

# Learning Trust in Dynamic Multiagent Environments using HMMs

Marie Elisabeth Gaup Moe, Mozghan Tavakolifard and Svein J. Knapskog

Centre for Quantifiable Quality of Service in Communication Systems<sup>1</sup>,

Norwegian University of Science and Technology,

marieeli,mozghan,knapskog@Q2S.ntnu.no

## Abstract

In open multiagent systems, agents are owned by a variety of stakeholders and can enter and leave the system at any time. Therefore, trust is a fundamental concern in effective interactions which is a key component of such systems. In this paper, we propose a trust model for autonomous agents in multiagent environments based on hidden Markov models and reinforcement learning. By this combination, the reliability of the hidden Markov model will be improved since its parameters are re-estimated after training of the model with the reinforcement learning module.

## 1 Introduction

The rapidly changing environments of the Internet suffer from problems related to fragile trustworthiness of its millions of active entities, which can be humans or mobile agents. This problem is nontrivial, as more and more commercial transactions get carried out over the Internet. Therefore, devising an effective approach for verification of trustworthiness in such complex environments is essential, since trust mechanisms play a key role in the security of the entities. Also the trust establishment is nontrivial, since the traditional and social means of trust cannot be applied directly to the virtual settings of these environments, because in many cases the involved parties did not have any previous interaction. In such scenarios, trust management techniques may be used to stimulate service quality and acceptable user behavior in online markets and communities, and also sanction possible unacceptable user behavior.

Application of autonomous agents in large-scale open distributed systems presents a number of new challenges such as:

- Agents with different characteristics can enter the system and interact with one another.
- Each agent tries to maximize its individual utility because it represents a specific stakeholder with various objectives.
- Agents may change their identities on re-entering the system to avoid punishment for any past wrong doing.
- Agents should decide how, when, and with whom to interact without any guarantees that the interaction will actually achieve the desired benefits.

---

<sup>1</sup>“Centre for Quantifiable Quality of Service in Communication Systems  
Centre of Excellence” appointed by The Research Council of Norway,  
funded by the Research Council, NTNU and UNINETT.  
<http://www.q2s.ntnu.no>

Agents are faced with significant degrees of uncertainty in making decisions since it is impossible to obtain perfect information about the environment and the interaction partners properties. In such circumstances, agents have to establish appropriate trust in each other in order to minimize the impact of the uncertainty associated with interactions [10].

The goal of an agent in a dynamic environment is to make optimal trust decisions over time. Learning trust serves such a purpose by biasing the agent's action choices through information gathered over time. An agent can base its action choice on prediction of the other agents' behaviors or directly on the reward (the outcome of the interaction) received from them. Reinforcement learning is a systematic method that associates an agent's action with its rewards.

In reinforcement learning, an agent need not explicitly model other agents since its action can be directly based on the rewards. Thus this learning method is particularly useful for cases where agents have little knowledge about each other. An agent in a multiagent system may know little about others because information is distributed. Even when an agent has some prior information about others, the behavior of others may change over time. It is therefore natural to apply a learning algorithm.

Our model consists of trust estimation and trust learning modules. The former and latter are constructed from *Hidden Markov Models* (HMM) and *Reinforcement Learning (RL)*, respectively. The model parameters of the HMM are re-estimated after having learnt about its environment from the reinforcement learning module. The proposed method enables us to improve the model reliability when dealing with a dynamic environment that changes over time.

## 2 Related Work

In [5] a trust model using a Markov model is proposed. In this work the Markov chain is defined as the chain of aggregated reputation values corresponding to a sequence of consecutive time slots. The current state vector shows the reputé value of the reputation queried at time slot  $N$ . The Markov matrix of a given agent denotes the probability of that agent transiting from one trustworthiness level to another trustworthiness level based on its past behavior captured using the Markov chain. In order to determine the probability of an agent transiting from trustworthiness level  $A$  to trustworthiness level  $B$ , based on the Markov chain, they use the ratio of the number of times that agent has transited from trustworthiness level  $A$  to trustworthiness level  $B$  to that of the total number of time that the agent has transited from trustworthiness level  $A$  to any other trustworthiness level. The future state vector of the agent is determined by multiplying the current state vector with the Markov matrix. The same authors also proposed a method for determining the effectiveness of their Markov model for predicting the future trustworthiness value of a given agent by utilizing simulation methods [6]. This paper presents the simulation method that they employed in order to determine the effectiveness of the Markov model in detail.

