# A Plan Classifier based on Chi-Square Distribution Tests

José A. Iglesias[1], Agapito Ledezma[1],
Araceli Sanchis[1] and Gal A. Kaminka[2]

[1]The CAOS Group, Department of Computer Science,
Carlos III University, Leganés, Spain
{jiglesia, ledezma, masm}@inf.uc3m.es

[2]The MAVERICK Group, Department of Computer Science,
Bar-Ilan University, Israel {galk}@cs.biu.ac.il

Corresponding author: José Antonio Iglesias
Tel: +34 916249111, Fax: +34 916249129
Email: jiglesia@inf.uc3m.es

**Abstract**

To make good decisions in a social context, humans often need to recognize the plan underlying the behavior of others, and make predictions based on this recognition. This process, when carried out by software agents or robots, is known as *plan recognition*, or *agent modeling*. Most existing techniques for plan recognition assume the availability of carefully hand-crafted *plan libraries*, which encode the a-priori known behavioral repertoire of the observed agents; during run-time, plan recognition algorithms match the observed behavior of the agents against the plan-libraries, and matches are reported as hypotheses. Unfortunately, techniques for automatically acquiring plan-libraries from observations, e.g., by learning or data-mining, are only beginning to emerge.

We present an approach for automatically creating the model of an agent behavior based on the observation and analysis of its atomic behaviors. In this approach, observations of an agent behavior are transformed into a sequence of atomic behaviors (events). This stream is analyzed in order to get the corresponding behavior model, represented by a distribution of relevant events. Once the model has been created, the proposed approach presents a method using a statistical test for classifying an observed behavior. Therefore, in this research, the problem of behavior classification is examined as a problem of learning to characterize the behavior of an agent in terms of sequences of atomic behaviors. The experiment results of this paper show that a system based on our approach can efficiently recognize different behaviors in different domains, in particular UNIX command-line data, and RoboCup soccer simulation.

**Keywords:** agent modeling, plan recognition, activity recognition, user modeling.

# 1 Introduction

To make good decisions in a social context, humans often need to recognize the plan underlying the behavior of others, and make predictions based on this recognition. This process, when carried out by software agents or robots, is known as *plan recognition*, or *agent modeling* [6, 11, 19, 27, 35].

One of the key tasks in agent modeling is *behavior classification* in which a stream of observations is categorized into pre-determined classes. The focus here is on recognizing patterns (possibly, multiple patterns) in the stream, that would allow its classification. This is in contrast to other agent modeling tasks, where the entire sequence of observed actions is to be recognized and matched against the plan library (e.g., to predict goals [22], or identify the sequence of actions that compose a plan [10, 11, 19, 27, 30, 31, 35, 40, 42]).

To carry out the classification, activity recognition algorithms rely on a plan library that encodes the patterns to be matched against the incoming observations. Successful matches indicate possible classifications. Such plan libraries may be built by hand, or automatically acquired. For instance, within the domain of robot soccer, Riley and Veloso [37] use hand-built models of ideal opponent behavior to classify opponent types in robot soccer. In contrast, Han and Veloso [20] use Hidden Markov Models (HMMs) trained to classify specific robot motions as specific behaviors (e.g., approach a ball). A HMM is a statistical technique for modeling based on the assumption that the process is a Markov process with hidden parameters. Indeed, HMMs and their many variants [33, 36] are a common tool in state-of-the-art activity recognition [8, 16].

This paper presents an alternative approach to behavior classification, based on sequence classification. The presented approach is called *ABCD* (Agent Behavior Classification based on sequence Distribution). It is based on representing the behavior of an agent as a distribution over sequences of observed atomic, where such sequences have been identified during training as statistically significant. When a new set of observations is given, the distribution of sequences in it is compared to the distribution of sequences in each of the classes, and the most closely matching model is selected. ABCD is appropriate for domains in which recognizing the atomic behaviors of agents is a tractable task, but the space of sequential combinations of these behaviors is practically unexplorable.

The approach presented in this paper is fully implemented and empirically evaluated in several domains. In the first, we use ABCD to recognize UNIX users based on previous traces of the commands they typed in a shell. In this environment, the goal is to develop a model or *profile* of the normal working state of a UNIX user with which its behavior can be recognized. We show that ABCD works successfully in two extensive UNIX command-line data sets, one with nine users, and one with 50 users. Moreover, ABCD is shown to be superior to the use of HMMs in these data-sets. We additionally use ABCD to recognize patterns used in the *RoboCup Soccer Coach Simulation* [2], which uses a simulated soccer domain [34].

This paper is organized as follows. First, Section 2 provides a brief overview of the background and related work relevant to this work. The approach (*ABCD*) is explained in detail in section 3. The different phases of the approach and its complexity are described. Section 4 describes the experiments and their results. Finally, section 5 contains future work and concluding remarks.

# 2 Background and Related Work

There are many different areas in which it is very useful to model, recognize, or classify the behavior of other agents. The literature of agent modeling is truly vast. We thus focus here only on the most related work in behavior classification.

Han and Veloso [21] recognize behaviors of robots using *Hidden Markov Models* and their

approach is evaluated in a real world scenario. In this case, states in the *HMMs* correspond to an abstracted decomposition of a robot's behavior. The observations of a robot represent its physical state and the corresponding set of Markov states represents its *mental state*. Then, as the intermediate states probabilities of a HMM indicate a behavior in progress, they can be used in anticipating the future behavior (states) of the robot. However, this approach makes a Markovian assumption (the probability of moving from the current state to another is independent of its previous states) in modeling an agent, whereas our proposal takes into account a short sequence of events to incorporate some of the historical context of the agent.

Riley and Veloso [37] propose a classification of the adversary behavior into predefined adversary classes in the domain of simulated robotic soccer. The behavior of the opponent is modeled by useful features (as determined by the developer) based on the areas in which the soccer events occur (i.e., spatial features). During classification, the system accumulates adversary position information in grids and then a decision tree is used for classifying it. In contrast, ABCD (presented here) examines *the temporal ordering* of events, but for the most part ignores their location. It is therefore a complementary approach.

Instead of describing the complete opponent behavior, Steffens [40] presents a feature-based declarative opponent-modeling (*FBDOM*) technique which identifies tactical moves of the opponent in multi-agent systems. In this case, the models built need distinct and stable features which describe the behavior of opponents. As in our approach, *FBDOM* is not restricted to any specific domain. However, it does not discover sequences. Instead, any temporal orderings are a-priori defined as features.

Time series and decision tree learning are used by Visser and Weland [44] to induce rules that describe the behavior of a team. The key idea of that research differs from ours. In their case, an object in a complex environment is seen as a time series. A qualitative abstraction of those time series is applied and an approach is used to discretize these time series in order to use the results for learning by C4.5, which cannot capture the temporal ordering of events (instead, the temporal ordering is captured by the qualitative abstraction of the time-series). In contrast, our work directly tackles the temporal ordering of events. A discretization (if needed) is applied to each point in the time series; ABCD is used to directly learn sequences of discrete events.

Work in plan recognition differs from classification, in that the entire sequence of observation must match the model. Tambe and Rosenbloom [42] infer opponent actions by using an agent's own behavior representation. Laird [30] uses the same idea in Quake, a real-time computer game. Once a complete sequence of behaviors matches, it can be used to distinguish the matching behavior from another. However, a key issue in plan recognition is that more than one model may match, and thus ambiguous matches are to be expected. AHMM [8] is an HMM variant that consists of a number of interacting Markov chains. Bui describes approximate-inference policy recognition algorithms for this model. A key difference with all of these methods is that they do not have a learning component, so the sequences must be manually constructed.

Carmel and Markovitch [9] propose a method to infer a model of the opponent's strategy which is represented as a Deterministic Finite Automaton. They provide a learning procedure, and show that the use of the model leads to improved results, due the model's predictions. In contrast, we focus on domains in which the behavior of the observed agent is non-deterministic, and likely too complex to model by a reasonably-sized finite automaton. Moreover, the technique we describe in this paper allows only classification of behavior, rather than predictions of actions.

Kaminka et al. [25] recognize basic actions based on descriptive predicates, and learn relevant sequences of actions using a statistical approach. Horman and Kaminka [23] expanded on this approach. A similar process is also used in [24] to create frequent patterns in dynamic scenes. However, these previous works focused on unsupervised learning, with no ability to classify behav-

iors into classes.

