EXPANDED HIDDEN MARKOV MODELS: ALLOWING SYMBOL EMISSIONS IN STATE CHANGES

Claudia Krull
Graham Horton
Institut für Simulation und Graphik
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, 39016 Magdeburg, Germany
E-mail: {claudia,graham}@sim-md.de

KEYWORDS

Hidden Markov Models, Stochastic Petri Nets.

ABSTRACT

In this paper we formally expand hidden Markov models (HMM) by symbol emissions in state changes. These expanded hidden Markov models (eHMM) can contain more information than original HMM with the same number of states. This is a necessary step towards the definition of hidden non-Markovian models on the basis of discrete stochastic models. These are most of the time event driven, which makes it necessary to attach information to the state changes that represent the events. The paper shows that the extended paradigm is to some extent equivalent to original HMM, and gives an example of the new possibilities using hidden non-Markovian models.

MOTIVATION

Simulation most of the time involves the need to model a real or planned system and then predict its behavior to some extent. This process heavily depends on structural and statistical knowledge about the system's behavior and parameters. For various reasons, sometimes one is not able or not willing to observe the complete system internals, but rather focuses on the system's visible interaction with the environment.

This could be the case, when one wants to diagnose the engine of a car without taking it apart, but by solely looking at some protocol that can be read out electronically. This error protocol can also be viewed as a sequence of signals that the hidden system (engine internals) emits. In case of a satellite in orbit it might not even be feasible to examine the object itself, but one has to rely on the signals emitted by the satellite in order to diagnose it, since the cost of sending a technician into space is just too high.

Many real systems can be represented using discrete stochastic models such as Petri nets. Extending these by the modeling and analysis of so-called rewards is a way to include observable output in these models. Petri nets are event-driven; the state transitions are the active elements and they model the dynamic elements of the real process. So-called impulse rewards can be associated with state transitions, and can model costs, protocol signals and much more. However, the analysis methods for Petri nets do not account for hidden information and the system is always assumed to be observable as a whole.

Hidden Markov models (HMM) on the other hand can model hidden processes with observable outputs. HMM consider the symbol being emitted in certain state, regardless of the previous or following state. However, the hidden model of HMM is usually a discrete-time Markov chain (DTMC). This restricts the modeling capabilities of the paradigm to geometric state duration distributions in the discrete case.

Combining the advantages of HMM and Petri nets, it would be possible to model more general hidden models which are event driven. Hidden non-Markovian models that can model more realistic behavior than DTMCs, could be analyzed with the methods of HMM, which expands the range of possible application areas.

The first step is to make HMM event-driven, which means associating the emission of symbols with a state transition, rather than with a state. The second step is then based on these eHMM the formalization of hidden non-Markovian models, which will not be described here. In this paper we formally define expanded hidden Markov models (eHMM) by associating symbol emissions with the state changes of HMM. We also adapt the three major algorithms that are used for the different analysis procedures, namely the forward algorithm for Evaluation, Viterbi for Decoding and Baum-Welch for Training of eHMM. The paper also gives an example of a hidden non-Markovian model to show the possible applications of the new paradigm.

STATE OF THE ART

Hidden Markov models and their application in speech recognition were first published around 1970; one of the first papers is (Baum et al. 1970). A comprehensive summary of the theory including algorithms and application examples can be found in (Rabiner 1989). The notation used in this current paper is based on the notation in (Rabiner 1989). In order to make the models more flexible, some research was done on explicit state duration densities (hidden semi-Markov models), which however complicated the solution algorithms (Russel and Moore 1985). Expanding all HMM states to a sub-HMM was also used to realize more general state duration distributions

(expanded state HMM) (Russel and Cook 1987). This increased the number of free parameters, the sub-HMM topologies were not very flexible, and the performance was tuned to speech-recognition systems. In (Wickborn et al. 2006) a method is described to find the trace probability and the corresponding state sequence for continuous stochastic models with generally distributed transitions, but the method is not applicable for the training of models. In (Isensee et al. 2006) a method for the training of hidden non-Markovian models using phase-type distributions was introduced.