In [12], the authors compare the effectiveness of probabilistic computational trust systems. They conclude that most probabilistic trust models are unrealistic, as the models allows for no dynamic behavior, and outline the idea of a trust model based on a hidden Markov model to cope with this problem. In [13], the authors have developed a hidden Markov model based approach to measuring an agent's reputation as a recommender. This approach models chained recommendation events as an HMM. The features of the trust model are: (1) no explicit requirement of chained recommendation reputations; (2) flexible recommendation network with presence of loops; and (3) integration of learning speed into trust evaluation reliability. In [7] an architecture for trust management in ubiquitous environments that deals with digital signatures and user presence in a uniform framework is proposed. This architecture includes inferences about user presence from incomplete sensor signals based on an HMM.

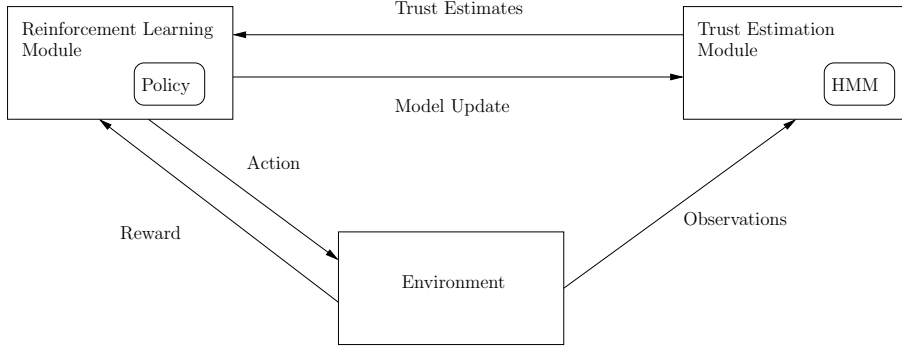


Figure 1: Architecture of the proposed trust model

### 3 The Proposed Model

Our model consists of trust estimation and trust learning modules constructed from *Hidden Markov Models* (HMM) and *Reinforcement Learning* (RL) as depicted in Figure 1. The model parameters of the HMM are re-estimated after having learnt about its environment from the reinforcement learning module. The proposed method enables us to improve the model reliability when dealing with a dynamic environment that changes over time. Similar approaches combining reinforcement learning and HMMs applied to motion recognition can be found in [4] and [2].

In the following sections we will start with describing some assumptions and limitations of this work, thereafter we present the stochastic modeling approach and details of the hidden Markov modeling.

#### 3.1 Model States

An agent can be in a *trusted*, *neutral* or *untrusted* state at any given time. An agent is in an untrusted state if it has been behaving in a malicious way in previous interactions, it is in a trusted state if it has shown good behavior. If its behavior has been a mixture of good and bad, or if it has not yet given any signs of its behavior, it is in the neutral state.

When an agent joins the system it is most likely in a neutral state, but there is also a small probability that it is in the untrusted state. Since it has not yet interacted with any of the other agents, no positive observations of its behavior have been made, so it can not be in a trusted state.

After joining the system the agent has a probability of becoming trusted or untrusted as it is interacting with other agents, depending on its behavior. We model trust as a dynamic variable, changing with time. This allow us to capture the behavioral characteristics of agents that are behaving good for a certain time, but then suddenly start misbehaving.

#### 3.2 Modeling of Agent Trustworthiness

In this section we will present an HMM for the trust relationship of agents in a multiagent system.

We model the agent interaction as a stochastic process. This means that we assume that there is a random time interval between each agent interaction and that the behavior of a node is only dependent on the current state of the node. The *state* of an agent can be characterized by whether or not it is behaving in a malicious manner in its interactions with other agents.

When using a continuous time Markov model to model the state of an agent, we make the following assumptions; all information about the agent is contained in the state, observations are independent given the current state, and state occupation time is negatively exponentially distributed.