As the main goal of this research is to classify an observed behavior, we consider that the actions performed by an agent (user) are usually influenced by past experiences. This aspect motivates the idea of automated sequence learning for behavior classification; if we do not know the features that influence the behavior of an agent, we can consider a sequence of past actions to incorporate some of the historical context of the agent.

Indeed, sequence learning is arguably the most common form of human and animal learning. Sequences are absolutely relevant in human skill learning [41] and in high-level problem solving and reasoning [4]. Taking this aspect into account in this paper, the problem of behavior classification is examined as a problem of learning to characterize the behavior of an agent in terms of sequences of atomic behaviors. Therefore, the behavior classification problem is transformed into a sequence classification problem where a sequence represents a specific behavior. This transformation can be done because it is clear that any behavior has a sequential aspect, as actions are performed in a sequence.

As in this research, there are many other areas in which sequential data need to be analyzed in order to solve a problem. In general, the sequence learning problem can be categorized in four basic categories: sequence prediction, sequence generation, sequence classification and sequential decision making. In this paper, the sequence classification is the category analyzed and developed.

Considering the sequence classification, the main reason to need to handle sequential data is because of the observed data from some environments are inherently sequential. An example of these data is the DNA sequence. Ma et al. [32] present new techniques for bio-sequence classification. Given an unlabeled DNA sequence $S$, the goal in that research is to determine whether or not $S$ is a specific promoter (a gene sequence that activates transcription). Also, a tool for DNA sequence classification is developed by Chirn et al. [13]. In a very different problem (computer intrusion detection problem), Coull et al. [14] propose an algorithm that uses pair-wise sequence alignment to characterize similarity between sequences of commands. The algorithm produces an effective metric for distinguishing a legitimate user from a masquerader. Schonlau et al. [39] investigate a number of statistical approaches for detecting masqueraders.

A very important issue in sequence learning is temporal dependencies. The following aspect is essential in our research: A current situation or the action that an agent performs usually depends on what has happened before. This aspect is taken into account in our research and in models such as HMMs; however, there are some other models which have problems dealing with such dependencies. For example, recurrent neural network models [18] or reinforcement learning cannot manage efficiently the long-range dependencies.

We focus in this research on learning segments of sequences whose frequency (*support*) within the training data is sufficiently high [3]. In addition, there exist other techniques ([43]) which can be combined with support. However, as we want to provide a *general* approach which can represent and handle different behaviors in a wide range of domains, those methods which require human expert guidance have been ignored.

# 3 ABCD: Agent Behavior Classification based on sequence Distribution

In this section, we present our approach for modeling and classifying agent behavior (where an *agent* could be a software agent, a robot or a human being). In order to recognize an observed behavior, our approach (as other learning-based agent modeling methods [37, 40]) uses a behavior-library in which all the different possible behaviors are stored; during run-time, the observing agent

matches observations against the different behaviors in the behavior-library. However, we depart from previous work in that we will be looking at classification across multiple libraries (to identify different agents, represented by different libraries).

Thus, as we show in Figure 1, our approach is divided in two main phases: Construction of Behavior Models (each model represented by a behavior library, one for each agent), described in Section 3.1 and Behavior Classification (Recognition), preferring one of the models over the others (Section 3.2). In addition, there are important questions of the complexity of the processes we introduce. We discuss those in Section 3.3.

### 3.1 Construction of Behavior Models

In many application domains, the actions performed by an agent are inherently sequential, and thus their ordering within the sequence should be considered in the modeling process. For example, in a human-computer interaction by commands, the specific ordering of commands within a sequence is essential for the result of the interaction[1].

Because of this, our focus in this paper is on behavior models that specifically encode the observed sequences of actions executed by the observed agents. In other words, the behavior library associated with an agent $A$ encodes sequences of actions that capture different behaviors which $A$ exhibits. The behavior library is then considered the model of $A$.

Construction of a behavior model is based on a stream of observed atomic discrete events, describing the behavior of the agent in its environment. Each *event* is an atomic observation that occurs in a certain place during a particular interval of time and defines a specific act of an agent. The kind of events and its features have to be determined by the designer taking into account the environment, and is beyond the scope of this paper; we note in passing that in general this capability exists even for domains in which observations are of continuous states, rather than discrete actions (e.g., [25] for RoboCup).

Once a sequence of events—representing the behavior of the agent—has been obtained, the *Creating Model Module (CMM)* constructs the corresponding agent model. The first step in the *CMM* is to extract the significant pieces of the sequence that can represent a repeating pattern of behavior. In many domains of interest, the temporal (non-Markovian) dependencies are very significant and we consider that a current event might depend on the events that have happened before it, and is possibly related to the events that will happen after it is observed.

We use the following example sequence to explain in detail the construction of behavior models. Let us consider we are observing an agent and its behavior is represented by the following sequence: $\{A \rightarrow B \rightarrow A \rightarrow B \rightarrow C\}$ where each different capital letter represents a different atomic event. We describe the process of constructing of a model from a single sequence of events. The sequence is then segmented into sub-sequences, and these are stored.

The event sequence needs to be segmented into several sub-sequences which will be inserted in the same model separately. This segmentation can be done by using some environment characteristic that can separate efficiently the sequence in several sub-sequences of uninterrupted events (for example, if we are modeling the behavior of a player in a soccer game, its sequence of events during a game can be divided by considering series of uninterrupted actions while he is the ball possessor). Otherwise, the sequence can be segmented by defining an appropriate maximum length and obtaining every possible ordered sub-sequence of that specific length. The length of these sub-sequences is an important aspect because it modifies both the size of the model and the final results quite significantly.

[1]For instance, consider the difference between the UNIX command sequence "`rm a.txt ; mv b.txt a.txt`", and the sequence "`mv b.txt a.txt; rm a.txt`".

For example, we can divide the example sequence ($A \rightarrow B \rightarrow A \rightarrow B \rightarrow C$) into sub-sequences of equal size. Let 3 be the sub-sequence length, then we obtain: $A \rightarrow B \rightarrow A$ and $B \rightarrow A \rightarrow B$ and $A \rightarrow B \rightarrow C$.

The sequences are stored in a *trie* data-structure [17, 28]. This follows in the footsteps of [23, 25]. When a new model needs to be constructed, we create an empty *trie*, and insert each sub-sequence of events into it, such that all possible sub-sequences are accessible and explicitly represented. Every trie-node represents an event appearing at the end of a sub-sequence, and the node's children represent the events that have appeared following this event. Also, each node keeps track of the number of times an event has been inserted on to it. When a new sub-sequence is inserted into the *trie*, existing nodes of the *trie* are modified and/or new nodes are created. As the dependencies of the events are relevant in an agent behavior, the sub-sequence suffixes (sub-sequences that extend to the end of the given sequence) are also inserted.

To illustrate, consider the previous example. The first sub-sequence ($\{A \rightarrow B \rightarrow A\}$) is added as the first branch of the empty *trie* (Figure 2-a). Each event is labeled with the number 1 that indicates that the event has been inserted in the node once (in Figure 2, this number is enclosed in square brackets). Then, the suffixes of the sub-sequence ($\{B \rightarrow A\}$ and $\{A\}$) are also inserted (Figure 2-b). Finally, after inserting the three sub-sequences and its remaining suffixes, the completed *trie* is obtained (Figure 2-c).

Once the *trie* is created, the sub-sequences that characterize the behavior have to be obtained (where a sub-sequence is a path from the *root* node to any other node of the *trie*). Thus, the *trie* is traversed to calculate the relevance of each sub-sequence. For this purpose, frequency-based methods [3] are used. In particular, in this approach, to evaluate the relevance of a sub-sequence, its relative frequency or support [3] is calculated. This value is the number of occurrences of a particular sub-sequence (of length $n$) divided by the total number of sub-sequences of equal length ($n$). As the sub-sequences in a *trie* are the different paths from the root to a node, the support value of a sub-sequence is stored in its last node. Therefore, in this step the *trie* is transformed into a set of sub-sequences labeled with a value (support). Note that this step does not necessarily have to be carried out separately, after the creation of trie. Rather, support counts can be updated during the insertion of every sub-sequence.

In the previous example, the *trie* consists of 9 nodes; therefore, the model consists of 9 different sub-sequences which are labeled with its support. Figure 3 shows the distribution of these sub-sequences.

The model of an agent, encoded by the behavior library, is then the distribution of sub-sequences within the library (stored in a trie). Agents then differ not in the sequences of action they produce, but in the relative likelihood of generating these sequences. Once a behavior model (distribution of relevant sub-sequences) has been created, it is stored in the *Behavior Model Library (BMLib)* (similar to the plan-libraries used in the plan recognition). Different created models are stored and labeled in the library with a *name* that identifies each model. In Section 3.3 we consider the complexity of this approach (a separate trie for each agent), in contrast to an approach utilizing a single trie for all agents.

