A New Approach for Computing Conditional Probabilities of General Stochastic Processes

Fabian Wickborn, Claudia Isensee, Thomas Simon, Sanja Lazarova-Molnar, Graham Horton
Computer Science Department, Otto-von-Guericke Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
fabian@sim-md.de (contact), claudia@sim-md.de, t.h.simon-mail@gmx.de,
sanja@sim-md.de, graham@sim-md.de

Abstract

In this paper Hidden Markov Model algorithms are considered as a method for computing conditional properties of continuous-time stochastic simulation models. The goal is to develop an algorithm that adapts known Hidden Markov Model algorithms for use with proxel-based simulation. It is shown how the Forward- and Viterbi-algorithms can be directly integrated in the proxel-method. The possibility of integrating the more complex Baum-Welch-algorithm is theoretically addressed. Experiments are conducted to determine the practicability of the new approach and to illustrate the type of analysis that is possible.

1. Introduction

Computer simulation has become a widespread analysis instrument for estimating the properties of stochastic continuous-time models. What-if-scenarios can be analysed without altering the real system or manufacturing prototypes. Simulation emulates the random behaviour of the real or fictional system and thereby produces data that is used for the gain of new knowledge and for decision making. In most cases the computation does not take any further information into account besides the simulation model itself.

In this paper, we are interested in analysing general stochastic simulation models for which we possess information about how they have behaved. We want to numerically compute conditional probability results that are valid under the constraint which is set by our information about the behaviour. Questions of interest are, for example, "What is the probability that the given behaviour can be exhibited by the model we have?", or "On the condition that some particular output is generated by the system, what is the probability that a particular chain of events has taken place?" If we modelled a fast food restaurant, one implementation of these abstract questions might be, "Given the receipts printed at the counter, was yesterday a typical day for the market?", or "What is the probability that the queue was full yesterday?", or even "What are the ten most probable combinations of five successive customer inter-arrival times that lead to the turnover which we experienced?"

Similar questions can be answered for memoryless discrete-time stochastic processes using Hidden Markov Models. In our research, we would like to answer them for non-Markovian models with a continuous time dimension by adapting the known HMM formalism and algorithms for proxel-based simulation. Being able to do so opens the door towards new kinds of analysis and applications of computer simulation.

1.1. Hidden Markov Models

In this section we will give a brief introduction to Hidden Markov Models and describe associated problems and algorithms. A Hidden Markov Model (HMM) is a stochastic model containing two Markov processes, in which a non-observable stochastic process is analysed indirectly using its random outputs. HMMs have been known for a long time and are often used in speech recognition, natural language processing and genomics to perform pattern recognition. So far, speech recognition is the most well known application of HMMs. It sees recorded spoken text as output of a HMM "natural language". The observed output sequence is a recorded audio signal divided into a sequence of (guessed) sounds, and a string of natural language text is considered to be the "hidden cause" of the signal. Here, the discrete "time" dimension is the sequence of recorded sounds.

However, in this paper, we want to use HMM techniques to compute conditonal probabilities with discrete-event simulation models with general firing times.

1.1.1. Definition of HMMs

A Hidden Markov Model consists of a Discrete Time Markov Chain (DTMC), whose behaviour cannot be observed directly, and an output alphabet, whose tokens are emitted from the DTMC states randomly, according to known probabilities. Since in general, different states can emit the same token, any given sequence of tokens may be be generated by more than one sequence of states. In formal notation ([8]) the definition of a HMM is given by $H = \{A, B, \pi\}$, with

- $S = \{s_1, \ldots, s_n\}$, a set of n (hidden) model states
- $A = \{a_{ij}\}$, a transition matrix for the inner Markov process (the hidden DTMC), where a_{ij} is the probability to change from state s_i to s_j.
- $\pi = \{\pi_1, \ldots, \pi_n\}$, a vector of state probabilities, where π_i is the probability to be in state s_i.
- $\Sigma = \{v_1, \ldots, v_m\}$, a set of m output tokens.
- $B = \{b_{ij}\}$, an emission matrix for the outer Markov process, where b_{ij} is the probability to emit token v_j from the state s_i.

1.1.2. Problems Solvable with HMMs

There are three standard problems which can be solved with Hidden Markov Models [8]. For all three, there is given a model description H and an output token sequence $O = \{o_1, \ldots, o_k\}$ with $o_1, \ldots, o_k \in \Sigma$.