When agents are interacting, an agent makes its opinion about the trustworthiness of the other agent based on the outcome of the interaction. After a random time interval these two agents meet again, and based on their belief about the other agent's trustworthiness they may decide whether or not to make an interaction. Since an agent's behavior can be changing with time it is not necessarily the case that an agent is in the same state as it were at the last encounter. An agent can only do its best guessing about the trustworthiness state of an other agent based on its own previous direct experiences with said agent and *recommendations* from other agents in the system. This means that the system state is hidden, and hence we use the HMM approach.

The system we consider is a multiagent system and we want to use the model to estimate the behavior of each single agent. An agent in the system rates all of the other agents after an interaction and uses an HMM per agent to decide and predict whether or not another agent is malicious. The HMM is updated from observations, that is the ratings after direct experiences or recommendations requested from other agents.

An HMM consists of a finite set of  $N$  hidden states  $S = \{s_1, \dots, s_N\}$  with an associated probability distribution. The state of the monitored agent is described by a discrete time Markov chain  $\mathbf{x}_k = x_1, x_2, \dots$  where  $x_k \in S$  is the possibly hidden state of the node at sampling instant  $k$ .  $\mathbf{P}_k = \{p_{ij}^k\}$  is the set of state transition probabilities,  $p_{ij}^k = P(x_{k+1} = s_j \mid x_k = s_i)$ ,  $1 \leq i, j \leq N$ , where  $x_k$  is the current state of the system.  $\pi = \{\pi_i\}$  is the initial state distribution, where  $\pi_i = P(x_1 = s_i)$ ,  $1 \leq i \leq N$ . The output from the agent ratings is classified by the set of observation symbols  $V = \{v_1, \dots, v_M\}$ . Let  $\mathbf{y}_k = y_1, y_2, \dots$  denote the sequence of observations, where  $y_k \in V$  is the observation made at sampling instant  $k$ . The HMM consists of two stochastic processes; the hidden process  $\mathbf{x}_k$ , and the observable process  $\mathbf{y}_k$  that depends on  $\mathbf{x}_k$ . The relation between  $\mathbf{x}_k$  and  $\mathbf{y}_k$  is described by the probability distribution matrix  $\mathbf{B} = \{b_j(m)\}$ , where  $b_j(m) = P(y_k = v_m \mid x_k = s_j)$ , for  $1 \leq j \leq N$ ,  $1 \leq m \leq M$ . See for instance [8] for a more extensive introduction to HMMs.

In our model we define three states, that can be characterized by the behavior of the agent, thus  $N = 3$  and each individual state is denoted  $S = \{s_1, s_2, s_3\}$ . The first state is the *trusted* state  $s_1$ , where the agent is not showing any malicious behavior, the second state is the *neutral* state  $s_2$ , where the outcome of interactions can be ambiguous, the third state is the *untrusted* state  $s_3$  where the agent is showing malicious behavior.

We have not made any assumptions about time between observations, and there is no direct relation between observations and state-changes. As a consequence the system could have made zero, one or more transitions during the time between to successive observations.

The time when observation number  $k$  is produced is denoted  $t_k$ . Time between observation  $k - 1$  and observation  $k$  is denoted  $\delta_k = t_k - t_{k-1}$ .

### 3.3 State Probability Distribution

The transition rate matrix  $\Lambda = (\lambda_{ij})$  is describing the dynamics of the system. To simplify the notation of equations and algorithms we will use  $i$  and  $j$  instead of  $s_i$  and  $s_j$ . The relation between system states and the transition rates is given by

$$\lambda_{ij} = \begin{cases} \lim_{dt \rightarrow 0} \frac{P(\mathbf{x}(t+dt) = j \mid \mathbf{x}(t) = i)}{dt} & \text{if } i \neq j \\ \sum_{j \neq i, j=1}^N -\lambda_{ij} & \text{if } i = j \end{cases}. \quad (1)$$