### 3.2 Behavior Classification

Once a set of models is available, a new stream of observations is classified. First, the stream of observations is segmented and leads to a new model stored in a new trie, as described in the previous section. This creates a distribution of sub-sequences, based on the observations (which serve as a sample) of the observed agent's behavior. Then, the model is matched with the models stored in the *BMLib*. Then, the behavior model (distribution of observed sub-sequences) is matched with all

the behavior models stored in *BMLib*. Thus given an observed agent and a set of agent behavior models ($\{ab_1, ab_2, ..., ab_n\}$) stored in the *BMLib*, the goal of this phase is to determine which model best fits the observed agent's action sequence. In Figure 1, this is the process in *Behavior Matching Module (BMM)*.

The matching of the new observed model to the models in the *BMLib* is done by a non-parametric (distribution-free) statistical test for comparing distributions. The choice of a distribution-free test is used so as to not bias the test in any way. We chose the two-sample *Chi-Square* test for this purpose.

To apply the *Chi-Square* test, the behavior model to classify is considered as an observed sample and all the behavior models stored in *BMLib* are considered as the expected samples. Then, this test compares the observed distribution with all the expected distributions objectively and evaluates whether a deviation appears. This is done by the comparison of two sets of support values, available in the trie; the *Chi-Square* is the sum of the terms $\frac{(Obs-Exp)^2}{Exp}$ calculated from the observed ($Obs$) and expected ($Exp$) distributions (models). Considering this sum of terms, the expected values (from one of the models stored in the *BMLib*) have to be compared with the observed values (from the model of the agent behavior to classify).

If an observed value is not represented in the expected distribution, it is not considered. Also, the amount of sub-sequences in an expected distribution is usually very large, so this kind of comparison can be very time-consuming. In order to solve this problem and to analyze all the observed sub-sequences, the way to compare the two distributions is modified to the sum of the terms $\frac{(Exp-Obs)^2}{Obs}$. Figure 4 represents graphically the idea of the proposed novel comparing method.

The key idea here is that the support value of each sub-sequence of the observed distribution is compared with the support value of the corresponding sub-sequence of the expected distribution. With this comparison, we obtain a value that indicates the difference (deviation) between the two distributions in terms of support (i.e., relative to the overall number of sequences). The lower the value, the higher the similarity between the two behaviors. An important advantage of the proposed test is its efficiency because only the observed sub-sequences are evaluated. However, there is no penalty for the expected relevant sub-sequences which do not appear in the observed distribution.

This comparison test is applied once for each behavior model stored in *BMLib*. The model obtaining the lowest value is considered as the most similar one. As the observed agent behavior is only classified in one of the behavior previously analyzed, it is not necessary to define a threshold for this process. Also, the number of terms to sum in each comparison is always the same: number of sub-sequences in the observed behavior model. It means that the degrees of freedom ($dof$) are the same in all the comparisons with the expected behavior models. Otherwise, a normalization of the results according to the $dof$ should be done.

As an example, let's consider the sequence that represents the observed behavior: $\{A \rightarrow B \rightarrow D\}$. Once its model (distribution) is created, it is compared to the distributions of the *BMLib* (*Expected Distributions*). We compare the two sets of frequencies using the sum of the terms $\frac{(Exp-Obs)^2}{Obs}$. Figure 5 shows the comparison between the previous expected distribution (*Expected Behavior Model 1*) and the observed distribution (*Observed Behavior Model*). The comparison value in this example is: $\frac{(0,42-0,33)^2}{0,33} + \frac{(0,5-0,5)^2}{0,5} + \frac{(0,42-0,33)^2}{0,33} + \frac{(0-1)^2}{1} + \frac{(0-0,5)^2}{0,5} + \frac{(0-0,33)^2}{0,33} =$ 1,88. This comparison is done with all the models stored in *BMLib*, and the observed model is classified into the model with obtains the lowest value.

### 3.3 The complexity of *ABCD*

In the *BMLib* described earlier, each behavior is represented by a distribution which is stored in a *trie* (a method that we call *K-Tries-Library* because the number of *tries* in the library depends on the number of behavior classes). However, the library could consist of a single *trie* in which all the behavior models are stored together, with appropriate annotation to distinguish classes. We call this method *One-Trie-Library*.

In the *One-Trie-Library* method, a new *trie* is created with the first sequence (behavior) to insert and the other sequences (behaviors) are added in the same *trie*. Therefore, each trie-node must contain information about the agents it represents, and the support for this specific node, for each of the agents. The fact that a sub-sequence could have already been inserted in the *trie* for other behaviors needs to be taken into account when the *trie* is being created.

Considering that $k$ is the number of behaviors to model; a *trie* node that stores an event in which $k$ models are represented is $k$ times bigger than the *trie* node that stores one behavior. Therefore, the disk space needed for a *K-Tries-Library* always will be bigger than the space needed for *One-Trie-Library*. However, the time consumed for creating the different libraries and for classifying a given agent behavior using the two libraries is very different. In the following sections, these two aspects are studied.

#### 3.3.1 K-Tries-Library vs One-Trie-Library : Creating the Library

Given the function *TimeInsert (n, l)* that returns the time consumed for inserting $n$ events in a *trie* using sub-sequences of length $l$ and assuming that all the behaviors have the same amount of sequence events: The Equation 1 gives the time consumed for creating a *trie* per behavior modeled. However, if we consider only a *trie* for representing all the behaviors, the new sub-sequences are always inserted in the same *trie*, so the time consumed for inserting the events in a single *trie* is represented in Equation 2.

$$T(\text{InsertKTries}) = O(k \times \text{TimeInsert}(n, l)) \tag{1}$$

$$T(\text{InsertOneTrie}) = O(\text{TimeInsert}(kn, l)) \tag{2}$$

where $n$ is the number of events to insert (using sub-sequences of length $l$) per behavior, and $k$ is the number of different behaviors.

For comparing these two equations, we have to consider that the more events inserted in the *trie*, the more time consumed for inserting a new event. Therefore, as the time for inserting a new event grows exponentially depending of the events already inserted, the creation of a *One-Trie-Library* is more time consuming than the creation of a *K-Tries-Library*.

Nevertheless, the creation of the library (*BMLib*) is done just once and the classification process is applied once per agent behavior to recognize. Hence, the time consumed for recognizing an agent behavior (studied in the next section) is an aspect more important in our approach.

#### 3.3.2 K-Tries-Library vs One-Trie-Library : Using the Library

Given a new sub-sequence of length $l$ to compare with the sub-sequences stored in *BMLib*: In a *K-Tries-Library*; the sub-sequence has to be searched separately in all the *tries*. Then, the time consumed, in the worst case, for searching the sub-sequence in the *trie* is:

$$T(\text{SearchKTries}) = O(lk). \tag{3}$$

On the other hand, using a *One-Trie-Library*, the time consumed, in the worst case, is the time of access to the corresponding node plus the time of access to the event of the different models represented in the current node:

$$T(\text{SearchOneTrie}) = O(l + k). \tag{4}$$

Considering these 2 equations, the time for searching a sub-sequence in a single *trie* is shorter. In *ABCD* this action is used several times in the classification process, so this aspect has been taken into account. Therefore, the experiments for this research have been performed by using a *One-Trie-Library*.

# 4 Experiments

In order to evaluate *ABCD*, we conducted extensive experiments in two different environments: *UNIX User Data* (Section 4.1) and *RoboCup Soccer Coach Simulation* (Section 4.2).

## 4.1 UNIX User Data

In this domain, the observed behavior of a UNIX user consists of the UNIX commands he/she typed during a period of time. The goal is to classify a given sequence of UNIX commands (user behavior) in one of the behavior models previously created and stored. This task is very useful in different application areas such as computer intrusion detection, intelligent tutoring systems, and more.

To evaluate *ABCD* in this environment, we have used two different sources of UNIX data with different number of users to classify:

- **Set of 9 UNIX Users**: Data[2] drawn from the command histories of 9 UNIX computer users at Purdue University over 2 years [15]. Each user file contains from about 10000 to 60000 commands and represents a specific UNIX users profile.
- **Set of 50 UNIX Users**: Data[3] used in the masquerade-detection studies done by Schonlau et al. [39]. In Schonlau research, commands from other users are interspersed as masqueraders data. However, in this research, the data of the 50 users are used without these commands interspersed. Each user file contains 15000 commands.

