Conversive Hidden non-Markovian Models

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: Dipl.-Inform. Robert Buchholz

geb. am 17. April 1983 in Berlin

Gutachter:

Prof. Dr.-Ing. Graham Horton
Prof. Khalid Al-Begain, PhD
Dr.-Ing. habil. Juri Tolujew

Magdeburg, den 04. Mai 2012

Abstract

Partially observable discrete stochastic systems are real-life systems whose stochastic internal behavior is not observable, but certain aspects of which cause the emission of externally observable signals. Examples for such systems include the undiagnosed illness of a patient that emits observable symptoms, and the unobserved mood of a car driver that affects his observable driving behavior. For many real-life systems it is desirable to be able to reconstruct the unobserved behavior based on the observations in order to gain insights into the unobserved behavior.

Hidden Markov models are a well-researched class of partially observable discrete stochastic models for which such a behavior reconstruction is possible. For HMMs, efficient algorithms to solve the four main behavior reconstruction tasks Evaluation, Decoding, Smoothing and Training exist. But a disadvantage of HMMs is that the internal behavior they can model is limited to discrete time and Markovian durations. Thus, HMMs are not applicable to many real-life systems. Certain extensions to HMMs exist that lift the limitation of discrete time, but those that additionally also lift the limitation of Markovian durations loose the ability to model concurrent behavior.

In this work, Conversive Hidden non-Markovian models are therefore developed as a new class of conceptual models that can represent stochastic systems that are continuous in time and may contain concurrent behavior with non-Markovian durations. For these CHnMMs, algorithms for the four basic behavior reconstruction tasks Evaluation, Decoding, Smoothing and Training are developed.

The developed algorithms compute exact results where comparable simulation algorithms so far only provided approximations. And the developed algorithms for all four tasks have been shown to be practically feasible for model sizes that allow for practical applications.

Thus, CHnMMs for the first time enable the behavior reconstruction of partially observable discrete stochastic systems in continuous time with concurrent non-Markovian behavior. They therefore enable practitioners to gain insights into the unobserved behavior of further systems and – depending on the application – may consequently be used to save time and money, or gain insights into so far unknown behavior.

Contents

Chapter 1

Introduction

1.1 Background

Many real-world systems are partially observable: their internal behavior is hidden from an outside observer, but the system emits observable signals that give hints on the current internal behavior.

One example of such a partially observable system is the behavior of a car driver. From a distance, his mood is not observable for an outsider, but the driving dynamics such as the acceleration of the car at a traffic light give an indication of what that mood could be.

For these systems it is often desirable to reconstruct the unobserved behavior based on a model of the system and some observations. In the case of car drivers, it could be worthwhile to reconstruct the mood of car drivers in different situations in order to determine whether certain traffic conditions negatively impact the mood of the driver and therefore increase the risk of accidents.

Using behavior reconstruction, the real system can be understood without having to go through the time-consuming, expensive and sometimes even unfeasible process of setting up additional sensors to measure the otherwise unobservable behavior. This allows the practitioner to understand the behavior of the real system without intervention in its behavior. Benefits of this understanding range from advancing understanding in scientific applications to saving money in manufacturing and business applications.

So-called partially observable discrete stochastic (PODS) systems are a well-researched class of partially observable systems [22, 48, 59, 62, 63, 67]. They can be modelled with a finite number of possible internal states - for example a car driver whose mood is either sad, neutral or angry instead of being represented by a real number - and their unobservable system behavior is influenced by randomness - for example the duration of shifting gears varies randomly. For many classes of PODS systems, efficient algorithms for the reconstruction of the likely unobserved behavior already exist.

1.2 Motivation and Scientific Challenges

However, behavior reconstruction is not yet possible for all PODS systems: Existing algorithms follow all possible internal behaviors of the model, and

determine the probability for each of them. But in many settings, the number of these internal behaviors increases exponentially with increasing length of the sequence of observations, which can easily render any approach unfeasible. Research therefore has so far been concentrated on developing algorithms for those simple classes of PODS for which the behavior reconstruction can be done efficiently.

This has lead to the current state of the art where behavior reconstruction is possible efficiently for some classes of PODS systems, and not possible at all for the remaining classes. In particular, behavior reconstruction is currently possible when either the internal unobserved activities[1] have random, arbitrarily distributed (i.e. non-Markovian) durations, or when multiple internal activities take place at the same time. However, no algorithms exist for systems which contain both concurrent activities and arbitrarily distributed activity durations at the same time. Developing algorithms for this class of PODS models is the focus of this work.

This class of systems with concurrent activities with arbitrary duration distributions seems especially relevant for practitioners, because in many instances models – and sometimes also observation protocols – of that type already exist, but there are no algorithms to make use of that data. For example, production facilities are often already modelled as PODS systems containing independent concurrent activities with arbitrary duration distributions (several machines are active at the same time, but duration of production steps are independent of each other) [57, 58, 74, 75]. Other application areas can be envisioned as well: Fastfood restaurants with multiple servers belong to this class of models, as might a model of the interactions of animals of different species in a given area.

The goal of this work is therefore to enable practitioners to reconstruct the behavior of systems represented as models with concurrent activities with arbitrarily distributed durations.

The scientific challenge here lies in the computational complexity: the more expressive a class of models is, the more different behaviors exist, and thus the more expensive the behavior reconstruction becomes. Since the class of PODS models with concurrent processes with arbitrarily distributed durations is the most extensive class of PODS models tackled so far, it is uncertain whether algorithms for its behavior reconstruction would be computationally feasible.

1.3 Goals

This work intends to enable practitioners to reconstruct the behavior of this more extensive class of PODS system in the same way that is possible with most basic PODS class of Hidden Markov Models (HMMs). For the behavior reconstruction of HMMs, four basic tasks and the corresponding solution algorithms exist [23, 62, 66]:

- Evaluation[2]: Determine the probability of a system to have created a given protocol. This task is performed in order to determine which one

[1] Throughout this work, we will use the word "activity" as defined in [1]: A time period of specified length.

[2] In the remainder of this work, the tasks Evaluation, Decoding, Smoothing and Training will be capitalized as proper nouns in order to differentiate them from other uses of these words.

of several competing models for an observation is most likely the correct one.

- Decoding: For a given model and a given sequence of observations, reconstruct the most likely internal behavior sequence that would have generated the observation protocol at hand. This task directly reconstructs the most likely internal behavior sequence of the system.
- Smoothing: Given a model and a sequence of observations, determine probabilities of the model to have been in each possible internal state at each point in time of a signal observation. For each time of a signal observation, the smoothing task takes into account all observations before *and after* that time. It thus reconstructs the probabilities of the system to be in each individual internal system state, whereas the Decoding task is is only concerned with reconstructing the most likely *sequence* of states.
- Training: Given an initial model and an observation sequence, the training task is to modify the model in order to better explain the observations. This is used to build a model that accurately explains the observations from a rudimentary initial model.

The goal of this work is consequently to provide algorithms that solve these four basic behavior reconstruction tasks on PODS models with concurrent activities with arbitrary non-Markovian durations.

Success Criteria The reaching of this goal is subject to two success criteria:

The mandatory criterion is that the tasks which compute an exact solution for HMMs (Evaluation, Decoding, Smoothing) should also not just provide approximations for our class of models, but compute exact solutions – given that the model and the observation sequences are correct. This criterion ensures that the reconstructed superior probability of a model, state or state sequence is indeed due to it being the best explanation of the observation, and not simply an approximation error. For the Training task, the predominantly used Baum-Welch algorithm for HMM only iteratively finds a better model and also only finds a locally optimal model, so that we too do not require an exact solution for the Training of our models.

Second, the purpose of this work is to enable practitioners to reconstruct the behavior of real-life systems without having to invest in expensive additional equipment. Thus, as a soft criterion we require that the algorithms to be developed are as efficient as possible with respect to computation time and memory consumption, in order to be executable on commodity computers.

1.4 Boundary Conditions and Differentiation

This section briefly lists the boundary conditions under which the algorithms will be developed, and serves the purpose of distinguishing the goal of this work from similar research areas that this work is not concerned with.

Every State Change Emits a Signal In all classes of PODS models for which behavior reconstruction algorithms exist, it is assumed that *every* change

of an internal state is related to an externally observable symbol to be emitted [22, 48, 53, 62, 63, 67, 69, 81]. This limitation can be a limitation to the expressiveness of the model class, but greatly decreases the computational effort required for the behavior reconstruction. Since the behavior reconstruction of our class of models with concurrent activities and arbitrarily distributed activity is thought to already be more expensive than is the case for existing models, we too require all internal state changes to emit an externally observable signal, and therefore the times of all state changes to be detectable. Behavior reconstruction of PODS systems without this limitation has been attempted before, but existing approaches have been shown to not be practically feasible [84].

No Experiments on Real-Life Systems The purpose of this work is solely to *enable future* practical applications, it is not concerned with proving the practical applicability. To that end, the algorithms developed are shown to be theoretically correct, and their behavior is tested on models that resemble potential application scenarios, but they are not tested on actual real-world data.

Consequently, the protocols of observations used in all experiments are not obtained from measurements on real-life systems. Instead, the systems in question are modeled in the discrete event simulation tool AnyLogic [8], and those simulation models are then used to generate artificial sequences of observations.

Systems are Modelled Manually The Training and Evaluation tasks are tasks that often occur in machine learning. One advantage that PODS behavior reconstruction has over other machine learning approaches is that a practitioner can use his experience, partial observations and measurements of the real system to manually build a model of the system. He can thus avoid a full-fledged machine learning approach to automatically build the model, which would require vast amounts of observation sequences and would still only result in an approximate model of the system.

Since this work deals with a class of PODS models, we assume as well that models will mostly be built manually by practitioners. To that end, the Training algorithms developed will be usable to refine an existing model, but not to set up the initial model structure. This assumption is also reflected in the document structure of this work: we will develop the algorithms to reconstruct the behavior of a model *before* we introduce the new model Training algorithms - which in fact are based on the behavior reconstruction algorithms.

1.5 Classification of Research

While this work deals with models that are a generalization of Hidden Markov Models, it touches other research areas of Computer Science as well:

- The new algorithms will be developed based on existing approaches from the field of state-space based *Simulation* of discrete stochastic systems.
- The Evaluation task, for which an algorithm is to be developed, can be used to determine which model best matches a given measurement, which is a classical task of *Pattern Recognition*.

- The Decoding and Smoothing tasks can be used to find relevant pieces of information in a large body of data (i.e. multiple observation sequences), which is a *Data Mining* task.
- The Training task parameterizes a model without human intervention solely based on measurements, as is the purpose of *Machine Learning* approaches.

Thus, all of algorithms to be developed have in common that they deal with the generation of *knowledge* from the raw data of observation sequences.

1.6 Methodology

The remaining work is guided by the following methodology: First, related work is presented in Chapter 2 in order to evaluate existing alternative approaches and familiarize the reader with background information relevant to the remaining document. Then, the class of partially observable discrete stochastic models for which the behavior reconstruction algorithms are to be developed is formally specified in Chapter 3. Based on this formal specification and borrowing ideas from the Proxel simulation method and the existing HMM forward algorithm, an algorithm to solve the Evaluation task for non-Markovian PODS models with concurrent activities is developed in Chapter 4. Chapter 5 slightly modifies that algorithm and adds backtracking in order to solve the Decoding task. Then, based on the Evaluation algorithm a complementary backward computation algorithm is developed in Chapter 6, which together with the Evaluation algorithm solves the Smoothing task. And finally, Training algorithms are developed in Chapter 7 based on the Smoothing algorithm. The document is then concluded in Chapter 8, where the findings are summarized, the success or failure is determined, and an outlook on benefits of this research and further research opportunities is given.

Chapter 2

Related Work

This chapter presents existing research results that are relevant to this thesis. Its purpose is twofold: First, it introduces existing approaches that this work is based on. And second, it aims to distinguish the approach developed in this thesis from existing research results. This is done by giving an overview of existing approaches that solve problems similar to the ones in this thesis, and by reasoning why those are unable to solve the specific tasks that are to be solved in this work.

To that end, the following topics are covered: As Hidden Markov Models and their basic behavior reconstruction tasks form the basis of this research, they are introduced in Section 2.1. This introduction includes the formal definition of HMMs and their behavior reconstruction tasks, and explains how these tasks are solved. Afterwards, existing approaches that may be suitable to solve these same tasks on non-Markovian models with concurrent activities are investigated. This includes derivatives of HMMs in Section 2.2, approaches for sequential data analysis in Section 2.3 and approaches from the field of pattern recognition in Section 2.4.

Following these alternative approaches, Section 2.5 introduces stochastic Petri nets as the class of conceptual models that will be used in the remainder of this work to visualize non-Markovian PODS models with concurrent activities. Finally, in Section 2.6 the Proxel simulation method is introduced. It is an algorithm that is able to simulate the behavior of those models whose behavior we wish to reconstruct and therefore provides several concepts that will be relevant for the behavior reconstruction tasks as well.

2.1 Hidden Markov Models

HMMs are a class of computational models to represent PODS systems in discrete time with time-homogenous internal behavior. Since the class of computational models that this work is concerned with is an extension of HMMs, there are two reasons to introduce HMMs:

First, a formal definition of the four basic HMM tasks needs to be introduced in order for the tasks to be defined and finally be solved for our class of models as well. And second, the HMM behavior reconstruction algorithms themselves will form the basis for the behavior reconstruction of our more expressive class

of PODS models.

Unless explicitly noted otherwise, all information in this section is taken from Rabiner [62].

Semantics HMMs are partially observable models in discrete time and with a discrete state space. Thus, an HMM is in one of a finite set of states at any given point in time. With each time step, the model stochastically changes its state based on a time-homogenous state transition probability - which includes the possibility of returning to the same state. After the state change the model emits an externally observable symbol. The particular symbol to be emitted is determine stochastically based only on the time-homogenous symbol emission probability of the state reached.

The specification of an HMM thus has to include a set of discrete states, a set of symbols, a matrix of state transition probabilities and the set of state symbol emission probabilities as well as a vector of initial state probabilities. When a real system is specified as such an HMM, and a sequence of observations (a so-called "trace") is available, the likely unobserved behavior of the real system can be reconstructed.

2.1.1 Formal Specification of HMMs

In this section, HMMs are formally specified. As far as possible, this definition will later be reused to define the class of non-Markovian PODS models with concurrent activities.

According to [62], an HMM has:

- a set of N discrete states $\{S_1, \ldots, S_N\}$
- a set of M observable symbols $\{V_1, \ldots, V_M\}$

In this context, a *trace* of observations O is a sequence of T individual symbol observations $O = o_1 \ldots o_T$ where each $o_t \in \{V_1, \ldots, V_M\}$. And $q_t \in \{S_1, \ldots, S_N\}$ is the system state after the tth symbol emission. Further, a *path* Q of length T is a sequence of traversed internal states $Q = q_1 \ldots q_T$. With these definitions, HMMs are specified by

- an initial probability vector $\Pi \in \mathbb{R}^N$ with elements $\pi_i = P(q_0 = S_i)$
- a matrix of state transition probabilities $A \in \mathbb{R}^{N \times N}$ with elements $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$
- a set of symbol emission probabilities $B = \{b_i(V_k)\}$, where each $b_i(k)$ is a function that maps each symbol V_k to the corresponding symbol emission probability in state S_i, i.e. $b_i(k) = P(o_t = V_k | q_t = S_i)$

Since the states S_i and symbols V_k are merely names that have no effect on the actual model behavior, an HMM λ is fully specified by $\lambda = (A, B, \Pi)$ alone.

For HMMs specified in this way, four basic behavior reconstruction algorithms exist [23, 62, 66]: Evaluation, Decoding, Smoothing and Training. These will be explained in detail in the following sections.

2.1.2 The Evaluation Task

The goal of the Evaluation task is to determine the probability that a model λ has caused a given trace of observations O, formally to determine the probability $P(O|\lambda)$. This task is usually performed when multiple competing models for a given trace exist [23], in order to determine the most likely model for that observation.

A naïve approach in computing this probability would be to individually compute the probabilities for all possible paths of internal states and to sum up these probabilities. This approach, however, would be generally computationally unfeasible: For each of the T time steps the internal state could be any of the N discrete states, requiring $O(N^T)$ time to compute the final Evaluation probability.

Instead, the Evaluation task for HMM is generally solved with the inductive Forward algorithm. For each time t it inductively computes the probabilities to be in each state based in the state probabilities of the previous time step. More precisely, it computes the Forward probabilities $\alpha_t(i)$ to be in state S_i at time t and having emitted the first t symbols of the trace,

$$\alpha_t(i) = P(q_t = S_i \cap o_1 o_2 \dots o_t).$$

These can be computed inductively with the initialization

$$\alpha_1(i) = \pi_i b_i(o_1)$$

and the induction over the number of time steps $t = 1 \dots T-1$:

$$\alpha_{t+1}(i) = \sum_{j=1}^{N} (\alpha_t(j)\ a_{ji}\ b_i(o_{t+1}))$$

With these joint probabilities, $P(O|\lambda)$ can then be computed using the law of total probability as:

$$P(O|\lambda) = \sum_{i=1}^{N} P(q_T = S_i \cap o_1 o_2 \dots o_T) = \sum_{i=1}^{N} \alpha_T(i)$$

With this approach, one only needs to compute N Forward probabilities for each of the T time steps. And each Forward probability depends only on the N Forward probabilities of the previous time step, yielding a time complexity of $O(N^2 T)$.

2.1.3 The Decoding Task

The Decoding task for a given model λ and a given trace O is to find the most likely sequence of internal system states Q of the model to have created the trace, formally to find[1]

$$\underset{Q=q_1 \dots q_T}{\arg\max}\ P(Q|O) = \underset{Q=q_1 \dots q_T}{\arg\max}\ \frac{P(Q \cap O)}{P(O)}.$$

[1] Formally, this probability as well as all following ones would need to be conditioned on the current model λ, here yielding $P(Q|O, \lambda)$. For legibility we omit this explicit conditioning on λ whenever it is clear from the context that the probability is to be computed for the current model.