Since observations are received at irregular intervals, the running transition probabilities  $p_{ij}^k = P(x(t + \delta_k) = j | x(t) = i)$  depend on the time since last observation  $\delta_k$ , and have to be calculated each time an observation is received. The running transition probability matrix  $\mathbf{P}_k = (p_{ij}^k)$  can be derived from Kolmogorov's equations [11] as follows

$$\mathbf{P}_k = e^{\Lambda \delta_k}. \quad (2)$$

For large state spaces this calculation can be quite expensive, but in our case the state space is small, and the calculations inexpensive. Let  $\gamma_k = (\gamma_k(i))$  denote the state probability distribution at time  $t_k$  given all observations received until time  $t_k$ ,  $\gamma_k(i) = P(x_k = i | \mathbf{y}_k)$  where  $\mathbf{y}_k = y_1, \dots, y_k$ . We will use ten observation symbols  $V = \{v_1, v_2, \dots, v_{10}\}$ , where the first five symbols are the ratings an agent make after a direct interaction and the last five symbols are ratings received as recommendations from other agents. The ratings are given in the form of trustworthiness values ranging from 1 to 5, where 1 corresponds to "very trustworthy", 2 to "trustworthy", 3 to "moderate", 4 to "untrustworthy", and 5 to "very untrustworthy". Algorithm 1 is used to update the current state distribution  $\gamma_{k-1}$ , based on the following inputs:  $k$  the observation index,  $\gamma = \gamma_{k-1}$  the current state distribution,  $y = y_k$  the current observation, and  $\delta = \delta_k$  the time between the current observation and last observation. In addition to the dynamic variables listed above, the following parameters are assumed to be available for the algorithm: the transition rates  $\Lambda$ , the initial state distribution  $\pi$ , and the two observation probability matrices  $\mathbf{B}^\psi$ , where  $\mathbf{B}^1$  is used for the direct observations  $v_1, \dots, v_5$ , and  $\mathbf{B}^2$  is used for the implicit observations in the form of recommendations  $v_6, \dots, v_{10}$ .

---

**Algorithm 1** Update state probability distribution

---

**Require:**  $k, \psi, \gamma, y, \delta$   
 $\mathbf{P}_k \leftarrow e^{\Lambda \delta}$   
 $\mathbf{B} \leftarrow \mathbf{B}^\psi$   
**if**  $k = 1$  **then**  
    **for**  $i = 1$  to  $N$  **do**  
         $\alpha(i) \leftarrow b_i(y) \pi_i$   
         $\gamma(i) \leftarrow \frac{b_i(y) \pi_i}{\sum_{j=1}^N b_j(y) \pi_j}$   
    **end for**  
**else**  
    **for**  $i = 1$  to  $N$  **do**  
         $\alpha(i) \leftarrow b_i(y) \sum_{j=1}^N \gamma(j) p_{ji}^k$   
    **end for**  
    **for**  $i = 1$  to  $N$  **do**  
         $\gamma(i) \leftarrow \frac{\alpha(i)}{\sum_{j=1}^N \alpha(j)}$   
    **end for**  
**end if**  
**return**  $\gamma$

---

Algorithm 1 was originally proposed in [3], it is based on dynamic programming and uses a set of temporary variables. During the processing of observation  $y_k$  the value stored in  $\alpha(i)$  represents the following probability  $\alpha(i) = P(\mathbf{y}_k, x_k = s_i)$ , also known as the *forward variable*. By using dynamic programming in the estimation of  $\gamma$ , the complexity of an update is reduced from  $O(2kN^k)$  for a straight forward calculation, to  $O(N^2)$ . Scaling of the  $\alpha(i)$  is used in order to prevent problems related to underflow, for more details see the forward-backward procedure described in [8]. It should be noted that Algorithm 1 is an on-line algorithm and very efficient,

it does not require the agents to keep any history of past observations in memory. The history of observations and the values of some of the running variables are, however, required for the more computationally expensive re-estimation of model parameters as explained later.

## 4 Learning of Model Parameters

The parameters of an HMM are usually set by offline training of the model with a large data set. Since we want the model to reflect the more realistic dynamic behavior of the multiagent system and also optimize the agents' trust-related behavior, we will use an online learning of the HMM parameters with *reinforcement learning*. Reinforcement learning (RL) [14] is a machine learning technique for solving decision problems of mapping actions to states based on interactions with the environment. The actions of an agent in the multiagent system could for instance be whether or not it should interact with another agent, based on its belief about the state of the other agent derived from the HMM. Such a mapping from state to action is called a *policy*. In RL the agents will learn policies based on feedback from the environment that is calculated based on a *reward function*.

