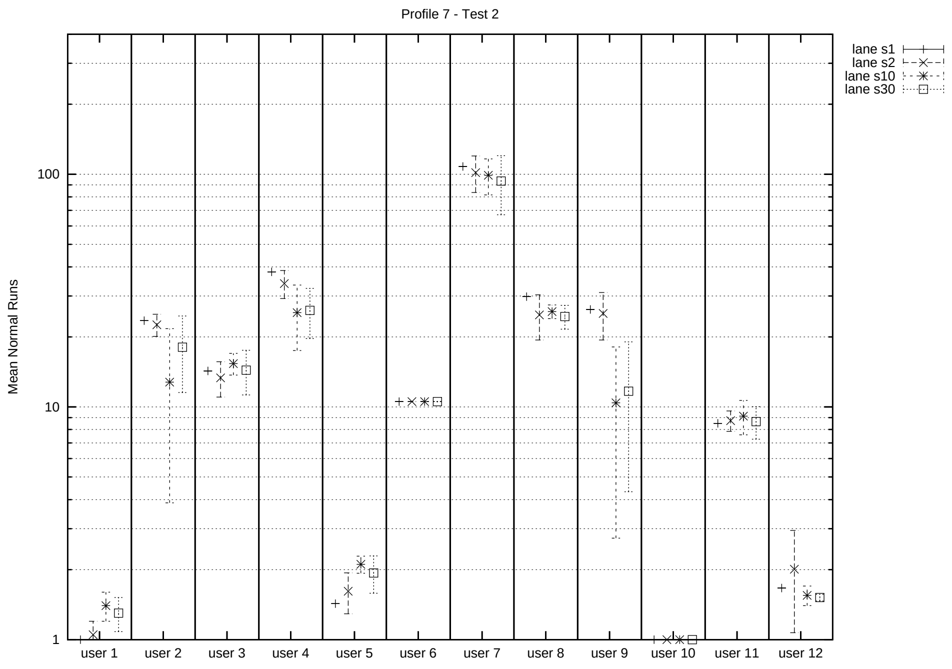


(b)

Figure 4.7: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER7's behaviors (Test Set 1 of 2).



(a)



(b)

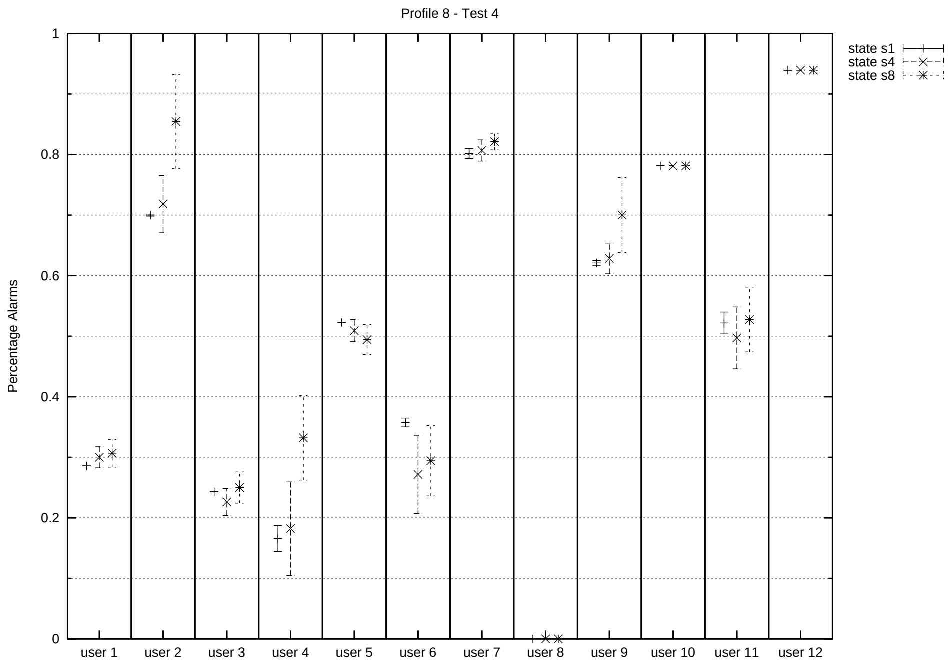
Figure 4.8: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER7's behaviors (Test Set 2 of 2).

4.6.3 Hidden Markov Model With State Transitions

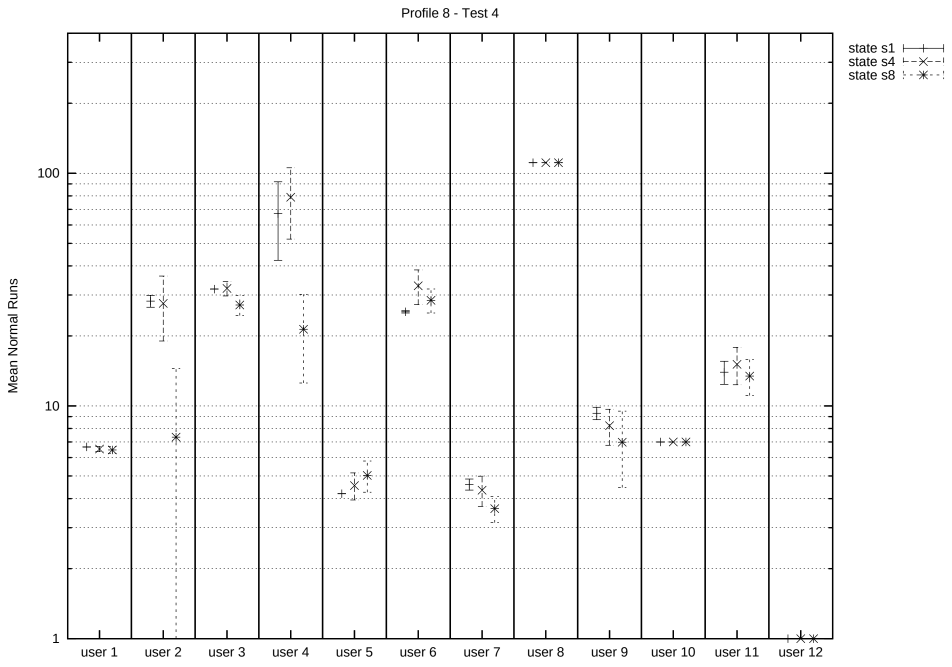
Using dynamic model extraction methods we have produced a finite prior model of the web application with 27 states. For each state we have recorded possible observations and using this information initialized the observation probabilities. As stated before, the extracted model can be partial or the user may not follow the view layer. Hence we have tested these models with 1, 4 and 8 additional nodes and 10 different random initialization of transition probabilities.

We have trained our HMMs with observations from training data using Baum-Welch Algorithm. We have calculated subsequences probabilities of parameterization data using Forward-Backward Algorithm. We have chosen the threshold t_{min} from the lower 5% quantiles of the non-parametric distribution of subsequences probabilities of parameterization data.

As the first interpretation of test results, the true accept and true detect performance increase, when the number of additional nodes increases. When the test number 4 of USER8 is considered as illustrated in the Figure 4.9, models with 1 and 4 additional nodes suffer from confusing USER8 with USER4. Only models with 8 additional nodes can truly detect the difference between these users. On the whole the performance of the models are satisfactory. However when users like USER7 which have continued to use the web application while listening music breaking the state diagram are considered, models with smaller number of additional nodes are not enough. This situation is illustrated in the Figure 4.10 and 4.11. The structure of this trend can be viewed as an indication that adding more additional nodes adds the capability of ordinary HMMs. However increasing the hidden state number augments the variance between test trials. This situation can be seen as an indication that depending on the random initialization of additional nodes these models can oscillate between ordinary HMMs and HMMs with prior estimation. Other charts for these models can be found in the Appendix B.

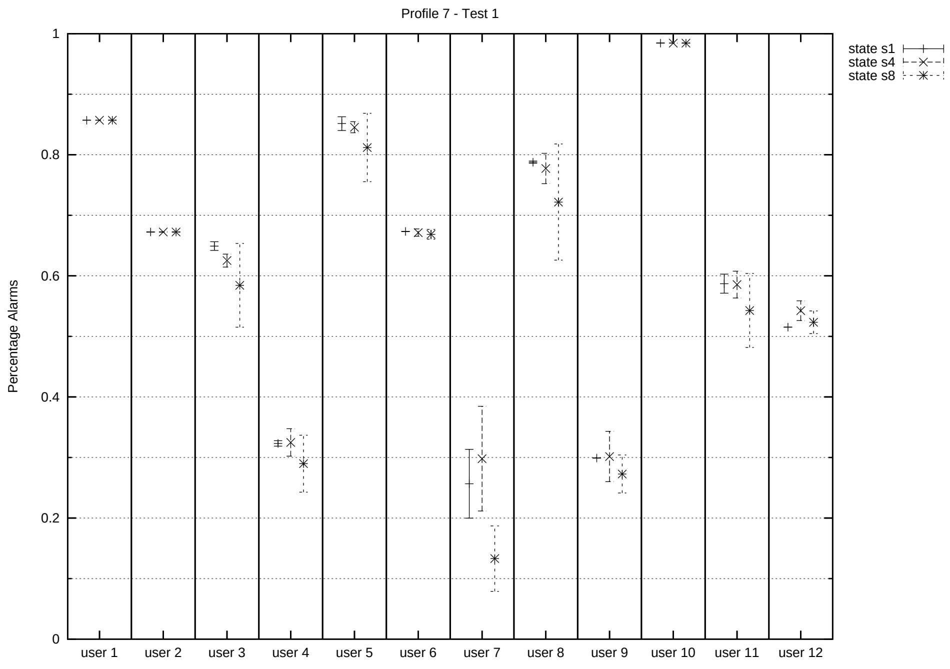


(a)

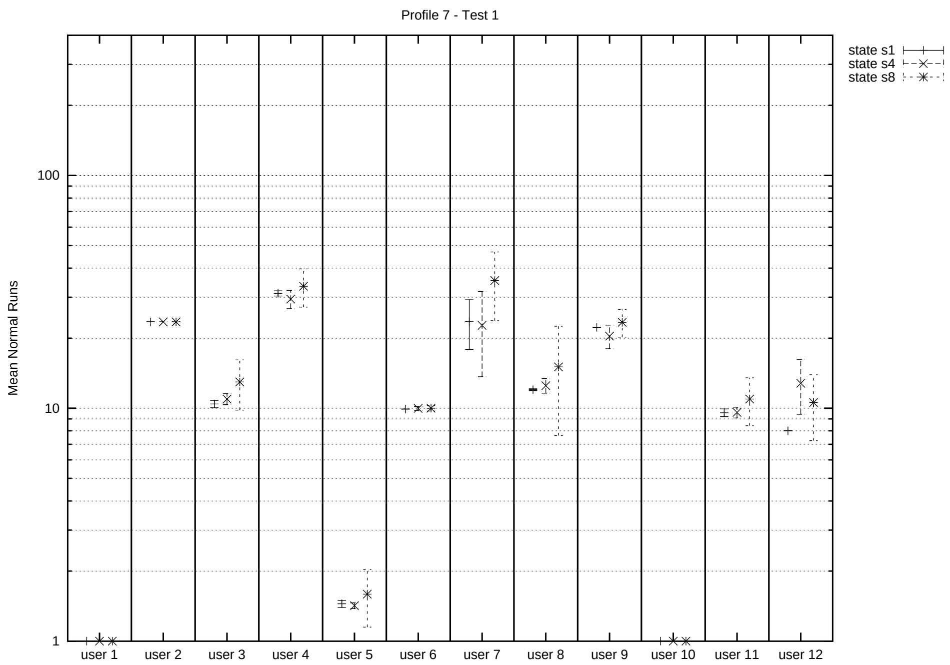


(b)

Figure 4.9: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER8's behaviors (Test Set 4 of 4).

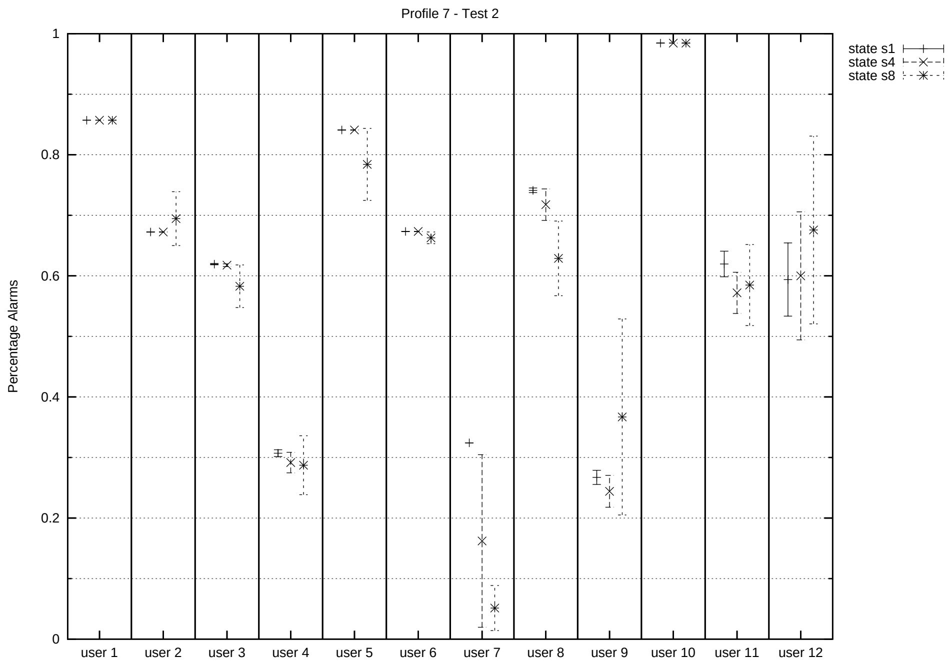


(a)

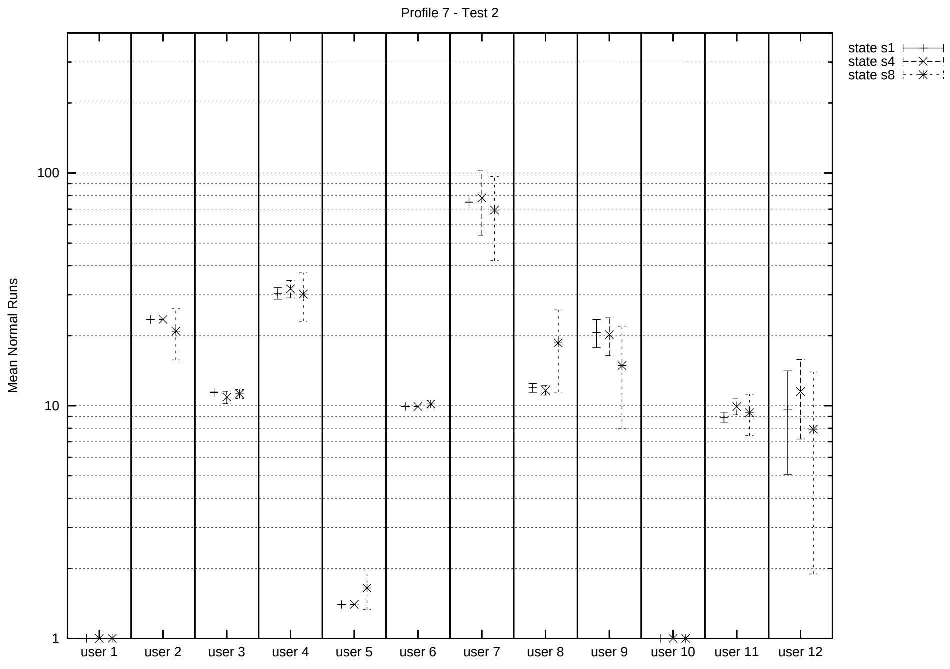


(b)

Figure 4.10: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER7's behaviors (Test Set 1 of 2).