The first problem is how to compute $P(O|H)$, i.e. the probability with which the model H will emit the sequence O. For our example of the fast food restaurant, this could be used to compute how likely a given sequence of meals and drinks is sold. The computation of $P(O|H)$ can be done with the Forward-algorithm developed by Baum et al. in the late 1960s [1, 3]. It computes the probability of the sequence beginning from the first time point to the last in k discrete steps.

The second problem is how to compute which sequence of DTMC states $Q = \{q_1, \ldots, q_k\}$ ($q_1, \ldots, q_k \in S$) will most likely generate O, i.e. the sequence of states which best explains the observations. In the example of a fast food restaurant, this means to track served customers by collected order receipts. This state sequence $Q(O)$ is computed by the so-called "Viterbi-algorithm" and is therefore also called the "Viterbi path". The algorithm was named after its developer who created it in 1967 [9, 4]. It implements a maximisation of the probability $P(Q|O, H)$, i.e. the probability that the sequence of visited states was Q, under the condition that the output sequence is O. Rabiner observes that it is structurally similar to the Forward-algorithm since both computations can be done by a lattice structure [8]. Therefore they can be implemented together in a single algorithmic loop which iterates m times.

The third problem is how to adjust the model parameters $H = \{A, B, \pi\}$ to maximize $P(O|H)$, i.e., to train the HMM to most likely generate the output. In that case, normally A, B and π are not known in advance but Σ, S and one or more output sequences O are. With the fast food restaurant that would be the training of a simulation model of the restaurant by receipts collected in the past. Rabiner evaluates this as "by far the most difficult" of the three HMM problems [8]. Its solution can be achieved with the Baum-Welch-algorithm (also known as Forward-Backward-Algorithm) which resembles a computation of maximum-likelihood-estimates for the HMM parameters [2].

All three computations have existed for more than three decades and can therefore be considered to be well-known and understood. The main disadvantage is the need for the model to be memory-less.

1.2. HMMs and General Stochastic Models

The possibility of answering questions for a simulation model of a real system under the condition that the model has generated a particular output sequence (which may have been generated by the real system) is very interesting for any kind of model. This is especially true if the observed output sequence can be generated in more than one way, that is, by more than one chain of events (or states).

The first two HMM computing problems are valuable tools for such analyses. They can directly be applied to a stochastic model with general firing times, if the model can first be translated into a DTMC. How to do this is the elaborated in this paper. The third HMM computation is a training algorithm which trains a discrete-time memory-less process model. For this algorithm to be applicable to general stochastic models, there would have to be a way of un-discretisating and back-fitting the memoryless transition probabilities to continuous stochastic distributions. Subsection 4.1 discusses the possibilities of implementing this approach.

1.3. Proxels as basis for HMM Analysis

Our goal is to be able to perform HMM-style computations to continuous-time stochastic models with generally distributed activities. The inner process of HMMs is required to be a DTMC, that is, to be both memoryless and discrete-time. For this reason, an algorithm for a general model must be able to separate the model into its discrete states, partition simulation time into discrete time steps and transform the model into a form in which it can be considered to be Markovian (i.e. memoryless).

Proxel-based simulation was introduced by Horton in 2002 [5] as a state space-based analysis method for general continuous-time stochastic models. It implements the method of supplementary variables to translate general distributions into a Markovian form and creates and analyzes the state space of a model on-the-fly. The proxel method performs a discretisation of simulation time and tracks all possible developments of the system behaviour in discrete time.

Let $s \in S$ be a discrete state of the model, τ be a vector of supplementary variables which store the duration of ongoing or pre-empted activities, and t be a time point in the discretised time dimension. Then a proxel $\alpha = (p, s\tau, t)$ is a vector consisting of a model state given by (s, τ, t) and the probability p for that state.

In the proxel method, the probability of all potential state changes is computed deterministically by the means of the instantaneous rate function of the state transition distributions. The assumption is made that the probability is negligible that more than one event will take place during any one time step. As a result of the proxel-based simulation, a transient solution for the state probability vector π of the model is computed for all discrete points in time. A detailed description of the algorithm can be found in [5].

Proxel-based simulation fulfils all of the requirements to translate generally distributed continuous-time models into a form in which HMM analysis algorithms can be applied. For that reason, we will exploit it for the purpose of reaching our goal.