A simple reward function for the multiagent trust model can be defined as follows:

1. If an interaction was made, and the agent's rating of the other agent's behavior was given the values 1, 2 or 3, a positive reward is given.
2. If an interaction was made, and the agent's rating of the other agent's behavior was given the values 4 or 5, a negative reward is given.
3. If no interaction was made, a zero reward is given.

The RL framework also includes a *value function*  $Q(s, a)$  which estimates the reward obtained if action  $a$  is performed in state  $s$ .

*Q-learning* [15] is a well-known RL algorithm that updates the value function in each step so that the agent policy converges to the optimal one. Q-learning works even though the state transition probabilities are unknown to the agent. In our approach we will use the output of the Q-learning to improve the HMM by updating the state transition rate matrix according to the learned optimal policy.

Since the current state of an agent is hidden in our trust model, we will instead use the state probability distribution  $\gamma$  that is calculated after each observation, and learn the function  $Q(\gamma, a)$ . A variant of the Q-learning algorithm suitable for this case where the current state is only partially observable, and accounting for the fact that the domain of  $Q$  is not discrete and finite, can be found in [1]. Following this approach we associate a value  $q(i, a)$  with each hidden state  $s_i$ , and approximate  $Q(\gamma, a)$  as

$$Q(\gamma, a) \approx \sum_{i=1}^N \gamma(i)q(i, a).$$

Learning  $Q$  is done by adjusting all the  $q$  values after each action  $a$  and immediate reward  $r$  according to the Q-learning rule:

$$q(i, a) = (1 - \beta\gamma_k(i))q(i, a) + \eta\gamma_k(i)(r + \sigma \max_a Q(\gamma_{k+1}, a)), \quad (3)$$

where  $\eta$  is the learning rate and  $\sigma$  is a discount factor. The learning proceeds as follows, when an agent encounters another agent it will get the current state probability distribution  $\gamma_k$  belonging to this particular agent from its corresponding HMM and execute the action with

the largest  $Q(\gamma_k, a)$ . After the action is performed, the agent receives the reward, the next step state probability distribution  $\gamma_{k+1}$  is output from the HMM, and the Q-learning updating rule from Equation 3 is applied. The process is repeated at the next encounter between the agents. It should be noted that the observations to the HMM coming from recommendations will not result in actions or rewards, only the direct experiences in the form of agent interaction will trigger the reinforcement learning module in the trust model.

When the agent encounters another agent for the first time, the parameters of the HMM will be set to default values, and the model might not properly predict the system dynamics. A newcomer to the system should not be expected to be in the trusted state, as such a starting point would encourage agents to change their identities and re-enter the system frequently. It is therefore better to assume that agents are most likely neutral or untrustworthy to start with, and then they have to prove themselves trustworthy by behaving good in the interactions.

As the agent learns about the behavior of the other agent through direct experience and recommendations, the HMM parameters should be updated in order to improve the predictiveness of the model. We suggest that the state transition rates  $\Lambda$  and the observation probabilities  $\mathbf{B}$  of the HMM are updated after a predetermined number of Q-learning steps, e.g. by the Baum-Welch algorithm which finds the maximum likelihood parameter estimate. See [8] for a detailed explanation of the Baum-Welch algorithm.

Given a sequence of  $K$  observations the Baum-Welch algorithm makes use of the *backward variable*  $\beta_k(i) = P(y_{k+1}, \dots, y_K | x_k = s_i)$ , which is the probability of the observation sequence from next step and until the end given that we are in the state  $s_i$  at time-step  $k$ . The backward variable is found inductively by setting  $\beta_K(i) = 1$  and then recursively calculating  $\beta_k(i) = \sum_{j=1}^N p_{ij} b_j(y_{k+1}) \beta_{k+1}(j)$ . The re-estimation procedure calculates the joint probability  $\xi(i, j) = P(x_k = s_i, x_{k+1} = s_j | \mathbf{y})$  of being in state  $s_i$  at time-step  $k$  and state  $s_j$  in time-step  $k + 1$  as

$$\xi(i, j) = \frac{\alpha_k(i) p_{ij} b_j(y_{k+1}) \beta_{k+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_k(i) p_{ij} b_j(y_{k+1}) \beta_{k+1}(j)}.$$