(a)



(b)

Figure 4.11: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER7's behaviors (Test Set 2 of 2).

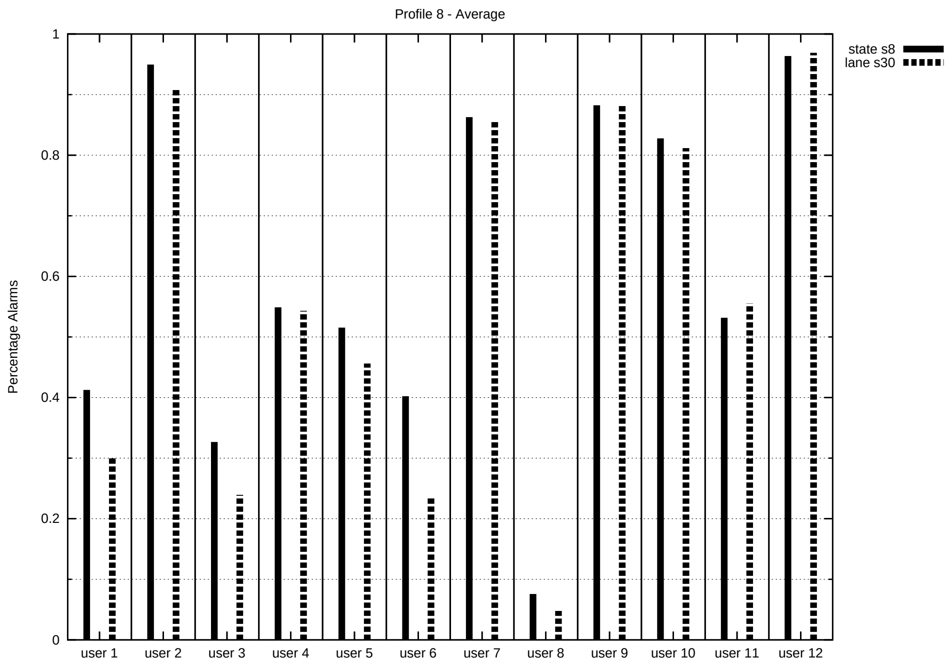
4.6.4 Comparison

Upto this section we have investigated each models in detail. In this section we will give brief comparison of two worthwile sensor models, ordinary HMMs and HMMs with prior estimation. In this comparison Markov chain models are eliminated because of the poor performance differentiating some users. The Figures 4.12, 4.13, and 4.14 illustrate average percentage alarms and mean normal runs for USER8, USER7, and USER4 respectively.

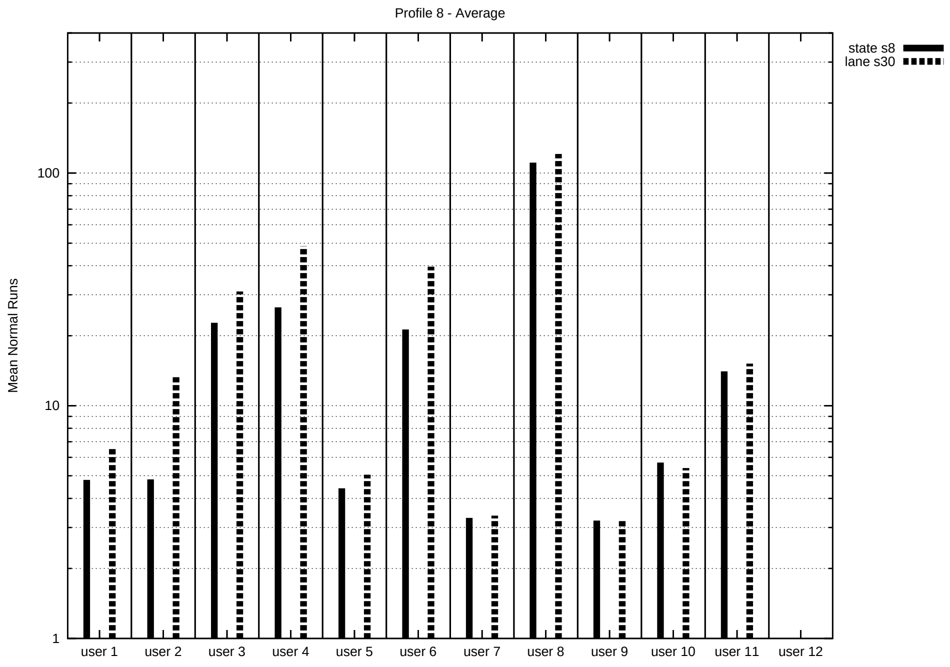
Although the acceptable false alarm rate parameter was tested as 0.05, some of the observed false alarm rates are greater than this. This is a result of the training and parameterization data failing to fully reflect the behavioral distribution present in the testing data. Because the user has changed behaviors or tasks over the interval between the generation of training and testing data, the profile does not include all of the behaviors present in the test data. This is actually caused by the batch-mode experimental setup used in this study.

If true detect capabilities are considered, HMMs with prior estimation have slightly better performance. Also mean normal run differences between the profiled user and the other users are slightly greater for this approach. However especially for USER7 this approach produces too many false alarm rates ($\sim$ 15%). As stated before USER7 have continued to use the web application while listening music which breaks the state diagram. If the file request actions while listening music are removed from the training set, false alarm rates decrease to a reasonable value. The Figure 4.15 shows the average percentage alarms and mean normal runs when the test data is filtered. For this data set, the initial window of user histories contain 500 observations of training group, 100 observations of parameter-selection group and 100 observations of testing group approximately and the window size of subsequences is 15. Since the testing data group is small, the mean normal run differences between the profiled user and the other users appear minor. However general performances of the sensors are acceptable. Again HMMs with prior estimation have slightly better performance as expected.

Finally, we will focus on the Figure 4.16 that illustrates the total results of all proposed approaches on all data sets. This figure can be interpreted as a different point of view of the problem. It illustrates the problem as clustering of the data into profiled user and other users. As it can be clearly seen from the figure, for Markov chain models it is not possible to differentiate profiled user from other users. If we consider percentage alarms, HMMs with prior estimation of state transitions clearly discriminate profiled user with an imaginary line. Again for the mean normal runs chart differentiation is clearly distinguishable excluding a data set that corresponds to USER7. As a conclusion this last figure evidently shows the performance difference between HMMs with prior estimation and ordinary HMMs highlighting the importance of prior estimation.

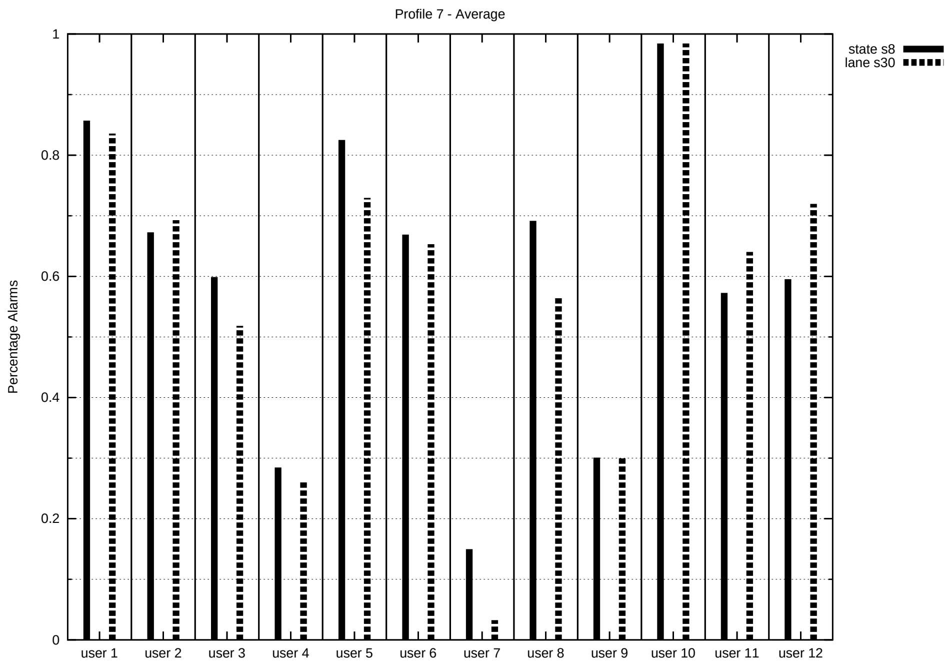


(a)

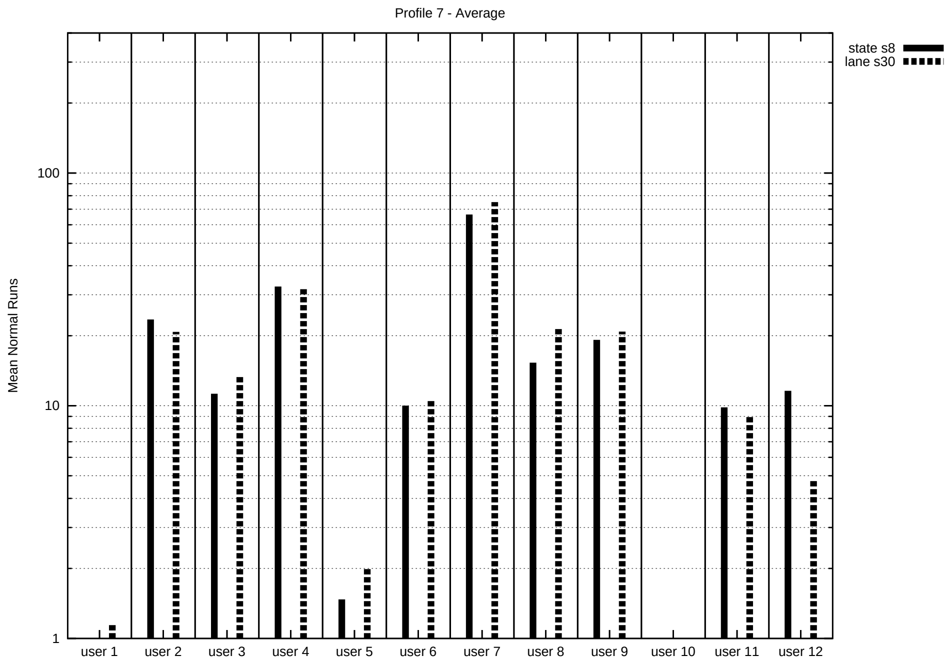


(b)

Figure 4.12: Average percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and normal HMMs of USER8's behaviors.

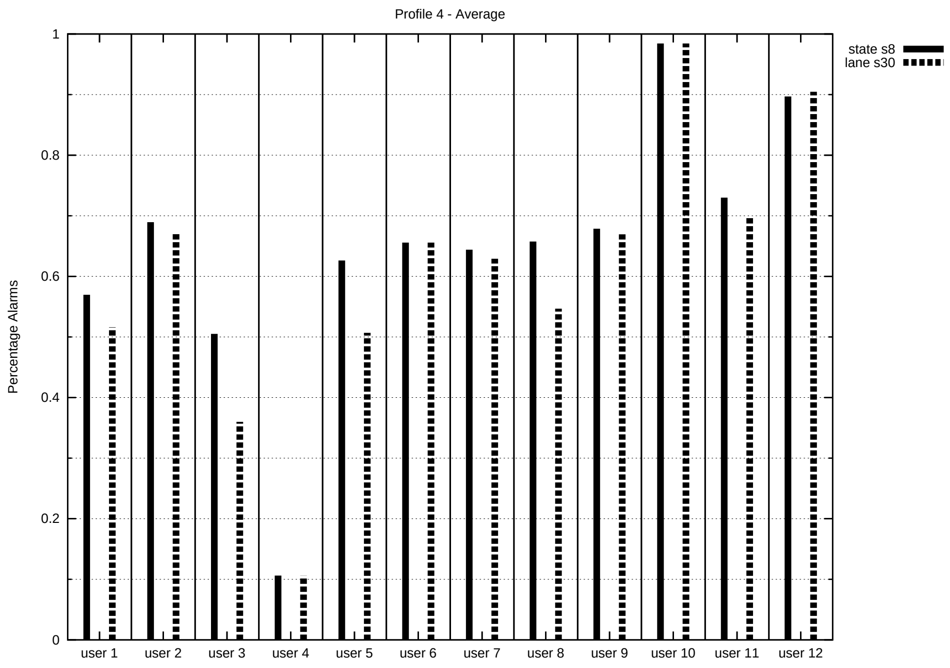


(a)

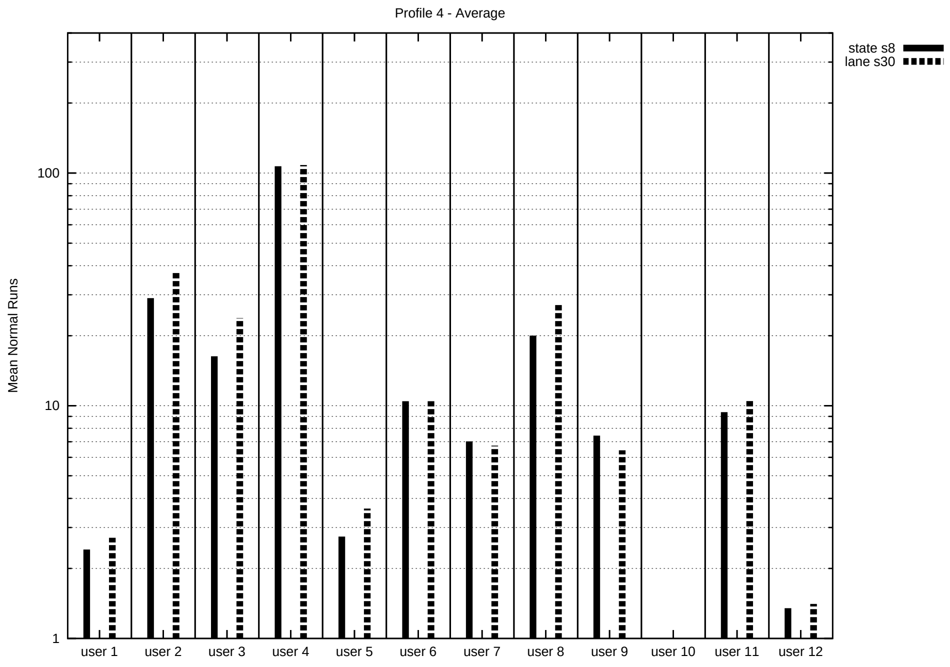


(b)

Figure 4.13: Average percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and normal HMMs of USER7's behaviors.