Hidden Markov Model Background

Classical HMM are discrete-time Markov chains (DTMC) that stochastically emit a symbol at every time step, depending on the state that the hidden process currently resides in. These are so-called double stochastic processes and are sometimes also called signal models. They are widely used in speech recognition systems and sometimes in pattern recognition. (Rabiner 1989)

A hidden Markov model can be described by a 5-tuple *(S,V,A,B,Π)*. *S* is the set of states of the DTMC. *V* is the set of output symbols. *A* is the transition probability matrix of the DTMC. *B* is the output probability matrix, with b_{ij} being the probability to output symbol v_j in state s_i. *Π* is the initial probability vector of the DTMC. A sequence of states of the DTMC is denoted as $Q=\{q_1,q_2,q_3,\ldots, q_T\}$ and an observed output sequence as $O=\{o_1,o_2,o_3,\ldots,o_T\}$ with *T* being the maximum number of steps of the DTMC. A parameterization of a specific HMM is often denoted by $\lambda=(A,B,\Pi)$ which fully defines the model.

There are three basic questions that can be answered for an HMM and a given output sequence (trace).

1. What is the probability of producing the given output sequence *O* with the given model λ?
2. What is the most probable state sequence *Q* of the model λ that produced the observed output sequence *O*?
3. Maximize the probability with which the model λ produces the output sequence *O* by training the parameters of the model.

The first question can be efficiently answered by the forward algorithm (Rabiner 1989). The most likely state sequence can be determined by the Viterbi algorithm (Viterbi 1967). The third task of training an HMM model is solved by the so-called Baum-Welch algorithm (Baum et al. 1970).

eHMM – DEFINITION AND NOTATION

First the basic definition of eHMM is adapted from HMM. Most of the components stay the same. The changes are that the state sequence is extended by one state and starts at time *t=0*. This makes it also one element longer than the symbol trace. The most important change can be found in the emission probabilities *B*. These are no longer attached to a specific state, but to a transition between two states.

$$5-Tuple:(S,V,A,B,\Pi)$$
$$Model:\lambda=(A,B,\Pi)$$
$$States:S=\{s_1,\ldots s_N\}$$
$$Symbols:V=\{v_1,\ldots v_M\}$$
$$Symbol\ Sequence:O=\{o_1\ldots o_T\}$$
$$State\ Sequence:Q=\{q_0\ldots q_T\}$$
$$Transition\ Matrix:$$
$$A=\{a_{ij}\}_{N\times N}$$
$$a_{ij}=P(q_t=s_j\mid q_{t-1}=s_i\}$$
$$Output\ Probabilities:$$
$$B(k)=\{b_{ij}(k)\}_{N\times N}\quad k=1\ldots M$$
$$b_{ij}(k)=P(v_k\ at\ t\mid q_{t-1}=s_i\wedge q_t=s_j)$$
$$Initial\ Probability\ Vector:$$
$$\Pi=\{\pi_i\}_N$$
$$\pi_i=P(q_0=s_i)$$

The emission probabilities *B* can no longer be stored in one matrix, but in a set of matrices, one for each symbol, that contains the probability that the symbol is emitted for a specific state change. The sum of the emission probabilities for each state change has to be 1. Therefore the following condition holds:

$$\sum_{k=1}^{M}b_{ij}(k)=1\ \forall i,j,a_{ij}>0$$

eHMM - THE THREE ANALYSIS ALGORITHMS

The three basic problems that can be solved using HMM are Evaluation, Decoding and Training. Each of them has different practical applications. These will now be adapted to eHMM.