Here, the denominator $P(O)$ is identical for every internal state sequence, thus does not impact the arg max and may be omitted, yielding:

$$\underset{Q=q_1 \dots q_T}{\arg\max}\, P(Q|O) = \underset{Q=q_1 \dots q_T}{\arg\max}\, P(Q \cap O)$$

The Decoding task is generally solved with the Viterbi algorithm [79], which works very similar to the Forward algorithm. Instead of inductively computing the *sum* of path probabilities $\alpha_i(t)$ that reached state S_i after emitting the partial trace $o_1 \dots o_t$, the Viterbi algorithm inductively computes the probabilities $\delta_i(t)$ of the one *most likely* path that reached S_i after emitting $o_1 \dots o_t$:

$$\delta_t(i) = \max_{Q=q_1 \dots q_{t-1}} P(Q \cap q_t = S_i \cap o_1 \dots o_t)$$

and for each $\delta_i(t)$ also stores the predecessor state $\psi_i(t)$ on its path. This way, the final state q_T of the most likely sequence of internal states is simply that S_i with the highest $\delta_i(T)$, and the stored predecessors $\psi_i(t)$ allow for the backtracking of the remaining path. Formally, the Viterbi algorithm is given by its induction initialization

$$\delta_1(i) = \pi_i \; b_i(o_1),$$

its induction step over all time steps $t = 1 \dots T-1$:

$$\delta_{t+1}(j) = \max_{i \in \{1 \dots N\}} \{\delta_t(i) \; a_{ij} \; b_j(o_{t+1})\}$$

$$\psi_{t+1}(j) = \underset{i \in \{1 \dots N\}}{\arg\max} \{\delta_t(i) \; a_{ij} \; b_j(o_{t+1})\}$$

and the final backtracking step

$$q_T^\star = \underset{i}{\arg\max}\{\delta_T(i)\}$$

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

that determines the most likely path of internal states $Q^\star = q_0^\star \dots q_T^\star$.

2.1.4 The Smoothing Task

Whereas the Decoding task was concerned with the single most likely path of internal system states to explain a given observation sequence, the Smoothing task is concerned with computing the probabilities of the system to be in a particular state at a particular time given a trace of observations:

$$\gamma_t(i) = P(q_t = S_i | o_1 ... o_t ... o_T)$$

It differs from the Forward probabilities computed for the Evaluation task in that the Forward probabilities take into account only past observations, whereas the Smoothing task is to compute the state probabilities given all past *and future* observations given by the trace.

The Smoothing probabilities $\gamma_t(i)$ can be computed from the Forward probabilities $\alpha_t(i)$ and the so-called Backward probabilities

$$\beta_t(i) = P(o_{t+1} ... o_T | q_t = S_i)$$

as follows:

$$\begin{aligned}
\gamma_i(t) &= P(q_t = S_i | o_1 \dots o_t \dots o_T) = \\
&= \frac{P(q_t = S_i \cap o_1 \dots o_t \dots o_T)}{P(o_1 \dots o_t \dots o_T)} \\
&= \frac{P(q_t = S_i \cap o_1 \dots o_t \dots o_T)}{\sum_j P(q_t = S_j \cap o_1 \dots o_t \dots o_T)} \\
&= \frac{P(q_t = S_i \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_i)}{\sum_j P(q_t = S_j \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_j)} \\
&= \frac{\alpha_t(i)\ \beta_t(i)}{\sum_j \alpha_t(j)\ \beta_t(j)}
\end{aligned}$$

So to perform the Smoothing task, one only needs to compute the Backward probabilities $\beta_i(t)$ that the system can still emit the remaining trace $o_{t+1} \dots o_T$ given that it is in state S_i at time t. These can be computed similarly to the Forward probabilities, with the difference that their induction is performed backwards with the induction initialization

$$\beta_T(i) = 1$$

and the induction steps for $t = (T-1) \dots 1$ are

$$\beta_t(i) = \sum_{j=1}^{N} \left(\beta_{t+1}(j)\ a_{ij}\ b_j(o_{t+1})\right).$$

With these and the Forward probabilities $\alpha_i(t)$ computed for the Evaluation task, the Smoothing probabilities $\gamma_i(t)$ are then simply the normalized product of the corresponding Forward and Backward probabilities:

$$\gamma_i(t) = P(q_t = S_i | O) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

2.1.5 The Training Task

The final basic HMM task is Training: given an initial model and a trace of observations, modify the model to better explain the observations (according to the Evaluation probability). Formally, this means that given a model λ and an observation sequence O, find a better model λ' with

$$P(O|\lambda') \geq P(O|\lambda).$$

The Training task is usually solved with the Baum-Welch algorithm [2], which is an implementation of the Expectation-Maximization paradigm [19]. Its basic approach is to use the Smoothing probabilities to compute the probabilities of certain relevant paths of internal states given the observation (the so-called path-counting), and to derive new model parameters from these path probabilities. Since an HMM is defined by $\lambda = (A, B, \Pi)$, a Training algorithm has to compute new values for each of the three elements Π, B and A.

The first subtask is thus to compute a new vector of initial state probabilities $\overline{\Pi}$. Its probabilities are formally the probabilities for the system to be in each

state at time $t = 0$ given the observed trace O. Thus, the new initial state probabilities are simply the Smoothing probabilities at time $t = 0$, so that the new vector $\overline{\Pi}$ consists of elements

$$\overline{\pi}_i = \gamma_i(0).$$

The second subtask is to determine adjusted symbol emission probabilities, i.e. the probabilities that the system emits a particular symbol given that it is in a particular state. One interpretation of this probability with respect to a trace of observation is that it is the ratio of the expected number of times that the system was in a particular state and emitted a particular symbol to the expected number of times that the system was in that state at all. Since exactly one symbol is emitted in each time step, the values for both given a trace can be computed from the Smoothing probabilities as follows:

$$\overline{b_i(k)} = \frac{\sum\limits_{t \in \{1..T | o_t = V_k\}} \gamma_t(i)}{\sum\limits_{t \in \{1..T\}} \gamma_t(i)}$$

The final subtask is to determine new state change probabilities. These probabilities $\overline{a}_{ij}$ can each be computed as the ratio of the expected number of times that the model performed the state change from S_i to S_j, to the the expected number of times that the model could have performed that state change, i.e. the expected number of times that is was in state S_i. To determine the numerator, first the probabilities for each individual state change at each time step given the whole trace are computed as:

$$\xi_t(i,j) = P(q_t = S_i \cap q_{t+1} = S_j | O) = \frac{\alpha_t(i)\ a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j)}{\sum\limits_{k=1}^{N}\sum\limits_{l=1}^{N} \alpha_t(k)\ a_{kl}\ b_l(o_{t+1})\ \beta_{t+1}(l)}$$

With these, the ratio of expected values for $\overline{a}_{ij}$ is

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}.$$

With these three subtasks, the Baum-Welch algorithm has been proven to determine a new model that is more or at least equally likely to explain the observation as the original model. In practical applications, the Baum-Welch algorithm is applied iteratively to obtain better and better models until its results converge to a *locally* most likely model to explain observation sequence O.

Summary HMMs are well-defined, and efficient solution algorithms exist for all four basic tasks. However, HMMs are not directly applicable to problems sought to be solved in this work, because they cannot operate on continuous time and cannot model non-Markovian state changes.

Therefore, in the next section existing extensions to HMMs will be evaluated with respect to their ability to reconstruct the behavior of non-Markovian PODS models with concurrent activities.

2.2 Existing Extensions to HMMs

To allow for efficient behavior reconstruction algorithms, HMMs have a number of limitations, which render them inapplicable to the behavior reconstruction of non-Markovian PODS models with concurrent activities. But extensions to HMMs that lift these limitations may exist. In particular, for our application we require the following limitations of HMMs to be lifted:

- HMMs operate in discrete time and are restricted to emit exactly one symbol in each discrete time step. To reflect the behavior of real systems we require that symbols can be emitted at arbitrary points in time.
- Since HMM transitions are given by constant probabilities, their state sojourn times necessarily follow a geometric distribution - the discrete time counterpart of the Markovian exponential distribution. However, many real-life processes are not Markovian [1] and we therefore require that activity durations may be arbitrarily distributed.
- In addition, we want to model independent concurrent activities, which means that some information about the elapsed activity durations must be retained over a state change. HMMs circumvent this requirement, because their implicit memoryless geometric distribution is invariant to the elapsed time. But any non-Markovian extension of HMMs would need to explicitly take care of retaining this memory.

In this section, we survey existing extensions to HMMs with respect to whether they lift these three limitations of HMMs.

2.2.1 Explicit-Duration HMMs

One early approach to add non-Markovian state sojourn times to HMMs were Explicit-Duration HMMs by Ferguson [22] in 1980. In HMMs, the geometrically distributed state sojourn times are implictly given by the constant state transition probabilities. To allow for arbitrary sojourn time distributions, Ferguson's model explicitly specifies probability mass functions for each state to describe the distribution of state sojourn times. In this model, the constant state change probabilities describe only the probability that the system changes its state to a given successor state *after* the explicitly modeled sojourn time has elapsed.

With this simple approach, Ferguson adds a basic notion of non-Markovian activity durations to HMMs. This extension has been shown to be sufficient to model systems such as rainfall seasonality [72] and failures in telecommunication systems [71]. However, it is still limited to discrete time and cannot model concurrent processes, since only a single probility mass function describes the sojourn time of a state. Additionally, it adds the complication that each duration probability mass function increases the number of free parameters of the model by the distributions maximum duration and consequently makes the Training of such a model far more difficult.

2.2.2 HMMs with Discrete Phase-Type Durations

Instead of modelling the discrete state durations by an explicit probability mass function that complicates the Training task, several researchers attempted to

instead model durations without changing the definition of the state change behavior of HMMs. They modelled sojourn times of states through sequences of discrete HMM states that form discrete phase-type (DPH) distributions [59] whose times-to-absorption approximate the desired non-Markovian duration distributions.

These models have been variously called Expanded State HMMs [67], Inhomogenous HMMs [63], and "HSMMs with durations from the exponential family" [53]. These classes of models only differ in their approach to the Training task: they use different DPH distribution topologies (cf. [4] for a survey of those topologies), and some first train parametric continuous duration distributions and later convert those to DPH distributions while others directly train the DPH distributions.

Consequently, all approaches share the same limitations. While they all allow for non-Markovian duration distributions and they mitigate the Training problem of Explicit-Duration HMMs, they still operate in discrete time and thus cannot model time-continuous behavior.

2.2.3 HMMs with Continuous Time

Other researchers developed HMM extensions can operate on continuous time, where observations consist not only of the observed symbol, but also the exact time of the observation.

Levinson's Continuously Variable Duration HMMs (CVDHMMs) are a continuous time version of the Explicit-Duration HMMs, where the state sojourn durations are given by continuous gamma distributions instead of discrete probability mass functions [47, 48]. Salfner's Generalized Hidden Semi-Markov Models (GHSMMs) follow the same approach, but allow for the state durations to be arbitrary continuous probability distributions [69, 70]. And Wei et al. instead modify the original HMMs structure by replacing the constant state change probabilities with parameterized exponential distributions and thus do not need to explicitly model state durations [81]. They consequently call their class Continuous Time HMMs (CT-HMMs).

While all three approaches fulfill our first criterion of allowing state changes to occur at arbitrary points in time, they fail in either the second or the third criterion: CVDHMMs and GHSMMs both model state durations by a single duration probability distribution and therefore cannot model multiple concurrent independent activites. And CVDHMMs as well as CT-HMMs do not allow activity durations to follow arbitrary non-Markovian distributions.

2.2.4 Hidden non-Markovian Models

Hidden non-Markovian Models (HnMMs) by Krull [36] can be seen as an extension of CT-HMMs in that they also specify the dynamic behavior of the system by continuous state change probability distributions and do not explicitly specify a state sojourn duration distribution. However, in HnMMs, those state changes are not limited to the exponential distribution, but can be arbitrarily distributed. And the semantics of HnMMs are that state changes are caused by the completion of activities, which may have started before the system entered the current state. Thus, HnMMs allow for the modelling of truly non-Markovian

concurrent activities, where the completion of one activity causes a state change but need not impact the time to completion of the other activities.

Beyond that, HnMMs differ from all previously shown HMM derivates in two additional fundamental ways: First, symbol emissions are caused by the completion of activities, and thus the symbol emission probabilities are specified for the activities and not for the states reached by the completion of these activities. And second, HnMMs do not require all activities to emit observable symbols, they are supposed to reconstruct the behavior of the hidden system even if the completion of some activities in the real system went unnoticed.

So, ostensibly HnMMs fullfil all requirements that we set forth for a computational model that can reconstruct the behavior of non-Markovian PODS models with concurrent activities. But a deeper analysis reveals that general HnMMs are not practically applicable: First, no complete set of behavior reconstruction algorithms exist; merely partial algorithms for the Evaluation and Decoding tasks exist [36, 84]. And even for these, the same researchers have shown that their algorithms seem to have an exponential time complexity in the number of observations and thus are practically unfeasible for all but the shortest traces of observations. It seems that the general class of HnMMs is too extensive to allow for efficient behavior reconstruction algorithms.

Subclasses of HnMMs Beyond the general class of HnMMs, the authors also proposed several subclasses of HnMMs [39]. These are categorized based on three binary attributes:

- Certainty of symbol emissions: determines whether observable symbols are emitted by either all (E_{all}) or just by some (E_{some}) state changes.
- Number of direct connections between states: for a single discrete state, there is either at most a single (SC_{one}), or there can be multiple (SC_{nT}) ways of reaching another discrete state directly (i.e. by a single discrete state change).
- Activity continuation: activities in the model are either terminated (T_{reset}) or may continue (T_{keep}) after a discrete state change.

Different classes of HnMMs are conceivable through different combinations of these attributes. In [36, 84], the infeasibility of the HnMM algorithms has only been demonstrated for the broadest possible class $\{E_{some}, SC_{nT}, T_{keep}\}$ of HnMMs. It is possible that a more restricted class of HnMMs is limited enough to allow for practically feasible behavior reconstruction algorithms and at the same time is extensive enough to model systems with concurrent activities with arbitrary duration distributions as required for this work. However, as of yet no feasible algorithms exist for any of these classes.

2.2.5 Summary HMM Extensions

Table 2.1 summarizes the findings on the suitability of existing HMM derivates to the behavior reconstruction task for our class of models. The properties of the computational models that are sufficient for our modelling class are marked bold.

Conceptual Model	Definition of Activity Durations	Time Between Observations	Concurrent Activities
HMM	constant probabilities	discrete	**yes**
HMM with DPH	DPH distributions	discrete	no
CVDHMM	gamma distribution	**arbitrary**	no
CT-HMM	markovian distribution	**arbitrary**	**yes**
GHSMM	**arbitrary distributions**	**arbitrary**	no
HnMM	**arbitrary distributions**	**arbitrary**	**yes**

Table 2.1: Overview over existing approaches of extending HMMs.

As the table shows, all conceptual models but HnMMs lack abilities required to model concurrent non-Markovian PODS systems.

And the consequences of those deficiencies for the behavior reconstruction are severe: Limiting the type of probability distributions as well as only approximating the desired probability distributions with DPH distributions causes a major discrepancy between the model and the actual system and thus causes an inacceptable error in the behavior reconstruction. Similarly, models that can only handle discrete times of observations require a discretization of observations times, a process that reduces information available to the model and thus its accuracy. And finally, those models that cannot model concurrent activities can simply not represent systems that contain those, or would have to view them as not being independent. This again causes a major discrepancy between real system and model and results in unusably inaccurate behavior reconstruction results.

Hidden non-Markovian Models have none of these deficiencies, but their behavior reconstruction has been shown to not be practically feasible due to their computational complexity.

So, there is currently no computational model derived from HMMs that is able to practically reconstruct the behavior of non-Markovian PODS systems with concurrent activities. It might however be possible to select a sufficiently expressive subclass of HnMMs and to develop efficient algorithms for that class. This approach is further investigated in Chapter 3.

2.3 Alternative Approaches for Sequential Data Analysis

Other potentially viable techniques for the behavior reconstruction of non-Markovian PODS systems with concurrent activities may be found in the research field of sequential data analysis [20]. Those algorithms also process data sequences to find the underlying cause of the sequence. This is similar to the tasks for PODS systems, which emit temporal data sequences (sequences of observations) and the causes of the observations are to be reconstructed.

Maximum Entropy Markov Models (MEMMs) MEMMs [51] model systems with a similar behavior as HMMs: the models contain a discrete state space and is discrete in time. In each time step the model changes its state once and emits a single externally observable symbol.

But MEMMs were explicitly developed for systems in which an observed is no just an symbol, but is comprised of – potentially overlapping – features. All state transition probabilities follow the same maximum entropy probability distribution and the actual transition probability from that distribution is selected based only on the weighted sum (where the weights depend on the next state) of the features present in the current observation. In contrast, we require the probabilities of state changes to vary only based on the duration probability distributions for the individual activities.

So, MEMMs in their current form cannot model time-continuous behavior. And even if they could be extended to do so, all state change probabilities in an MEMMs still need to follow the same probability distribution in order for the MEMM training algorithms to work. This violates our requirements that activity durations may follow arbitrary continuous probability distributions.

Conditional Random Fields Conditional Random Fields [42, 80] are a generalization of MEMMs. While in MEMMs the next internal state of a model depends only on the current state and the next observation, there can be undirected dependencies between an internal state and its predecessor *and* successor in the special case of a linear chain CRF, and dependencies between an internal state and potentially all prior and later internal states in a general case CRF.

But with respect to our application CRFs share the limitations of MEMMs: they have no notion of continuous time and the dependencies between the internal states necessarily follow an exponential model, while we require continuous, arbitrarily distributed duration of activities to be modelled.