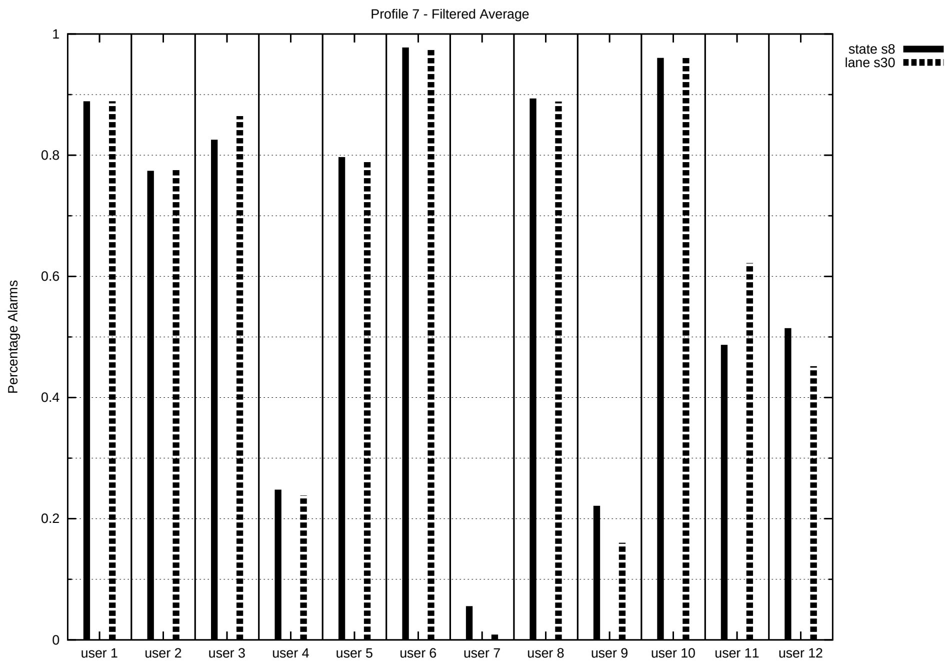


(a)

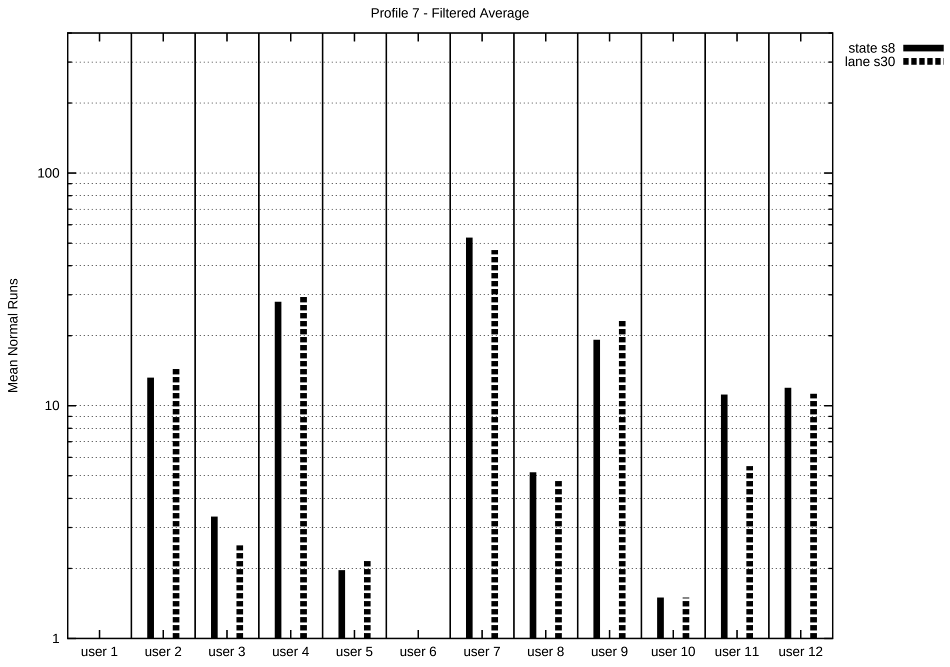


(b)

Figure 4.14: Average percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and normal HMMs of USER4's behaviors.

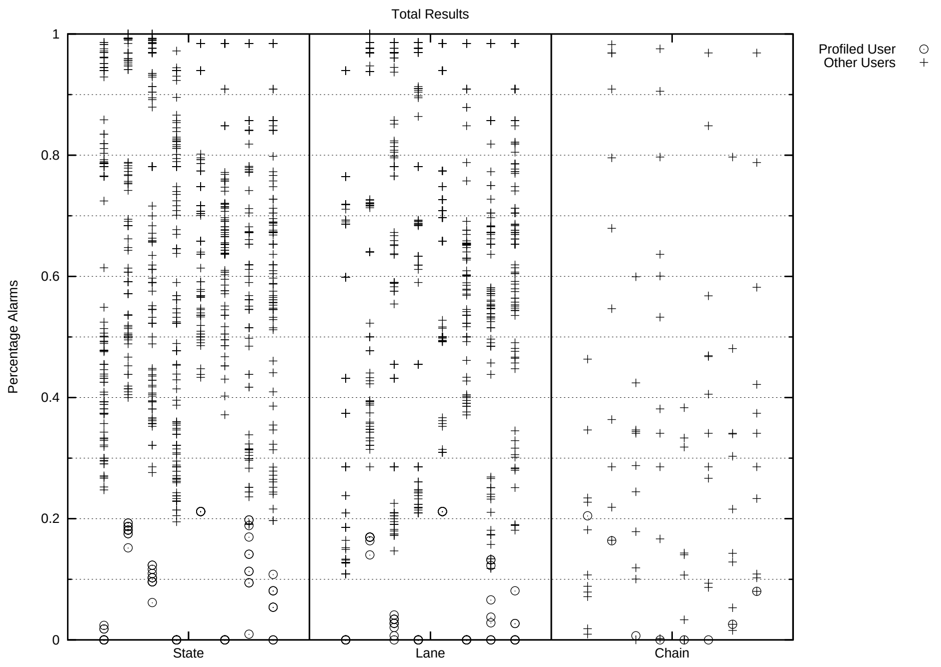


(a)

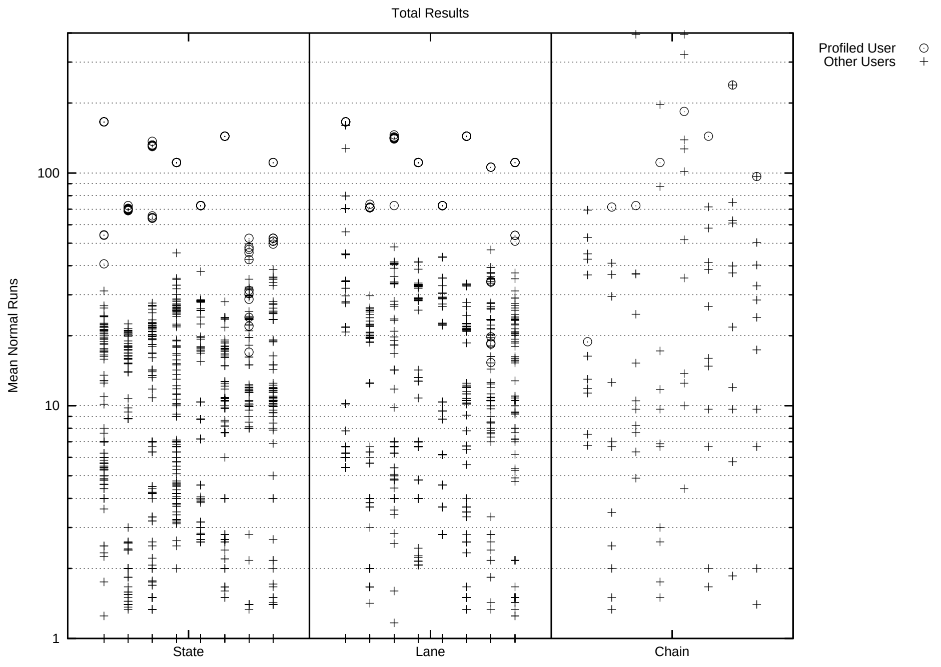


(b)

Figure 4.15: Filtered average percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and normal HMMs of USER7's behaviors.



(a)



(b)

Figure 4.16: Total percentage alarms (a) and mean normal runs (b) for all proposed approaches.

Chapter 5

Conclusion

Briefly, in this study we have aimed to develop a model or profile of normal working state of web application user and to detect anamolous conditions as deviations from the expected behavior patterns. For this purpose we have collected audit data from a music sharing application. We have preprocessed and transformed the data labeling as observations and breaking into sessions. We have trained variations of Markov chains and hidden Markov models and tested these models according to the percentage alarms and the mean normal runs.

As a conclusion, we have demonstrated the use of Markov chains and hidden Markov models for user profiling in the domain of anomaly detection in web applications. The key results of the empirical investigation are:

- Markov chains are insufficient to model user behaviors for anomaly detection. Because these models can only represent a single context for any given symbol.
- HMMs can be used to identify users by their usage patterns.
- Larger HMMs were found to be more effective at identifying the valid user, while smaller HMMs were generally better at discerning impostors.
- Although the optimal number of hidden states is user dependent and appears to reflect a measure of syntactic complexity present in the data, in a web application the number of states in the structural model of the application can be good estimate for the number of hidden states in HMMs.
- Explicit state model of the web application can be used as prior estimation to initialize HMMs.

Moreover our empirical investigation showed us that in a web application users demands from an application do not change rapidly and frequently, and also their current actions and responses reflect the unique demands of the current situation, but they are also shaped out of previously learned behaviors and skills. Nevertheless the training and parameterization data have failed to fully reflect the behavioral distribution present in the testing data. This is an evidence for the need of investigating extensions of HMM anomaly detection sensors to online mode.

To use our proposed approaches in a real application, there are two important points that should be taken into considerations:

- All possible action requests should be identified and labeled according to reflect user preferences and habits. Some high level observations should be divided into sub-categories to reflect additional user differentiation. As stated before for example, in a financial web application one can label a payment action with amount scale to classify users according to the scale of their payment amount.
- The structural model of the application covering most of the internal states should be extracted. This will decrease the need of additional hidden states in HMMs. Also prior estimation of initials in HMMs will affect the quality of sensors.

5.1 Future Work

In this section we will focus on how can we extend these approaches to improve anomaly detection performance in web applications and discuss possible future works in this domain.

As stated in the previous section being not able to fully reflect the behavioral distribution present in the testing data using the training and parameterization data is a direct evidence for the need of investigating extensions of HMM anomaly detection sensors to online mode [33]. Also we believe that sensors should not rely on least recently observed actions, besides they should not forget the past. Because in a web application sessions observed in the near future are likely to observed similarly again. Hence it is expected that performance improvements will be realized by using a weighted learning strategy.

The structure of all web applications do not need to have enough number of depths and branching factors to be able to produce distinguishable models of usage patterns. In such applications these models will not be enough to differentiate all users. Hence combining other features with action request behaviors can help better identification of users. For example, in a web application we can consider the time difference between observations and relate them with the speed of filling forms.

Moreover modeling interaction with user interface is an open question. Client side javascript can collect user interface interaction information about the web user. For example, the use of keyboard or mouse while filling form elements like comboboxes and checkboxes can be useful feature in this domain. Also mouse usage patterns are not easily reproducible, hence modeling them can be valuable for user identification and anomaly detection.

References

[1] Lane T. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, Department of Electrical and Computer Engineering, 2000.

[2] Lane T. Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users*, pages 35–44, 1999.

[3] Lane T. and Brodley C. E. An empirical study of two approaches to sequence learning for anomaly detection. *Machine Learning*, 51:73–107, 2003.

[4] Denning D. E. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[5] Kumar S. *Classification and Detection of Computer Intrusions*. PhD thesis, Department of Computer Sciences, Purdue University, 1995.

[6] Axelsson S. Research in intrusion-detection systems: A survey. Research survey, The Swedish National Board for Industrial and Technical Development, 1999.

[7] Axelsson S. Intrusion detection systems: A survey and taxonomy. Research survey, The Swedish National Board for Industrial and Technical Development, 2000.

[8] Lee W., Stolfo S., and Mok K. W. Mining audit data to build intrusion detection models. In *Knowledge Discovery and Data Mining*, pages 66–72, 1998.

[9] Kang D., Fuller D., and Honavar V. Learning classifiers for misuse detection using a bag of system calls representation. Technical report, Computer Science Department, Iowa State University, 2005.

[10] Weiss S. M. Rule based machine learning methods for functional predicting. *Artifical Intelligence Research*, pages 383–403, 1995.

[11] Chan P., Mahoney M. V., and Arshad M. H. A machine learning approach to anomaly detection. Technical report, Department of Computer Sciences, Florida Institute of Technology, 2003.

[12] Ludovic Me. Genetic algorithms, an alternative tool for security audit trails analysis. *1st Workshop on Recent Advances in Intrusion Detection*, 1998.

[13] Lane T. and Brodley C. E. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[14] Forrest S., Hofmeyr S. A., Somayaji A., and Longstaff T. A. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128.