For re-estimation of  $p_{ij}$  according to Baum-Welch we use

$$p_{ij} = \frac{\sum_{k=1}^{K-1} \xi(i, j)}{\sum_{k=1}^{K-1} \gamma_k(i)},$$

where the nominator is the expected number of transitions from state  $s_i$  to state  $s_j$ , and the denominator is the expected number of transitions from state  $s_i$ . For re-estimation of the observation symbol probabilities the following equation is used

$$b_j(m) = \frac{\sum_{k=1, \text{s.t. } y_k = v_m}^K \gamma_k(j)}{\sum_{k=1}^K \gamma_k(j)},$$

where the nominator is the expected number of times in state  $s_j$  and observing the symbol  $v_m$  and the denominator is the expected number of times in state  $s_j$ .

Algorithm 2 implements the re-estimation of the parameters. In order to avoid problems related to underflow we use scaling of the backward variable as described in [8] and [9]. The proof of correctness of the scaling procedure is given in the Appendix A. Algorithm 2 takes as input all the  $K$  different  $\alpha_k$  and  $\gamma_k$  vectors, which are calculated by Algorithm 1, as well as all the different  $\mathbf{P}_k$ . This means that these values need to be stored in the agent's memory together with the history of observations for the re-estimation procedure. To simplify the implementation we will combine the two observation probability matrices into one single matrix  $\mathbf{B}$  for the re-estimation algorithm. The observation probabilities are modeling the uncertainties

---

**Algorithm 2** Re-estimation of parameters

---

**Require:**  $P, \gamma, \mathbf{y}, \alpha, \mathbf{B}, K$

```
for  $i = 1$  to  $N$  do
   $\hat{\beta}_K(i) \leftarrow \frac{1}{\sum_{i=1}^N \alpha_K(i)}$ 
end for
for  $k = K - 1$  to  $1$  do
  for  $i = 1$  to  $N$  do
     $\hat{\beta}_k(i) \leftarrow \frac{1}{\sum_{j=1}^N \alpha_k(j)} \sum_{j=1}^N p_{ij}^k b_j(y_{k+1}) \hat{\beta}_{k+1}(j)$ 
    for  $j = 1$  to  $N$  do
       $\xi^k(i, j) \leftarrow \gamma_k(i) p_{ij}^k b_j(y_{k+1}) \hat{\beta}_{k+1}(j)$ 
    end for
     $\hat{\gamma}_k(i) \leftarrow \sum_{j=1}^N \xi^k(i, j)$ 
  end for
end for
for  $i = 1$  to  $N$  do
  for  $j = 1$  to  $N$  do
     $p_{ij} \leftarrow \frac{\sum_{k=1}^{K-1} \xi^k(i, j)}{\sum_{k=1}^{K-1} \hat{\gamma}_k(i)}$ 
  end for
end for
for  $m = 1$  to  $M$  do
  for  $j = 1$  to  $N$  do
     $b_j(m) \leftarrow \frac{\sum_{k=1, \text{s.t. } y_k = v_m}^K \hat{\gamma}_k(j)}{\sum_{k=1}^K \hat{\gamma}_k(j)}$ 
  end for
end for
return  $\mathbf{B}, \mathbf{P}$ 
```

---

associated with observations. This means that when we re-estimate  $b_j(m)$  elements associated with the direct observations, i. e. the agent's own ratings after interactions, we are evaluating the reliability of the agent itself when it comes to making good trust decisions. The elements associated with the recommendations are re-estimated as an evaluation of the reliability of the recommending agents. In this context we do not model the trustworthiness of each recommender separately, but this could be done as an extension of the model.

## 5 Conclusions and Future Work

We have proposed a novel trust model for multiagent systems where the goal of an agent is to make optimal trust decisions over time in a dynamic environment. An agent bases its action choice on a prediction of the other agents' behaviors according to the HMM trust estimation module following the Q-learning greedy policy. Since this is a policy that tends to the optimal one over time, the model learning algorithm will be training the HMM with sequences of observations that will positively impact the end goal.