Bayesian Belief Networks Bayesian Belief Networks are a general way of graphically modelling conditional dependencies between random variables as a directed acyclic graph [28]. Using Bayes' theorem, joint probabilities of any subset of the network's random variables can be computed from the graph structure and the annotated conditional probabilities.

Hidden Markov models have been shown to be representable as a special case of BBNs. Here, each discrete internal state is only dependent on the previous internal state, and each observation is only dependent on the current internal state. From this graph structure, the usual HMM behavior reconstruction algorithms can be derived directly from the corresponding BBN structure [28].

But for non-Markovian models with concurrent processes whose behavior we which to reconstruct, the actual conditional dependencies are unclear. The next discrete state that the model will be in depends on the current discrete state, the next observation and the duration that each activity has been going on so far. And these possible activity durations in turn depend on how long the model has been in discrete states in which each activity was active, and thus potentially on all previous discrete states. Thus, to determine the probabilities of the model to reach each possible next discrete state using a BBN, one would have to evaluate all possible sequences of previous discrete states. The number of those sequences increases exponentially with each discrete state change, rendering the approach unfeasible for all but very short observation sequences.

Conclusion Maximum Entropy Markov Models, Conditional Random Fields and Bayesian Belief Networks are well suited to reconstruct the behavior of mod-

els based on non-temporal sequential observations. But none is easily extended to continuous time and non-Markovian activity durations.

2.4 Alternative Pattern Recognition Approaches

Pattern recognition approaches solve similar tasks to our Training and Evaluation tasks: real systems emit tuples of observations (the so-called "feature vector") and a set of these feature vectors is used to train a model of the behavior of the real system or to build a model that can discriminate between alternate models. For an arbitrary new feature vector the model (or models) can then be used to decide which real system most likely generated it.

But pattern recognition approaches make certain assumptions that are violated by our class of models.

Naive Bayes Naive Bayes classifiers use Bayes' theorem and the assumption that individual observations are statistically independent in order to compute the probability of each model to have created the feature vector [7]. But in our setting the completion of an activity determines which activities are performed and thus can be completed next, so that subsequent activity completions are correlated. And since the observations that comprise the feature vector are caused by the completion of activities, those are correlated as well. Thus, Naive Bayes classifiers are not applicable to our class of models.

K-Means, Decision Trees and SVMs K-Means clustering, decision trees, and support vector machines [86] are all classification approaches that attempt to subdivide the feature space (the space of all feature vectors of a given length) into regions where all feature vectors inside a region are attributed to a single cause, i.e. to same most likely model. The approaches only differ in the the way of subdividing the feature space: K-Means clustering uses Voronoi cells around sample points, decision trees use axis-aligned planes and support vector machines use arbitrary planes or arbitrary "warped" planes when using the optional "kernel trick".

In theory, those approaches could be trained with labeled example observation sequences to determine the location and label of the regions, i.e. the regions of the most likely generating real system. And the trained system could then be used to determine the most likely real system to have created the observation sequence, akin to the HMM Evaluation task. Yet, the recognition accuracy of decision trees on non-Markovian PODS systems even without concurrent activities has already been shown to be close to random guessing [9]. And this accuracy is unlikely to improve substantially for k-means clustering and SVMs, because the randomness inherent in non-Markovian PODS systems makes the structure of the true region boundaries highly complex and irregular, and neither K-Means clustering nor support vector machines approximate those very well.

Multilayer Perceptrons Multilayer Perceptrons (MLP) are artificial neural networks with a feedforward structure that – when fed with a n-dimensional

input vector (e.g. a sequence of n observations) – output an m-dimensional output vector (e.g. the probability that the input sequences was caused by each one of the m possible explainations) [73]. They have been shown to be able to approximate all continuous functions [64] and may thus also be able to reconstruct the behavior of non-Markovian PODS systems. But they, too, have been shown to only yield recognition accuracies close to random guesses on the behavior reconstruction of non-Markovian PODS systems [9]. This is likely to be due to the effect that even slight changes in the sequence of observations (e.g. the variation of a single observation time) can cause potentially dramatic changes in the probability that a given model could have created the trace. To train an MLP that accurately reflects such a system would likely require a sizable MLP and an unfeasibly extensive set of Training data which would additionally result in an unfeasible computation time needed for Training.

Conclusion So, well known pattern recognition algorithms are not viable approaches to the behavior reconstruction of non-Markovian PODS systems: Naive Bayes is not applicable since it requires the observations to be conditionally independent, and all other evaluated approaches have shown very poor accuracy in the behavior reconstruction of PODS systems (violating the success criterion of exactness formulated in Section 1.3) and are additionally limited to input of fixed size, i.e. the length of the observation traces has to be fixed during training. This means that the additional information contained in longer observation sequences is not available to those algorithms, and shorter observation sequences cannot be dealt with at all.

Furthermore it has been shown in previous sections that approaches for sequential data analysis are also unsuitable for the behavior reconstruction of non-Markovian PODS systems with concurrent activities. And existing extensions to HMMs can reconstruct the behavior of either non-Markovian PODS systems or PODS systems with concurrent activities, but cannot reconstruct the behavior of systems with both properties. Thus, no approach currently exists that can accurately reconstruct the behavior on non-Markovian PODS systems with concurrent activities. The most promosing course of actions to fill this gap appears to be to further extend HMMs and to develop dedicated behavior reconstruction algorithms for that extended model class.

2.5 Augmented Stochastic Petri Nets

The task of this thesis is to formally specify a *computational* model for the class of non-Markovian PODS systems with concurrent activities, and to develop efficient algorithms to reconstruct the behavior of those models. While a formal definition is necessary for the algorithms themselves, the models are better presented to reader in terms of a easily comprehensible *conceptual* model. For this work, we augment non-Markovian stochastic Petri nets as used in [36] as our class of conceptual models and introduce them in this section.

Stochastic Petri nets [55] are a class of conceptual models that can visualize various different kinds of discrete systems [5, 50]. The stochastic Petri net (SPN) as used in [36] has the following elements:

- Places drawn as hollow circles model physical locations or logical states.

- Tokens drawn as small filled circles located in places model entities in locations, or the system to be in a particular logical state. The distributions of tokens in places determines the discrete state of the SPN called its "marking".
- Timed transitions, drawn as narrow hollow rectangles model activities that "fire" a random time interval after they have been activated. This firing changes the marking of the SPN.
- Immediate transitions, drawn as narrow filled rectangles model an instantaneous change of the marking caused by a condition becoming true.
- Input arcs, arrows connecting places to transitions, determine which places need to be populated by tokens in order for the transition to become activated - and at the same time in which places tokens are destroyed when the transition fires.
- Output arcs, arrows connecting transitions to places, determine in which places tokens are created when the transition fires.
- Inhibitor arcs, drawn as lines connecting a place to a transition with a circular end at the transition disable that transition if the place contains enough tokens.
- Multiplicities can be applied to all arcs and determine the number of tokens relevant to the arc.

Semantics A transition in a Petri net is enabled when in all places connected to it by input arcs enough tokens according to the arc's multiplicity are present, and when all places connected to it by inhibitor arcs contain fewer tokens than the inhibitor arc's multiplicity. Enabled immediate transitions fire right away, while enabled timed transitions fire a random time interval (given by a continuous probability distribution) after being enabled. The firing of a transition destroys tokens in all places connected to it by input arcs according to the arcs' multiplicities, and creates new tokens in those places connected to it by output arcs again according to the arcs' multiplicities. The behavior of timed transitions when they are deactivated before they can fire depends on their *age policy*. Those with the default policy RACE_ENABLE forget the duration the transition has been active. Those marked to have the RACE_AGE policy memorize the elapsed time and continue from there once they are activated again.

Additional Restrictions and Augmentations SPNs as defined above can model arbitrary discrete stochastic systems. In this work, we slightly deviate from this definition in order to ensure that every system modelled as an SPN can be converted to a model that belongs to our new class of computational models, as will be defined in Chapter 3. To that end we require that the firing of any two concurrently active timed transitions must not result in the same marking. And only one immediate transition may be active at any given time.

The definition of SPNs so far does not reflect the stochastic symbol emissions that would enable the behavior reconstruction of a system modelled in such a way. We therefore augment SPNs to reflect these symbol emissions: for each

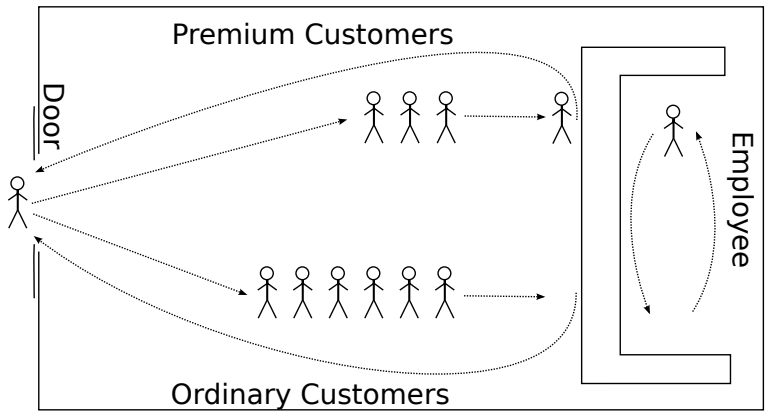

Figure 2.1: The Car Rental Agency model, shown schematically.

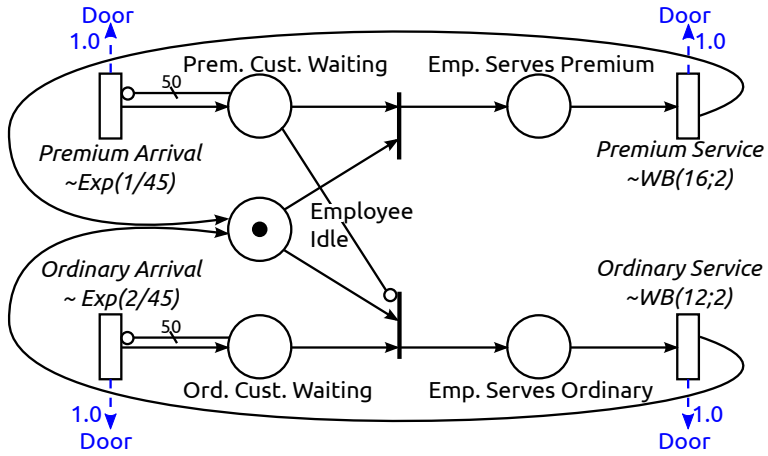

Figure 2.2: The Car Rental Agency model as an augmented SPN.

timed transition, each possible symbol emission is reflected by dashed arrows starting from the transition and is annotated with the symbol name and the probability that the symbol is emitted. For reasons that will become apparent in Chapter 3, we require the symbol emission probabilities for each timed transition to sum up to one. Immediate transitions may not emit symbols, since they always fire at the same time as timed transitions and the symbol to be emitted for that time is already determined by the timed transition.

We call this variant of SPNs an "augmented stochastic Petri net" (ASPN). It will be used throughout the remaining work to represent models of PODS systems.

Example Model: The Car Rental Agency One example model that will be used throughout this work is shown schematically in Figure 2.1 and as an ASPN in Figure 2.2. It models a car rental agency with a single employee. This particular agency serves ordinary as well as premium customers. All of these enter and leave through the same automatic sliding door. Thus, the door

operation protocol reflects the arrival and service completion of customers.

Inside the car rental agency, two separate lines are formed for ordinary and premium customers. Each of the two lines can hold at most 50 customers. Any arriving excess customer will be turned away.

The employee serves any premium customer before serving an ordinary customer, but will not interrupt the service of a customer once it has started. When the service of a customer has finished, that customer leaves right away. The inter-arrival times of customers as well as the service durations are all random, following known continuous probability distributions.

In the ASPN, the top left and bottom left places represent waiting customers. The place located between these indicates whether the employee is currently idle and thus ready to serve the next customer. Whenever the employee is idle and at least one customer is waiting, the next customer will be served. Here, the inhibitor arc ensures that premium customers have precedence. The rightmost places determine whether the employee currently serves an ordinary or a premium customer. When the service finishes the employee returns to the "ready" state. The firing of all timed transitions (arrival of a customer, service completion) causes the door of the car rental agency to open and thus the emission of the externally observable symbol "Door".

2.6 The Proxel Method

The Proxel simulation method [29, 43, 44] is an approach to simulate models of non-Markovian discrete stochastic systems with concurrent activities. It cannot reconstruct the behavior of those systems – as we would like to – , but its inner workings may provide insights into how behavior reconstruction algorithms can be developed for this class of models.

The Proxel method is a state-space based simulation algorithm that uses supplementary variables [16, 27] to handle non-Markovian activities. Its basic approach is to observe the model at equidistant points in time and for those times to determine the probabilities of the model to be in each possible state. Those probabilities are computed inductively by using the probabilities of every possible state from the previous point in time and computing the probabilities for every possible completion of a single activity during the time step in order to determine individual state probabilities for each state at the end of the time step.

The basic computational element of the Proxel simulation is a so-called Proxel, a tuple $P = (q, \vec{\tau}, p)$. It consists of a discrete marking[2] q representing a discrete state that the model is in, the age vector $\vec{\tau}$ containing the currently elapsed durations for all non-Markovian active or race-enabled activities, and the probability p of the model to be the state $(q, \vec{\tau})$ at the given point in time.

Algorithm A modelling expert has to supply a discrete stochastic model of the system to be simulated, the desired discrete time step size with which the simulation is to be carried out, and the set of Proxels representing the possible states the system may be in at time $t = 0$.

[2] In [29] the discrete marking has the symbol m. We use q here to conform with the notation introduced for HMMs so that later in this work elements of HMMs and Proxel simulation can easily be combined.

The simulation is then carried out inductively by iterating over all Proxels of one point in time and determining the possible successor Proxels for the next point in time by assuming that at most one activity finishes in each time step. The probability that no activity has been completed is based on the *hazard rate functions* [77] $\mu(t)$ of the probability distributions of all n enabled activities. It is given by the ordinary differential equation (ODE)

$$\frac{d\Pi_{sojourn}}{dt} = -\Pi_{sojourn} \sum_{i=1}^{n} \mu_i(t + \tau_i) \tag{2.1}$$

where the inital value $\Pi_{sojourn}(0)$ is the probability of the Proxel currently processed.

For all other Proxels representing the cases that a single activity has been completed, the successor's discrete state can be found in the model specification and its age vector is a modified version of the predecessor Proxel's age vector reflecting the passing of a time step and whether each activity has finished, is still going on, has been interrupted or has been cancelled during that time step. Its probability is derived using $\Pi_{sojourn}$ from Equation 2.1 as

$$\frac{d\Pi_{complete_i}}{dt} = \Pi_{sojourn}\ \mu_i(t + \tau_i).$$

Using these two formulas, the successor Proxels for all Proxels of a time step are computed, and the process is repeated for every time step until the desired simulation end time has been reached. In practical implementations the ODEs are solved using Euler's approximation method. And whenever the set of Proxels for a given point in time contains multiple Proxels with identical discrete state and age vector, they are merged into a single Proxel by summing up their probabilities. This Proxel merging limits the growth of the set of Proxels for each subsequent time step and makes the method computationally feasible.

While generally being more accurate than the Monte Carlo Simulation of a model, the Proxel method nevertheless computes only an approximation of the exact simulation results for two reasons: The method considers the completion of at most one activity per time step, even though a discrete stochastic model allows for multiple activities to be completed in any given time interval. And the Euler integration of the ODEs is only an approximation of the actual successor probabilities.

Conclusion The Proxel method computes an approximate simulation result for discrete stochastic systems. It is not directly applicable to the behavior reconstruction of non-Markovian PODS systems, but two of its concepts may be used to develop those behavior reconstruction algorithms: the concept of using Proxels containing age vectors to store the probabilities of the system to be in a given state at a given time, and the ODEs used to compute the probabilities of the model's stochastic behavior.

2.7 Summary

This chapter introduced Hidden Markov Models and the basic tasks Evaluation, Decoding, Smoothing and Training which can be solved on HMMs, and are

to be solved on our class of continuous time non-Markovian models. It was argued why existing machine learning approaches cannot solve these tasks on our class of systems. Furthermore, augmented stochastic Petri nets (ASPNs) were introduced as a conceptual model for our class of systems and the Proxel method was shown to be able to simulate possible behavior of the class of models for which we seek to reconstruct unobserved behavior; the Proxel method might therefore be a value tool in developing these behavior reconstruction algorithms.

The next chapter formally specifies the class of models of discrete stochastic models with concurrent activities of non-Markovian durations, and the following chapters then develop algorithms to solve the basic HMM tasks on that class of models.

Chapter 3

Defining Conversive Hidden non-Markovian Models

In this chapter the class of computational models to represent non-Markovian PODS systems with concurrent activities is determined. HnMMs (cf. Section 2.2.4) have been shown to be the only existing conceptual model able to faithfully represent non-Markovian concurrent activities, but also to elude efficient behavior reconstruction. We therefore select our class of conceptual models by imposing additional limitations on the definition of HnMMs, assuming that these will make it possible to develop efficient behavior reconstruction algorithms. We then proceed to derive a formal specification of this class of models and conclude the chapter by applying this specification to two example models in order to verify the usability of this specification.

3.1 Identifying an Adequate Conceptual Model

To model non-Markovian PODS systems with concurrent activities, we need most of the expressiveness of HnMMs, especially

- a discrete state space
- stochastic non-Markovian durations for activities
- activities that occur concurrently, but begin and end independently of each other
- symbol emissions at arbitrary points in time, resulting in observations to consist of the time stamp of the observation in addition to the observed symbol.

However, as was noted as a boundary condition for this work in Section 1.4, we content ourselves with the limitation that all internal state changes of the real system are assumed to emit an observable symbol. This limitation is not present in general HnMMs, but is otherwise shared by HMMs and all HMM extensions (cf. Section 2.2).