2. Implementation

In this section we show how to apply the first two fundamental HMM analyses to the proxel-based simulation algorithm by implementing the Forward and the Viterbi algorithm. The third algorithm (Baum-Welch) has not yet been implemented, but will be theoretically elaborated in subsection 4.1. First, a reference example is given and it is shown how to pre-process (i.e. discretise) the continuous-time output sequence to make it usable by the analysis algorithms.

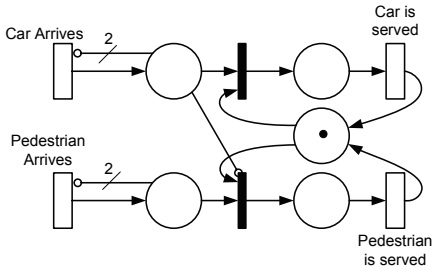

Figure 1. Petri net model of the fast food restaurant

2.1. Accompanying Example

To illustrate the following explanations, we use a model of a small fast food restaurant as an example. The restaurant has a single employee who serves customers one at a time. The customers come by car or on foot and arrive in differently randomly distributed intervals and wait in separate queues for their service. The service times are of random length and are differently distributed for cars and pedestrians. The employee gives customers arriving by car higher priority, so he or she will ignore all pedestrians until there are no cars in service or in the queue.

In Figure 1, a Stochastic Petri Net (SPN) representation of the model is shown. The distributions for the event times are:

- Car Arrives: Exponential(Mean = 10.0)
- Pedestrian Arrives: Weibull(Scale = 10.0, Shape = 1.5)
- Car is served: Uniform (a = 0.0, b = 0.4)
- Pedestrian is served: Weibull(Scale = 10.0, Shape = 1.5)

Customers can order either a soft drink or a meal. Car drivers buy drinks with a probability of 70% and meals with a probability of 30%. Pedestrians stop by for a meal in 60% of all cases or purchase a soft drink with 40% probability. At service completion, a receipt is created which reports the current time and the product which was served to the customer. The model emits tokens from the set $\Sigma = \{\text{“}S\text{”}', \text{“}M\text{”}'\}$ at each completion-of-service event with the probabilities given above, where “S” stands for a soft drink and “M” for a meal. The arrival events in this model do not emit tokens.

Using this model as an example, it will be explained how output sequences are discretised and the model

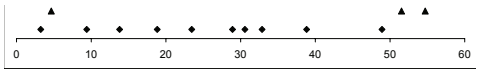

Figure 2. Example output sequence

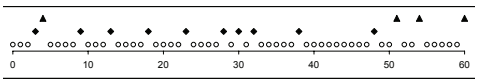

Figure 3. Discretised output sequence

is simulated and analysed with the Forward and the Viterbi algorithm.

2.2. Discretisation of the Output Sequence

For the analysis to function, there must be some kind of signal sequence emitted by the model. In case of a HMM, this sequence O consists of a discrete enumeration of tokens. With continuous-time stochastic models, the observed output sequence O is no longer just an enumeration of tokens $v_i \in \Sigma$ but of ordered pairs (t_i, v_i), where t_i is a point in simulation time and $v_i \in \Sigma$ is the token emitted at t_i. Tokens are emitted while performing a state transition, that is, the emission is part of a simulation event. In the example, tokens are only emitted at the end of a service activity.

Figure 2 shows one possible output sequence of the fast food model plotted over time. Triangles represent the emission of a meal token and diamonds represent the emission of a soft drink token. The time unit is one minute. Each token is emitted at a unique point in the continuous time dimension.

The HMM algorithms demand that the output sequence O be discrete-time and the emission to happen from a particular state with a certain probability. Therefore, the output sequence of the model of interest has to be discretised. The discretisation is a preprocessing prior to the simulation of the model. The continuous time dimension is partitioned into multiple intervals with the same step size Δ. Δ has to be chosen so that at most one token is emitted in each discrete time step. Each token of the output sequence is mapped to the time step during which it was emitted. An artificial null token ϕ is inserted into the sequence at each time step in which no meal or soft drink token was emitted,

Input : Sequence O, Simulation time $t_{\max}$
Output: Discretized sequence: diseq$[0 \dots t_{\max}]$
Data: Step $k = 0$, Sequence Index $j = 0$
while $k\Delta \leq t_{\max}$ **do**
  if $O[j]$.*timestamp* $< (k+1)\Delta$ **then**
    diseq$[k] = O[j]$.token
    $k = j + 1$
  else diseq$[k] = \phi$
  $k = k + 1$

Algorithm 1: Discretising the sequence

in particular: ϕ is emitted with a probability of 1.0 in all time steps in which no state change occurs. The discretisation has linear computational effort with regard to the number of discrete time steps. The discretisation can be expressed in pseudo-code notation as in (Algorithm 1.