In both cases, the data is drawn from *tcsh* history files and pre-processed to remove file names, user name, directory structures, etc. Command names, flags, and shell meta characters have been preserved. This analysis is only based on two fields: *Command name* and *User*. Thus, a user is identified by a set of commands concatenated by date order; for example the first 10 commands of the *User1* in the *50 Users set* are: *cpp, sh, xrdb, cpp, sh, xrdb, mkpts, env, csh, csh.*

### 4.1.1 Experiment Design

In order to measure the performance of the proposed classifier using the above data, we use 10-fold cross-validation. Thus, the commands typed by a user (training set) are divided into 10 disjoint subsets with equal size. Each of the 10 subsets is left out in turn for testing. The remaining 9 subsets, in each round, is used for training. The process is repeated 10 times, each time with a different training and testing sets. The average results over these 10 rounds are reported.

[2]Available from http://archive.ics.uci.edu/ml/datasets/UNIX+User+Data (UCI Machine Learning Repository)
[3]Available from the Schonlau web page: http://www.schonlau.net/intrusion.html

In each round the training sets for all users are used to create new models (as explained in Section 3.1). Each of the test sets is then classified using ABCD, using the statistical method explained in Section 3.2. The user is classified into the most similar distribution (lowest result of the comparisons).

The number of UNIX commands analyzed per user is very relevant for the result of the classification. Therefore, we have performed several experiments with different number of UNIX commands (50, 100, 500, 1000, 5000 and 10000) per user. These commands are selected from the last commands typed by a user. Also, in the phase of behavior model creation, the length of the sub-sequences in which the original sequence is segmented (used for creating the *trie*) is a relevant parameter: Using a longer length, the time consumed for creating the *trie* and the number of relevant sub-sequences in the corresponding distribution increase drastically. In the experiments presented in this paper, three different segmentation values for the sequence (sub-sequence lengths) are evaluated: 3, 5 and 10.

#### 4.1.2 Results

The UNIX command sequence (*Test Distribution*) is classified into the user behavior (*Training Distribution*) with the smallest deviation in the comparison process. Also, this process generates a ranked list with the most likely user at the top. There are users whose behavior is quite similar and the comparison result could be similar too. However, in the proposed experiments, the classification is correct only if the user who typed the sequence of commands to classify holds *the first position* of the ranking list. Thus this is a very conservative test.

The results are listed in Table 1. Each major row corresponds to a test-set size (from 50 commands, to 10,000). Each such row is further subdivided into experiments with different segment lengths (3, 5, and 10[4]). The columns show the average classification success and the standard deviation, for the 10-fold cross validation experiments, in the 9-user and 50-user data sets. Each cell therefore corresponds to the results of 10 runs, and overall, the table shows the results of 34 experiments, each consisting of 10 runs.

The results for the *Set of 9 Users* show that even with 50 commands (45 per training and 5 per testing), the classification rate is very high (around 80%). However, the results obtained with different sub-sequence lengths for creating the *trie* (3, 5 and 10) show that the higher classification rates are not obtained using a higher length. From the results for *Set of 50 Users*, we can see that the classification rate is smaller because of the large number of users to classify. In this case, this rate increases considerably with increasing the number of commands to analyze. Using 10000 commands (9000 for testing and 1000 for testing), the classification rate is close to 80%.

#### 4.1.3 *ABCD* vs *HMMs* in the UNIX Environment

To put these results in context, we compare this table to a similar table, in which the classification results were obtained using a standard HMM technique. Recent works have demonstrated the effectiveness of Hidden Markov Models (*HMM*s) for information extraction, in particular in classification of sequential data (see, e.g., [21, 33]).

We thus compare *ABCD* with a classifier based on HMMs. An HMM consists of a finite set of states, each of which is associated with a probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In each particular state, an observation can be generated, according to the associated probability distribution (it is only the

[4]The results for 50 commands using a sub-sequence length of 10 for creating the *trie* cannot be calculated because the testing file has only 5 commands (10%).

observation, not the state visible to an external observer). See [7, 36] for more details on HMMs.

To define a *HMM* completely, the following elements are need: (1) the number of states of the model, $N$; (2) the number of observations symbols in the alphabet, $M$; (3) a state transition probabilities matrix, $A$; (4) an observation probability distribution in each of the states, $B$; and (5) the prior state distribution, $\Pi$. To use this technique for classifying the behavior of UNIX users, a *HMM* is created for each user where the number of observations symbols is the number of different commands typed by the user. Also, in order to compare the obtained *ABCD* results to the *HMM* results, the number of states of a *HMM* corresponds with the length of the sub-sequences used to create the *trie*.

The toolkit UMDHMM [26] has been used to create each UNIX user behavior model from the corresponding training data files. Once the *HMMs* that represent the different UNIX user behaviors have been created, the *Forward Algorithm* is used to calculate the probability of an observed UNIX user sequence given a *HMM*. Finally, the sequence of commands is classified into the *HMM* with the highest likelihood.

Table 2 shows the results obtained using a classifier based on HMMs and using the same data than in the previous experiments. It follows the same structure as Table 1.

A careful comparison of the two tables reveals that the ABCD technique is superior to HMMs when used in smaller data-sets, i.e., when the number of examples is small. However, HMMs are superior to the ABCD when the data-set is very large (somewhere between 5000 and 10000 commands).

Figures 6 and 7 show the relevant differences between the classification rates obtained using both *ABCD* and *HMMs* for classifying UNIX users. It is remarkable the high classification rate obtained by *ABCD* with a low number of commands for training and classifying. For areas such as computer intrusion detection, this aspect is really important because the detection can be done when the user only have typed a few commands.

### 4.2 RoboCup Soccer Coach Simulation

To further challenge ABCD's scope of application, we further evaluated the use of ABCD in classifying simulated robot behavior, in the RoboCup soccer simulation. The simulated soccer environment is very different from the UNIX domain used above. The *Soccer Server System* [34] is a server-client system which simulates a soccer game, and has been used annually in the RoboCup soccer world-cup competitions. Software agents interact in a complex and noisy multi-agent environment. Eleven players (agents) can only perceive objects that are in their field of vision and both the visual information and the execution of the actions are noisy. Additionally, the server allows an agent to connect as a *coach* client [38] who has a global view of the world without noise, and whose only action is to send messages to the players while the ball is out of play. In particular, in this research we use the environment of the *RoboCup Coach Competition*. The first *Coach Competition* was held in 2001, but the goal of this *Competition* changed recently in order to emphasize opponent-modeling approaches.

#### 4.2.1 RoboCup Soccer Team Behavior Classification

The main goal in this environment is to classify the behavior of a soccer simulation opponent team by observing its actions (there is an underlying assumption that the behavior of the players does not vary significantly over the course of the game). In this domain, a team behavior consists of the sum of the behavior of its individual players: The team of agents cooperates to achieve a common goal. Therefore, the changes made to the environment are not the result of the behavior of a single

agent, but the interaction of the agent with each other and the world in which they act. As a consequence, the emergent behavior is usually hard to understand because the global behavior is not the sum of the local behaviors of the agents.

As in the previous environment, the observed behaviors are initially analyzed, and then its corresponding models are stored in the *BMLib*. After that, a new game is observed and the behavior models (from *BMLib*) followed by the team members must be recognized. The construction of models is done considering only the behavior followed by a few players (usually 1, 2 or 3 players), what we call *player behavior*. However, the behavior to classify is the sum of several *player behaviors*, what we call *team behavior*.

The construction of models is done by analyzing several game *log files* (*Game Training files*) in which different *player behaviors* are followed by a few players (a priori we do not know the players that follow the behavior). Then, it is observed a new game in which several *player behaviors* (usually 4 or 5) are activated at the same time (*team behavior*). The goal in this environment is to classify the game by reporting the *player behaviors* activated in the observed game.

**Construction of Behavior Models.** The first step in this process is to create a sequence of atomic behaviors from a given game. Kaminka et al. [25], and later Kuhlmann et al. [29] describe a procedure to identify high-level events in a soccer game (an event represents a recognized atomic behavior). Based on that work, in order to create the sequence of events that characterizes the behavior of a group of players (obtained from the *log file* game), two stages are used: features extraction and event recognition.

*Features extraction:* The important features over all the information of the game are processed from the *log files*. The necessary information to identify high-level events is extracted: *Cycle* (value that enables arranges the events), *Ball and Players position* (in the *Cartesian coordinate system*), *Ball Possessor* (value that indicates who the owner of the ball is).