With this limitation in mind we are able to select a subclass of HnMMs as our computational model. This will be done by looking at the binary classification

attributes for HnMMs (cf. Section 2.2.4) and reasoning for each whether the more restrictive attribute value is sufficient for our purposes or whether the more permissive value needs to be selected. Our final class of computational models is then the subclass of HnMMs conforming to all selected attribute values.

Attribute "Certainty of Symbol Emissions" : This attribute determines whether all (E_{all}) or just some (E_{some}) of the state changes of the model cause an observable symbol to be emitted. As noted before, we expect our models to always emit a symbol whenever an activity ends and thus the discrete state of the model changes, conforming to the more limited attribute value E_{all}.

Attribute "Activity Continuation" : This attribute determines whether all activities cease whenever the system changes its discrete state (T_{reset}), or whether activities continue beyond state changes (T_{keep}). A defining aspect of our conceptual model is the representation of concurrent activities with non-Markovian durations. Those were explained to be activities whose times of occurence overlap, but which begin and end independent from each other. Having concurrent activities thus implies that the completion of one activity - which causes a discrete state change - does not affect the time of completion of the other activity. Thus, information of the elapsed time of activities has to be retained beyond discrete state changes, requiring the more permissive attribute realization T_{keep}.

Attribute "Number of direct connections between states" : This attribute determines whether there is exactly one (SC_{one}) or if there are multiple (SC_{nT}) activities that can lead from one discrete state to a particular other one. None of our requirements to a class of computational models is affected by this attribute. So we chose to determine the realization of this attribute with regard to conciseness and intelligibility of the resulting notation: For classical HMMs and their extensions, the behavior of changing the system state from one state to another is given by a single entry in the state transition matrix, and is therefore inherently limited to SC_{one}. In order to also be able to adopt this concise matrix notation for our class of models and therefore ensure intelligibility of the algorithm descriptions, we decided to also select the more restricted attribute value SC_{one}. The definition of ASPNs as the conceptual model for this class of computational models also contains the corresponding limitation (cf. Section 2.5).

Naming the Class With all attribute values being selected we consequently define our class of conceptual models for the behavior reconstruction of non-Markovian PODS systems with concurrent activities as the class of HnMM models conforming to the attributes E_{all}, T_{reset} and SC_{one}. We call this class

Conversive Hidden non-Markovian Models

(CHnMMs), since – due to E_{all} – the system communicates the times of all internal state changes.

3.2 Limitations of CHnMMs Compared to HnMMs

This subclass was defined as narrowly as possible in order to enable the development of efficient behavior reconstruction algorithms. While the attribute values were selected so that all non-Markovian PODS systems with concurrrent activities that conform to our boundary condition can still be modelled, it is worth discussing which other types of systems are excluded by these limitations.

By selecting the attribute value E_{all} we exclude all systems where any internal state change is not externally observable. In many complex systems, at least some activities are completely hidden from an outside observer, making it impossible to model them as a CHnMM. Ostensibly, it may seem possible to convert general HnMMs to CHnMMs by introducing a *null* symbol and have every state change that would emit no symbol emit that *null* symbol instead, thereby conforming to E_{all}. But behavior reconstruction always requires a complete trace of observations in addition to a model. And thus a complete trace would have to include the times at which unobservable state changes occured and emitted the *null* symbol. Yet, since those state changes where unobservable to begin with, it is impossible - even when accepting increased computational complexity - to convert a general HnMM to a CHnMM.

The attribute value T_{keep} does not impose any limitations on the expressiveness of CHnMMs, since it is more permissive than the alternative T_{reset}. The latter would have prevented the modelling of concurrent activities and together with E_{all} would have resulted in the models to be GHSMMs, for which efficient algorithms already exist [69, 70].

The final attribute value SC_{one} imposes slight limitations on the modelling power of CHnMMs as it prevents multiple activities from having the same consequences. However, the class SC_{one} was selected only for conciseness of the formal specification and the algorithms developed in the subsequent chapters. It has no practical impact: First, all algorithms developed could be applied to SC_{nT} models as well, one only needs to change the data representation, but algorithm complexity and formulas remain unchanged. And second, SC_{nT} models can be converted to SC_{one} without loss of functionality: whenever the completion of two activities would lead to the same discrete state, one simply duplicates that state and lets each activity lead to a different copy.

Thus, the only real limitation on the applicability of CHnMMs compared to HnMMs is to require that the completion of all activities is externally observable. This limitation is thought to be necessary, because it is essential for efficient behavior reconstruction.

In the next section, the informally determined CHnMM class will be defined formally.

3.3 Formal Specification

The formal specification of CHnMMs will be split into two parts: First, we specify the elements of the original HMMs (cf. Section 2.1) that are adequate for CHnMMs as well. Afterwards, the new addition that are specific to CHnMMs are specified. The parts of the HMM specification that apply to CHnMMs as well are:

- The internal behavior of the model causes the emission of symbols from the set $\{V_1, \ldots, V_M\}$, which can be observed externally.
- CHnMMs have a discrete state space consisting of N *states* $S_1, \ldots, S_N$. However, in contrast to HMMs, for CHnMMs the discrete state no longer fully specifies the current model state alone; the elapsed durations of all activities has to be included in a state description as well.
- The discrete state after the tth symbol emission is represented by the symbol $q_t \in \{S_1, \ldots S_N\}$
- The probability of the model to be in each of the N states at time $t = 0$ is given by an *initial probability vector* $\Pi \in \mathbb{R}^N$ with elements $\pi_i = P(q_0 = S_i)$. Since all elements are probabilities, $\forall i : \pi_i \geq 0$ and $(\sum_i \pi_i) = 1$ have to hold.
- The state change behavior of the model is given by a matrix A with size $N \times N$, whose entries a_{ij} describe the conditions under which the state change from discrete state S_i to discrete state S_j occurs. However, for CHnMMs the matrix elements are no longer numbers, but have to contain more complex specifications of the activities whose completion changes the discrete state. The exact definition of this state transition matrix will be given in the following.

Furthermore, CHnMMs have additional properties beyond HMMs:

- Since CHnMMs model systems with activities of non-Markovian durations, they can no longer be specified by constant probabilities. Instead, the state change behavior of the model is directly specified by a set of K activities $TR = \{TR_1, \ldots, TR_K\}$, of which each activity TR_i [1] is specified by a tuple $(dist, id, b(v), aging)$:
  - $dist$ specifies the continuous probability distribution that determines the duration of the activity until it is completed and causes a discrete state change.
  - $id \in \mathbb{N}$ is a unique identifier of the state transition with the fixed value $TR_i.id = i$. The id is used to track activities that begin in one discrete state and continue in another. This is necessary since CHnMMs are of type T_{keep} and thus the probabilitiy that an activity causes the next discrete state change depends on how long it has been active in previous discrete states.
  - $b(v) : \{V_1, \ldots, V_k\} \mapsto \mathbb{R}_0^+$ is a function that determines the probability with which the activity emits each symbol when causing a discrete state change. Since CHnMMs fall into the HnMM class E_{all}, each activity is guaranteed to emit exactly one symbol upon causing a state change, formally requiring that

$$\forall i \in \{1, \ldots, K\} : \left(\sum_{j=1}^{M} TR_i.b(V_j) \right) = 1$$

[1] The more obvious choice of a symbol for activities would be A_i, but A is already used for the state transition matrix in accordance with the HMM specification. TR_i is used instead, since activities cause discrete state **tr**ansitions.

This definition mimics the definition of the symbol emission probabilities in HMMs (cf. Section 2.1).

- $aging \in \{true, false\}$. An activity may be interrupted by a state change, i.e. it may be active in one discrete state and inactive in the next one without having caused the state change and thus without having been completed. In those cases the value $aging$ determines whether the activity continues when it is activated again ($= true$) or whether it restarts from the beginning ($= false$)

- The state transition matrix A specifies the discrete state change behavior of the model. For HMMs it contained constant probabilities. Since in CHnMMs discrete state changes are caused by the completion of activities, here the matrix A contains the activities from TR. More concretely, every matrix entry a_{ij} of the state transition matrix A is either the one activity from $\{TR_1, \ldots, TR_K\}$ that causes the state change from S_i to S_j, or the special symbol $\varnothing$ when no such state change is possible.

  One additional limitation on the matrix A is that activities cannot at the same time cause and not cause the next state change. To that end, each activity can be active at most once per discrete state and thus appear at most once in each row of A, formally requiring that

  $$\forall i,\ \forall j \neq l : a_{ij} = TR_k \Rightarrow a_{il} \neq TR_k$$

  Note that ASPNs (cf. Section 2.5) as a conceptual model for CHnMMs also enforce this limitation by requiring that at most one immediate transition is active at any given time and thus when an activity ends there is exactly one possible outcome.

All these symbols together define the behavior of a CHnMM. But since the states S_i and symbols V_j are merely names that do not affect the model behavior, and since all elements of TR are also present in A, a CHnMM λ is already fully specified by the matrix A and the initial probability vector Π, i.e. $\lambda = (A, \Pi)$.

3.3.1 Semantics

The semantics of a CHnMM specified that way are as follows: At any given time, the model is in one of the specified discrete states. Depending on that state, multiple activities may be ongoing at the same time. The durations of all activities are random (distributed according to known probability distributions) and depend on how long – if at all – the activities have already lasted in previous discrete states. The first activity to finish causes the next state change to another discrete state, and also the emission of an observable symbol. In this new state another set of activities is active. The activities from the old state may either continue in the new state, be interrupted to continue in a future discrete state (for *aging* activities), or may be terminated and restart in a future state. The activities that are active in the new state may either continue from the previous state or from the time when they were last interrupted by a state change (for *aging* activities), or may restart.

3.3.2 Additional Notation

In addition to the specification of CHnMMs, the following notation will be used in this work in conjunction with CHnMMs:

- An observation o of a symbol emission is a tuple $o = (v, e)$, where $v \in \{V_1, \ldots, V_M\}$ specifies the observed symbol and e specifies the time of the symbol emission.
- A *trace* $O = o_1 o_2 \ldots o_T$ is a sequence of observations in chronological order, i.e. $\forall i < j : o_i.e < o_j.e$.
- For an activity TR_i with duration probability distribution $dist$, the functions $pdf(dist)$,$cdf(dist)$ and $hrf(dist)$ define the probability density function, cumulative distribution function and hazard rate function of the distribution, respectively. For example, $cdf(TR_i.dist)(5)$ returns the value of TR_i's cdf at position $t = 5$.
- For a probability distribution $dist$, the boolean function $isExp(dist)$ returns whether the probability distribution is an exponential distribution and therefore Markovian.

This concludes the definition of CHnMMs and the introduction of additional notation that is used throughout the remaining work. For quick reference, all definitions given in this section are listed concisely in Appendix C. In the next sections we will show that these definitions are able to represent realistic systems by showing how realistic problems informally specified as ASPNs can be converted to strictly formal CHnMMs.

3.4 Conversion of ASPNs to CHnMMs

So far we have defined ASPNs as our class of conceptual models in which practitioners can model their behavior reconstruction problems. We have also defined CHnMMs as our class of computational models which will be used to solve these behavior reconstruction tasks. However, we have yet to show how to convert the former into the latter.

Algorithms already exist for converting a generalized stochastic Petri net (GSPN) to a continuous time Markov chain (CTMC) [50]. Our classes of ASPNs and CHnMMs are extensions of GSPNs and CTMCs, respectively, augmented by the ability to allow for non-Markovian durations and the emission of externally observable symbols. Thus, the following procedure for converting an ASPN to a CHnMM is quite similar to the existing approach.

A general correspondence exist between ASPNs and CHnMM in that ASPN markings correspond to CHnMM states and ASPN timed transitions correspond to CHnMM activities. With these in mind, the idea of the algorithm to convert an ASPN to a CHnMM is as follows:

1. From the known initial marking of the ASPN recursively determine all N reachable tangible markings, i.e. reachable markings in which no immediate transition is active. In theory, the number of tangible markings in an ASPN may be infinite, but in practical applications usually only a finite

set of markings has non-zero probabilities[2]. For each of those markings of the ASPN, a discrete state S_i is created for the CHnMM.

2. For each of the M unique symbols that the timed transitions of the ASPN emit, a corresponding CHnMM symbol V_i is created.

3. For each of the K timed transitions in the ASPN, a CHnMM activity TR_i is created, whose elements $(dist, id, b(v), aging)$ are given as follows:

   - $dist$ is the duration probability distribution of the timed transition
   - id of the ith activity is i in accordance with the definition of CHnMMs
   - $b(v)$, the mapping from emitted symbol to probability, is directly taken from the symbol emission probabilities specified for the corresponding ASPN timed transition
   - $aging$ is $true$ when the ASPN transition is of type RACE_AGE (cf. Section 2.5), and $false$ otherwise.

4. The *initial probability vector* Π of length N (the number of discrete states in the CHnMM and equivalently the number of tangible markings in the ASPN) has a probability value of one for the discrete state S_i that corresponds to the known initial marking of the ASPN, and probability values of zero for all other entries.

5. The state transition matrix A of dimension $N \times N$ is initially filled with the symbol $\varnothing$ in each of its entries. Then, for each of the tangible markings S_i and each timed transition TR_k that is enabled in said marking, the tangible marking S_j is determined that is reached when transition TR_k fires in S_i and all subsequently active immediate transitions in subsequent markings of the same time step fire as well. With this found S_j, the entry a_{ij} of A is set to TR_k. Since the ASPN definition requires that only one immediate transition is active at a given time, the reached tangible marking is unique and thus can be determined[3].

This procedure converts an ASPN to a CHnMM $\lambda = (A, \Pi)$, which can then be used in conjunction with the algorithms developed in the following chapters to reconstruct the unobserved system behavior.

3.5 Example Models Defined as CHnMMs

We will now use this procedure of converting ASPNs to CHnMMs in order to exemplarily specify two CHnMMs of realistic systems with partially observable

[2]For ASPNs with an infinite number of reachable markings no CHnMM can be built. However, for these, exact behavior reconstruction as attempted in this work would not be possible in any case, since exact reconstruction would need to take into account all of the infinite number of markings, which is not feasible.

[3]One exception is that an infinite sequence of immediate transitions may fire after a single timed transition has fired and thus no tangible state is ever reached. In this case the ASPN is deadlocked, no tangible target state exists and the ASPN cannot be converted to a CHnMM. However, such deadlocking ASPNs are not practically relevant since the deadlock means that the system can never progress beyond the point in time at which the deadlock occurred, which does not occur in real-life systems.

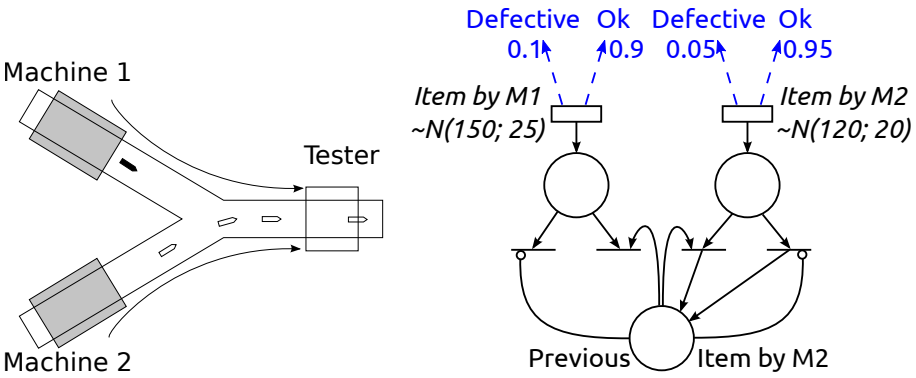

Figure 3.1: The quality tester model, shown schematically (left) and as an augmented SPN (right).

behavior. These two CHnMMs will also be used repeatedly in the remainder of this work to experimentally test the algorithms developed.

The two systems were selected to show potential practical applications of CHnMMs and at the same time cover different kinds of CHnMMs: They differ in the model complexity, where one model has only a single marking, whereas the other one has about 5000 reachable markings. And they differ in the amount of information present in the observations: In one model, different transitions emit symbols with different probabilities, so that the type of emitted symbol contains information on what activity is more likely to just have been completed. In the other one, every transition emits the same single symbol so that the information of the observation sequence lies only in the times of the symbol emissions, but not in the symbols themselves.

3.5.1 The Quality Tester

The "Tester" example model represents a partially observable part of a production line (cf. Figure 3.1): Two machines are imperfectly producing indistinguishable items, causing some items to be defective. Producing an item takes a random amount of time and the two machines have different continuous probability distributions to describe this randomness, because they are different models.

The items are then put on conveyor belts to be fed to a single automatic quality tester for quality control. This tester logs the time of each quality test and the test result ("ok" or "defective"). Since the time that the items spend on the conveyor belts is known, constant, and identical for both machines, the times at which each item was produced can directly be derived from the test protocol. This yields a production protocol containing the times of item production and the quality predicate ("ok" or "defective") of each item produced.

In this scenario, the owner or the operator of the production facility may be interested in finding out which of the two machines has a higher ratio of defective items. For safety reasons, the area between the two machines and the tester is unobservable and thus the only source of even partial observations of the system behavior is the quality test protocol. Determining the source of defective items thus requires behavior reconstruction and thus prior to that the

modelling of the system as an ASPN and a conversion to an CHnMM.

The simple schematic version of this model (cf. left hand side of Figure 3.1) can almost directly be converted into an ASPN, since the system has only a single discrete state and two distinct activities (the production of items by the two machines) that are always active. The only obstacle is that independently of which machine has produced an item, the system is alwas in the same marking, which violates one of the limitations of ASPNs. As a remedy, the procedure developed to circumvent the SC_{one} limitation of CHnMMs can be applied (cf. Section 3.2): by storing the source of the previously produced item the single marking of this system is duplicated and the firing of the state transition representing the two machines are made to reach different clone markings. The resulting ASPN can be seen on the right-hand side of Figure 3.1).