Problem I : Evaluation

The evaluation problem takes a given model $\lambda=(A,B,\Pi)$ and a given output sequence *O*. The question is the following: What is the probability that the given output sequence was produced by the given model?

$$O,\lambda\Rightarrow P(O\mid\lambda)$$

Forward Algorithm

The forward algorithm is an iterative algorithm, which step by step computes the probabilities α of the growing sub-sequences. Thereby it has a complexity of $O(N*T)$ as opposed to the brute force approach $O(N^T)$ that evaluates all possible paths separately.

$$\alpha_t(i)=P(o_1o_2\ldots o_t,q_t=s_i\mid\lambda)$$

Initialization:

$$\alpha_0(i)=\pi_i$$

Induction :

$$\alpha_t(i) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_{ij}(o_t) \quad t = 1 \ldots T$$

Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

The algorithm can be easily modified to allow for symbol emissions in state changes. The value of $\alpha_t(i)$ now describes the probability of having emitted the symbol sequence $o_1 \ldots o_t$ and ending up in state s_i.

Backward Algorithm

The backward algorithm works analogously to the forward algorithm, but starts to evaluate the trace at the end. It is mentioned here for completeness and because the β values are needed later on for the Baum-Welch algorithm.

$$\beta_t(i) = P(o_{t+1} \ldots o_T \mid q_t = s_i, \lambda)$$

Initialization :

$$\beta_T(i) = 1$$

Induction :

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_{ij}(o_{t+1}) \beta_{t+1}(j) \quad t = T-1 \ldots 0$$

Termination :

$$P(O \mid \lambda) = \sum_{i=1}^{N} \beta_0(i) \pi_i$$

The algorithm can also be easily modified to allow for symbol emissions in state changes. The value of $\beta_t(i)$ now describes the probability of having emitted the symbol sequence $o_{t+1} \ldots o_T$ when starting in state s_i.

Problem II : Decoding

The decoding problem takes a given model $\lambda=(A,B, \Pi)$ and a given output sequence O. One then tries to retrace/reconstruct the behavior of the hidden model that produced the given output most likely according to some measure.

$$O, \lambda \Rightarrow \arg\max_{Q} P(O \mid Q, \lambda)$$

Individual most likely states

Using the original HMM definition, one way of decoding a given trace is to determine the individually most likely state to produce a certain output symbol in the sequence and then take that as the solution. The problem is that the resulting state sequence might not be valid, since transitions between two successive states might not be possible if the transition probability is 0.

This approach does not work here, since two successive most likely state changes cannot be as easily connected as two successive states. The target state of the first transition and the source state of the second transition have to be the same to produce a valid state sequence. This is even more serious than having zero probability to change between two successive states.

Viterbi Algorithm

The Viterbi algorithm decodes a given symbol sequence by determining the most probable sequence of states to have produced it. The algorithm has a similar structure to the forward algorithm.

$$\delta_t(i) = \max_{q_0 \ldots q_{t-1}} P(q_0 \ldots q_t = s_i, o_1 \ldots o_t \mid \lambda)$$

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij} b_{ij}(o_{t+1}) \right]$$

Initialization :

$$\delta_0(i) = \pi_i$$

$$\psi_0(i) = 0$$

Recursion:

$$\delta_t(j) = \max_i \left[\delta_{t-1}(i) a_{ij} b_{ij}(o_t) \right] \quad t = 1 \ldots T$$

$$\psi_t(j) = \arg\max_i \left[\delta_{t-1}(i) a_{ij} b_{ij}(o_t) \right] \quad t = 1 \ldots T$$

Termination:

$$P^* = \max_{i=1 \ldots N} \left[\delta_T(i) \right]$$

$$q_T^* = \arg\max_{i=1 \ldots N} \left[\delta_T(i) \right]$$

Backtracking :

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$

The algorithm can also be easily modified to allow for symbol emissions in state changes. The value of $\delta_t(i)$ now contains the probability of the most likely path that emitted the symbol sequence $o_1 \ldots o_t$ and ends up in state s_i. The most likely state sequence so far for each current state s_i, which is needed for backtracking, is stored in $\psi_t(i)$. The resulting most likely state sequence to have produced the output O is also called the Viterbi path.

Problem III : Training

The problem of training HMMs is by far the most difficult, and is more an optimization task than a direct solution. The training problem takes a given output sequence O and a model size given by the model states S and the output symbols V. The task is to find the most likely model to have produced this output sequence.

$$O \Rightarrow \lambda = \arg\max_{\lambda} P(O \mid \lambda)$$

Baum-Welch Algorithm

The Baum-Welch algorithm is an optimization algorithm that takes an initial model specification and improves it successively through several iterations called epochs, ensuring that the probability to emit the output sequence is increased in every iteration.