[15] Ghosh A. K., Schwartzbard A., and Schatz M. Using program behavior profiles for intrusion detection. *Proceedings of 1st Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, 1999.

[16] Cho S. and Park H. Efficient anomaly detection by modeling privilege flows using hidden markov model. *Computers and Security*, 22(1):45–55, 2003.

[17] Cho S. and Han S. Two sophisticated techniques to improve hmm-based intrusion detection systems. *Lecture Notes On Computer Science*, 2820:207–219, 2003.

[18] Hoang X. D., Hu J., and Bertok P. A multi-layer model for anomaly intrusion detection using program sequences of system calls. *The 11th IEEE International Conference on Networks*, pages 531–536, 2003.

[19] Pettit S. Anatomy of a web application: Security considerations. White paper, Sanctum Inc., 2001.

[20] D'Ambrosio B. Modeling web visitor behavior: Real-time dynamic response for 21st century e-commerce using relational bayesian models. White paper, CleverSet Inc., 2004.

[21] Tonella P. and Ricca F. Dynamic model extraction and statistical analysis of web applications. In *Proceedings of the Fourth International Workshop on Web Site Evolution*, 2002.

[22] Hafri Y., Djeraba C., Stanchev P., and Bachimont B. A markovian approach for web user profiling and clustering. *Lecture Notes in Artificial Intelligence*, 2637:191–202, 2003.

[23] Cadez I. V., Heckermen D., Meek C., Smyth P., and White S. Model-based clustering and visualization of navigation patterns on a web site. *Journal of Data Mining and Knowledge Discovery*, 7(4):399–424, 2003.

[24] Manavoglu E., Pavlov D., and Giles C. L. Probabilistic user behavior models. In *Proceedings of the Third International Conference on Data Mining*, 2003.

[25] Kruegel C. and Vigna G. Anomaly detection of web based attacks. In *Proceedings of 10th ACM Conference on Computer and Communications Security*, 2003.

[26] Chen Y., Åström M., and Wang L. Session comparison measurement and learning in masquerading detection. Technical report, Department of Business Administration and Social Sciences, Division of Systems Sciences, Sweden, 2004.

[27] Huang Y., Huang S., Lin T., and Tsai C. Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web*, pages 148–159, 2003.

[28] Kıcıman E. and Fox A. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, 2005.

[29] Rabiner L. R. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of The IEEE*, volume 77, pages 257–286, 1989.

[30] Juang B. and Rabiner L. R. A probabilistic distance measure for hmms. *AT&T Technical Journal*, 64(2):391–408, 1985.

[31] Rabiner L. R. and Juang B. An introduction to hidden markov models. *IEEE Acoustics, Speech, and Signal Processing Mag.*, 3(1):4–16, 1986.

[32] Juang B. and Rabiner L. R. The segmental k-means algorithm for estimating the parameters of hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1639–1641, 1990.

[33] Digalakis V. V. Online adaptation of hidden markov models using incremental estimation algorithms. *IEEE Transactions on Speech and Audio Processing*, 7(3):253–261, 1999.

[34] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2001.

[35] Dugad R. and Desai U. B. A tutorial on hidden markov models. Technical report, SPANN Laboratory Indian Institute of Technology, Bombay, 2001.

Appendix A

An Overview of Hidden Markov Models

In section 2.4, we have seen the notation and three main problems of HMMs. We present brief descriptions of the commonly employed techniques for the solutions of these problems here.

Calculation of Observation Probabilities: A most straightforward way to determine $P(O|\lambda)$ is to find $P(O|Q,\lambda)$ for a fixed state sequence $Q = q_1, q_2, \ldots, q_T$ then multiply it by $P(Q|\lambda)$ and then sum up over all possible Q's. We have:

$$P(O|Q,\lambda) = b_{q_1}(O_1)b_{q_2}(O_2)\cdots b_{q_T}(O_T) \tag{A.1}$$

$$P(Q|\lambda) = \pi_{q_1}a_{q_1q_2}a_{q_2q_3}\cdots a_{q_{T-1}q_T} \tag{A.2}$$

Hence we have:

$$P(O|\lambda) = \sum_Q P(O|Q,\lambda)P(Q,\lambda) \tag{A.3}$$

$$= \sum_Q \pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)\cdots a_{q_{T-1}q_T}b_{q_T}(O_T) \tag{A.4}$$

where $Q = q_1, q_2, \ldots, q_T$.

This equation involves N^T distinct possible state sequences I, which yields order of $2TN^T$ multiplications which is not feasible. This problem is solved with a dynamic programming algorithm called *Forward-Backward Algorithm* that employs the Markov property (finite memory) to avoid computation of all possible state sequences.

Consider the forward variable $\alpha_t(i)$ defined as:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{A.5}$$

i.e. the probability of the partial observation sequence up to time t and the state S_i at time t, given the model λ. We can solve for $\alpha_t(i)$ inductively, as follows:

1. Initialization:
$$\alpha_0(i) = \pi_i b_i(O_1) \tag{A.6}$$
where $1 \leqslant i \leqslant N$.

2. Induction:
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \tag{A.7}$$
where $1 \leqslant t \leqslant T-1$ and $1 \leqslant j \leqslant N$.

3. Termination:
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{A.8}$$

In the induction step we want to compute the probability of partial observation sequence up to time $t+1$ and state S_j at time $t+1$; state S_j can be reached (with probability a_{ij}) independently from any of the N states at time t. The summation in the induction step refers to this fact. Also the summand gives observation sequence up to time t. In the termination step we just sum up all possible (independent) ways of realizing the given observation sequence. The total number of multiplications involved in this algorithm is $N + N(N+1)(T-1)$ i.e. the order of N^2T.

In a similar manner we may define a backward variable $\beta_t(i)$ as:
$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T | q_t = S_i, \lambda) \tag{A.9}$$
i.e. the probability of the observation sequence from $t+1$ to T given the state S_i at time t and the model λ. Again we can solve for $\beta_t(i)$ inductively as follows:

1. Initialization:
$$\beta_T(i) = 1 \tag{A.10}$$
where $1 \leqslant i \leqslant N$.

2. Induction:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{A.11}$$
where $1 \leqslant t \leqslant T-1$ and $1 \leqslant i \leqslant N$.

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1)\beta_1(i) \qquad \text{(A.12)}$$

The computation of $P(O|\lambda)$ using $\beta_t(i)$ also involves the order of N^2T calculations.

Determination of Optimal State Sequences: There are several possible ways of finding the optimal state sequence associated with the given observation sequence. To choose the states q_t which are individually most likely, we define the variable:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \qquad \text{(A.13)}$$

i.e. the probability of being in state S_i at time t, given the observation sequence O and the model λ. This variable can be simply expressed by forward-backward variables:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \qquad \text{(A.14)}$$

since $\alpha_t(i)$ accounts for the partial observation sequence $O_1, O_2, \ldots, O_t$ and state S_i at t, while $\beta_t(i)$ accounts for the remainder of the observation sequence $O_{t+1}, O_{t+2}, \ldots, O_T$ given state S_i at t. Using $\gamma_t(i)$, we can solve for the individually most likely state q_t at time t, as:

$$q_t = S_{argmax_{1\leqslant i\leqslant N}[\gamma_t(i)]} \qquad \text{(A.15)}$$

where $1 \leqslant t \leqslant T$. Although this maximizes the expected number of correct states by choosing the most likely state for each t, there could be some problems with the resulting state sequence. For example, when the HMM has state transitions which have zero probability ($a_{ij} = 0$ for some i and j, the optimal state sequence may not even be a valid sequence.

Hence we have to modify the optimality criterion. The most widely criterion is to find the single best state sequence, i.e. to maximize $P(Q|O, \lambda)$ which is equivalent to maximizing $P(Q, O|\lambda)$. A formal technique for finding this single best state sequence exists, based on dynamic programming methods, and is called the *Viterbi Algorithm.* We need to define a variable:

$$\delta_t(i) = max_{q_1,q_2,\ldots,q_{i-1}} P(q_1, q_2, \ldots, q_{t-1}, q_t = S_i, O_1, O_2, \ldots, O_t|\lambda) \qquad \text{(A.16)}$$

i.e. the best score along a single path, at time t, which accounts for the first t observations and ends in state S_i. By induction we have:

$$\delta_{t+1}(j) = max_i \left[\delta_t(i)a_{ij}\right] b_j(O_{t+1}). \qquad \text{(A.17)}$$

To actually retrieve the state sequence, we need to keep track of the argument which maximized above equation, for each t and j.We do this via the array $\psi_t(j)$. The complete procedure for finding the best state sequence can now be stated as follows:

1. Initialization:

$$\delta_1(i) = \pi_i b_i(O_1) \tag{A.18}$$

$$\psi_1(i) = 0 \tag{A.19}$$

   where $1 \leqslant i \leqslant N$.

2. Recursion:

$$\delta_t(j) = max_{1 \leqslant i \leqslant N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_t) \tag{A.20}$$

$$\psi_t(j) = argmax_{1 \leqslant i \leqslant N} \left[\delta_{t-1}(i) a_{ij}\right] \tag{A.21}$$

   where $2 \leqslant t \leqslant T$ and $1 \leqslant j \leqslant N$.

3. Termination:

$$p^* = max_{1 \leqslant i \leqslant N} \left[\gamma_T(i)\right] \tag{A.22}$$

$$q_T^* = argmax_{1 \leqslant i \leqslant N} \left[\gamma_T(i)\right] \tag{A.23}$$

4. Path backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \tag{A.24}$$

   where $t = T-1, T-2, \ldots, 1$.

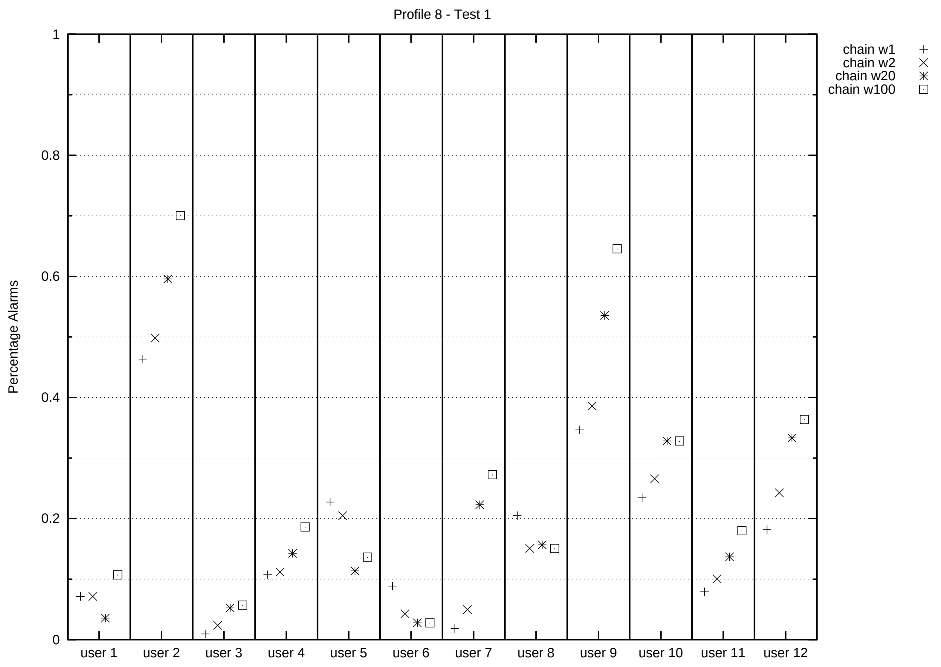
A little reflection over the above steps will show that computationally the algorithm is similar to the forward-backward procedure except for the comparisons involved for finding the maximum value. Hence its complexity is also of the order N^2T.

Training the Model: The third, and by far the most difficult, problem of HMMs is to determine a method to adjust the model parameters (A, B, π) to maximize the probability of the observation sequence given the model. There is no known way to analytically solve for the model which maximizes the probability of the observation sequence. In fact, given any finite observation sequence as training data there is no optimal way of estimating the model parameters. There are two methods for solving this problem with some limitations, one chooses $\lambda = (A, B, \pi)$ such that $P(O|\lambda)$ is locally maximized called Baum-Welch Algorithm and the other chooses $\lambda = (A, B, \pi)$ such that $P(O, Q|\lambda)$ is maximized called Segmental K-Means Algorithm. These methods are deeply explained in studies [31], [32], [34], and [35].

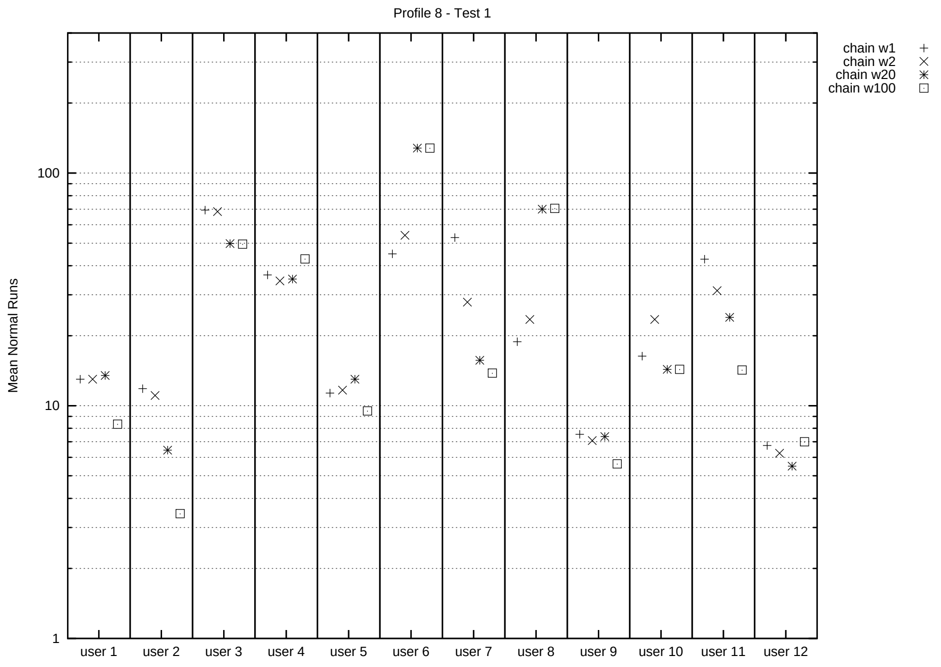
Appendix B

Other Charts for Test Results

This section includes other charts for test results which are not directly referenced in this study. All of these charts include both percentage and mean normal runs. These charts are organized in the order of approaches presented in the Section 3.3. For each approach charts are presented in the order of decreasing profiled user number and increasing test set number.