*Event Recognition:* With the previous data, what events have occurred must be inferred. We follow [25, 29] in recognizing eight different events: *Pass, Dribble, Intercept pass, Steal, Goal, Missed shot, Foul* and *Hold*. Also, each event is characterized by the players that have executed the action and its team (*L*-Left or *R*-Right). In this work, we have used the same recognition of events, and the result of this phase is a sequence of events ordered by time. This sequence is called *behavior sequence* and it may look as follows: $\{Pass1to2(R) \rightarrow Dribble2(R) \rightarrow Steal3(L) \rightarrow ...\}$

After recognizing all the events of a game, the sequence is segmented considering only the actions performed by the team to analyze its behavior (opponent team). Therefore, in order to divide the *behavior sequence* into different sub-sequences, we consider as a sub-sequence the list of events executed by the opponent team while it is possessor of the ball. Using these sub-sequences from the different *log files*, the corresponding *tries* are created and the models obtained by using *ABCD*.

**Classification of an observed team.** In this phase, a new team is observed *online* by the coach agent. Then, the *team behavior* model is created and matched to the models of the *player behaviors* stored in the *BMLib* using the *ABCD* statistical methods. This process returns a ranked list with the most likely *player behavior* model at the beginning.

### 4.2.2 Results

For the experiments in this domain, we have used the rules from the *RoboCup 2006 Coach Competition* [1] and the experiments have been performed in the same way that this competition. The competition consists of 3 different rounds with different *player behaviors* to analyze and different

*team behaviors* to classify. We report here on the results of the first round obtained using *ABCD* for being the most representative.

In this first round, 17 different *player behaviors* are analyzed (available from [2]) and stored in *BMLib* by using a full program (*coach*) implemented based on *ABCD*. The games analyzed (*Training Files*) in this case are around 1000 to 3000 cycles games (in our case, the number of atomic behaviors identified for each game is usually around 150). Then, in each iteration of the round, a different game where four or five different *player behaviors* have been activated is observed (*team behavior*) by the *coach* which is connected to the Soccer Server. For these experiments, in order to recognize better the *player behaviors*, 3 games with the same *team behavior* (same *player behaviors* activated) have been observed. After observing the 3 games, a ranking list with the most likely *player behavior* is reported. For evaluating the result, we consider the order in the list of the *player behaviors* activated.

Table 3 shows the ranking list obtained for the 3 iterations of the first round. As it is indicated with the number in brackets, in the first round there are 4 *player behaviors* activated and 5 in the second and third iterations. The *player behaviors* are identified with a number (from pattern00 to pattern16) and in table 3 the *player behaviors* that have been activated are marked with an asterisk. As we can see, the result are very promising since in the five first places of the 3 rounds, there are 3 *player behaviors* that have been correctly identified as activated.

Analyzing these results, the *player behaviors* named *pattern04* and *pattern16* have been classified correctly in first position. Although the way to define these *player behaviors* is using a special language called *CLang* [12] (with which the behavior of the simulated soccer player can be modified), we describe these *player behaviors* as follows:

- ***Pattern04:*** *The players 6, 7 and 8 always pass the ball to a specific point in the field.*
- ***Pattern16:*** *Player 1 always pass to player 3 or 4. Player 3 and 4 always dribble to a fix space.*

However, the following *player behaviors* are not recognized correctly:

- ***Pattern14:*** *The player 3 dribbles to the space where the player 5 is situated. The player 5 dribbles to the space where the player 9 is situated. The player 9 dribbles to the space where the player 3 is situated.*
- ***Pattern08:*** *If the ball is situated in a defined area, player 8 dribbles to a fix space. Otherwise, player 8 passes to player 0.*

As we can see from the results, *ABCD* works successfully when the *player behavior* to recognize is related to the actions of the players. Other types of *player behavior* (related to the different field regions in which the action occurs or the cycle when it occurs) could not be detected. Although there are *player behaviors* that are not related to actions, some of them are recognized for the way they play.

## 5 Conclusions and Future Works

This paper presents a novel approach for modeling and classifying behaviors from observation (called *ABCD* - Agent Behavior Classification based on Distributions of relevant events). The underlying assumption in this approach is that the observed behavior can be transformed into a sequence of ordered atomic behaviors. If this transformation can not be done or the defined events

do not capture the observed behavior properly, the proposed approach is not useful. The obtained sequence is segmented and stored in a *trie* and then the relevant sub-sequences are evaluated by using a frequency-based method. The main aspect in *ABCD* is that the model of an agent behavior is represented by a distribution of relevant sub-sequences. Finally, for classifying a given behavior, the *Chi-square Test* for two samples is proposed.

Also, an important aim in this work is to provide a *general* approach which can represent and handle different behaviors in a wide range of domains. Therefore, *ABCD* is generalizable to modeling and classifying behaviors represented by a sequence of events (such as GUI events, network packet traffic and so on). In order to demonstrate this generalization, *ABCD* performance has been experimentally evaluated in two very different domains: *UNIX User Classification* and *RoboCup Soccer Coach Simulation*. A large set of experiments were conducted in both domains.

The experiments show that a system based on *ABCD* is very effective for classifying a UNIX user, even with a very limited number of training examples, and testing data. For areas such as computer intrusion detection, these results are very encouraging. On the other hand, HMMs proved superior to ABCD in this domain, when the number of examples was 2-3 orders of magnitude larger.

In the real-time multi-agent domain of *RoboCup Soccer Coach Simulation*, the results of using *ABCD* are very satisfactory (similar or even better than those obtained by the *RoboCup 2006 Coach Competition* champion). In this environment, a correct and rapid classification of the opponent can be very advantageous. However, these results are dependent on the events defined. When such events do not capture salient information of some agents, their behavior was unrecognizable.

An important aspect that has not been tackled in this paper is to consider that many agents change their behavior and their preferences over time, which means that their models should be frequently revised to keep it up to date. This aspect could be solved by using *Evolving Systems* [5] and it is proposed for future work. Also, to use the result of the classification for carrying out effective actions in the environment (implementation of the *Reasoning Module*) will be considered in our future work.

## References


[1] RoboCup Committee. RoboCup 2006 Official Rules for the Competition (http://ce.sharif.edu/ m_sedaghat/ robocup/orga/rc06/rules0.0.pdf), 2006.

[2] The RoboCup 2006 Coach Competition Web Page (http://ssil.uni-koblenz.de/rc06/), 2006.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering (ICDE-95)*, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[4] J. Anderson. *Learning and Memory: An Integrated Approach*. John Wiley and Sons., New York, 1995.

[5] P. Angelov. *Evolving Rule-based Models: A Tool for Design of Flexible Adaptive Systems*. Springer-Verlag, Heidelberg, New York, 2002.

[6] D. Avrahami-Zilberbrand and G. A. Kaminka. Incorporating observer biases in keyhole plan recognition (efficiently!). In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*, pages 944–949. AAAI Press, 2007.

[7] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[8] H. Bui. A general model for online probabilistic plan recognition. In G. Gottlob and T. Walsh, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1309–1318. Morgan Kaufmann, 2003.

[9] D. Carmel and S. Markovitch. Opponent modeling in multi-agent systems. In *Proceedings of the IJCAI-95 Workshop on Adaption and Learning in Multi-Agent Systems*, pages 40–52. Springer-Verlag, London, UK, 1996.

[10] S. Carrbery. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[11] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[12] M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. Soccerserver manual ver. 7, 2002.

[13] G. Chirn, J. T. Wang, and Z. Wang. Scientific data classification: A case study. In *Proceedings of the Ninth International Conference on Tools with Artificial Intelligence (ICTAI-97)*, page 216, Washington, DC, USA, 1997. IEEE Computer Society.

[14] S. E. Coull, J. W. Branch, B. K. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC-03)*, pages 24–33, 2003.

[15] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[16] T. Duong, H. H. Bui, D. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov models. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR-2005)*, San Diego, CA, 2005. IEEE Computer Society.

[17] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[18] C. L. Giles and M. Gori, editors. *Adaptive Processing of Sequences and Data Structures*, volume 1387 of *Lecture Notes in Computer Science*. Springer, 1998.

[19] R. Goldman, C. Geib, and C. Miller. A new model of plan recognition. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 245–254, San Francisco, CA, 1999. Morgan Kaufmann.