Figure 3 shows the discretisation version of the example output sequence given above. Circles represent the emission of ϕ, i.e. the case that no meal or soft drink was sold during that time step. The discretisation was performed with the step size $\Delta = 1$ minute.

2.3. Sequence Probability and Viterbi Path

The idea behind our modified proxel-method for computing $P(O|H)$ and Q is to use a proxel $\alpha = (p, s, \tau, t, o)$ to represent a state s of the underlying model and the probability p to be in that state at time t and to generate the token o_e ($t = e\Delta, 1 \leq e \leq k$) from that state. For any proxel α , we can determine all possible successor states, compute the probabilities that the state transition to those states will occur within the time step Δ and that the token o_{e+1} will be generated at that transition, and create new proxels accordingly. Δ is again the discretisation parameter of the time dimension and has to be of the same size as the step used for the discretisation of the output sequence (or at least be an integer divider for it).

If a state transition is enabled in α but cannot generate o_e, then no proxel is generated. For this reason, some branches of the logical tree of proxels are pruned. Since we need to visit all discrete paths to compute the Viterbi-path we are not allowed to aggregate proxels representing the same state (contrary to the usual implementation of proxel-based simulation).

We define X as a stack for the temporary storage of proxels. Since we do not aggregate proxels that have stored probability valid for the same model state $(s.\tau, t)$, we can use the simple stack to minimise the memory requirements, which results in a depth-first exploration of the possible paths [5]. C is an array of size k that stores the probabilities that the sub-sequences $O_e \subset O$ have been generated by the model. T_{ij} is the state transition from s_i to s_j. h_{ij} is the appropriate instantaneous rate function, and b_{ije} is the probability of emitting o_e at the event of T_{ij}.

Below, a pseudo-code notation of the algorithm is

given. For the sake of clarity, we set aside the question of how to compute the age vector of a successor proxel (which can be found in [5]).

```
1  X = ∅
2  addProxel(1.0, s, 0.0, 0.0)
3  while X ≠ ∅ do
4      α =getproxel()
5      if α.t < kΔ then
6          forall T_ij that are enabled in α.s do
7              if b_ije > 0 then
8                  C[t/Δ] =
                   C[t/Δ]+bije×α.p×δ×h_ij(α.τ)
9                  addproxel(bije × α.p × δ ×
                   h_ij(α.τ), succ(α.s, T_ij),
                   aging(α.τ, α.s, T_ij, p.t + Δ))
10         if o_e = φ then
11             C[t/Δ] = C[t/Δ] + bije × α.p ×
               (1 − δ × Σ_T h_ij(α.τ))
12             addproxel(bije × α.p × (1 − δ ×
               Σ_T h_ij(α.τ)), α.s, aging(α.τ, α.s, 0),
               p.t + Δ)
```

Algorithm 2: $P(O|H)$ and the Viterbi path

The algorithm makes use of the following functions (see [5] for details):

addproxel(α) pushes a proxel on the stack X

getproxel() pops one proxel from the stack X

succ(s, T) computes the successor state that is reached from the state s at the event of T happening

aging(τ, s, T) computes a new supplementary variable vector for the event of T happening in state s

The modified algorithm is similar to the original one and differs primarly at the constraints in lines 7 and 10, in which the computation is restricted to only creating proxels that generate the output token of the according time step. In lines 8 and 11, the probability $P(O_e|H)$ of generating the subsequences O_e at all is stored.

As described in 1.1.2, the Viterbi-path is the sequence of model states that most likely generates the output sequence. Since each proxel that is valid for a time step e holds the probability to have reached one particular state by a unique path at the time $e\Delta$, the probability of a proxel α can be considered as the probability of the complete path by which α was reached. For that reason, the proxel $\max\{\alpha.p|\alpha.t = e\Delta\}$ is the endpoint of the Viterbi-path $Q_e(O)$. Since the proxel-based simulation already creates each possible path, one can gain the Viterbi path by simply comparing the probability of the proxels at given time $e\Delta$. The proxel with the maximum probability together with all of its predecessors form the Viterbi-path.