The $\alpha_t(i)$ and $\beta_t(i)$ are the terms calculated by the forward and backward algorithm. $\gamma_t(i)$ contains the probability of being in state s_i given the current model configuration and output sequence. The summation of the $\gamma_t(i)$ over the length of the trace yields the total

probability of being in state *i* given the current model and trace.

$\xi_t(i,j)$ contains the probability of the state change from s_i to s_j emitting the given symbol o_t from step time t-1 to t. This is only defined if $a_{ij}>0$. The summation of the $\xi_t(i,j)$ over the length of the trace yields the total probability of a state change from i to j given the current model and trace.

$$\xi_t(i,j) = P(q_{t-1} = s_i, q_i = s_j \mid O, \lambda)$$
$$= \frac{\alpha_{t-1}(i) a_{ij} b_{ij}(o_t) \beta_t(j)}{P(O \mid \lambda)}$$
$$= \frac{\alpha_{t-1}(i) a_{ij} b_{ij}(o_t) \beta_t(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_{t-1}(i) a_{ij} b_{ij}(o_t) \beta_t(j)}$$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \quad t = 0 \ldots T-1$$

$$\sum_{t=0}^{T-1} \gamma_t(i) = \text{expected number of transitions from } s_i$$

$$\sum_{t=1}^{T} \xi_t(i,j) = \text{expected number of transitions from } s_i \text{ to } s_j$$

These values are now used to re-estimate the model parameters $\lambda=(A,B,\Pi)$ as follows. The re-estimation step needs to be performed until a predefined stability criterion is satisfied, that signifies, that no further improvement can be found for the model parameters.

$$\text{Re} - \text{estimation Formulas}$$

$$\overline{\pi_i} = \text{expected frequency in state } s_i \text{ at time } t = 0$$
$$= \gamma_0(i)$$

$$\overline{a_{ij}} = \frac{\text{expected number of transitions from } s_i \text{ to } s_j}{\text{expected number of transitions from } s_i}$$
$$= \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(i)}$$

$$\overline{b_{ij}(k)} = \frac{\text{expected number of transitions from } s_i \text{ to } s_j \text{ observing } v_k}{\text{expected number of transitions from } s_i \text{ to } s_j}$$
$$= \frac{\sum_{t=1 \& o_t = v_k}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \xi_t(i,j)}$$

The re-estimation formulas can be transformed to fit the new model definition. Therefore the expanded HMM can also be trained using the Baum-Welch algorithm. The resulting procedure should still suffer from the same drawbacks as the original models. It is still a local optimization method, which moves in the direction of steepest descent and is very expensive, having to run through the forward and backward procedures for every epoch.

EQUIVALENCE OF eHMM

In this section we will show that a standard HMM can be transformed into an expanded HMM, which makes the latter at least as powerful as HMM.

The example system is a small web server that cannot be observed directly. An internet user is trying to deduce the server's current condition based on its answers to ping requests. The HMM models the web server that can be either working in the states *Idle* or *Busy* or that can be in state *Failed* (see Figure 1). To the outside user in the web the actual state of the web server is not visible, he can only observe the answer to the ping request once every time unit, which is either a *ping* reply or a *timeout*. By analyzing the observed traces, he can draw inferences on the actual state of the web server. The parameter of the model are the initial state probability vector Π, the state transition probability matrix A and the output probability matrix B, which can all be derived from the Figure.