With the procedure outlined in Section 3.4, this ASPN can then be converted to the following CHnMM:

- States: $S_{prev\_from\_1}, S_{prev\_from\_2}$
- Symbols: $V_{ok}, V_{defective}$
- Activities:
  - TR_{M1}: $(\sim N(150, 25), 0, V_{ok} \rightarrow 0.9; V_{defective} \rightarrow 0.1, false)$
  - TR_{M2}: $(\sim N(120, 20), 1, V_{ok} \rightarrow 0.95; V_{defective} \rightarrow 0.05, false)$
- State transition matrix $A = \begin{pmatrix} TR_{M_1} & TR_{M_2} \\ TR_{M_1} & TR_{M_2} \end{pmatrix}$
- Initial probability vector $\Pi = (1, 0)$

With only two discrete states and two concurrent activities, this model is close to the most simple CHnMM imaginable. It can thus be used to illustrate the upper limit of computational performance for the CHnMM algorithms to be developed, and is used as an application example for those algorithms.

3.5.2 The Car Rental Agency

The Car Rental Agency model (cf. Figure 2.2 on page 21) that models the behavior of customers and the only employee in said rental agency has already been introduced in Section 2.5 as an example of ASPNs. In this section, we only present the so far omitted representation of this system as a CHnMM.

In this model, a marking consists of the numbers of ordinary and premium customers currently standing in line, as well as the type (ordinary or premium) of the customer who is currently being served. With the system allowing for up to 50 customers per queue the resulting CHnMM will have more than 5000 discrete states and the state transition matrix consequently more than 25 million entries. In order to give a feasible representation of the state space we use the set-builder notation. And we only list those elements of the state transition matrix A that are not equal to $\varnothing$.

Discrete States We name states using a tuple (X, i, j) where $X \in \{O, P\}$ denotes whether the currently served customer is an ordinary (O) or a premium (P) customer, i denotes the number of waiting ordinary and j the number of waiting premium customers. The set of discrete states of the Car Rental Agency model is then

$$\{S_{idle}\} \cup \{S_{O,i,j} | i, j \in \{0 \dots 50\}\} \cup \{S_{P,i,j} | i, j \in \{0 \dots 50\}\}$$

Set of Symbols The set of externally observable symbols contains only a single symbol V_{Door}, since all discrete state changes (arrival and service completion of a customer) cause the door of the rental station to open.

Activities The only activities than occur in this model are the arrivals and service completions of ordinary and premium customers, respectively:

- $TR_{premiumArrive}$: $(\sim Exp(1/45), 0, V_{Door} \to 1, true)$
- $TR_{premiumService}$: $(\sim Weibull(16, 2), 1, V_{Door} \to 1, true)$
- $TR_{ordinaryArrive}$: $(\sim Exp(2/45), 2, V_{Door} \to 1, true)$
- $TR_{ordinaryService}$: $(\sim Weibull(12, 2), 3, V_{Door} \to 1, true)$

State Transition Matrix Since it is impractical to given the whole state transition matrix with its more than 25 million entries, we only list those entries that represent possible state changes, i.e. matrix entries that do not contain the special symbol $\varnothing$.

- $S_{O,0,0} \Rightarrow S_{idle} : TR_{ordinaryService}$
- $S_{P,0,0} \Rightarrow S_{idle} : TR_{premiumService}$
- $S_{idle} \Rightarrow S_{O,0,0} : TR_{ordinaryArrive}$
- $S_{idle} \Rightarrow S_{P,0,0} : TR_{premiumArrive}$
- $\forall i \in \{0 \dots 50\}, j \in \{0 \dots 49\} : S_{O,i,j} \Rightarrow S_{O,i,j+1} : TR_{premiumArrive}$
- $\forall i \in \{0 \dots 49\}, j \in \{0 \dots 50\} : S_{O,i,j} \Rightarrow S_{O,i+1,j} : TR_{ordinaryArrive}$
- $\forall i \in \{0 \dots 50\}, j \in \{1 \dots 50\} : S_{O,i,j} \Rightarrow S_{P,i,j-1} : TR_{ordinaryService}$
- $\forall i \in \{1 \dots 50\} : S_{O,i,0} \Rightarrow S_{O,i-1,0} : TR_{ordinaryService}$
- $\forall i \in \{0 \dots 50\}, j \in \{0 \dots 49\} : S_{P,i,j} \Rightarrow S_{P,i,j+1} : TR_{premiumArrive}$
- $\forall i \in \{0 \dots 49\}, j \in \{0 \dots 50\} : S_{P,i,j} \Rightarrow S_{P,i+1,j} : TR_{ordinaryArrive}$
- $\forall i \in \{0 \dots 50\}, j \in \{1 \dots 50\} : S_{P,i,j} \Rightarrow S_{P,i,j-1} : TR_{premiumService}$
- $\forall i \in \{1 \dots 50\} : S_{P,i,0} \Rightarrow S_{O,i-1,0} : TR_{premiumService}$

Initial Probability Vector The initial probability vector Π contains the probability 1.0 for the initial marking (no customers are present in the rental station and the clerk is thus idle) and 0.0 probability otherwise:

$$\Pi : \pi_i = \begin{cases} 1.0 & \text{if } S_i = S_{idle} \\ 0.0 & \text{otherwise} \end{cases}$$

This model has an extensive discrete state space 2500 times the size of the Tester state space. And the model has three concurrent activities (two arrival activities and the service of either an ordinary or a premium customer) in most discrete states. Thus, computation time and memory consumption will likely be far higher for this model than for the Tester model. The Car Rental Agency model is therefore used in this work to test the limits of practical applicability of the algorithms developed.

3.6 Summary

In this chapter, the requirements to a class of computational models that can represent the behavior of partially observable discrete stochastic models with concurrent non-Markovian activities have been analyzed, and a computational model - called Conversive Hidden non-Markovian Models - has been specified. Finally, two example models of that class have been introduced informally, and were specified formally by the means of CHnMMs, supporting the statement that the definition of CHnMMs is usable to specify real-life systems.

Due to being a subclass of HnMMs, the existing partial HnMM behavior reconstruction algorithms would be applicable to the class of CHnMMs as well. However, since the HnMM algorithms have been shown to not be practically feasible, the remainder of this work will be concerned with the development of behavior reconstruction algorithms tailored specifically to CHnMMs. It is assumed that the restrictions of the class of CHnMMs compared to HnMMs makes those algorithms feasible.

In the following chapters, the specification presented in this chapter will be used to develop solution algorithms to the four basic behavior reconstruction tasks Evaluation, Decoding, Smoothing and Training for CHnMMs.

Chapter 4

The Evaluation Task

4.1 Introduction

In this chapter a solution algorithm for the CHnMM Evaluation task will be developed. Evaluation is the task of determining the probability that a given observation sequence was caused by a given model, or formally [66] to determine[1]:

$$P(O|\lambda)$$

A typical practical application for Evaluation of HMMs is pattern recognition [23]: For each pattern (e.g. drawn gestures, handwriting, movement gestures [17, 18] or even a microphone to be classified [14]) to be recognized, a separate model is built. For such a pattern recognition task an observation sequence of the user input consists of characteristic points of the input such as direction changes and zero crossings. Then, to determine the pattern the user attempted to convey, the Evaluation probability of the observation sequence is computed for each model in turn, and the model with the highest probability is taken as the most likely user input.

To solve the Evaluation task for CHnMMs, we will extend the inductive HMM Evaluation algorithm (The so-called "Forward algorithm", cf. Section 2.1.2) with elements of the Proxel simulation method (cf. Section 2.6), and we will develop further additions that ensure the computation of exact probabilities where the Proxel method only provides approximations.

The next section will give a detailed overview of which parts from the HMM Forward algorithm and the Proxel simulation algorithm can be reused, and which have to be developed from scratch.

4.2 Existing Groundwork and Unresolved Issues

Applicable Concepts from the HMM Forward Algorithm As with HMMs, it appears to be futile for CHnMMs to follow all possible paths of internal system behaviors that are consistent with a given observation sequence: The

[1] In the remainder of this work the explicit conditioning of probabilities on the model λ is omitted when it is clear from the context that the desired probabilities are computed given the current model

number of those paths would increase exponentially in the number of individual observations in the trace.

Instead, the basic inductive approach of the HMM Forward algorithm appears to be applicable to CHnMMs as well: The initial probability vector Π can be interpreted as a vector of initial *joint probabilities* of the model to be in each state at time $t = 0$ *and* having emitted all observations made so far, since at $t = 0$ no observation has yet been made and thus the latter is always true. If an induction step could be developed that uses the set of those joint probabilities for the partial observation sequence of the first n observations to compute them for the first $n + 1$ observations, then this induction could be applied iteratively for the whole trace of observations. The result of this induction would be individual probabilities of the model to be in each reachable state and having emitted all of the observations. The sum of those probabilities would be the probability of the system to be in any state and having emitted the whole trace, and thus the desired Evaluation probability.

Applicable Concepts from the Proxel Method While the overall approach of the HMM Forward algorithm seems to be suitable for CHnMMs as well, computation of the actual joint probabilities has to differ in order to account for the non-Markovian concurrent behavior of CHnMMs. Some approaches to compute this behavior can be taken from the Proxel method, which was developed to *simulate* concurrent non-Markovian behavior (cf. Section 2.6):

1. While the state transition probabilities in HMMs were constant, for CHnMMs they vary following a continuous duration probability distribution. The Proxel method provides formulas to compute approximations of these state transition probabilities and provides the mathematical background on which exact formulas may be derived.

2. HMMs were inherently Markovian, meaning that the future behavior of an HMM only depends on its current discrete state and not on its past. Thus, the inductive forward algorithm could compute the joint probabilities after the n+1th observation solely based on the joint probabilities of the discrete states and the nth observation.

   In CHnMMs, activities are non-Markovian. Their state transition probabilities depend on how long the activity has been going on. So, to predict future behavior it is insufficient to know the current discrete state of the system, one also has to retain detailed information on the past. This ostensibly makes it impossible to develop an efficient inductive algorithm where the current system state alone is able to predict the future behavior.

   The Proxel simulation method has the same challenge, and solved it through an extension of the state description. In the Proxel algorithm, the current system state consists not only of the discrete state, but is extended by *supplementary variables* storing the durations for which each activity has been going on so far (the so-called "age vector"). This extended state description fully encodes the relevant history into the state and thus completely determines the possible future behavior of the model.

   For CHnMMs, basing an inductive Forward algorithm on this extended state description would allow it to again compute the joint probabilities

of the $n+1$th symbol emission based only on the joint probabilities of the nth symbol emission.

3. In an HMM, the set of states is constant. Thus, the joint probabilities for each time step can be stored in a fixed size vector.

   In the Proxel simulation and for CHnMMs, each element of the extended state's age vector can potentially take on any positive real value, meaning that the set of possible states is infinite and cannot be stored. For each time of a symbol emission, the set of states that have non-zero probability however *is* finite, but the actual states in that set vary from observation to observation and are not predictable beforehand. Thus, storage in a fixed-size vector is not possible.

   The Proxel method provides a solution for this problem which is applicable to CHnMMs as well: instead of using a single vector to store the association between each state and its probability, the Proxel method stores each individual state-probability association in a separate tuple, a so-called Proxel. For each time step, the method maintains a set of arbitrary size of those Proxels with non-zero probability.

4. Since the state description of a Proxel is sufficient to completely predict its future, two Proxels with identical state (i.e. identical discrete state and age vector) share the same future. Thus, the Proxel method - and any algorithm on CHnMMs - can *merge* those Proxels with identical state by adding their probabilities. This merging significantly reduces the computational cost of the approach, since through it an inductive CHnMM forward algorithm has to compute the probability of every possible behavior only once.

Unresolved Issues So, the HMM Forward algorithm provides the basic algorithm structure of an algorithm solving the CHnMM Evaluation task, and the Proxel method provides concepts on how to structure the data for that algorithm. Yet, some aspects of a CHnMM Evaluation task go beyond the HMM Forward algorithm and the Proxel simulation method:

First, the core formula of the Proxel is the approximation of discrete state change probabilities during the interval of a single time step. For CHnMMs where the trace of observations is given, the exact times at which discrete state changes occur are known in advance and no state changes can occur in the time intervals between them.

Thus, the Proxel transition probability computation cannot be used for CHnMMs, and a new one to compute the conditional state transition probabilities at exact points in time (the times of the observations) given the observations has to be derived.

Second, the computation of the probability for each reachable state will be performed inductively over the times of symbol emissions. In CHnMMs, these intervals are of variable length whereas the Proxel method and HMMs work with equidistant time steps. However, such a modification to variable time steps has already been attempted for the Proxel method [37, 82, 83] itself. They have shown to only impact the numerical domain of the age values (which are no longer always multiples of the fixed time step size, but arbitrary positive real values) and the computation of the state transition probabilities. Thus,

the only difficulty here is to ensure that the formulas for the state transition probabilities, which are derived in the next section, work correctly even under symbol emissions at arbitrary times.

4.3 Computing Exact State Change Probabilities

An inductive CHnMM Forward algorithm would be provided with the joint probabilities $P(q_t = S_i \cap age_t = \tau \cap o_1 \dots o_t)$ of the model at the time of the tth symbol emission ($o_t.e$) to be in each state (discrete state S_i and age vector τ) and to have emitted the first t elements of the observed trace. Its task is then to use those to compute the probabilities $P(q_{t+1} = S_j \cap age_{t+1} = \tau' \cap o_1 \dots o_{t+1})$ of the model at the time of the $t+1$th symbol emission ($o_{t+1}.e$) to be in each possible successor state (discrete state S_j and age vector τ') and have emitted the first $t+1$ elements of the trace.

The latter is the product of the prior multiplied by the probability that the first discrete state change after $o_t.e$ is at $o_{t+1}.e$, changes the discrete state from S_i to S_j, and emits the symbol $o_{t+1}.v$. Using the definition of conditional probability and taking into account that the emission of a symbol depends only on the actual activity that causes the symbol-emitting state change, this state change probability can be split into the product of three individual probabilities:

1. The *sojourn* probability that no activity is completed between $o_t.e$ and $o_{t+1}.e$
2. The *change* probability that the state change from discrete state S_i to S_j occurs exactly at time $o_{t+1}.e$, given that no state change has occured between $o_t.e$ and $o_{t+1}.e$
3. The probability that the signal $o_{t+1}.v$ is emitted given that the state change between states S_i and S_j has occured

The third probability is given by the specification of the CHnMM as $a_{ij}.b(o_{t+1}.v)$, but the other two have yet to be derived.

4.3.1 State Sojourn Probability

The first formula to be derived is the probability that no state change occurs (i.e. no activity is completed) between the current and the next symbol emission. For intelligibility, we call the time of the current symbol emission $o_t.e$ simply t_{old} and the time of the next symbol emission $o_{t+1}.e$ simply t_{new}.

Let $\{TR_1, TR_2, ..., TR_n\}$ be the set of activities that may occur in the current state S_i with age vector $\vec{\tau}$. For each TR_i, its stochastic duration is given by the corresponding cdf F_i. Furthermore, none of the activities have been completed until time t_{old} (otherwise the activity would not be active in the current state). And at t_{old}, each activity TR_i has been active for a duration of τ_i.

We are interested in the state sojourn probability between t_{old} and t_{new}, i.e. the probability that no activity finishes until t_{new}, given that no activity has finished up until t_{old}:

$$P_{sojourn} = P(\forall 1 \leq i \leq n : TR_i > t_{new} | \forall 1 \leq i \leq n : TR_i > t_{old})$$

In this formula, $TR_i > t_{new}$ means that the activity TR_i is completed later than t_{new}.

Since the completion of the activities are statistically independent (the time at which activity TR_i finishes does not depend on any other activity), this probability can be written as the product of the individual sojourn probabilities:

$$P_{sojourn} = \prod_{i=1}^{n} P(TR_i > t_{new} | TR_i > t_{old})$$

According to the definition of conditional probability [54], this can be written as:

$$P_{sojourn} = \prod_{i=1}^{n} \frac{P(TR_i > t_{new} \cap TR_i > t_{old})}{P(TR_i > t_{old})}$$

Here, $TR_i > t_{new}$ is a subset of $TR_i > t_{old}$ (whenever TR_i finishes later than t_{new}, it also has to finish later than t_{old} since $t_{new} > t_{old}$). Thus $P(TR_i > t_{new} \cap TR_i > t_{old}) = P(TR_i > t_{new})$, yielding:

$$P_{sojourn} = \prod_{i=1}^{n} \frac{P(TR_i > t_{new})}{P(TR_i > t_{old})}$$

For a given activity these probabilities can be computed from its cumulative distribution function: $F(t)$ is the probability for an activity to have finished in less than t time, and thus $1 - F(t)$ is the probability that the activity has finished after a duration of t. Thus,

$$P_{sojourn} = \prod_{i=1}^{n} \frac{1 - F_i(\tau_i + (t_{new} - t_{old}))}{1 - F_i(\tau_i)}. \tag{4.1}$$

This equation can be used for any discrete state and age vector to compute the exact sojourn probability to stay in the current state from t_{old} to t_{new} given that the system was still in that state at t_{old}.

4.3.2 Virtual Probabilities

Before we derive a formula for the *instantaneous state change probability*, we shortly discuss the peculiar features of the expected solution.

The probability in question is the probability of an activity to be completed exactly at the time of a symbol emission given that it has not been completed before. Formally, this means to determine the probability that a continuous probability distribution (describing the activity duration) takes on a single discrete value (the time of the symbol emission). For all probability distribution functions this value is always infinitesimal and thus numerically zero. Yet, evaluating all these probabilities to zero would result in an Evaluation probability of zero as well, irrespective of the model or the observation sequence. This would be a useless result to the Evaluation task.