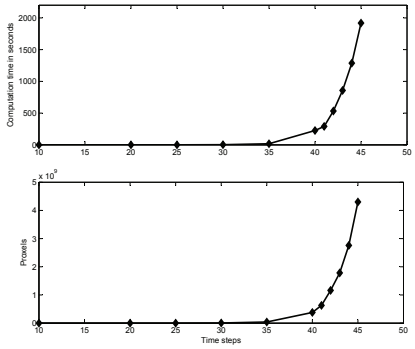

Figure 4. Computation time and processed proxels over time

3. Experiments

In this section, we report the results that were obtained from computational experiments. We discuss the efficiency of the algorithm by studying the following topics: the computational effort as a function of the analysed time span, the number of possible paths as a function of time span and output sequence, and the Viterbi path. For the sake of computational feasibility, we limited the state space of the model by setting a maximum of queue length of two customers for each queue. The resulting SPN model has a reachability graph consisting of 19 discrete markings and 27 arcs. All computations were performed on a Pentium IV processor with 3 GHz. Also, in all cases, we used a depth-first enumeration of the state space to minimise memory requirements (as proposed in [5]). The time step size for both the discretisation of the output sequence and the simulation was $\Delta = 1$ minute.

3.1. Computational Complexity

To illustrate the complexity of the algorithm we performed analyses for the fast food model using subsequences of the discretised output sequence O given in Figure 3. We used values from 20 to 45 as length of the sub-sequence. Figure 4 illustrates the computation time and the number of processed computational units (proxels).

The computational complexity grows exponentially with the number of time steps, that is, the length of the

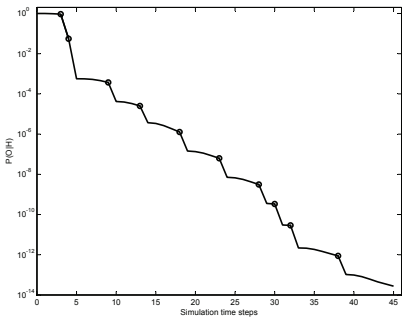

Figure 5. Probability of a growing sub-sequence of the example output

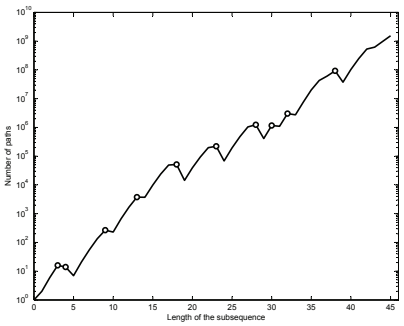

Figure 6. Number of paths that generate the sub-sequence

analysed sub-sequence. This behaviour is not surprising, since it is due to the exponentially growing complexity of the proxel-based simulation algorithm which is executed during the analysis. Although the algorithm is feasible for small sub-sequences, the analysis is too expensive to be practical for larger sequences. For the analysis of the complete output sequence (containing 120 tokens), estimates yield a runtime of more than 715 years. Evidently, techniques are needed to reduce the computational complexity of the method.

3.2. Probability of the Output Sequence

Next, we consider the probability $P(O|H)$, i.e. the probability, that the model will generate the observed output sequence, over time. We performed an analysis of the fast food model, again with the output sequence of Figure 2. For each time step up to $t = 45$ minutes, we computed the probability that the sub-sequence starting at the beginning up to the current time step will be generated by the model.

Figure 5 shows how the probability of the sub-sequences decreases with increasing length of the sub-sequence. Notable is the sharper decrease of the probability following a time step in which a non-null token was emitted. These time steps have been marked with a circle in the figure. The decrease occurs due to the reduced ambiguity with which the sub-sequence has been generated.

The reduction of the ambiguity is further illustrated in Figure 6, in which the number of possible paths that create the sub-sequences is plotted as a function of the length of the sub-sequence. The number of paths for a time step is equal to the number of proxels that are processed at this time step, because all proxels which are generated belong to paths that can validly generate the sub-sequence.