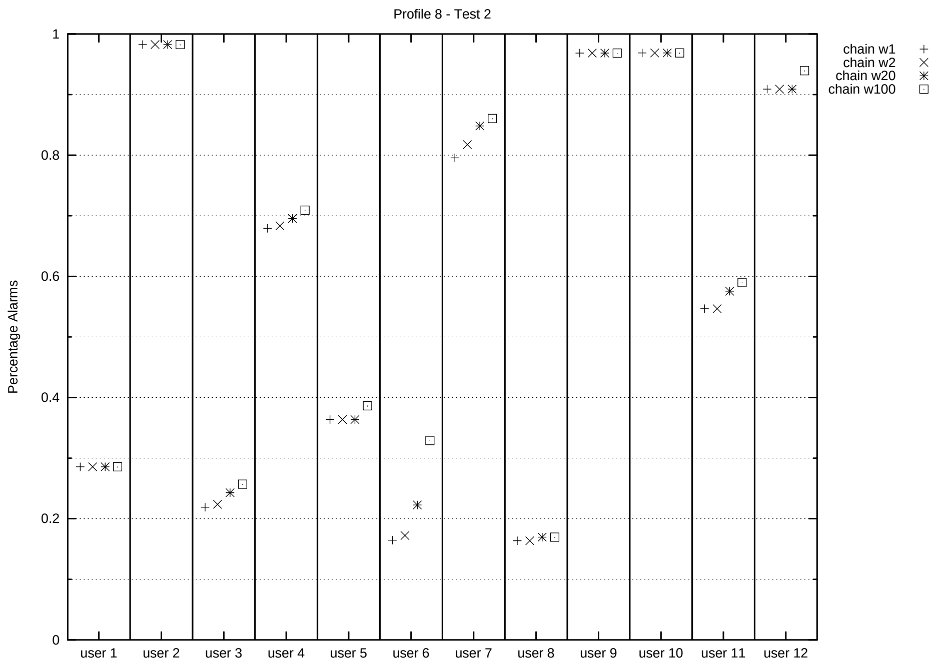


(a)

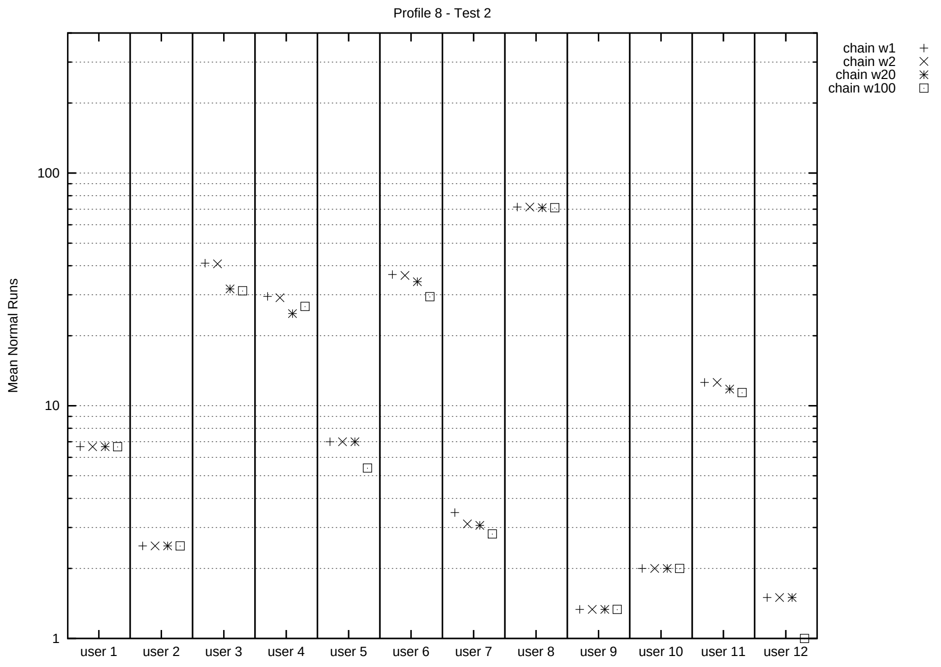


(b)

Figure B.1: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER8's behaviors (Test Set 1 of 4).

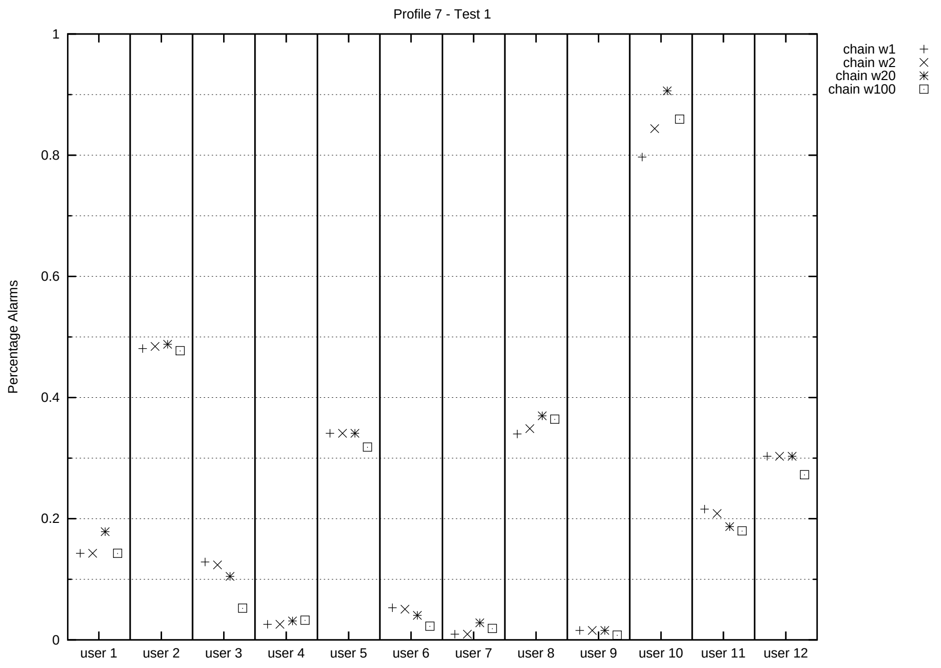


(a)

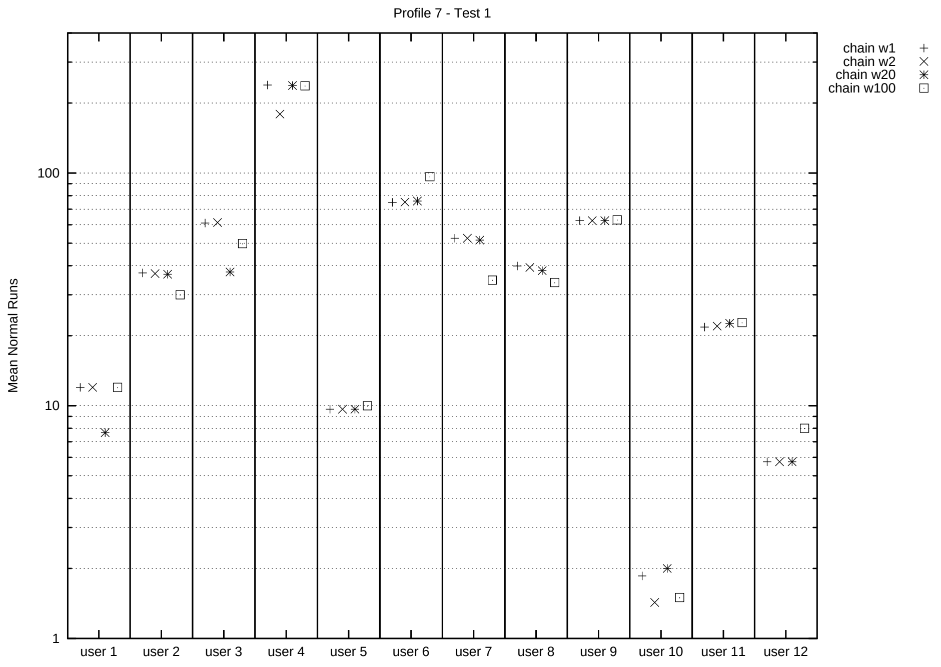


(b)

Figure B.2: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER8's behaviors (Test Set 2 of 4).

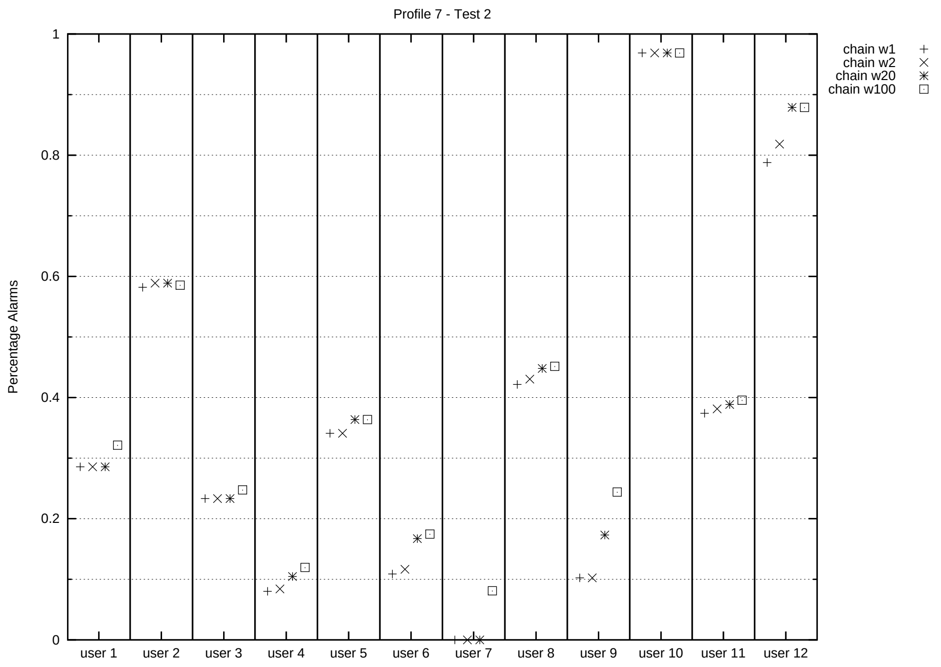


(a)

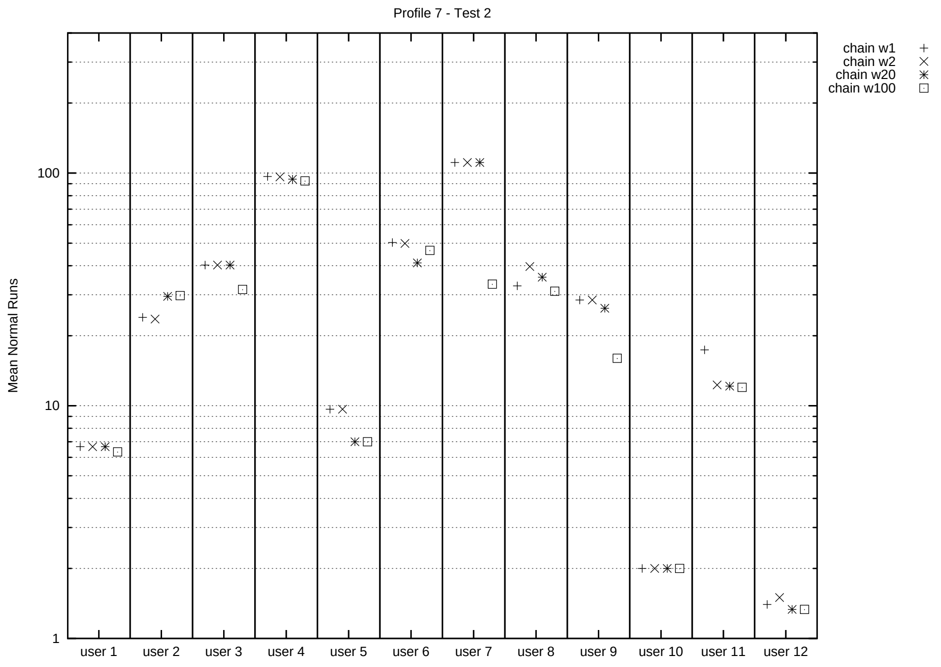


(b)

Figure B.3: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER7's behaviors (Test Set 1 of 2).

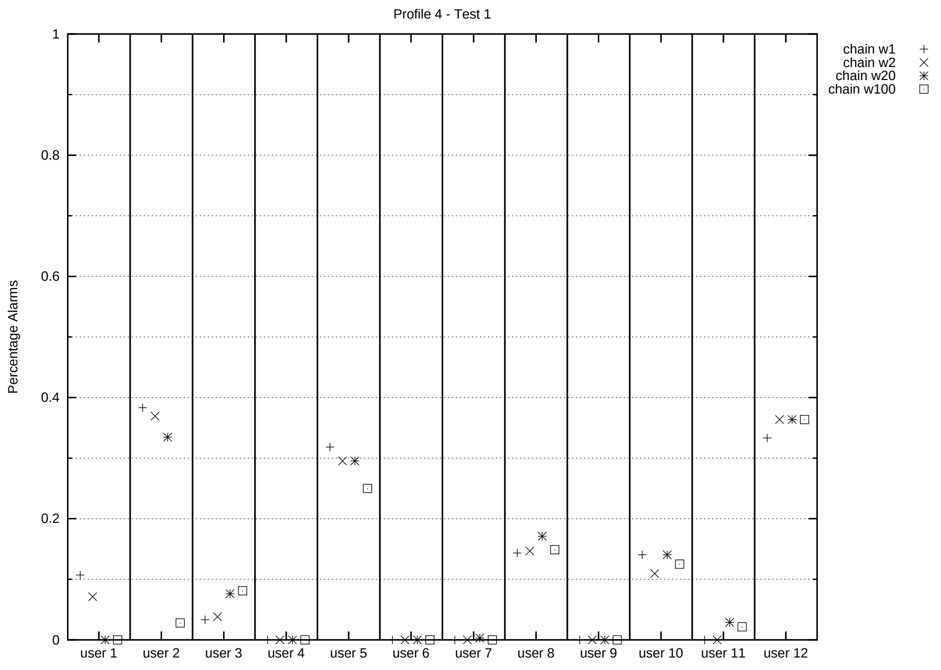


(a)

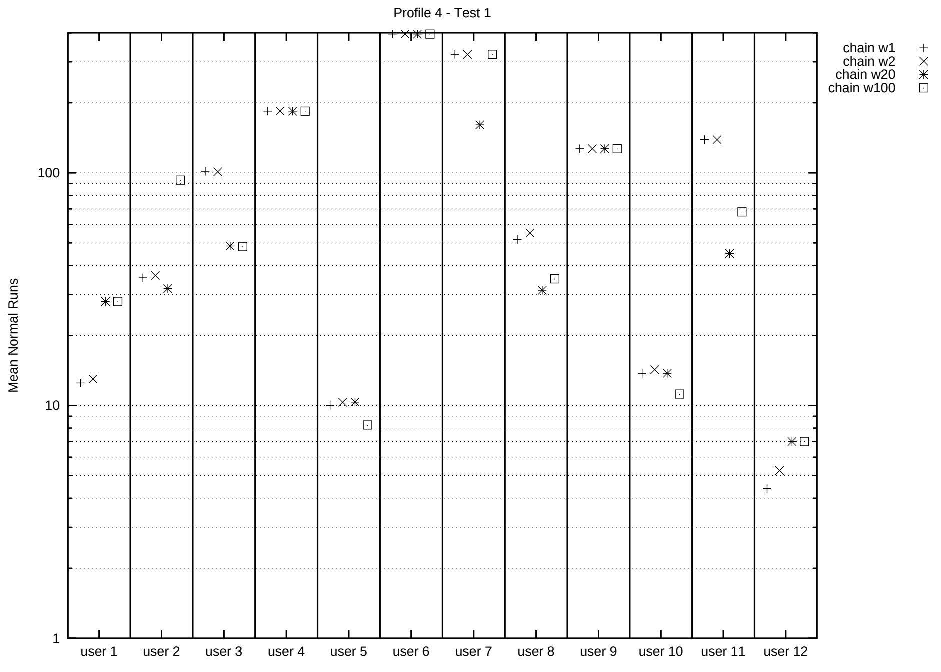


(b)

Figure B.4: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER7's behaviors (Test Set 2 of 2).