A potential solution to this problem is based on the observation that these infinitesimal probabilities are by no means all identical. On the contrary, the usual representation of a continuous probability distribution as a *probability distribution function* (pdf) is a common way of giving different numerical values

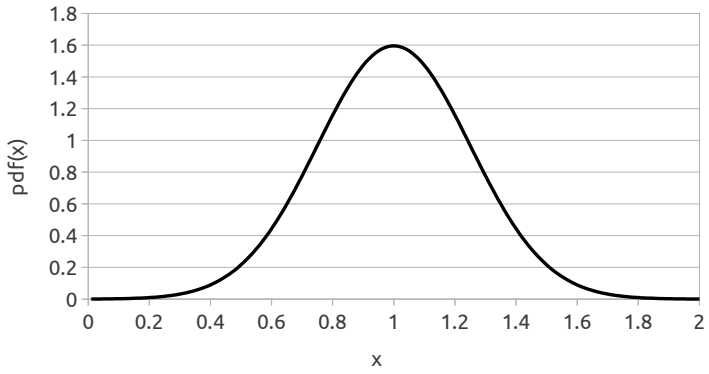

Figure 4.1: Graph of the probability distribution function for the normal distribution with mean 1.0 and standard deviation 0.25.

to these infinitesimal quantities. For example, the *pdf* of a normal distribution $\sim N(1, 0.25)$ evalutes to about 1.6 at position 1 and 0.21 at position 0.5 (cf. Figure 4.1). These values must not – and cannot – be interpreted as probabilities, and strictly speaking the probability of that normal distribution to take on either value is infinitesimal. But what the *pdf* can be used for is to determine the probabilities of the normal distribution (or any other probability distribution) to take on different values *relative* to each other. So while the *pdf* values given before must not be interpreted as absolute probabilities, the can be used to assert that the normal distribution $\sim N(1, 0.25)$ is about eight times as likely to take on a value of 1.0 (or rather a value in the ϵ-environment of 1.0) than it is to take on the value 0.5.

Similar to the *pdf*, in the next section we will derive a formula for our *instantaneous state change probabilities* that assigns numerical values to infinitesimal probabilities. We call these numerical values *virtual probabilities*, since they behave similar, but not identical to actual probabilities. In particular, virtual probabilities are also measures of likelihood for a given event to occur. And as with values of a *pdf*, two virtual probabilities are in the same ratio as the corresponding (potentially infinitesimal) probabilities and can thus be used to assess the relative differences between the two corresponding events. However, virtual probabilities differ from actual probabilities in that a single virtual probability cannot be interpreted a probability measure: A single virtual probability has no meaning by itself in the same way that a single value of a *pdf* has no meaning by itself.

Thus, virtual probabilities can be used as stand-ins for infinitesimal actual probabilities. In the following, we will derive a formula for the virtual *instantaneous state change probability*, which will result in the final Evaluation probability to be a virtual probability as well. Similarly, the intermediate result for the Decoding, Smoothing and Training tasks will be virtual probabilities as well. These virtual probabilities will then be shown to either be a useful result to the task at hand by themselves, or it will be shown that the actual probability to be computed for that task can be derived from these virtual probabilities.

4.3.3 Instantaneous State Change Probability

The second probability to be derived is the probability that the completion of an activity causes a discrete state change at an exact point in time given that the activity has not been completed before.

The derivation of this probability has been attempted before in [40] for general HnMMs, but their final result was incorrect for two reasons: First, they incorrectly assumed that the state change probability to be computed may be conditioned on the next observation. But the Forward variables of HnMMs (including CHnMMs) are *joint* probabilities to be in a given state and having emitted the trace so far. Thus, to compute a Forward probability from such a conditional probability, one would need to eliminate the conditioning on the next observation. This, however, requires the determination of the absolute probability of said observation, which the authors neglected to determine. And second, their informal argument implicitly and unwarrantly additionally conditions that state change probability on the current model state. Thus, their approach is of little use for us.

Instead, we start from scratch by noting that the probability for an activity with continuously distributed duration to finish at a discrete point in time (as opposed to finishing in a certain time interval) is infinitesimal. We therefore seek to represent this quantity as a virtual probability.

To obtain such a numerical value for the infinitesimal quantity we study the ratio of two arbitrary instantaneous state change probabilities as the limit that the activities are completed in an infinitesimal time interval around their respective durations τ_1 and τ_2:

$$\frac{P_{change_1}}{P_{change_2}} = \lim_{\Delta t \to 0} \frac{P(\tau_1 < TR_1 \leq \tau_1 + \Delta t | TR_1 > \tau_1)}{P(\tau_2 < TR_2 \leq \tau_2 + \Delta t | TR_2 > \tau_2)}$$

The right-hand side quotient may be expanded by ${}^1/_{\Delta t}$ yielding

$$\frac{P_{change_1}}{P_{change_2}} = \lim_{\Delta t \to 0} \frac{\frac{P(\tau_1 < TR_1 \leq \tau_1 + \Delta t | TR_1 > \tau_1)}{\Delta t}}{\frac{P(\tau_2 < TR_2 \leq \tau_2 + \Delta t | TR_2 > \tau_2)}{\Delta t}}.$$

As long as the limit of the denominator does not evaluate to zero, we can split the limit into two separate limits:

$$\frac{P_{change_1}}{P_{change_2}} = \frac{\lim_{\Delta t \to 0} \frac{P(\tau_1 < TR_1 \leq \tau_1 + \Delta t | TR_1 > \tau_1)}{\Delta t}}{\lim_{\Delta t \to 0} \frac{P(\tau_2 < TR_2 \leq \tau_2 + \Delta t | TR_2 > \tau_2)}{\Delta t}}$$

Here, each limit corresponds to the definition of the hazard rate $\mu(t) = \frac{f(t)}{1-F(t)}$ [52] of the duration probability distribution of the corresponding activity, and thus

$$\frac{P_{change_1}}{P_{change_2}} = \frac{\mu_1(\tau_1)}{\mu_2(\tau_2)}.$$

So, while we cannot numerically represent the infinitesimal P_{change} itself, any two non-zero P_{change} are in the same ratio as the corresponding values of their hazard rate functions. Or put another way, each infinitesimal probability P_{change} is proportional to the value of the hazard rate function of the corresponding activity probability distribution. The proportionality constant s

is unknown, but it is identical for all instantaneous state change probabilities. Thus, the quantity

$$\overline{P}_{change} = s * P_{change} = \mu(t) \tag{4.2}$$

is a virtual probability that assigns a numerical value to the infinitesimal P_{change}. Since the only difference between an ordinary probability and the corresponding virtual probability is an (unknown) scaling factor, the behavior of virtual probabilities is similar enough to that of ordinary probabilities to allow us to solve the CHnMM Evaluation task.

First, virtual probabilities can be added and multiplied just like ordinary probabilities. This allows us to perform all computations of a CHnMM Forward algorithm and to solve the Evaluation task as if all values were ordinary probabilities. Second, the ratio of two virtual probabilities has the same value as would the ratio of the corresponding but intractable ordinary probabilities:

$$\frac{\overline{P}_{change_1}}{\overline{P}_{change_2}} = \frac{s * P_{change_1}}{s * P_{change_2}} = \frac{P_{change_1}}{P_{change_2}} \tag{4.3}$$

Thus, if one occurence has a virtual probability of 0.2 and another one has a virtual probability of 0.6, then the second one is three times more likely than the first one, even though the numerical values must not be interpreted as literal probabilities.

So when the virtual Evaluation probabilities of multiple models for the same observation are given, then these virtual probabilities alone can be used to determine which model is the most likely explanation of the observation, and by what factor it is more likely than the other models. Thus, the virtual probabilities to be computed by the CHnMM Evaluation task can be used in the same ways as the ordinary probabilities computed for the HMM Evaluation task.

In the remainder of this work, we will use virtual probabilities for the computations of intermediate probabilities for all algorithms. For each algorithm we will then show that either the computed virtual probabilities can be used directly in place of ordinary probabilities (as was just done for the Evaluation task), or that the desired ordinary probability can be computed from the virtual probabilities obtained from the algorithm.

4.3.4 Summary: State Change Probability

Using the results above, the (virtual) state change probability that the first discrete state change after $o_t.e$ is at $o_{t+1}.e$, changes the discrete state from S_i to S_j, and emits the symbol $o_{t+1}.v$ can be computed as the product of:

1. the probability $P_{sojourn}$ to stay in state S_i for the time interval $o_t.e$ to $o_{t+1}.e$
2. the virtual conditional probability $\overline{P}_{change}$ to change state from S_i to S_j at the exact time of $o_{t+1}.e$, given that no state change has occurred since $o_{t+1}.e$
3. the conditional probability $a_{ij}.b(v_k)$ to emit signal v_k given that a state change from S_i to S_j takes place

Now that the state change probability can be computed, it can be used along with the known probability to be in each state at time $t = 0$ to compute the virtual probabilities to be in every possible successor state at the time of the first symbol emission and having emitted that first symbol. This approach can then be used iteratively for each observation from a trace to inductively compute the Forward probabilities for each time of symbol emission from the Forward probabilities of the previous symbol emission. The sum of these virtual Forward probabilities for the final observation is then the virtual probability of the model to have caused the whole trace of observations, and is thus the solution to the Evaluation problem.

It should be noted that the actual Evaluation probability for any CHnMM without deterministic activity durations is always infinitesimal, since it depends on point probabilities of continuous probability distributions (as shown in the derivation of $\overline{P}_{change}$). The introduction of virtual probabilities allows us to assign numerical values to these otherwise intractable probabilities. With these, Evaluation can be used to select the most likely model out of a set of models to explain a given trace of observations, since the virtual Evaluation probabilities of different models are in the same ratio as the actual unknown infinitesimal probabilities (cf. Equation 4.3). However, a single virtual Evaluation probability is only a *stand-in* and must not be interpreted as the actual infinitesimal probability of a model to have caused a given trace of observations.

So the HMM Forward algorithm provides a general algorithm structure for an algorithm to solve the CHnMM Evaluation task, the Proxel simulation model provides data structures that help adapt the HMM algorithm to non-Markovian systems, and the formulas derived allow for exact computations of the probabilities needed for such an algorithm. In the next section, all these individual parts will be assembled to form the CHnMM Forward algorithm

4.4 Result: The CHnMM Forward Algorithm

The resulting algorithm is given in Algorithm 1. It uses Proxels ρ defined as $\rho = (q, \vec{\tau}, \alpha)$, where q and $\vec{\tau}$ are the discrete state and age vector, respectively, and α is the virtual Forward probability for that state. Its input is the model specification $\lambda = (A, \Pi)$, the trace O, the set of activities TR and the numbers of states N, observations T and activities K, respectively. It assumes that a dummy observation at time $t = 0$ with an arbitrary symbol has beed added to the trace O.

It initializes the set of Proxels for $t = 0$ from the initial probability vector given by the CHnMM specification (line 1). It then iterates over all symbol emissions from the given trace (line 2) in order to inductively compute the joint probabilities of the model for each time step to be in a given state (disrete state and age vector) and having caused all symbol emissions of the trace up to that time step.

For each time of a symbol emission it iterates over all Proxels of the previous symbol emission time (line 6) to determine all possible successor Proxels. So for each Proxel, the sojourn probability (line 8) to stay in the current discrete state between the previous and the current symbol emission is computed. The algorithm then iterates over all possible state changes that may have caused the current symbol emission (line 9).

Algorithm 1: CHnMMForward

Input: A, Π, O, TR, N, T, K

$R_0 = \bigcup_{i \in \{1,\dots N | \pi_i \neq 0\}} \{(i, \vec{0}, \pi_i)\}$;

for $t = 1$ **to** T **do**

  $v = o_t.v$;

  $\Delta t = o_t.e - o_{t-1}.e$;

  $R_t = \varnothing$;

  foreach $\rho \in R_{t-1}$ **do**

    i $= \rho.q$;

    $P_{sojourn} = \prod_{j \in \{1,\dots,N |\ a_{ij} \neq \varnothing\}} \frac{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id} + \Delta t)}{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id})}$;

    foreach $j \in \{1, \dots, N | a_{ij} \neq \varnothing\}$ **do**

      $Row_i = \{a_{i1}, \dots, a_{iN}\}$;

      $Row_j = \{a_{j1}, \dots, a_{jN}\}$;

      $\vec{\tau}' : \tau'_k =$

$$\begin{cases} \tau_k + \Delta t & \text{if } TR_k \in Row_i \wedge TR_k \neq a_{ij} \wedge \neg isExp(TR_k.dist) \wedge \\ & (TR_k \in Row_j \vee TR_k.aging) \\ \tau_k & \text{if } TR_k \notin Row_i \wedge TR_k.aging \wedge \neg isExp(TR_k.dist) \\ 0 & \text{otherwise} \end{cases}$$

      $\mu = hrf(a_{ij}.dist)$;

      $\rho' = (j, \vec{\tau}', \rho.\alpha * P_{sojourn} * \mu(\rho.\tau_{a_{ij}.id} + \Delta t) * a_{ij}.b(v))$;

      if $\rho'.\alpha = 0$ **then** continue;

      if $\exists \rho'' \in R_t$ *with* $(\rho''.q = \rho'.q \wedge \rho''.\vec{\tau} = \rho'.\vec{\tau})$ **then**

        $\rho''.\alpha += \rho'.\alpha$;

      else

        $R_t = R_t \cup \{\rho'\}$;

return R_T

For each of these state changes the adjusted age vector for the new state is computed (line 12): The age is increased by the length of the current time step (the time between the previous and the current symbol emissions) for all those activities that were active for the whole time step and may still continue. This includes all those activities that did not cause the current state change and are either still active in the new discrete state or are of type RACE_AGE and thus may be interrupted to be resumed in a later discrete state. For all activities that where not active in the current state and are of type RACE_AGE, their age does not change from the age they had before the time step. And in all other cases, the age of the activity is reset to zero. This includes exponentially distributed activities, which are memoryless and whose behavior is thus age-independent, the activity that was just completed and caused the state change, and those activities that where active in the previous state but are not in the new one and thus are cancelled.

With the known age vector and discrete state after the current symbol emission a Proxel is created for that new state and its probability is computed according to the formulas in Section 4.3 (line 14). If the set of Proxels for the successor time step does not already contain a Proxel with the same discrete state and age vector, then the newly created Proxel is added to that set (line 19). Otherwise, the two Proxels are merged by adding the new probability to the already existing Proxel (line 17).

The algorithm terminates once the set of Proxels for the time of the final observation from the trace have been created, and returns that set. Each of these Proxels then contains the (virtual) joint probability of the model to be in the state represented by the Proxel and having emitted the whole trace. According to the law of total probability, summing up all those probabilities yields the desire virtual Evaluation probability of the model to have created the observation trace:

$$\overline{P}(O) = \sum_{\rho \in R_T} \rho.\alpha$$

Thus, this algorithm solves the CHnMM Evaluation task.

4.4.1 Differences to the Proxel Simulation Algorithm

The CHnMM Forward algorithm is generally similar to the Proxel simulation method: all possible single model state changes are followed for each time step, and the resulting states are stored in Proxels. However, in detail there are major differences between the two algorithms:

Virtual Probabilities In the CHnMM Forward algorithm, the Proxels store virtual probabilities which have no meaningful interpretation as an actual probability. In addition the CHnMM Forward algorithm stores joint probabilities of a given state and the emission of a given trace in those Proxels, whereas the Proxel simulation algorithm stores absolute state probabilities.

Accuracy The state change probabilities for the CHnMM Forward algorithm are not approximations of the solution to an ODE, but are computed through closed-form exact formulas.

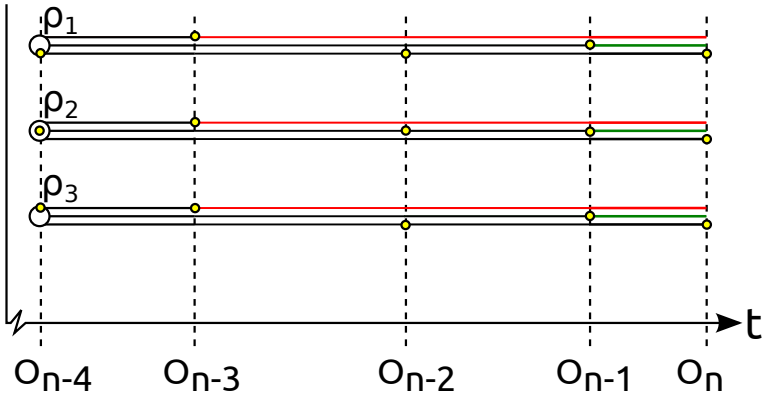

Figure 4.2: Schematic representation of the conditions under which Proxels have identical age vectors and can thus be merged.

Additionally, the Proxel method made the unwarranted assumption that only a single discrete state change may occur during each time step. This assumption was made deliberately[29] in order to make the approach computationally feasible, but at the same time introduced a noticable approximation error. The CHnMM Forward algorithm also makes this assumption, but for CHnMMs this assumption is valid: since all discrete state changes emit an observable symbol, there can be no discrete state change in the time interval between two symbol emissions. This and the previous point together mean that the CHnMM Forward algorithm does not contain any of the approximation errors of the Proxel method and is thus accurate.

Time Step Size In the Proxel method, the time step size has to be chosen carefully by the user to select a tradeoff between computation time and result accuracy[43]. For the CHnMM Forward algorithm the variable time step sizes are selected automatically to conform to the time intervals between symbol emissions, and this way eliminate all approximation errors. Thus, the CHnMM Forward algorithm contains no user-tunable parameters and is thus easier to use by practitioners.

Proxel Merging Proxel simulation and the CHnMM Forward algorithm both employ Proxel merging: Whenever two Proxels for the same time step exist that present the same discrete state and age vector, they are merged to form a single Proxel with the combined probability of the two. This merging prevents both algorithms from otherwise suffering an exponential increase in the number of Proxels with each new time step, since each Proxel from one time step would always have multiple successors in the next time step. Through merging, several of those successors are combined to form a single Proxel whereby the number of Proxels per time step is bounded for many models. Yet, both algorithms differ in the circumstances under which mergable Proxels exist:

In the Proxel simulation, the age vector elements are discretized according to the time step size and so Proxels that would otherwise only have similar age vectors thus have identical ones and can be merged. In CHnMMs, age values

are not discretized and can take on potentially any positive real value, virtually eliminating chance mergings. This potentially reduces the probability of Proxel mergings and thus increases the required computational effort.