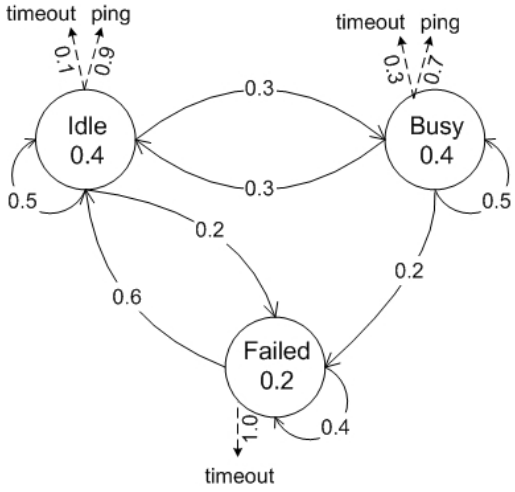

Figure 1 : Original HMM with Parameters

This model $\lambda=(A,B,\Pi)$ can easily be transformed into an eHMM $\lambda'=(A',B',\Pi')$:

- The transition probability matrix stays the same $A' := A$
- The output probabilities shift from the states to the state changes, by using the output probabilities of the target state for a state transition $b_{ij}'(k) := b_j(k)$
- The initial probability vector has to be computed by solving the following linear system of equations (assuming it to be a row vector) $\Pi = A\,\Pi'$.

The new calculation of the initial probability vector becomes necessary, because the state sequence of the eHMM has one more element than the output sequence or the original state sequence of the HMM.

After these transformations, the model shown in Figure 2 is produced. The changed elements are marked

in red. The major change is the shifting of the output probabilities from the states to the state transitions.

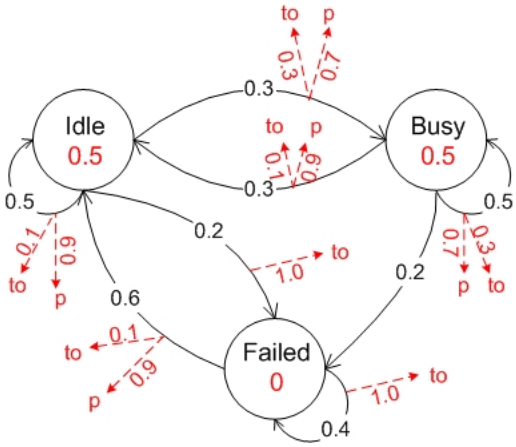

Figure 2 : Equivalent Expanded HMM with Parameters

The models are equivalent in the sense that they produce the same output sequences with the same probabilities. As an example the output probabilities of two sequences were computed using the original and the modified forward algorithm. The Viterbi algorithm was used to compute the most likely state sequences that produced the output.

The internet user has observed two sequences: (*ping, ping, ping*) and (*ping, timeout, ping*). He wants to know the absolute probability of each trace and the most likely hidden state sequences that produced them. The results of the forward and Viterbi algorithm for the given models is shown in Table 1. $P(O|\lambda)$ is the absolute probability of emitting trace O with the original HMM; $P(O|\lambda')$ is the probability of emitting trace O with the equivalent eHMM. The probabilities are exactly the same in both cases. Q is the Viterbi path of the HMM, and Q' the Viterbi path of the eHMM. F denotes the hidden model state *Failed* and I the state *Idle*. The last three elements of the paths are equivalent.

Table 1 : Results of HMM and eHMM

Sequence *O*	***(ping,ping,ping)***	***(ping,timeout,ping)***
HMM Probability	$P(O\|\lambda)=0.26448$	$P(O\|\lambda)=0.132672$
eHMM Probability	$P(O\|\lambda')=0.26448$	$P(O\|\lambda')=0.132672$
HMM Viterbi path	$Q=I,I,I$	$Q=I,F,I$
eHMM Viterbi path	$Q'=I,I,I,I$	$Q'=I,I,F,I$

The equivalence of the output trace probabilities can even be proven inductively, as the following formulas show.