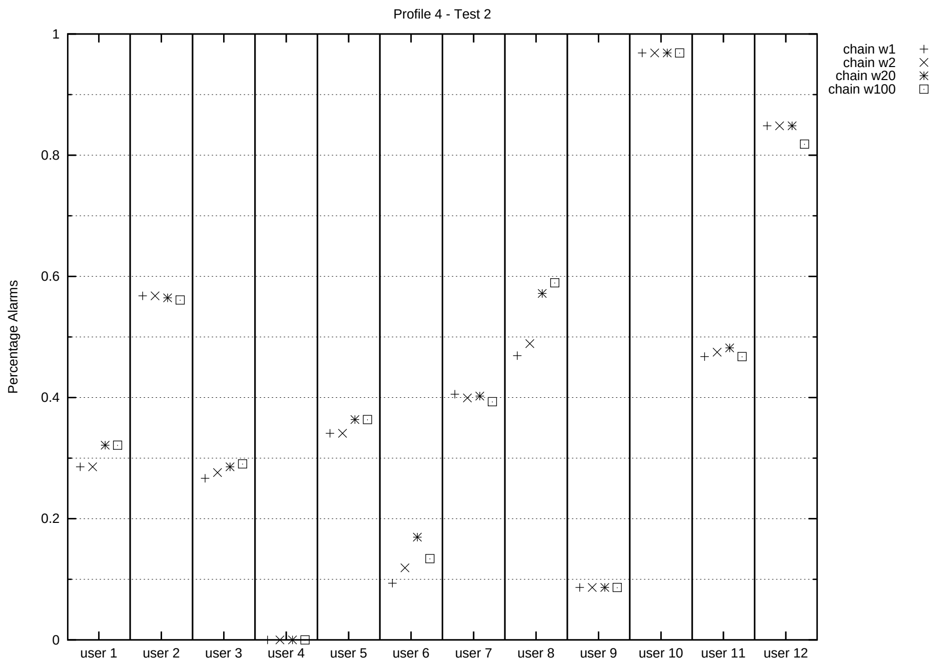


(a)

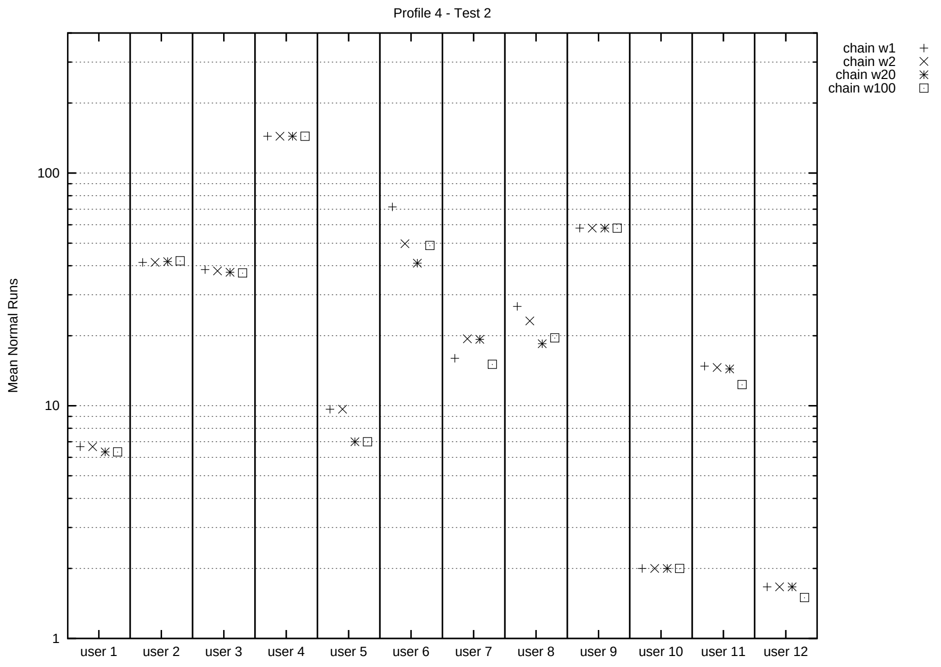


(b)

Figure B.5: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER4's behaviors (Test Set 1 of 2).

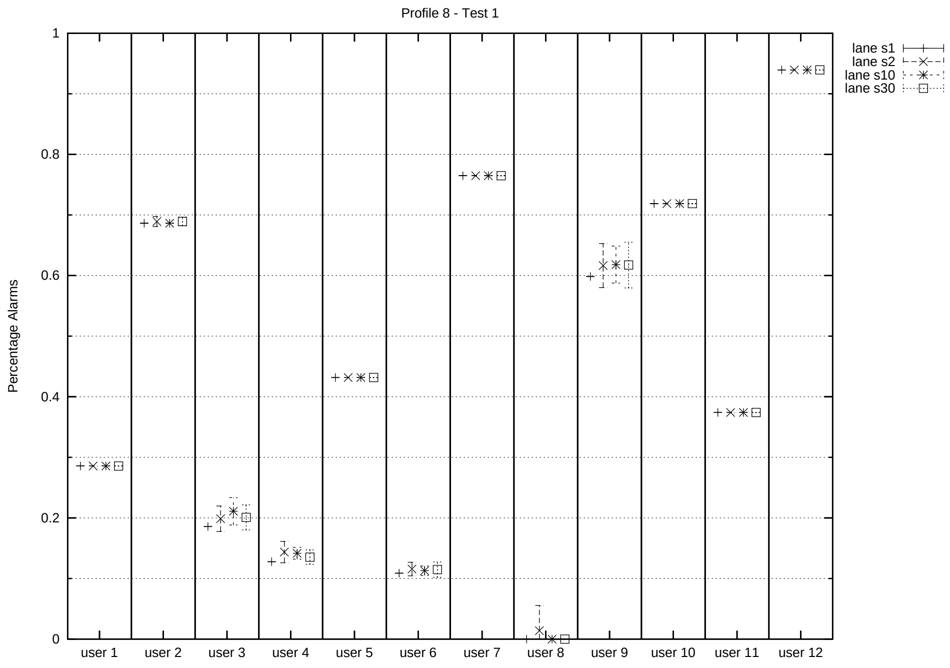


(a)

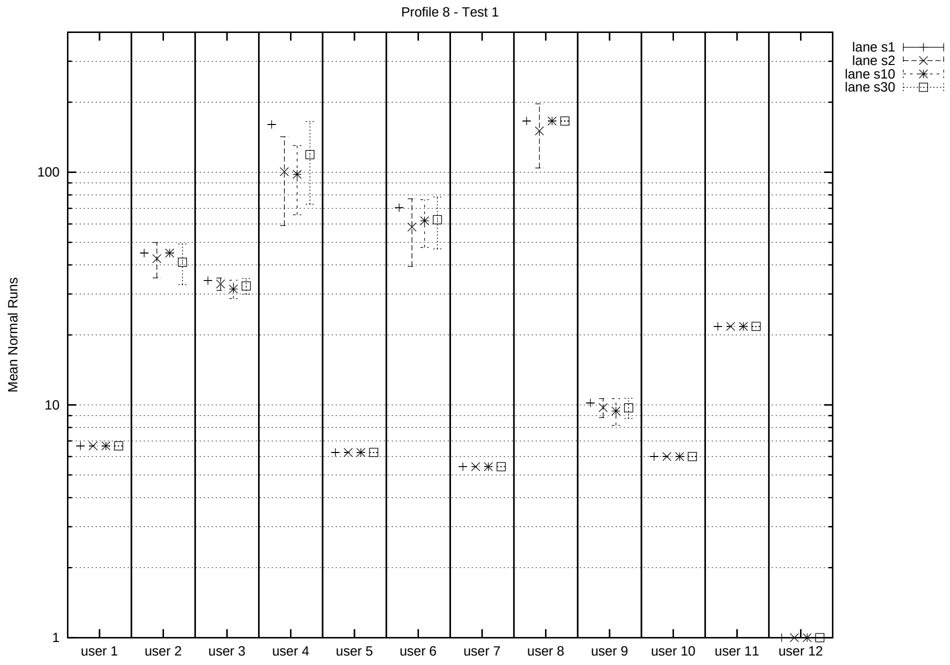


(b)

Figure B.6: Percentage alarms and mean normal runs for Markov chain models with different increment weights of USER4's behaviors (Test Set 2 of 2).

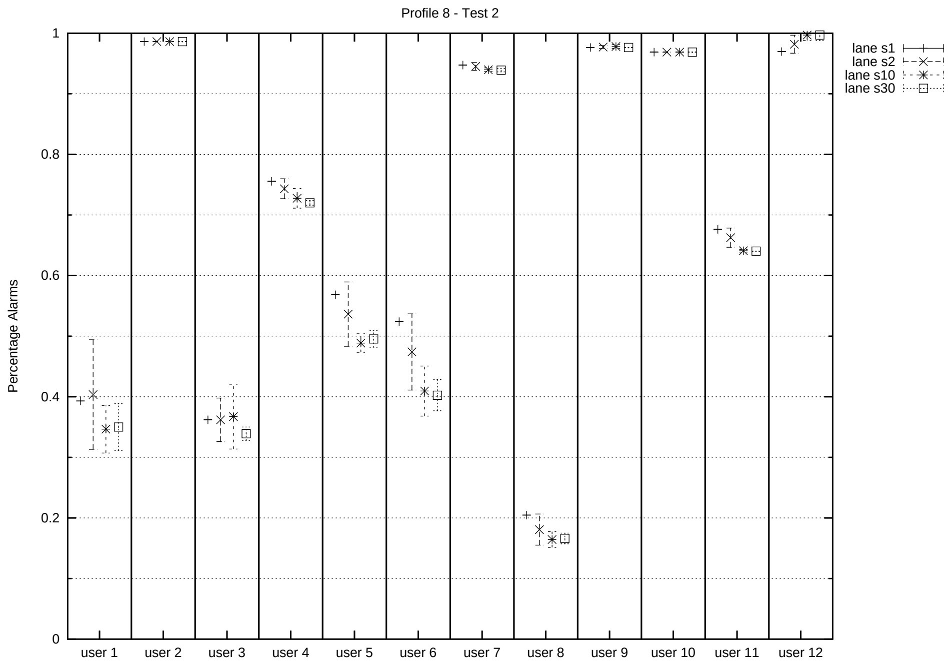


(a)

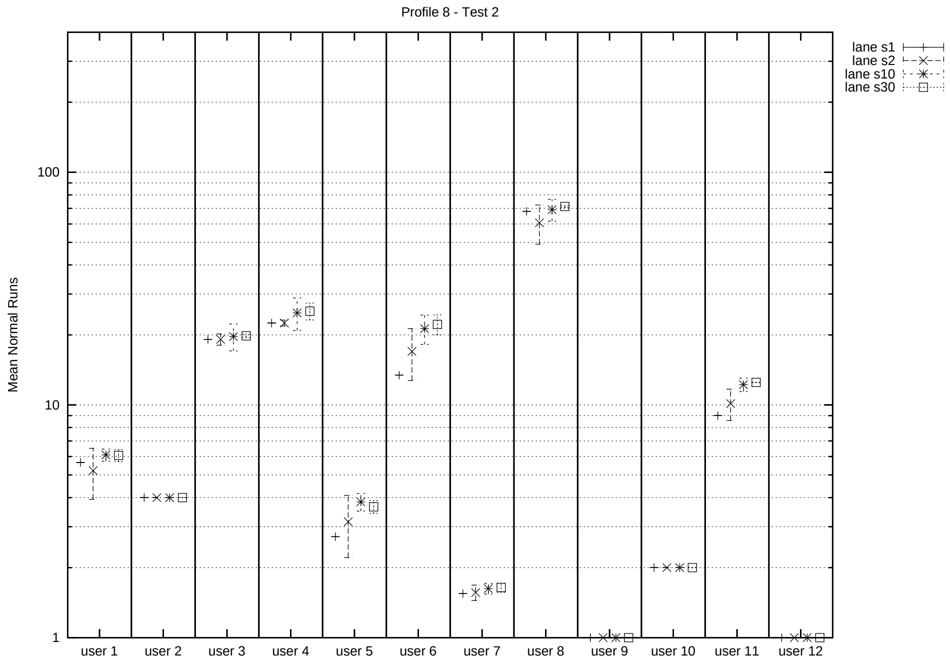


(b)

Figure B.7: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER8's behaviors (Test Set 1 of 4).