Besides those virtually impossible chance mergings, Proxels can only be merged in CHnMMs when the most recent symbol emission at which each activity was completed matches between the Proxels. Figure 4.2 visualizes this behavior: In a model with three activities (the sets of three parallel horizontal lines), three proxels ρ_1, ρ_2 and ρ_3 may have arbitrary age vectors at the time of the O_{n-4}th symbol emission and thus cannot be merged. However, their age vector values at time $O_n.t$ (length of the colored line segments) depend only on the time that has passed since each activity has been completed (yellow circles) most recently. Thus, in this example, irrespective of

- the prior age vector values of ρ_1, ρ_2 and ρ_3
- which activity has been completed at $O_{n-4}.t$
- whether the second or third activity has been completed at $O_{n-2}.t$,

as long as the O_{n-3}th observation is caused by the completion of the first activity, the O_{n-1}th observation by the second activity and the O_nth observation by the third activity, the Proxels will have identical age vectors at time $O_n.t$ and – when their discrete states match as well – will be merged in the Forward computation step for O_n. So, even though age vector elements can take on potentially any real number, the set of possible age values at the time of a particular observation only depends on the combinatorics of which activity may have cause what prior observation. Proxels for which these combinatorics result in identical age vectors can then be merged when their discrete states match as well. This condition for Proxel merging holds far more often than chance mergings and is likely to be a major contributing factor in the practical feasibility of the approach.

Furthermore, in CHnMMs successor Proxels are only created for state changes and not also for inactivity (as with the Proxel simulation). This reduces the number of successors of each Proxel by one and thus reduces the factor by which the number of Proxels could multiply in each time step. And in CHnMMs, Proxels are only created for the times of symbol emissions, whereas the Proxel simulation has to create Proxels in discrete time steps which are usually far smaller than the time intervals between discrete state changes in order to reduce the approximation errors inherent in the method. So the number of times at which the number of Proxels may multiply is also far lower for CHnMMs.

Thus, even though the CHnMM Forward algorithm does not use discretized age vector values as the Proxel simulation method and is thus less likely to merge Proxels by chance, it may still be practically feasible. The actual Proxel merging behavior and the general behavior of the CHnMM Forward algorithm are tested experimentally in the next section.

4.5 Experiments

In this section, an application example is given for the newly developed CHnMM Forward algorithm, and the algorithm is tested for its practical feasibility with

respect to computation time and memory consumption.

4.5.1 Application Example

The Tester model (cf. Figure 3.1 on page 32) can serve as an example for the application of the CHnMM Evaluation task. Assume that during normal operation the two machines both produce defective items with a probability of only 0.01. But suddenly the overall defect probability as recorded by the quality tester increases to about 0.05[2]. There are three likely scenarios that could explain than increase of defects:

- The first machine was damaged and thus produces more defective items
- The second machine was damaged and thus produces more defective items
- The quality of the raw materials used has degraded causing both machines to produce more defective items

To determine which of the three cases occurred the CHnMM Evaluation task can be used: Three variations of the CHnMM Tester model are built to represent the three cases. In each model and based on the expected values of the machine production durations, the defect probabilities of the two machines are adjusted in order to account for the total number of defects recorded in the observation trace. Then the Evaluation task is performed for each model with the trace of observations in question. The resulting $P(O|\lambda_i)$ all have numerical values of about 10^{-3000}, but since these are *virtual* probabilities they cannot directly be interpreted as probabilities. However, they can be used to compute the conditional probability for each model to have generated the trace of observations given that the trace had to be caused by one of the three models, as follows:

$$\begin{aligned}
P(O \text{ by } \lambda_i | O \text{ by } \lambda_1, \lambda_2 \text{ or } \lambda_3) &= \frac{P(O \text{ by } \lambda_i \cap O \text{ by } \lambda_1, \lambda_2 \text{ or } \lambda_3)}{P(O \text{ by } \lambda_1, \lambda_2 \text{ or } \lambda_3)} \\
&= \frac{P(O \text{ by } \lambda_i)}{\sum_{j=1}^{3} P(O \text{ by } \lambda_j)} \\
&= \frac{P(O|\lambda_i)}{\sum_{j=1}^{3} P(O|\lambda_j)} \\
&= \frac{s^n \; P(O|\lambda_i)}{\sum_{j=1}^{3} s^n \; P(O|\lambda_j)} \\
&= \frac{\overline{P}(O|\lambda_i)}{\sum_{j=1}^{3} \overline{P}(O|\lambda_j)}
\end{aligned}$$

Thus, this conditional probability simplifies to normalizing each virtual probability by dividing it by the sum of all three virtual probabilities. The results are shown in the following table:

[2]The reader is reminded that in this work no real-life data was used. Instead, all traces of observations are synthetic traces created with Monte-Carlo simulations using the AnyLogic simulation software [8].

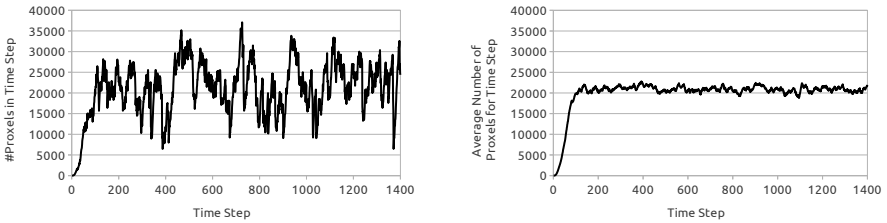

Figure 4.3: Plot of the number of Proxels created for each of the first 1400 time steps of the CHnMM Forward computation of the Car Rental Agency model, for a single trace (left) and the average over 100 traces (right).

Scenario	**Normalized Probability**
Machine 1 damaged	~ 1
Machine 2 damaged	$1.22 * 10^{-26}$
Raw materials degraded	$2.89 * 10^{-7}$

In this case, the first scenario is more than ten million times more likely than the other two combined. Thus, if the assumption holds that only those three causes for defects exist, then Machine 1 is overwhelmingly likely to be damaged and needs to be repaired.

In a real production facility this Evaluation approach could be used to find defective equipment based solely on existing data and without the need for a physical inspection – which would require downtime – of the machines.

4.5.2 Time Complexity

While the approach has been shown to be applicable to practical problems, it is still necessary to asses its limits of practical applicability.

To that end, we assess the number of Proxels per time step and the closely linked computation time subject to increasing trace length: If the number of Proxels per time step stays about constant with increasing trace length, then the computation time per time step would also stay constant and the total computation time for a trace would increase only linearly with increasing trace length, allowing for almost arbitrary trace length. If, on the other hand, the number of Proxels per time step increases with each time step then the total computation time would increase superlinearly with increasing trace length and could quickly render the Evaluation of longer traces infeasible.

To asses both properties, the CHnMM Forward algorithm was applied to the Car Rental Agency model and the number of Proxels in each time step as well as the cumulated computation time were recorded. In the Tester model, the Evaluation of a trace for one day took only about 70ms and never needed more than 12 Proxels per time step. Both are considered to be too low to make realistic assessments.

The left-hand side of Figure 4.3 shows the number of Proxels for each time step of a single trace of the Car Rental Agency model with about 1400 observations. Initially, the number of Proxels increases exponentially with each time step. For the remaining trace, the number of Proxels per time step varies unpredictably. The likely reason is that some successor Proxels are not just

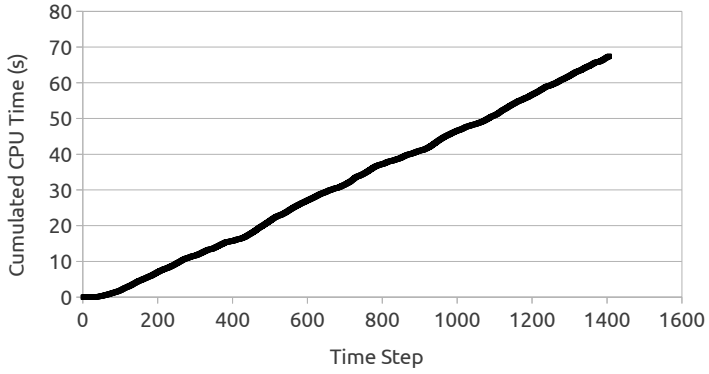

Figure 4.4: Plot of the cumulated computation time required for the CHnMM Forward computation of the Car Rental Agency model.

unlikely but impossible, i.e. either $P_{sojourn}$ or P_{change} is zero. In these cases the successor Proxel is omitted completely (cf. Line 16 in the CHnMM Forward pseudocode). So, in instances where the majority of successors have zero probability, the number of Proxels per time step declines. Otherwise it may grow exponentially. How often each of these situations occurs depends on the time and symbol of the current and potentially of all previous symbol emissions. Since those are random, the number of Proxels per time step varies randomly as well.

To better characterize the number of Proxels per time step the experiment was repeated 100 times with different traces that were generated through 100 independent replications of the simulation model. The right-hand side of Figure 4.3 shows the average number of Proxels per time step over these traces. Here, it is apparent that the number of Proxels per time step is bounded - for the Car Rental Agency model about 22000 Proxels are generated per time step on average - and this number does not grow with increasing trace length. This means that the potential exponential growth of the number of Proxels per time step due to each Proxel having multiple successors is successfully counteracted by Proxel merging and the elimination of *impossible* Proxels, and does not limit the practical feasibility of the approach for longer traces.

Figure 4.4 shows the corresponding cumulated CPU time for each observation, i.e. the computation time required to execute the CHnMM Forward task up to the given observation. The total computation time was only about a minute for a trace of 1400 symbols corresponding to an observation time of about 160h. Thus, for this model, even real-time online behavior reconstruction of CHnMMs would be possible.

For the first few observations there is an exponential relationship between the number of observations analyzed and the time spent, corresponding to the initial exponential growth in the number of Proxels. But for longer traces, the relationship is mostly linear, matching the boundedness of the number of Proxels per time step.

Thus, the computation time of the CHnMM Forward algorithm seems to

increase linearly with the length of the observation sequence. Practically, this means that the Evaluation even for very long traces is feasible.

Limits of Practical Applicability The CHnMM Evaluation task has been shown to be practically feasible for the given models and trace lengths, but its general limits of applicability have not yet been investigated. The algorithm would be unfeasible if the number of Proxels per time step is too high to allow for efficient behavior reconstruction. Depending on the application area, this may mean that either real-time Evaluation is no longer possible or that the task takes longer than a few hours.

The experiments on computational complexity have already shown that the number of discrete states is a relevant but not a limiting factor for this practical feasibility: While the Evaluation of the Tester model with two discrete states took 0.07s, Evaluation of the Car Rental Agency model with 2500 times as many discrete states took about 1000 times as long. Thus, it may be assumed that the computation time increases approximately proportionally to the size of the reachable set of discrete states. Consequently, an increase in the size of the discrete state space does not dramatically impact the practical feasibility.

The other influencing factor on the number of Proxels is the set of possible age vectors, since Proxels with different age vectors may not be merged. This number is heavily influenced by the number of concurrent activities, because those determine the dimension of the age vector[3].

To test this influence of the number of concurrent activities on the computation time, we performed experiments on variations of the Tester model with different numbers of machines that are concurrently producing the items. The results for a trace covering an observation time of 100000s can be seen in Figure 4.5. The computation time grows exponentially with increasing number of concurrent activities. For two concurrent activities, the Evaluation takes less than 0.1s, for four activities already more than a minute and for six activities more than 45 hours, which is slower than real-time.

Since the number of concurrent activities is not the only factor affecting computation time, the results presented here are not generalizable. Yet, they show a certain trend: for pattern recognition in human-computer interaction, computer feedback on human input has to be instantaneous. In that application area, models may thus have at most two to three concurrent activities. In other scenarios restrictions on permissible computation time may be more relaxed. But even there six concurrent activities seem to be the upper limit for practically feasible Evaluation.

In summary, the practical applicability of the CHnMM Forward algorithm depends somewhat on the size of the discrete state space, and is severely limited by the number of concurrent activities.

The next two sections of this chapter contain ideas for further applications and possible extensions to the CHnMM Evaluation algorithm as well as implementation considerations for the algorithm.

[3]Strictly speaking, the dimension of the age vector is given by the *total* number of activities. But inactive activities of type RACE_ENABLE have an age value of zero and therefore do not impact the size of the set of possible age vectors.

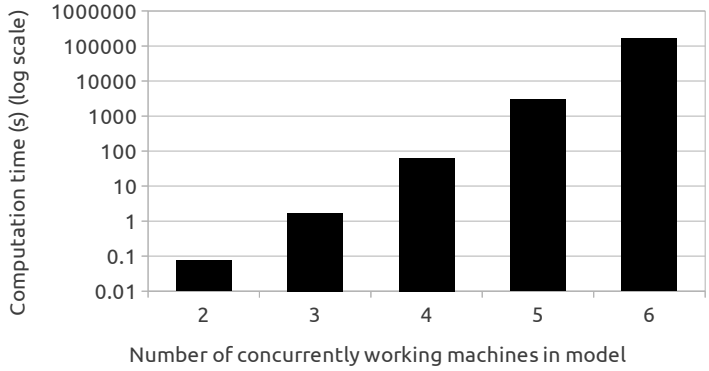

Figure 4.5: Computation time of the CHnMM forward algorithm for the Tester model with different numbers of concurrently active machines for traces of the same duration.

4.6 Further Applications and Extensions

Solving the Filtering Task In addition to the four basic HMM behavior reconstruction tasks for which CHnMM algorithms are to be developed, other HMM tasks exist. One of those is the Filtering task [66], which computes the conditional probabilities of the model to be in each state given that all elements from the trace are emitted so far. For CHnMMs, this task is solved as a side-effect of the CHnMM Forward algorithm developed for the Evaluation task: The Proxels in the CHnMM Forward algorithm represent the corresponding virtual joint probabilities and thus normalizing those so that they sum up to one converts them into conditional probabilities.

The Filtering task could be used to reconstruct individual probabilities of the model to be in a particular state at a particular time given the partial trace of observations from the past of that time. However, Filtering is rarely used, because the Smoothing algorithm developed in Chapter 6 solves the similar probability of the model to be in a particular state at a particular time, given the complete trace containing observations from the past and future of that time. Smoothing therefore uses more information than Filtering and can thus make more accurate predictions.

Performance Counters For some application scenarios CHnMMs may be extended by performance counters than count the occurence of certain events, e.g. the number of defective items from Machine 1 in the Tester model. Such a counter can be added as another supplementary variable to each Proxel, and each successor Proxel would either inherit the counter value from its predecessor, or would change it when certain conditions occur – e.g. by increasing it by one whenever Machine 1 produces a defective item.

An execution of the CHnMM Forward algorithm and normalization of the Proxel probabilities for the final time step would then compute the virtual probabilities that the model is in each reachable state with each possible given counter value. From those Proxels, statistics on the counter values such as their distri-

bution (as a histogram) or their expected values could be derived in order to assess the model behavior under the trace [13].

In the general case, this additional information is bought through an increase in computational complexity, because Proxels may now only be merged when their discrete state and all supplementary variables (the age vector and the counters) match. This increases the number of Proxels by a factor according to the number of reachable different values for the counter.

However, if only an expected value of the counter is desired, the computation need not slow down noticably compared to the normal Filtering algorithm. In this case, the merging condition can be relaxed to allow the merging of Proxels even when their counter values do not match. The counter value of the merged Proxel will instead be set to the weighted average of the two source counter values, weighting them by their probabilities. This effectively precomputes the expected counter values over the merged Proxels and does not change the final result. Yet it does not increase the state space of the model compared to the general application of performance counters and thus is far more efficient.

More Efficient Computation of Evaluation Probabilities The CHnMM Forward algorithm contains a number of redundancies [11]: State change probabilities are recomputed for every Proxel, even though they may be identical for several Proxels. And after the creation of each Proxel, Proxels that share its state have to be found in the *whole* set of Proxels as a prerequisite for merging, even though the creation of Proxels follows certain patterns.

For example, in the Car Rental Agency model the probability to change the current discrete state only depends on what type of customer is currently being served and how long that service has been going on. It does not depend on the number of people waiting in line, and yet the same value is recomputed even for Proxels that only differ in that number of waiting people. Also, for each Proxel a state change through the activity "A premium customer arrives" predictably changes the discrete state by always increasing the number of waiting premium customers by one. Yet this structure is not exploited to make the behavior reconstruction more efficient.

MultiProxels [11] cluster states according to those exploitable model structures, and store probabilities for each state of a cluster, instead of generating individual Proxels for each state. With MultiProxels, redundant state change probability computations need only be performed once per cluster and not for individual Proxels anymore. And for all states clustered in a MultiProxel, duplicates that should be merged tend to be located in only a single other cluster and are often found there directly instead of having to scan the whole set of Proxels. Since the MultiProxel approach only reorders the computations of the CHnMM Forward algorithm, the results are not altered and are still exact.

Experiments have shown that the MultiProxel approach can reduce the computation time for the CHnMM Forward algorithm by up to 80%. Their drawbacks are the more complex algorithm, and potentially an increased memory consumption and even a slowdown on some models. The latter occurs when many states of a MultiProxel state cluster have a probability of zero and yet need to be stored and processed since a MultiProxel is always processed completely. So, MultiProxels can dramatically speed of CHnMM Forward computations, but the decision on whether to use them currently requires expert knowledge.

4.7 Implementation Considerations

While the pseudocode of the CHnMM Forward algorithm shows *what* needs to be computed when, it does not in full detail explain *how* the respective behavior can be implemented efficiently, and what other things need to be considered in an actual implementation of the algorithm. These issues are addressed here.