Basis

$$\alpha_1(j) = \pi_j b_j(o_1) \quad (\pi_j = \sum_{i=1}^{N} a_{ij}\pi'_j \text{ by construction})$$
$$= \sum_{i=1}^{N} \pi'_j\, a'_{ij} b_j(o_1) \quad (b_j(o_1) = b'_{ij}(o_1) \text{ by construction})$$
$$= \sum_{i=1}^{N} \alpha'_0(i) a'_{ij}\; b'_{ij}(o_1)$$
$$= \alpha'_1(j) \quad \forall j$$

Assumption

$$\alpha_k(i) = \alpha'_k(i) \quad \forall i$$

Inductive Step

$$\alpha_{k+1}(j) = \sum_{i=1}^{N} \alpha_k(i) a_{ij} b_j(o_{k+1}) \quad (b_j(o_t) = b'_{ij}(o_t) \text{ by construction})$$
$$= \sum_{i=1}^{N} \alpha'_k(i) a'_{ij}\, b_{ij}(o_{k+1})$$
$$= \alpha'_{k+1}(j)$$

Conclusion

$$\alpha_t(i) = \alpha'_t(i) \quad \forall i, t = 1 \ldots T$$

The terms calculated by the forward algorithm are the same for the HMM as for the equivalent eHMM. All terms of the eHMM are marked by an apostrophe. The Basis shows that the terms $\alpha_1(i)$ and $\alpha'_1(i)$ are equivalent, by replacing the single terms by their equivalents, based on the defined transformation rules. Based on the assumption that the terms for step k are equal, it is concluded, that the terms for k+1 must be equal. Therefore this equality holds for all $t>0$.

The proof of equivalence of the Viterbi path can be done analogously to some extent. The assumption is, that the last T elements of the eHMM Viterbi path are the same as the HMM Viterbi path. The idea is that if the first element of the HMM Viterbi path is the same as the second element of the eHMM Viterbi path, then the following elements will also be equivalent. The problem is, that the base assumption is not given in all cases, since the two-element eHMM Viterbi path might not contain the most likely starting state $max(\pi_i)$ of the HMM, since the π_i are the weighted sum rather then the maximum of the π'_i. However, in the given example, the Viterbi paths of the two models are equivalent.

These results show that the example transformation was done correctly and the adapted algorithms produce results that are equivalent to the original ones. The equivalence was proven for the forward algorithm, and was tested on an example for the Viterbi algorithm. We did not test the Baum-Welch algorithm for several reasons. First of all, the equivalence of the results might not be given, since a different model is trained, with more free parameters, and secondly an implementation of the adapted algorithm is still future work.

The reverse transformation of an arbitrary eHMM into an HMM is not as intuitive as the approach described in this section. Extra states have to be introduced to account for the increased amount of information in an eHMM. Artificial constructs might become necessary such as states with no probability to stay in. The dual graph that could be constructed using graph theory, is only defined on planar graphs, which does not have to be the case for Markov chains.

MOTIVATING EXAMPLE

An example of an interesting hidden non-Markovian model is given in this section. It will show the increased

modeling power of the new paradigm, which involves symbol emissions in state changes. The example is that of a machine in a factory. This machine is essential for the availability of the whole production line and is often maintained to prevent failures. Even though, the machine fails from time to time, and can therefore be in the three state *OK*, *Failed* or *Maint.* The times between failures and the maintenance intervals can have any distribution type, for example Weibull for the failure (T_F) and Normal for the maintenance interval (T_{MI}). The maintenance of a machine can produce cost of three different categories, depending on the extent of repair necessary (10€, 20€ or 30 €), each having a different probability to occur. Similarly the cost for the repair of a failed machine can be grouped into 3 categories, depending on the severity of the disruption (30€, 100€ or 200 €). The time necessary for either repair or maintenance can again have any kind of probability distribution such as Lognormal. The production manager only gets the record of the bookings that were the results of the repairs or maintenances, but not the exact events that produced them.