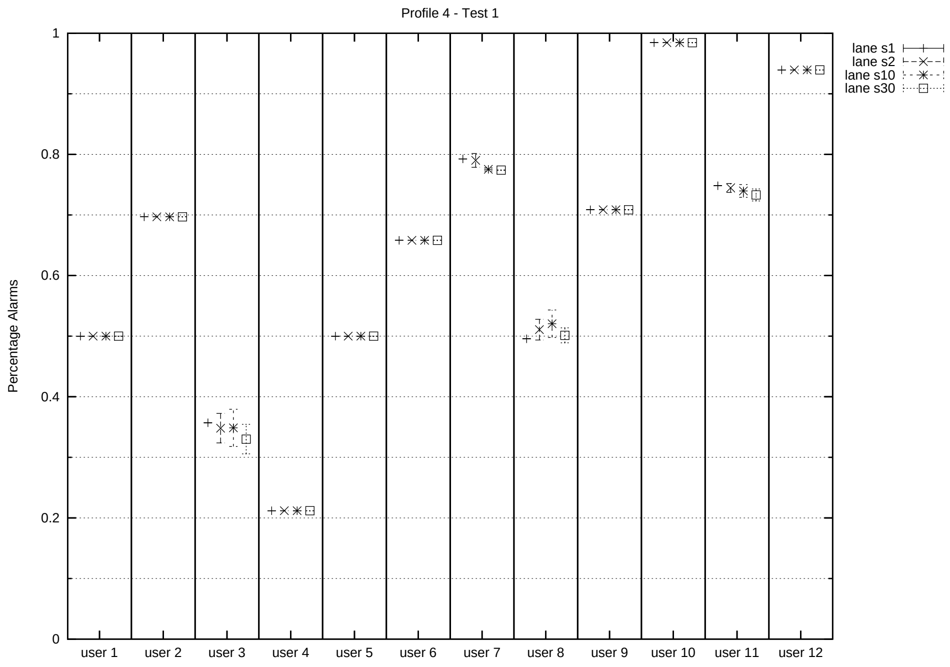


(a)

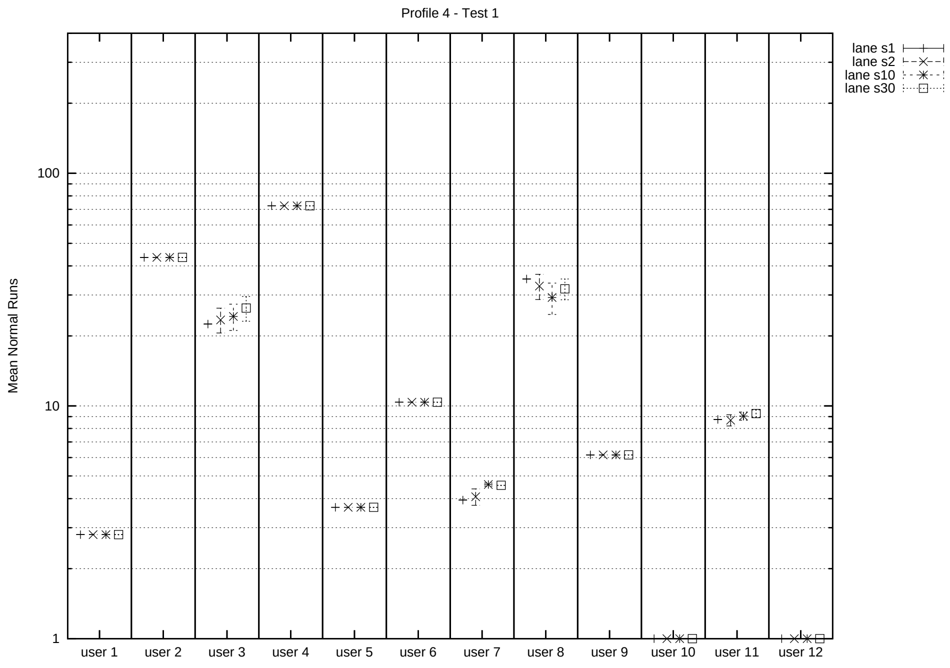


(b)

Figure B.8: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER8's behaviors (Test Set 2 of 4).

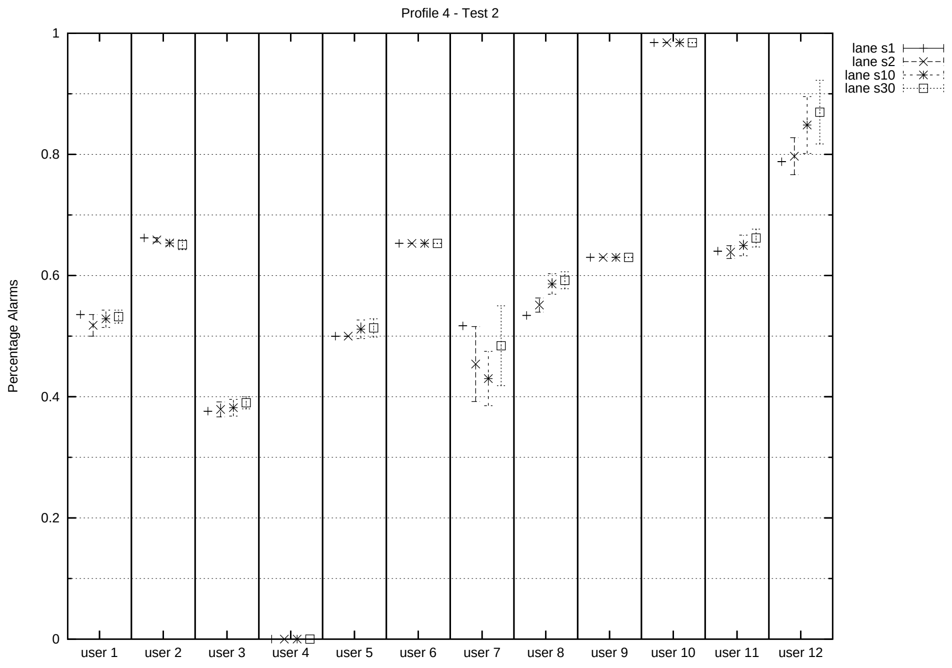


(a)

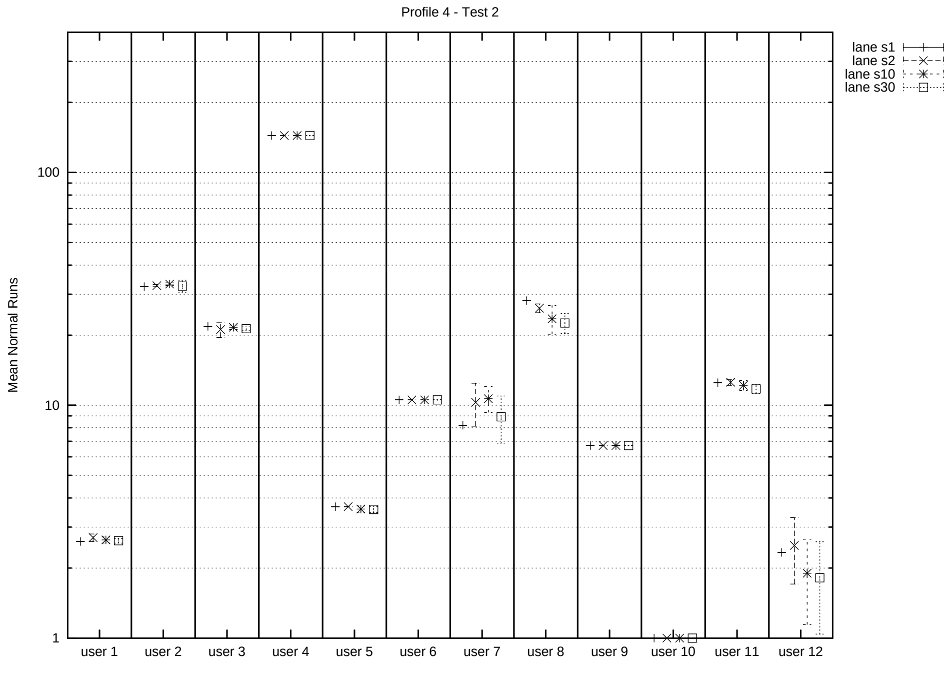


(b)

Figure B.9: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER4's behaviors (Test Set 1 of 2).

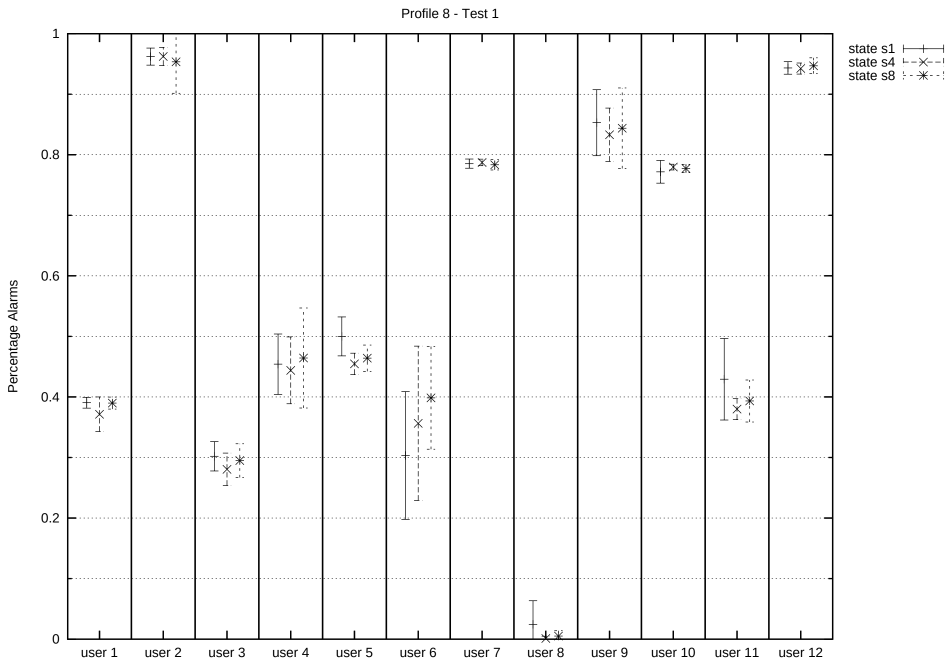


(a)

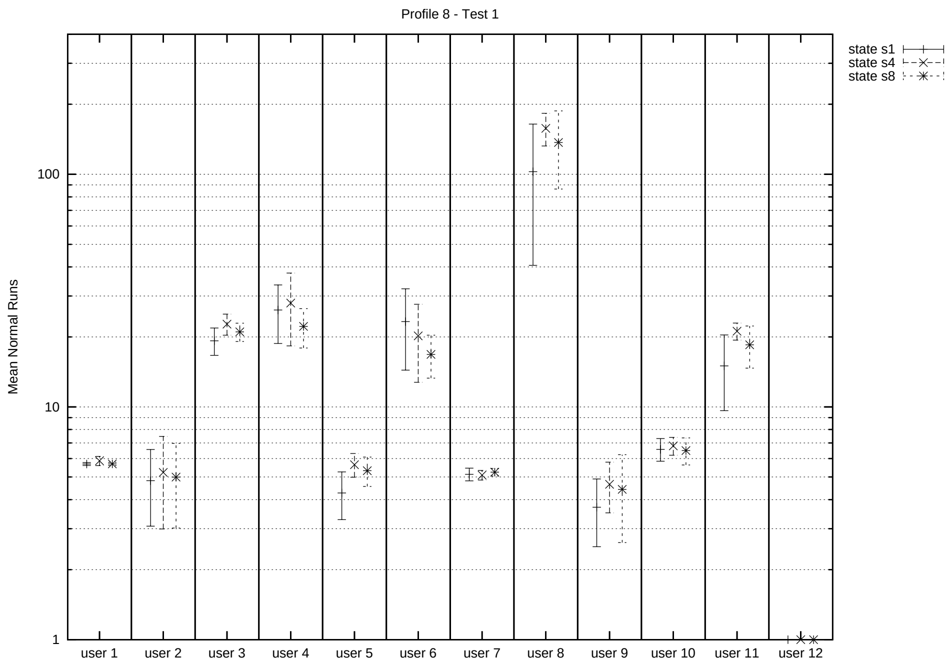


(b)

Figure B.10: Percentage alarms (a) and mean normal runs (b) for HMMs with different state numbers of USER4's behaviors (Test Set 2 of 2).

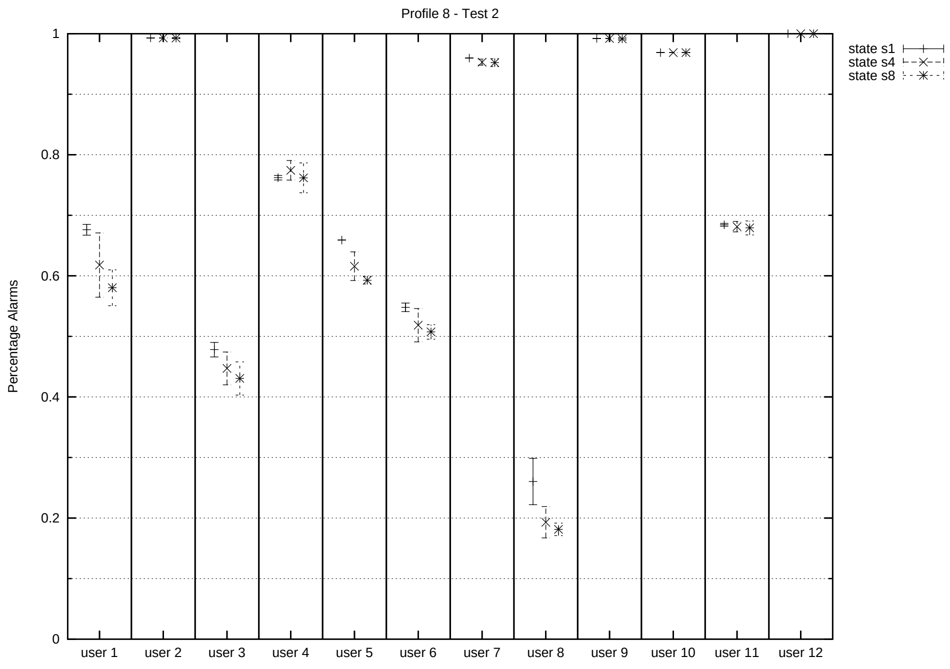


(a)

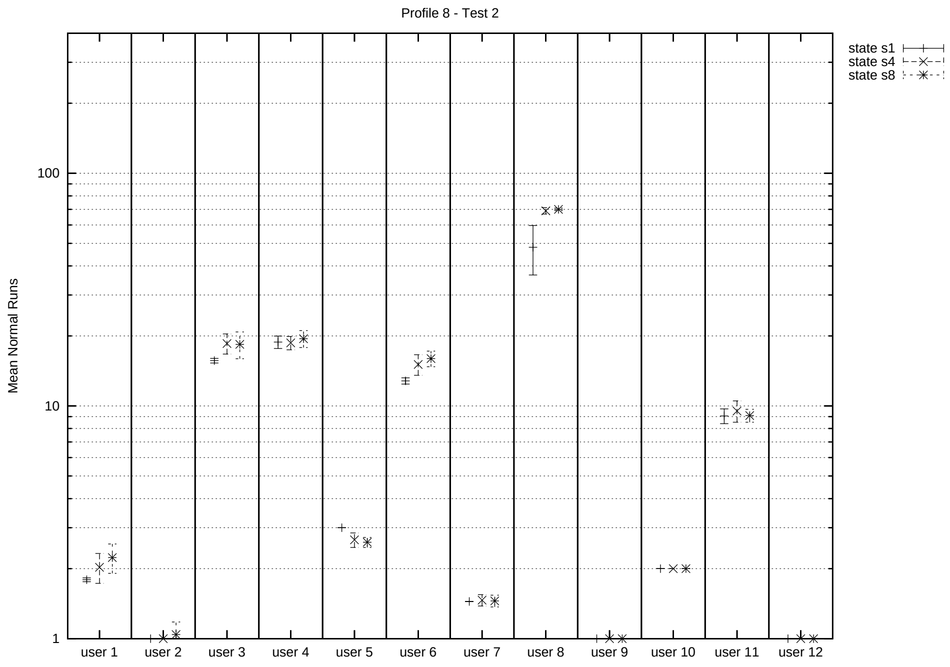


(b)

Figure B.11: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER8's behaviors (Test Set 1 of 4).