At the emission of a non-null token (again illustrated by a circle) only those state transition are processed that are able to emit the token. This results in a pruning in the set of possible paths and thereby to a temporary reduction in the number of processed proxels. In our example, this means that every time a token is emitted we know that the service of a customer has been completed. All paths in which no service was completed at that time step are unable to generate the model and may therefore be discarded.

For our example model, the number of sequence-generating paths increases for consecutive time steps in which the null token is emitted. At the same time, $P(O|H)$ decreases. This means that an even higher number of paths create an even smaller amount of probability. We believe that this property might be exploitable to improve the efficiency of the method.

3.3. Viterbi Path

In the next experiment, the probability of the Viterbi path was recorded as a function of time. In Figure 7 the probability of the Viterbi path for time $t = 45$ is shown. The lower (blue) graph shows the probability that the model will develop in that path over time. The probabilities are equal to the individual probabilities of the proxel forming the path. The upper graph is the conditional probability of the path under the condition that the model generates the first 45 tokens from the output sequence of Figure 3. The sequence of states that is most likely traversed within 45 time steps has a conditional probability of 4.15E-6 (a chance of 1:240000).

Taking the assumption of at most one event happening during a time step into account, the output sequence is associated to a unique chain of events. These events

Table 1. Viterbi path in term of events

t	Event	t	Event
0	Car arrives	23	Car served
2	Car arrives	24	Car arrives
3	Car served	26	Car arrives
4	Car served	27	Pedestrian arrives
7	Pedestrian arrives	28	Car served
12	Car arrives	30	Car served
13	Car served	32	Pedestrian served
16	Pedestrian arrives	36	Pedestrian arrives
18	Pedestrian served	38	Pedestrian served
22	Car arrives		

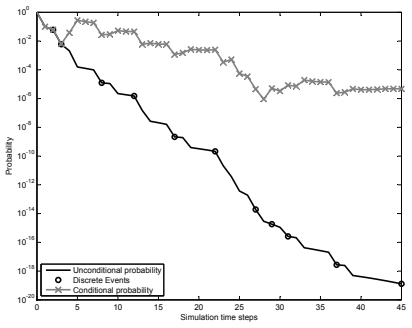

Figure 7. Probability and conditional probability of the Viterbi path

are given in the table 3.3.

Since the proxel-based algorithm computes all possible state paths, it is easy not only to track the Viterbi path but to generate a top-n list of the most likely paths. For the first five output tokens of the example output sequence the four most likely paths to generate them are (conditional probabilities are given in brackets):

1. $t = 0$: Car arrives, $t = 2$: Car arrives, $t = 3$: Car served, $t = 4$: Car served (0.2789)

2. $t = 1$: Car arrives, $t = 2$: Car arrives, $t = 3$: Car served, $t = 4$: Car served (0.2777)

3. $t = 0$: Car arrives, $t = 1$: Car arrives, $t = 3$: Car served, $t = 4$: Car served (0.2312)

4. $t = 1$: Pedestrian arrives, $t = 2$: Car arrives, $t = 3$: Pedestrian served, $t = 4$: Car served (0.1130)

4. Future Work

4.1. Automated Model Generation with HMMs

Using proxels, the Forward and Viterbi algorithms can be extended to non-Markovian models. However, training model parameters in order to fit the hidden models output to a specific sequence is not possible using this approach, since the DTMC of the model is only partly created during the proxel simulation and not stored as a whole. Here we present an idea how the Baum-Welch algorithm could be nevertheless applied to non-Markovian models.

In order to use the Baum-Welch-Algorithm for training non-Markovian models, these models need to be turned into a DTMC. If a general transition is simply replaced by an exponential one, the algorithm can be applied, but the resulting model would not represent the true system. A transition can also be replaced by an approximation, which consists of a series of Markov chain states. These so-called phases form a phase-type distribution [7]. A fitting algorithm has been developed [6] that approximates generally distributions by a discrete phase-type distribution of variable order. The order of the distribution, i.e. the number of phases, is notdependent on the number of time steps of the original distribution.

A non-Markovian model with unknown transition distributions could be turned into a DTMC by replacing the generally distributed transitions by a phase type distribution of a fixed structure, a limited number of phases and an initial standard parameterization, such as an Erlang distribution. The model parameters can now be fit to the desired output sequence using the Baum-Welch algorithm, changing the appropriate transition probabilities of the DTMC accordingly. Ideally, the resulting model still contains the introduced phase-type approximation, whose parameters have also changed.