As this is just a preliminary theoretical model, without any simulation results yet, there are many directions of research that may be explored to improve our work. We would like to see if it is possible to additionally train the model to making the best decisions about when to ask other agents for recommendations. Application of the model to a variety of trust scenarios and a comparison to other proposed trust models is of course the prime interest of our future work.

## References

- [1] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI Press, 1992.
- [2] K. Hamamoto, K. Morooka, and H. Nagahashi. Motion Recognition By Combining HMM and Reinforcement Learning. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2004.
- [3] Kjetil Haslum, Marie Elisabeth Gaup Moe, and Svein Knapskog. Real-time Intrusion Prevention and Security Analysis of Networks using HMMs. In *Proceedings of the 4th IEEE LCN Workshop on Network Security (WNS)*. IEEE, 2008.
- [4] M. A. T. Ho, Y. Yamada, and Y. Umetani. An HMM-based Temporal Difference Learning with Model-Updating Capability for Visual Tracking of Human Communicational Behaviors. In *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition (FGR'02)*. IEEE, 2002.
- [5] F. K. Hussain, E. Chang, and T. S. Dillon. Markov model for modeling and managing dynamic trust. In *Proceedings of the 3rd IEEE International Conference on Industrial Informatics, INDIN'05*, pages 725–733. IEEE, 2005.
- [6] F. K. Hussain, E. Chang, and T. S. Dillon. Quantification of the Effectiveness of the Markov Model for Trustworthiness Prediction. In *Proceedings of the International Conference 9th Fuzzy Days*. Springer, 2006.
- [7] J. Noda, M. Takahashi, I. Hosomi, H. Mouri, Y. Takata, and H. Seki. Integrating presence inference into trust management for ubiquitous systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 59–68. ACM Press New York, 2006.
- [8] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.
- [9] Ali Rahimi. An Erratum for “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”. <http://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html>, 2000.
- [10] S.D. Ramchurn, D. Huynh, and N.R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(01):1–25, 2005.
- [11] Sheldon M. Ross. *Introduction to Probability Models*, chapter Continuous-Time Markov Chains, pages 349–390. Academic Press, New York, 8th edition, 2003.
- [12] V. Sassone, K. Krukow, and M. Nielsen. Towards a Formal Framework for Computational Trust. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects (FMCO 2006)*, volume LNCS 4709, page 175. Springer, 2006.
- [13] W. Song, V.V. Phoha, and X. Xu. The HMM-Based Model for Evaluating Recommender’s Reputation. In *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, pages 209–215. IEEE, 2004.
- [14] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.

## Appendix A Scaling of the forward and backward variables

We want to use scaling of the forward variable and the backward variable, to avoid problems related to underflow. Since, without scaling, these variables consist of a large number of terms of value significantly less than 1. This appendix will give a detailed explanation of the scaling procedure for the implementation of the Algorithms 1 and 2, as proposed in [8] and [9]. We have that

$$\alpha_k(i) = \begin{cases} \pi_i b_i(y_k) & \text{if } k = 1 \\ b_i(y_k) \sum_{j=1}^N \alpha_{k-1}(j) p_{ji} & \text{if } k > 1 \end{cases} \quad (4)$$

where  $\alpha_k(i)$  is the forward variable without any scaling. We want to compute the scaled forward variable

$$\hat{\alpha}_k(i) = C_k \alpha_k(i), \quad (5)$$

with scaling coefficient

$$C_k = \frac{1}{\sum_{j=1}^N \alpha_k(j)}. \quad (6)$$

Let us use the notation  $\bar{\alpha}_k(i)$  for the running value of the forward variable as it is used in the implementation before scaling, and  $c_k$  for the running value of the scaling coefficient. The recursion for calculating the scaled forward variable can then be expressed as:

Initialization:

$$\begin{aligned} \bar{\alpha}_1(i) &= \alpha_1(i) \\ \hat{\alpha}_1(i) &= \frac{\alpha_1(i)}{\sum_{j=1}^N \alpha_1(j)} \end{aligned}$$