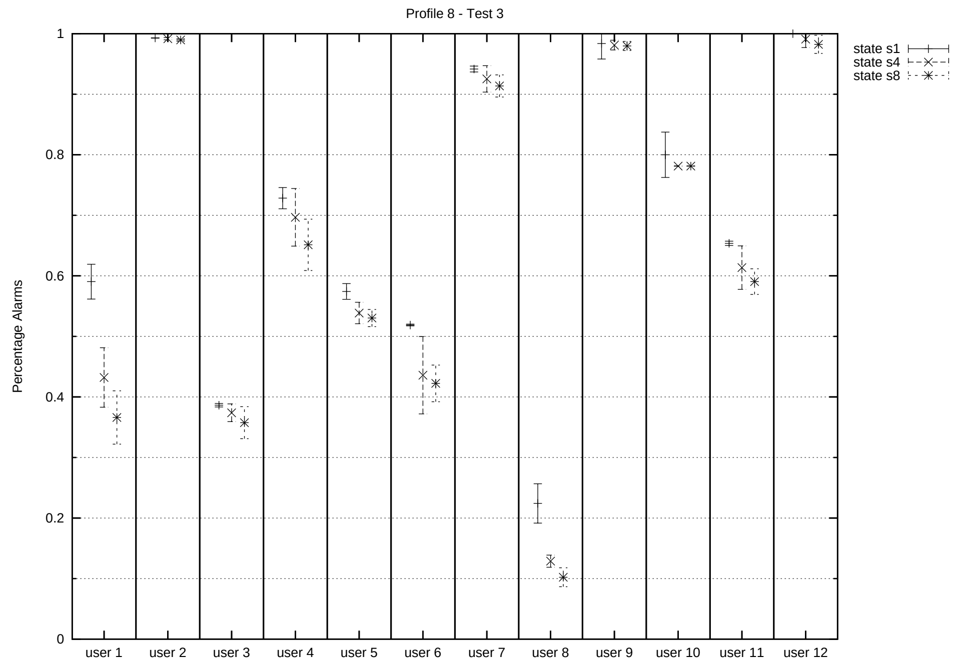


(a)

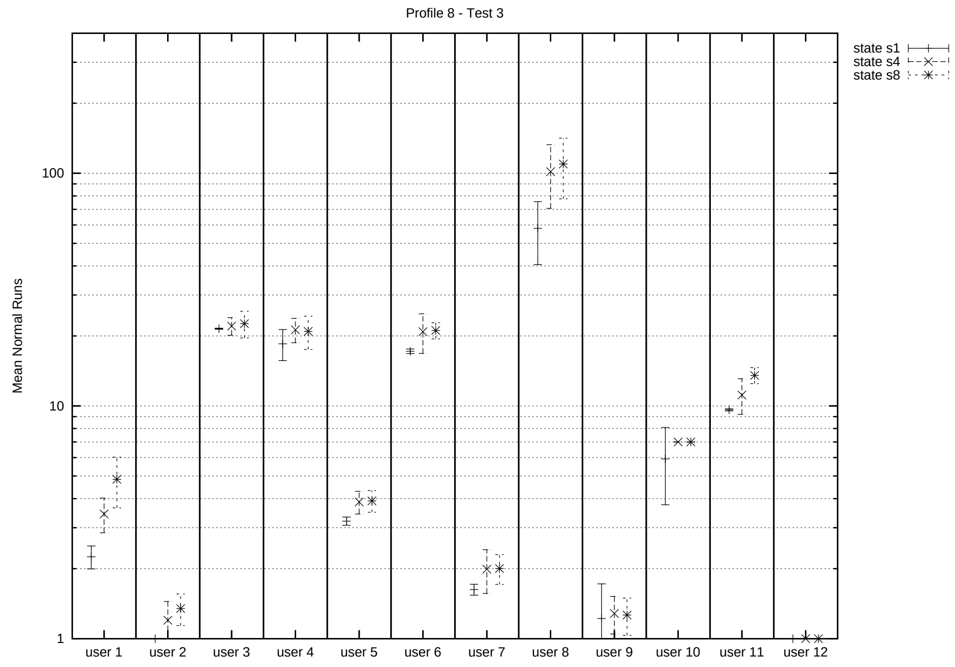


(b)

Figure B.12: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER8's behaviors (Test Set 2 of 4).

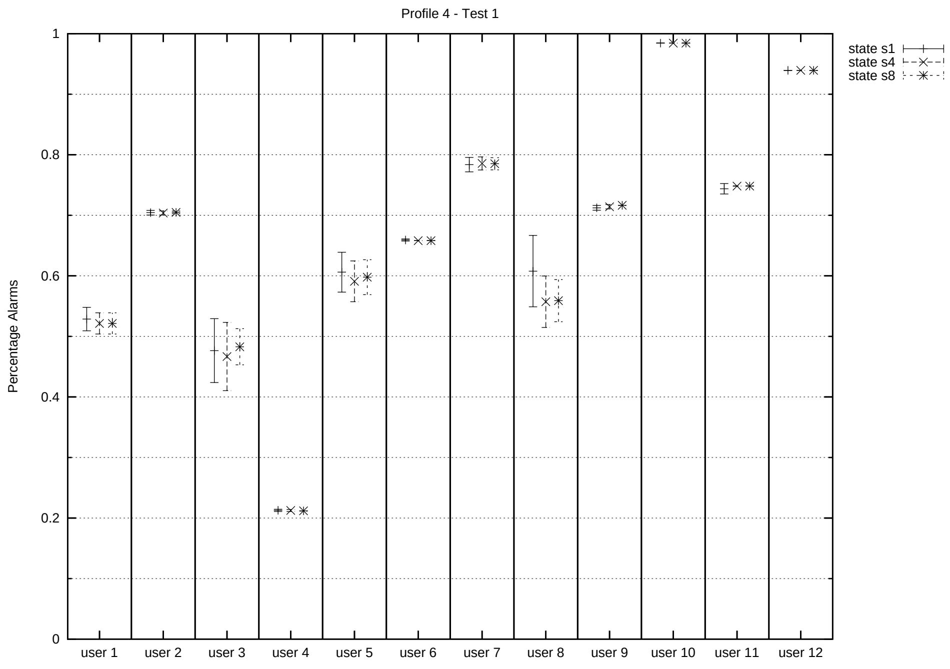


(a)

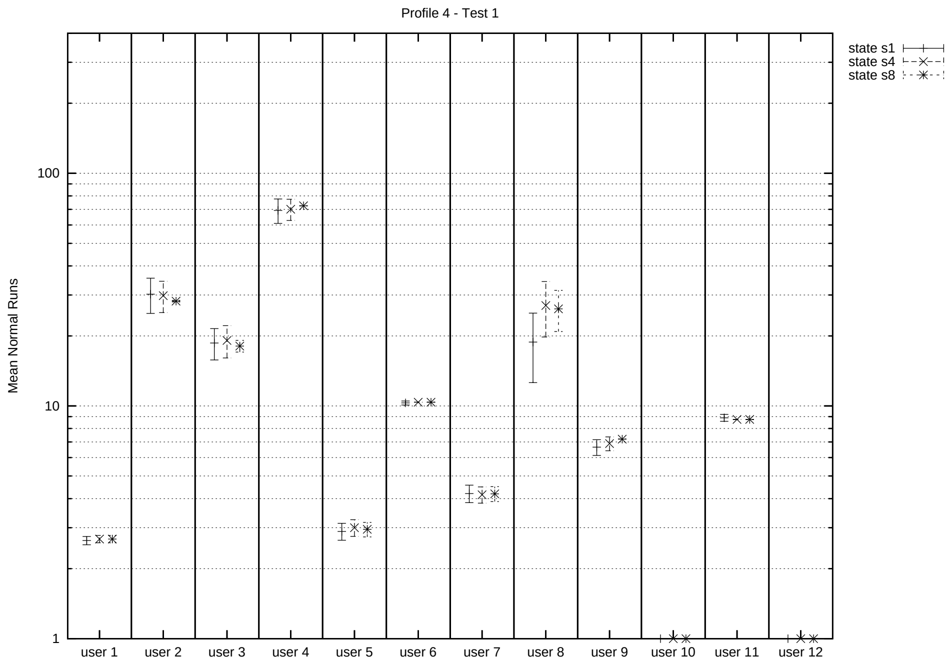


(b)

Figure B.13: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER8's behaviors (Test Set 3 of 4).

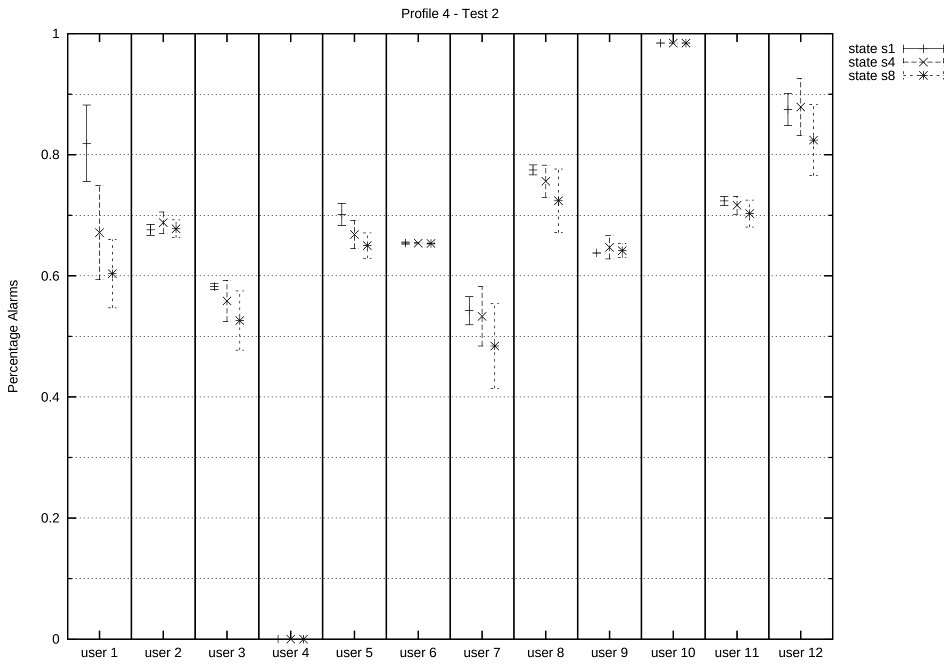


(a)

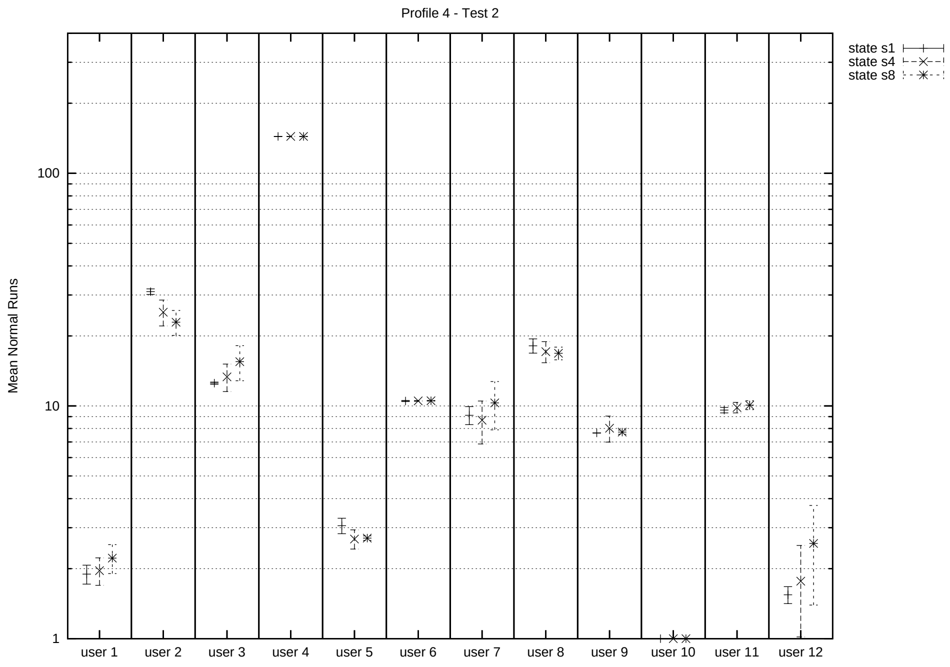


(b)

Figure B.14: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER4's behaviors (Test Set 1 of 2).



(a)



(b)

Figure B.15: Percentage alarms (a) and mean normal runs (b) for HMMs with prior estimation and different additional state numbers of USER4's behaviors (Test Set 2 of 2).