[20] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In J. Hollerbach and D. Koditschek, editors, *Proceedings of the IJCAI-99 Workshop on Team Behavior and Plan-Recognition*, pages 199–204, London, 1999. Springer-Verlag. Also appears in Proceedings of the 9th International Symposium of Robotics Research (ISSR-99).

[21] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of IJCAI-99 Workshop on Team Behaviors and Plan Recognition*, 1999.

[22] J. Hong. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, 15:1–30, 2001.

[23] Y. Horman and G. A. Kaminka. Removing biases in unsupervised learning of sequential patterns. *Intelligent Data Analysis*, 11(5):457–480, 2007.

[24] Z. Huang, Y. Yang, and X. Chen. An approach to plan recognition and retrieval for multi-agent systems. In *Proceedings of the AORC-2003 Workshop on Adaptability in Multi-Agent Systems*, 2003.

[25] G. A. Kaminka, M. Fidanboylu, A. Chang, and M. Veloso. Learning the sequential coordinated behavior of teams from observations. In *Proceedings of the Robot Soccer World Cup VI (RoboCup-2002)*, volume 2752 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2002.

[26] T. Kanungo. UMDHMM: A hidden markov model toolkit. In *Proceedings of the Extended Finite State Models of Language*. Cambridge University Press, 1999.

[27] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 32–37. AAAI press, 1986.

[28] D. Knuth. *The Art of Computer Programming, Vol. 3.* Addison-Wesley, Reading, 1973.

[29] G. Kuhlmann, P. Stone, and J. Lallinger. The UT Austin Villa 2003 Champion Simulator Coach: A Machine Learning Approach. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *Proceedings of the Robot Soccer World Cup VII (RoboCup-2003)*, volume 3020 of *Lecture Notes in Computer Science*, pages 636–644, Berlin, 2004. Springer Verlag.

[30] J. E. Laird. It knows what you're going to do: adding anticipation to a quakebot. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 385–392, Montreal, Canada, 2001.

[31] A. Ledezma, R. Aler, A. Sanchis, and D. Borrajo. Predicting opponent actions by observation. In *Proceedings of the Robot Soccer World Cup VIII (RoboCup-2004)*, volume 3276 of *Lecture Notes in Computer Science*, pages 286–296. Springer, 2004.

[32] Q. Ma, J. T. Wang, D. Shasha, and C. H. Wu. DNA sequence classification via an expectation maximization algorithm and neural networks: a case study. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 31(4):468–475, 2001.

[33] E. Marhasev, M. Hadad, G. A. Kaminka, and U. Feintuch. The use of hidden semi-markov models in clinical diagnosis maze tasks. *Intelligent Data Analysis*, 13(6):To Appear, 2009.

[34] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: a tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2):233–258, 1998.

[35] A. Quilici, Q. Yang, and S. Woods. Applying plan recognition algorithms to program understanding. *Automated Software Engineering*, 5(3):347–372, 1998.

[36] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[37] P. Riley and M. Veloso. On behavior classification in adversarial environments. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Proceedings of the Distributed Autonomous Robotic Systems 4 (DARS-2000)*, pages 371–380. Springer-Verlag, 2000.

[38] P. Riley and M. Veloso. Coaching a simulated soccer team by opponent model recognition. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, pages 155–156. ACM, 2001.

[39] M. Schonlau, W. DuMouchel, W. Ju, A. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, 2001.

[40] T. Steffens. Feature-based declarative opponent-modelling in multi-agent systems. Master's thesis, Institute of Cognitive Science Osnabrueck, 2002.

[41] R. Sun, E. Merrill, and T. Peterson. From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, 25(2):203–244, 2001.

[42] M. Tambe and P. S. Rosenbloom. Resc: An approach for dynamic, real-time agent tracking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[43] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41, New York, NY, USA, 2002. ACM.

[44] U. Visser and H. G. Weland. Using online learning to analyze the opponent's behavior. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *Proceedings of the Robot Soccer World Cup VI (RoboCup-2002)*, volume 2752 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2002.

Table 1.Classification Results using ABCD. 9 and 50 Users.

| | | **ABCD Classifier Results** | | | |
|---|---|---|---|---|---|
| | | **Set of 9 UNIX Users** | | **Set of 50 UNIX Users** | |
| Number of commands | Subseq Length | Classification rate % | Standard Deviation | Classification rate % | Standard Deviation |
| 50 | 3 | 80,00 | 1,40 | 48,20 | 8,99 |
| | 5 | 78,89 | 1,34 | 48,80 | 7,73 |
| 100 | 3 | 80,00 | 0,96 | 53,40 | 8,42 |
| | 5 | 76,67 | 1,08 | 51,40 | 9,81 |
| | 10 | 78,89 | 0,83 | 54,80 | 6,99 |
| 500 | 3 | 90,00 | 1,08 | 64,00 | 9,16 |
| | 5 | 91,11 | 1,27 | 64,20 | 10,17 |
| | 10 | 86,67 | 1,49 | 63,80 | 12,48 |
| 1000 | 3 | 87,78 | 1,53 | 72,00 | 10,14 |
| | 5 | 87,78 | 1,30 | 71,20 | 10,49 |
| | 10 | 81,11 | 1,84 | 69,00 | 11,69 |
| 5000 | 3 | 85,56 | 1,23 | 75,80 | 12,05 |
| | 5 | 87,78 | 1,30 | 76,60 | 12,26 |
| | 10 | 84,40 | 1,54 | 75,00 | 12,64 |
| 10000 | 3 | 88,87 | 1,53 | 76,20 | 12,14 |
| | 5 | 90,00 | 1,40 | 78,80 | 13,14 |
| | 10 | 91,40 | 1,66 | 79,00 | 13,39 |

Table 2.Classification Results using HMMs. 9 and 50 Users.

| | | **HMMs Classifier Results** | | | |
|---|---|---|---|---|---|
| | | **Set of 9 UNIX Users** | | **Set of 50 UNIX Users** | |
| Number of commands | Subseq Length | Classification rate % | Standard Deviation | Classification rate % | Standard Deviation |
| 50 | 3 | 52,22 | 2,23 | 30,40 | 14,08 |
| | 5 | 54,44 | 2,06 | 32,40 | 14,72 |
| | 10 | 54,44 | 2,08 | 34,80 | 15,02 |
| 100 | 3 | 64,44 | 1,49 | 39,40 | 8,72 |
| | 5 | 61,11 | 1,53 | 40,00 | 8,58 |
| | 10 | 62,22 | 1,60 | 40,40 | 8,94 |
| 500 | 3 | 63,33 | 1,22 | 42,20 | 6,19 |
| | 5 | 68,89 | 1,30 | 48,20 | 6,03 |
| | 10 | 66,67 | 1,26 | 51,20 | 5,86 |
| 1000 | 3 | 63,33 | 1,20 | 46,20 | 4,69 |
| | 5 | 68,89 | 1,32 | 49,20 | 4,55 |
| | 10 | 66,67 | 1,09 | 53,20 | 4,47 |
| 5000 | 3 | 80,00 | 1,05 | 54,20 | 3,89 |
| | 5 | 82,22 | 0,90 | 58,20 | 3,53 |
| | 10 | 88,89 | 0,97 | 62,20 | 3,45 |
| 10000 | 3 | 89,90 | 1,22 | 76,40 | 3,35 |
| | 5 | 93,11 | 1,32 | 78,80 | 3,41 |
| | 10 | 93,32 | 0,97 | 80,20 | 3,29 |

Table 3.Results for the *RoboCup Coach Competition*. Round1

| *Round1-Iter1* (4) | *Round1-Iter2* (5) | *Round1-Iter3* (5) |
|---|---|---|
| ***pattern04 (*)*** | ***pattern16 (*)*** | ***pattern04 (*)*** |
| pattern16 | ***pattern01 (*)*** | ***pattern02 (*)*** |
| ***pattern00 (*)*** | pattern00 | pattern13 |
| pattern12 | ***pattern13 (*)*** | pattern05 |
| ***pattern15 (*)*** | pattern05 | ***pattern12 (*)*** |
| pattern03 | pattern09 | ***pattern00 (*)*** |
| pattern09 | ***pattern07(*)*** | pattern01 |
| pattern05 | pattern03 | ***pattern06 (*)*** |
| pattern01 | pattern10 | pattern03 |
| pattern06 | ***pattern08(*)*** | pattern10 |
| pattern08 | pattern16 | pattern07 |
| pattern13 | pattern15 | pattern16 |
| pattern10 | pattern02 | pattern11 |
| pattern11 | pattern12 | pattern08 |
| ***pattern14 (*)*** | pattern04 | pattern15 |
| pattern02 | pattern11 | pattern09 |
| pattern07 | pattern14 | pattern14 |

**Figure captions:**

Fig. 1: Agent Behavior Classification based on Distributions of relevant events ($ABCD$): Framework. The process "Construction of Behavior Models" is described in Section 3.1. The "Behavior Classification" process is described in Section 3.2.