For  $k > 1$ :

$$\begin{aligned} \bar{\alpha}_k(i) &= b_i(y_k) \sum_{j=1}^N \hat{\alpha}_{k-1}(j) p_{ji} \\ c_k &= \frac{1}{\sum_{j=1}^N \bar{\alpha}_k(j)} \\ \hat{\alpha}_k(i) &= c_k \bar{\alpha}_k(i) \end{aligned}$$

We want to prove that this recursion realizes the scaling as expressed in Equation 5. For  $k = 1$  the scaling coefficient is  $c_1 = C_1 = \frac{1}{\sum_{j=1}^N \alpha_1(j)}$ , so the scaling is exactly as in Equation 5. For  $k > 1$  we can use a proof of induction as follows:

$$\begin{aligned} \bar{\alpha}_k(i) &= b_i(y_k) \sum_{j=1}^N \hat{\alpha}_{k-1}(j) p_{ji} \\ &= b_i(y_k) \sum_{j=1}^N C_{k-1} \alpha_{k-1}(j) p_{ji} && \text{(by application of Equation 5)} \\ &= C_{k-1} \alpha_k(i) && \text{(by application of Equation 4)} \end{aligned}$$

This gives us the relation

$$c_k = \frac{1}{\sum_{j=1}^N \bar{\alpha}_k(j)} = \frac{1}{C_{k-1} \sum_{j=1}^N \alpha_k(j)} \quad (7)$$

We can then express  $\hat{\alpha}_k(i)$  as

$$\hat{\alpha}_k(i) = c_k \bar{\alpha}_k(i) = \frac{C_{k-1} \alpha_k(i)}{C_{k-1} \sum_{j=1}^N \alpha_k(j)} = \frac{\alpha_k(i)}{\sum_{j=1}^N \alpha_k(j)}$$

so the recursion used is indeed realizing the scaling as given in Equation 5, which was what we wanted to show. Furthermore, by combining Equations 6 and 7, we get the relation

$$C_k = C_{k-1} c_k = \prod_{\kappa=1}^k c_\kappa$$

Recall the definition of the backward variable:

$$\beta_k(i) = \begin{cases} 1 & \text{if } k = K \\ b_i(y_{k+1}) \sum_{j=1}^N \beta_{k+1}(j) p_{ij} & \text{if } k > K \end{cases}$$

For the scaling of the backward variable let us define the scaling coefficient

$$D_k = \prod_{\kappa=k}^K c_\kappa$$

which gives us the relation

$$C_k D_{k+1} = \prod_{\kappa=1}^k c_\kappa \prod_{\kappa=k+1}^K c_\kappa = C_K$$

Let us denote by  $\hat{\beta}_k(i)$ , the scaled backward variable

$$\hat{\beta}_k(i) = D_k \beta_k(i) \quad (8)$$

and let us denote by  $\bar{\beta}_k(i)$ , the running backward variable as it is used in the implementation before scaling. The recursion for calculating the scaled backward variable is given by

Initialization:

$$\bar{\beta}_K(i) = \beta_K(i) = 1$$

$$\hat{\beta}_K(i) = D_K = c_K$$

for  $k < K$ :

$$\bar{\beta}_k(i) = b_i(y_{k+1}) \sum_{j=1}^N \hat{\beta}_{k+1}(j) p_{ij}$$

$$\hat{\beta}_k(i) = c_k \bar{\beta}_k(i)$$

Note that the running value of the scaling coefficient will be the same  $c_k$  as was used in the scaling of the forward variable. For  $k = K$  the Equation 8 is trivially fulfilled. The correctness

of the recursion for  $k < K$  can be shown by induction as follows

$$\begin{aligned}
\bar{\beta}_k(i) &= b_i(y_{k+1}) \sum_{j=1}^N \hat{\beta}_{k+1}(j) p_{ij} \\
&= b_i(y_{k+1}) \sum_{j=1}^N D_{k+1} \beta_{k+1}(j) p_{ij} \\
&= D_{k+1} \beta_k(i)
\end{aligned}$$

we can then rewrite  $\hat{\beta}_k(i)$  as

$$\hat{\beta}_k(i) = c_k \bar{\beta}_k(i) = c_k D_{k+1} \beta_k(i) = D_k \beta_k(i)$$

hence, the recursion is realizing the scaling in Equation 8.