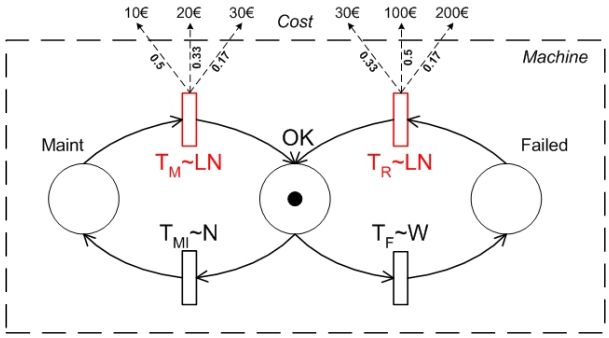

Figure 3 : Example of Hidden non-Markovian Model

This system could be interpreted as a hidden non-Markovian model (see Figure 3), with the machine being the internal hidden part and the bookings as observable symbol emissions. It would not be possible to model this using an HMM, since the underlying model is not a DTMC. Several interesting questions could be answered using this abstraction:

- How probable is a certain booking record? Which series of events produced that record most likely?
- What is the machine availability of the possible generating paths, and how does that compare to the actual observed availability of the machine?
- How would the behavior of the system change, when changing maintenance intervals?
- Are there any frequent failure sequences? What produced them? Could they be prevented by using more reliable system components?

These questions can be answered using hidden non-Markovian models and the analysis methods of both HMM and Petri nets. The new combined paradigm enables the direct utilization of system output for the parameterization, diagnosis and analysis of not directly observable behaviour. This was not possible by using either HMM or Petri nets alone.

CONCLUSION

In this paper we expanded the definition of hidden Markov models to symbol emissions in state changes. We first expanded the basic definition of HMM and then adapted the three basic analysis algorithms to this new definition. We then gave an example of the equivalence to the old notation, and an example of the usefulness of hidden non-Markovian models.

This new definition shifts the focus of the model from the states to the state changes, the events. This is necessary to be able to analyze discrete stochastic systems using the proposed methods, since there the events are the driving force of the model behavior. Using this newly defined paradigm of eHMM we can now proceed to the formalization of hidden non-Markovian models. These will open up a whole new range of possible applications of the analysis methods of HMM. These applications might include the analysis of systems by only looking at failure protocols or the prediction of machine performance based on the previous machine behavior.

REFERENCES

Baum, L.E., T. Petrie, G. Soules, N. Weiss. 1970. “A Maximization Technique in the Statistical Analysis of Probabilistic Functions of Markov Chains”. In *The Annals of Mathematical Statistics*, vol.41, no.1, pp.164-171.

Isensee, C., F. Wickborn and G. Horton. 2006. “Training Hidden non-Markov Models”. In *Proc. of the ASMTA’06*.

Rabiner, L. R.. 1989. “A tutorial on hidden Markov models and selected applications in speech recognition”. In *Proceedings of the IEEE.* vol.77, no.2, pp.257–286.

Russel, M. J. and R. K. Moore. 1985. “Explicit modelling of state occupancy in hidden Markov models for automatic speech recognition”. In *Proc. ICASSP’85*, pp. 5-8.

Russell, M. J. and A. E. Cook. 1987. “Experimental evaluation of duration modelling techniques for automatic speech recognition,” in *Proc. ICASSP’87*, pp. 2376–2379.

Viterbi, A. H.. 1967. “Error bounds for convolutional codes and an asymptotically optimal decoding algorithm”. In *IEEE Transactions on Information Theory*, IT-13:260–269.

Wickborn, F., C. Isensee, T. Simon, S. Lazarova-Molnar, G. Horton. 2006. “A New Approach for Computing Conditional Probabilities of General Stochastic Processes”. In *Proc. ANSS’06.*

AUTHOR BIOGRAPHIES

CLAUDIA KRULL studied Computer Science at the Otto-von-Guericke-Universität Magdeburg and spent an exchange year at the University of Wisconsin, Stevens Point, where she graduated in 2002. She was awarded her German University diploma in September 2003. Since October 2003 she is a PhD student at the “Lehrstuhl für Simulation” at the Otto-von-Guericke-Universität Magdeburg. Her e-mail address is: `claudia@sim-md.de.`

GRAHAM HORTON studied Computer Science at the University of Erlangen, obtaining his Masters degree ("Diplom") in 1989. He obtained his PhD in Computer Science in 1991 and his "Habilitation" in 1998 at the same university, in the field of simulation. Since 2002, he is Professor for Simulation and Modelling at the Computer Science department of the University of Magdeburg. His email address is: `graham@sim-md.de.`