Fig. 2: Steps of creating an example trie.

Fig. 3: Distribution of sub-sequences.

Fig. 4: Agent Behavior Classification Process.

Fig. 5: Observed and Expected Comparison Example.

Fig. 6: ABCD vs HMMs. Set of 9 Users.

Fig. 7: ABCD vs HMMs. Set of 50 Users.

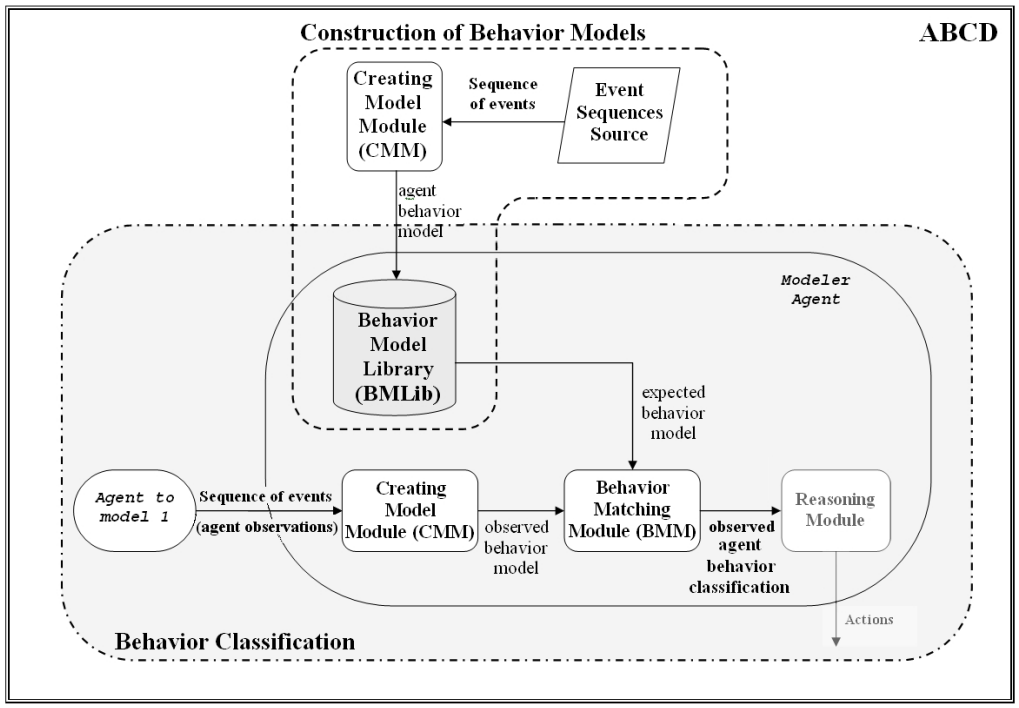
Construction of Behavior Models
ABCD
Creating Model Module (CMM)
Sequence of events
Event Sequences Source
agent behavior model
Modeler Agent
Behavior Model Library (BMLib)
expected behavior model
Agent to model 1
Sequence of events (agent observations)
Creating Model Module (CMM)
observed behavior model
Behavior Matching Module (BMM)
observed agent behavior classification
Reasoning Module
Actions
Behavior Classification

Fig. 1.

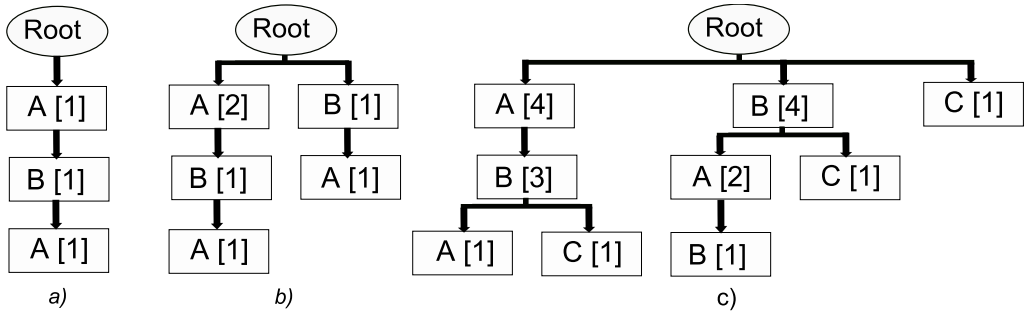


Fig. 2.

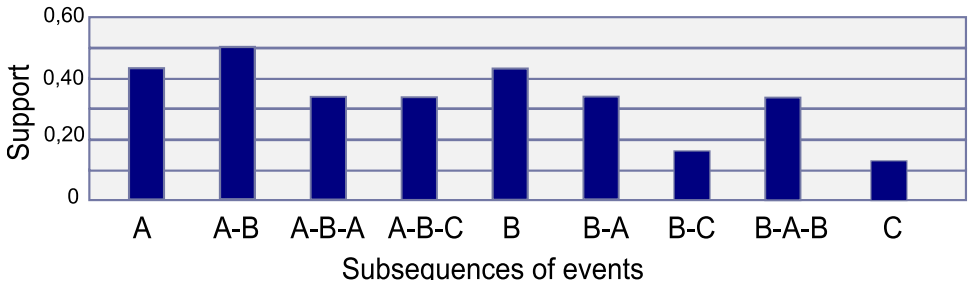
0,60
0,40
0,20
0
Support
A
A-B
A-B-A
A-B-C
B
B-A
B-C
B-A-B
C
Subsequences of events

Fig. 3.

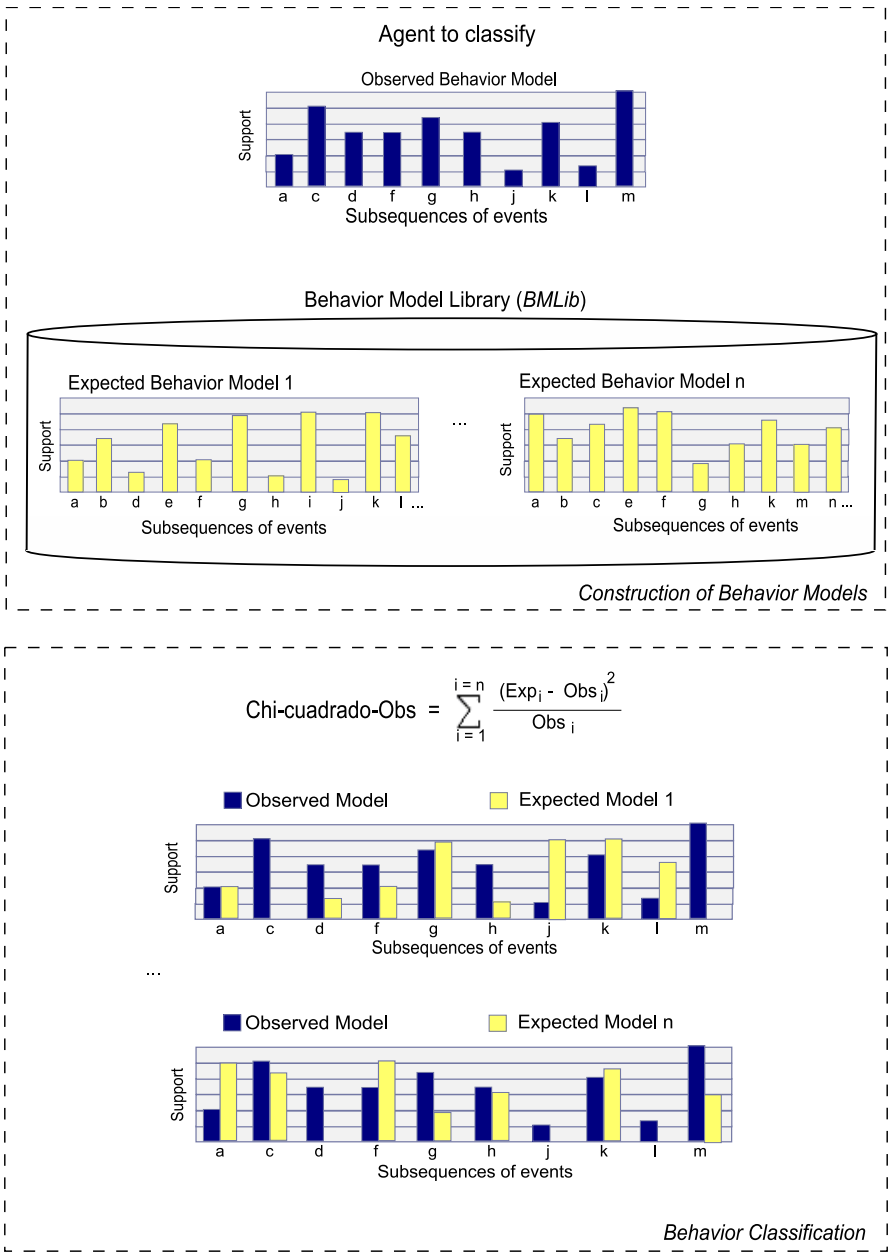
Agent to classify
Observed Behavior Model
Support
a c d f g h j k l m
Subsequences of events
Behavior Model Library (BMLib)
Expected Behavior Model 1
Support
a b d e f g h i j k l ...
Subsequences of events
...
Expected Behavior Model n
Support
a b c e f g h k m n ...
Subsequences of events
Construction of Behavior Models
Chi-cuadrado-Obs = $\sum_{i=1}^{i=n} \frac{(Exp_i - Obs_i)^2}{Obs_i}$
Observed Model
Expected Model 1
Support
a c d f g h j k l m
Subsequences of events
...
Observed Model
Expected Model n
Support
a c d f g h j k l m
Subsequences of events
Behavior Classification