One could now extract this phase approximation and analyze it separately, by generating the output of this particular Markov chain segment. This output should correspond to the actual distribution of the originally replaced transition. Finding this distribution might be done by expertise and guessing the correct distribution or by using a kind of backward fitting.

This approach has not been tested, but it presents exciting possibilities, since most real world models are non-Markovian. Some interesting questions are: Does the Baum-Welch algorithm preserve the structure of the phase type distribution? Do the resulting phase-type distributions actually fit a known general distribution? Can some kind of automated backward fitting be applied or is user-knowledge neccessary? When replacing

the phase-type distributions by their fits, does the resulting model accurately produce the output sequence?

If the answers to these questions are positive, it might be possible to algorithmically train a general stochastic process to generate a particular behaviour. Such algorithm would certainly prove to be useful in the process of automatic generation of simulation models.

4.2. Effort Reduction by Adaptive Time Steps

From the experiments we learned that the analysis has to deal with possible model behaviours that create sequences of consecutive null tokens. Such subsequences result in a large number of paths which can generate them and, therefore, in a high computational complexity. Many of these paths occur due to the lack of events happening, that is, because the model stays in the same state for many consecutive time steps. This, in turn, is due to the small discretisation parameter Δ.

To reduce the computational complexity, the use of adaptive time steps for both the discretisation of the continuous output sequence and the proxel-based simulation might prove useful. For the latter it has already been shown that the introduction of adaptive time steps leads to an improvement in the efficiency of the computation [10]. We are optimistic that the insights gained there can be applied to the above problem to reduce its computational complexity. This would allow the method to be applied to larger simulation models or longer output sequences.

4.3. Validation and Verification of Models

One further possible application of the new approach introduced in this paper might be the exploitation of the Forward algorithm for the validation and verification of simulation models. The method is able to compute the probability by that a model generates a given output sequence. During the modelling process, this computation can be used to verify if the model is able to generate a sequence which was originally generated by the modelled system. If the computed probability is zero, then one can conclude that the model is unable to recreate that sequence and therefore is invalid or has been incorrectly implemented. This process can easily be automated and, for that reason, come to use in a model validation and verification tool chain.

5. Summary and Outlook

This paper presented an approach to implement the Forward and the Viterbi algorithm for the simulation of stochastic continuous-time signal models using Hidden Markov Chains. The implementation performs a discretisation of the time dimension for the stochastic process and the output sequence. For the first, the proxel-based simulation algorithm is exploited. The major achievement of this paper is the introduction of an algorithm with which simulation models containing any type of random distribution can be analysed under the assumption of a given system output. Since the computational complexity of our simulation algorithm grows exponentially, at the moment it is feasible only for small models and short output sequences. Our future research on the topic of proxel-based simulation aims to weaken this drawback.

So far, the accuracy of the method has not been addressed. Further experimental studies have to be made and a theoretical model for the loss of accuracy for the conditional probability values due to the discretisation has to be created.

References

[1] L. E. Baum and J. A. Egon. An inequality with applications to statistical estimation of probabilistic functions of a markov process an to a model for ecology. *Bull. Amer. Math. Soc.*, 73:360–363, 1967.

[2] L. E. Baum, T. Peterie, G. Souled, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist.*, 41(1):164–171, 1970.

[3] L. E. Baum and G. R. Sell. Growth functions for transformation on manifolds. *Pac. J. Math.*, 27(3):211–227, 1968.

[4] G. D. Forney. The viterbi algorithm. *Proc. IEEE*, 61:268–278, March 1973.

[5] G. Horton. A new paradigm for the numerical simulation of stochastic petri nets with general firing times. In *Proceedings of the European Simulation Symposium 2002*. SCS European Publishing House, 2002.

[6] C. Isensee and G. Horton. Approximation of discrete phase-type distributions. In *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[7] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. The John Hopkins University Press, 1981.

[8] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[9] A. H. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theory*, IT-13:260–269, April 1967.

[10] F. Wickborn and G. Horton. Feasible state space simulation: Variable time steps for the proxel method. In *Proceedings of the 2nd Balkan conference in informatics*, pages 446–453, Ohrid, Macedonia, November 2005.