Fig. 4.

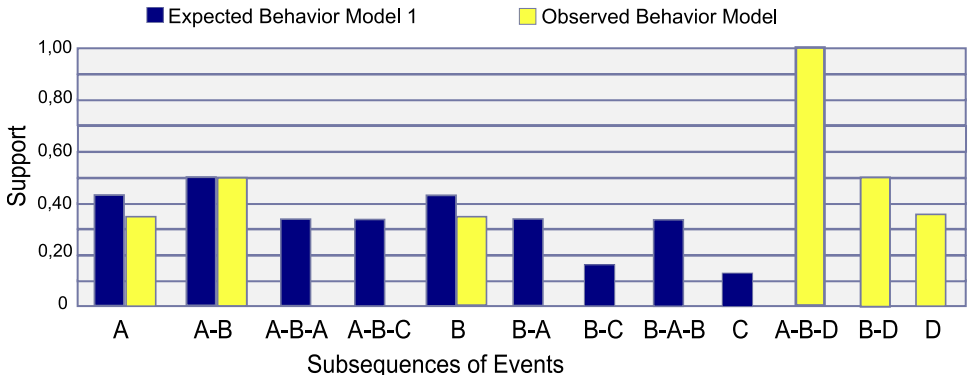
Expected Behavior Model 1
Observed Behavior Model
1,00
0,80
0,60
0,40
0,20
0
Support
A
A-B
A-B-A
A-B-C
B
B-A
B-C
B-A-B
C
A-B-D
B-D
D
Subsequences of Events

Fig. 5.

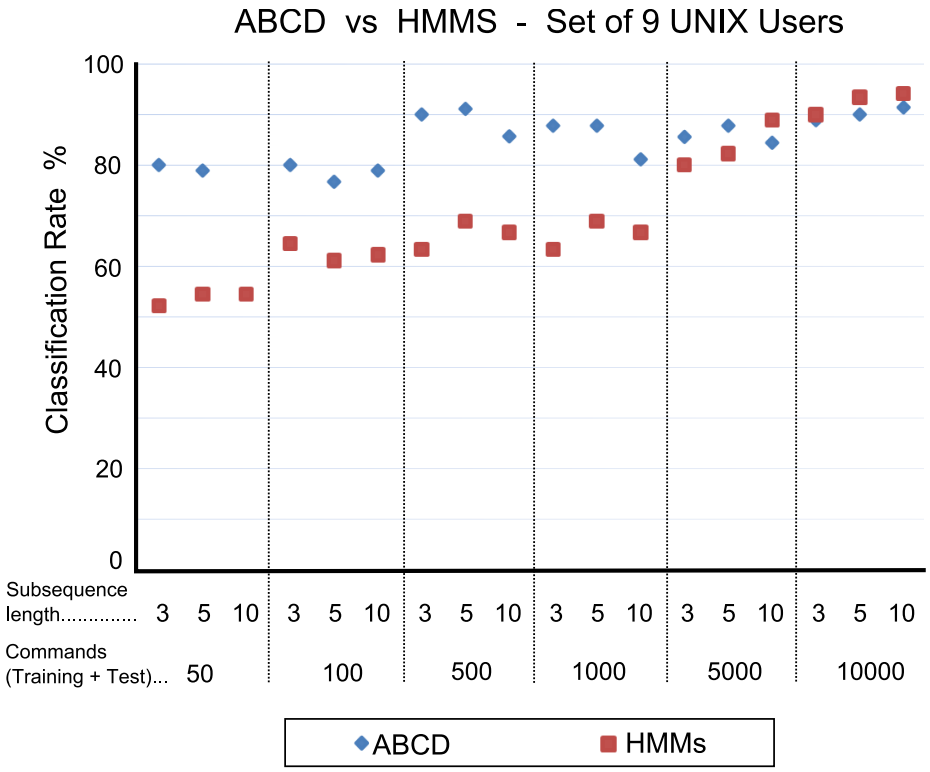
ABCD vs HMMS - Set of 9 UNIX Users
Classification Rate %
100
80
60
40
20
0
Subsequence length.....
3 5 10 3 5 10 3 5 10 3 5 10 3 5 10 3 5 10
Commands (Training + Test)...
50
100
500
1000
5000
10000
ABCD
HMMs

Fig. 6.

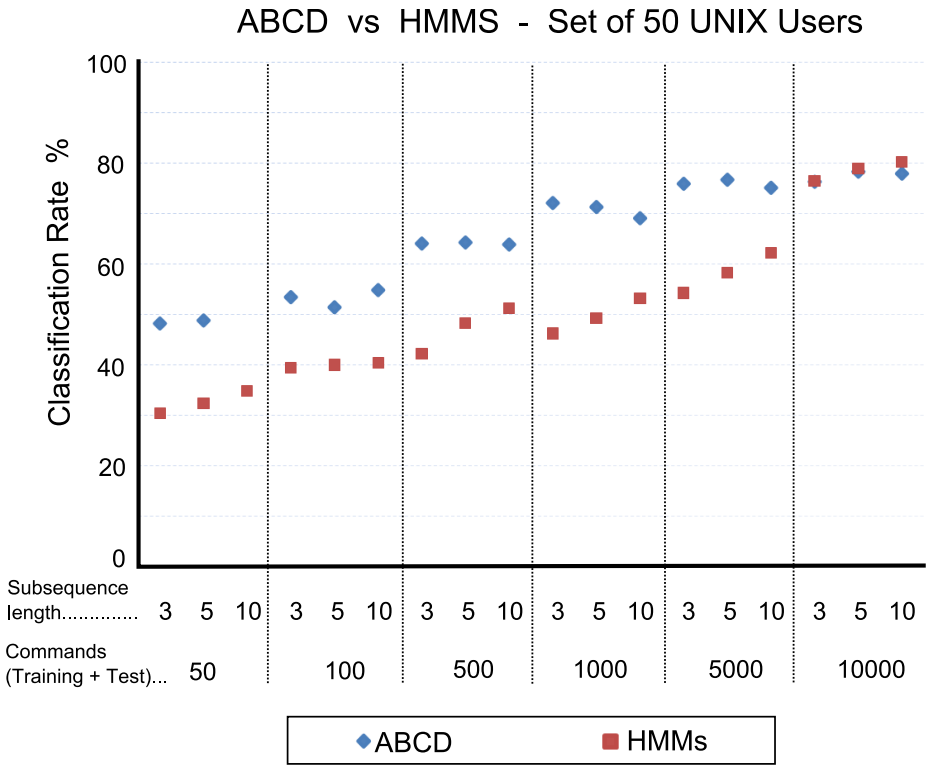
ABCD vs HMMS - Set of 50 UNIX Users
Classification Rate %
100
80
60
40
20
0
Subsequence length.....
3 5 10 3 5 10 3 5 10 3 5 10 3 5 10 3 5 10
Commands (Training + Test)...
50
100
500
1000
5000
10000
ABCD
HMMs

Fig. 7.