4.7.1 Implementation Considerations Derived from Proxel Simulators

The inductive CHnMM Forward algorithm computes all Proxels of a given time step based only on the set of Proxels of the previous time step. Furthermore, the result of the algorithm is only the set of Proxels of its final time step. Thus, whenever the successors of a Proxel have been computed, the Proxel itself has become irrelevant and can be discarded to save memory. When discarding individual Proxels is too inefficient, region-based memory management [3] may be used instead to efficiently allocate memory for individual Proxels of a time step, but to discard the whole set at once.

The Matrix A of a CHnMM is usually sparsely populated, since the width and height of the matrix are determined by the size of the state space, but the number of elements populated in a matrix row is given by the number of concurrently active transitions. In the Car Rental Agency model with its over 5000 states, the matrix would have over 25 million elements, but only about 15000 of those are populated. Thus, to save memory A should not be stored as a full matrix, but as an adjacency list, i.e. for each discrete state to store a list of which activities are occurring in that state, and which discrete state is reached by each activity being completed. A positive side effect of this implementation is that in order to compute all successor Proxels one does not have to traverse a complete matrix row, but only the short list of activities occurring.

The modification of age vector values of a new Proxel (i.e. whether each age value is kept from the predecessor, increased, or set to zero) depends only on the completed activity and the discrete states of the Proxel and its predecessor. Those decisions are thus constant for each time step and may be precomputed for each non-$\varnothing$ entry of A, instead of having to make the decisions for every Proxel again.

And finally, a core concept of the Proxel-based CHnMM Forward algorithm is the merging of Proxels with identical discrete state and age vector. This implies that the efficiency of an implementation of the algorithm is heavily impacted by its ability to efficiently find duplicate Proxels. Known efficient ways are based on storing the Proxels of a time step in a dictionary [68] indexed by the discrete state and age vector, for example in a hash table or a self-balancing binary tree [35].

All of these issues occur in Proxel simulators as well and have been solved there in the same fashion [43].

4.7.2 Implementation Considerations Derived from HMM Forward Implementations

While the Proxel method computes the probability of the model to be in each reachable state, the CHnMM Forward algorithm computes only those joint prob-

abilities of the model to be in a given state *and* having emitted all observations prior to the current time step. It ignores the probabilities of the model to be in any state while not having emitted all prior observations, since these are not relevant for the Evaluation task. Consequently, while the Proxel method guarantees a probability sum of one in each time step, the CHnMM Evaluation algorithm is not conservative and "loses" probability mass in each time step.

The actual loss of probability mass varies per time step and even per Proxel, and is not easily predicted beforehand. So, while a Proxel in one time step may have the lowest probability of all Proxels, one of its successors may have the highest probability of all Proxels in the next time step. The consequences of this behavior are twofold.

First, in the Proxel method, Proxels with very small probability (e.g. $< 10^{-15}$) are assumed to not noticably impact the final simulation result and are thus ignored to speed up the simulation. In CHnMMs this is not possible, because the successor Proxels of a Proxel with very low probability may still have a comparatively high probability and thus a heavy impact on the result.

And second, since probability mass is lost in each time step, the remaining probabilities may become very small. In the experiments on the Forward algorithm (c.f. Section 4.5), a typical Evaluation probability has been shown to be about 10^{-3000}. And since this is the probability sum of all Proxels of the final time step, the individual Proxel probabilities are even lower. These numbers are too low to be represented by the IEEE 754 `double` type which is used to represent floating point numbers on x86 commodity hardware.

For HMMs the standard solution to this problem [23, 62] is to store logarithmic probabilities and perform direct addition and multiplication of these logarithmic values by applying the product rule of logarithms and the Kingsbury-Rayner formula [32]. This solution also works sufficiently well for CHnMMs, even without dedicated hardware support for logarithmic arithmetic [15].

With these details, it should be possible to implement the CHnMM Forward algorithm.

4.8 Conclusion

In this Chapter, an exact algorithm for the CHnMM Evaluation task has been developed. Experiments have demonstrated the applicability of the algorithm to the pattern recognition problem. And the computation time of the algorithm on small models has been shown to be low enough to perform the Evaluation task in real-time on the commodity hardware of desktop computers.

Thus, the first goal of this work to provide an exact solution to the CHnMM Evaluation problem has been reached. In the next chapter, this Forward algorithm is modified to solve the CHnMM Decoding problem.

Chapter 5

The Decoding Task

While the Evaluation task determined only a single virtual probability to characterize an unobserved behavior, the Decoding task directly reconstructs the complete internal behavior of the model during the time period of the observations. Its goal is to determine the most likely sequence of internal unobservable discrete states that a model has passed while emitting a given trace. Formally, this means finding

$$\underset{Q=q_0 \ldots q_T \in \{S_1,\ldots,S_N\}^{T+1}}{\arg\max} P(q_0 \ldots q_T | o_1 \ldots o_T).$$

Here, the relevant probability can be rewritten to

$$\begin{aligned} P(q_0 \ldots q_T | o_1 \ldots o_T) &= \frac{P(q_0 \ldots q_T \cap o_1 \ldots o_T)}{P(o_1 \ldots o_T)} \\ &= \frac{s^T\ P(q_0 \ldots q_T \cap o_1 \ldots o_T)}{s^T\ P(o_1 \ldots o_T)} \\ &= \frac{\overline{P}(q_0 \ldots q_T \cap o_1 \ldots o_T)}{s^T\ P(o_1 \ldots o_T)}. \end{aligned}$$

In this representation, the denominator does not depend on the actual paths of internal states, and therefore does not affect the arg max. Omitting it yields the alternative more practical representation of the Decoding task to find

$$\underset{Q=q_0 \ldots q_T \in \{S_1,\ldots,S_N\}^{T+1}}{\arg\max} \overline{P}(q_0 \ldots q_T \cap o_1 \ldots o_T). \tag{5.1}$$

Here, the virtual probabilities over which the arg max is to be determined are individual joint path probabilities of the model to traverse the given sequence of discrete internal states while emitting the given observation sequence. The Forward algorithm developed for the Evaluation task would already compute those probabilities if no Proxel merging took place. However, if Proxel merging was simply disabled in the CHnMM Forward algorithm, then the number of Proxels would grow exponentially with each time step, rendering the approach unfeasible.

Instead, in order to solve this tasks for CHnMMs, we will follow the same approach as for HMMs (cf. Section 2.1.3) and modify the corresponding Forward

algorithm to omit those paths that cannot possibly yield the highest joint path probability. The resulting CHnMM algorithm is usually practically feasible, but for long traces may consume too much memory. We will therefore also develop a modified algorithm for the Decoding task that is slightly slower than the original one, but requires far less memory.

5.1 The Basic CHnMM Decoding Algorithm

A naïve approach to the Decoding problem would be to simply compute the probability of all possible paths of internal states and would then select the one with the highest probability. As Equation 5.1 shows, the probabilities to be computed for this task are essentially the joint probabilities that the Forward algorithm would generated without Proxel merging: with Proxel merging, the Proxel probabilities represent the probability of the model to be in a given state at a given time and having emitted the trace so far. Without Proxel merging, each Proxel is caused by a single path of internal states, since each Proxel has exactly one direct successor (the one Proxel that caused its creation). Thus, each Proxel's probability of the final time step represents the desired joint probability of the model to have passed through a single path of internal states and have emitted the observation sequence. Selecting the one Proxel with the highest probability from the Proxel set of the final time step would solve the Decoding problem.

Unfortunately, as shown for the Forward algorithm, the computation without Proxel merging would result in an exponential growth in the number of Proxels with each additional time step and is thus unfeasible for all but very short traces.

The solution to this problem is to allow merging, but to change its semantics: When two Proxels are to be merged in the Forward algorithm, this is because the model has reached the same state (i.e. same discrete state and same age vector) through two different paths. Merging is possible, because the possible future behavior depends only on the current (identical) state and not the (different) prior paths of internal states. Thus, the one Proxel with higher probability will also cause all paths of future behavior to have a higher probability than the corresponding paths of future behavior from the other Proxel. This guarantees that of two Proxels to be merged the one Proxel with the lower probability cannot lay on the path with the highest probability to explain the observation sequence.

Consequently, when two Proxels with identical state for the same time step exist in the Decoding task, the one with the lower probability can be discarded. For Proxel merging this means that the merged Proxel does not contain the probability *sum* of all paths that pass through it, but only the probability of the most likely path.

With this small change the modified Forward algorithm outputs a set of Proxels for the final time step where each Proxel represents the probability of the most likely path to end in that state. The highest of these probabilities is then the probability of the most likely path to explain the given trace of observations, and the discrete state of the corresponding Proxel is the final discrete state of that path.

What is missing to complete the Decoding task is to determine the remaining states on that path. This is easily done by adding links to the predecessor

Proxel: even *with* the modified Proxel merging of the Decoding task, each Proxel of that task after the first time step has a unique predecessor Proxel (it has to have at least one predecessor, since Proxels are only created by following possible activity completions from other Proxels; and it has at most one predecessor, since all others are discarded through the modified merging). To solve the Decoding problem one simply adds enough information to each Proxel to uniquely determine its predecessor, i.e. the predecessor's discrete state and age vector. Then, when the Proxel representing the final state of the most likely path has been determined by the modified Forward algorithm, its links and those of its predecessors can iteratively be followed backwards to reconstruct the complete most likely path of (discrete) states, thereby solving the Decoding problem.

The complete Decoding algorithm using modified Proxel merging and Proxel backlinks is given as pseudocode in three parts: Algorithm 2 details the computations for a single time step of the modified Forward algorithm with alternative Proxel merging semantics used to find the end state of the most likely path, Algorithm 3 details the computations for a single backtracking step used to find a given predecessor on the most likely path, and Algorithm 4 uses the other two algorithms to solve the Decoding task for a given trace of observations.

All three algorithms assume that Proxels contain links to their predecessors, so that the normal Proxel definition of $\rho = (q, \vec{\tau}, \alpha)$ needs to be extended to $\rho = (q, \vec{\tau}, \alpha, q_{parent}, \vec{\tau}_{parent})$.

Algorithm 2: DecodingForwardStep

Input: $A, TR, R_t, v_{t+1}, \Delta t, N, T, K$

$R_{t+1} = \varnothing$;
foreach $\rho \in R_t$ **do**
  i = $\rho.q$;
  $$p_{sojourn} = \prod_{j \in \{1,\ldots,N|\ a_{ij} \neq \varnothing\}} \frac{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id} + \Delta t)}{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id})};$$
  foreach $j \in \{1, \ldots, N | a_{ij} \neq \varnothing\}$ **do**
    $Row_i = \{a_{i1}, \ldots, a_{iN}\}$;
    $Row_j = \{a_{j1}, \ldots, a_{jN}\}$;
    $$\vec{\tau}' : \tau'_k = \begin{cases} \tau_k + \Delta t & \text{if } TR_k \in Row_i \wedge TR_k \neq a_{ij} \wedge \neg isExp(TR_k.dist) \wedge \\ & (TR_k \in Row_j \vee TR_k.aging) \\ \tau_k & \text{if } TR_k \notin Row_i \wedge TR_k.aging \wedge \neg isExp(TR_k.dist) \\ 0 & \text{otherwise} \end{cases}$$
    $\mu = hrf(a_{ij}.dist)$;
    $\rho' = (j, \vec{\tau}', \rho.\alpha * p_{sojourn} * \mu(\rho.\tau_{a_{ij}.id} + \Delta t) * a_{ij}.b(v_{n+1}), \rho.q, \rho.\vec{\tau})$;
    if $\rho'.\alpha = 0$ **then** continue;
    if $\exists \rho'' \in R_{n+1}$ *with* $(\rho''.q = \rho'.q \cap \rho''.\vec{\tau} = \rho'.\vec{\tau})$ **then**
      if $\rho'.\alpha > \rho''.\alpha$ **then**
        $R_{n+1} = (R_{n+1} \backslash \{\rho''\}) \cup \{\rho'\}$;
    else $R_{n+1} = R_{n+1} \cup \{\rho'\}$;
return R_{n+1}

Algorithm 2 is almost identical to a single step of the CHnMM Forward algorithm (cf. Algorithm 1 on Page 46) Its inputs arethe state transition matrix M, the set of activities TR, the set of Proxels after the tth observation R_t, the symbol v_{t+1} of the next observation, the time Δt between the last and the next observation, and the numbers of discrete states N, observations T and activities K, respectively. The only differences to the Forward algorithm are the addition of links to predecessor Proxels and the modified merging, which discards the source Proxel with the lower probability.

Note that as with the Forward algorithm, all probabilities stored in Proxels for the Decoding task are actually virtual probabilities that must not be interpreted as literal probabilities. However, since the ratio of two virtual probabilities is the same as the ratio of the corresponding unknown actual probabilities, the maximum of two virtual probabilities can be used in place of the maximum of the unknown actual probabilities, and will yield the same path.

Algorithm 3: DecodingBacktrackingStep

Input: $R_t, \rho_{max_{t+1}}$
Result: Proxel of the tth time step that is part of the most likely path of internal states.
1 **return** only $\rho \in R_t$ with $\rho.q = \rho_{max_{t+1}}.q_{parent} \wedge \rho.\vec{\tau} = \rho_{max_{t+1}}.\vec{\tau}_{parent}$

Algorithm 3 simply uses the links of a single Proxel $\rho_{max_{t+1}}$ to find its predecessor from the set of Proxels R_t of the previous time step.

Algorithm 4: Decoding_Iterative

Input: A, TR, Π, O, N, T, K
Result: Sequence of internal states $q_0, \ldots, q_T$ that is most likely to have created the sequence of observations O

$R_0 = \bigcup_{i \in \{1,\ldots N | \pi_i \neq 0\}} \{(i, \vec{0}, \pi_i, \varnothing, \varnothing)\}$;
for $t = 1$ **to** T **do**
  $R_t =$ `DecodingForwardStep` $(A, TR, R_{t-1}, o_t.v, o_t.e - o_{t-1}.e, N, T, K)$;
$\rho_{max} = \arg\max_{\rho \in R_T}(\rho.\alpha)$;
$q_T = \rho_{max}.q$;
for $t = T$ **to** 1 **do**
  $\rho_{max} =$ `DecodingBacktrackingStep` $(R_{t-1}, \rho_{max}))$;
  $q_{t-1} = \rho_{max}.q$;
return $(q_0, \ldots, q_T)$

Algorithm 4 uses the two other algorithms to solve the Decoding problem. Its inputs are the same as for the Forward algorithm: The model specification (A, Π), the set of activities TR, the trace of observations O and the numbers of discrete states N, observations T and activities K. Like the Forward algorithm it assumes that the trace being with a dummy observation o_0 with an arbitrary symbol and time $o_0.e = 0$.

It first creates all Proxels for all time steps with the modified Forward algorithm (lines 2 and 3), then finds the Proxel representing the final state of the most likely path (line 4) and finally iteratively follows the links of that Proxel and its predecessors backwards to retrieve the complete path (lines 6–8).

Properties of the Algorithm The modified Forward algorithm and its backtracking as shown in Algorithm 4 together solve the CHnMM Decoding problem. The computational complexity is identical to that of the normal CHnMM Forward algorithm that solves the Evaluation task: for the Forward part of the computation, the same number of Proxels are generated in each time step of the two algorithms, since successor Proxels with the same discrete state and age vector are created in both algorithms and Proxels are also merged under the same conditions. The adding of predecessor links to Proxels adds only a small constant time overhead, and the following of links to predecessors in the backtracking part of the algorithm requires only the lookup of a single Proxel for each time step. Thus, judging by the computational complexity alone, the algorithm for the Decoding task is practically feasible whenever the Forward algorithm for the CHnMM Evaluation task is feasible and so the limitations of practical feasibility of the Forward algorithm (cf. Section 4.5.2) ostensible apply to the algorithm for the Decoding task as well.

However, the Decoding algorithm has a very different memory complexity: In order to follow the links back to predecessors, those predecessor Proxels need to exist and thus the algorithm must hold all Proxels of all time steps in memory at the same time, whereas the Forward algorithm needed to store at most two time steps at once.

This has a noticable influence on the memory consumption as the following example shows: a single Proxel for the Decoding task of the Car Rental Agency model has a size of 96 bytes on x86-64 hardware. For each time step the algorithm generates on average 25000 Proxels, resulting in a memory requirement of $2.4MB$ per time step. For the CHnMM Forward algorithm, this translates to less than five megabytes of memory that are required to store all Proxels for the respective current and next time step. To perform the Decoding task for the Car Rental Station model with the trace used in the Experiments in Section 4.5 containing about 1400 observations would however require more than three gigabytes of memory. That amount of memory is barely available for a single application on 2011 commodity hardware.

The computational complexity of the algorithms has been shown experimentally to be linear in the number of observations, meaning that behavior reconstruction for far longer traces (e.g. 100000 observations and more) is computationally feasible. For those traces, however, the memory consumption of the modified Forward algorithm solving the Decoding task would be several hundred gigabytes and thus would render such a task unfeasible with the current algorithm on commodity hardware. In the next Section, an alternative algorithm to solve the CHnMM Decoding task is therefore developed that has a much more favorable memory consumption pattern.

5.2 Reducing the Memory Consumption

One observation that enables the development of a more memory-efficient algorithm for the Decoding task is that the existing algorithm indeed stores the Proxels of all time steps, but does not require true random access over them. It first linearly processes each time step from first to last, requiring only access to the most recently processed time step. And after finding the Proxel representing the final state of the most likely path it linearly processes each time step last to first in order to determine the remaining discrete states on that path, requiring only access to one time step at a time.

Ideally, one would like to have an inverse modified Forward algorithm that takes the set of Proxels from one time step and reconstructs the set of Proxels from the *previous* time step. This way, the Decoding Forward computation could be executed by discarding all but the most recent time step as is done for the Evaluation problem. And to reconstruct the most likely path, one would iteratively use such an inverse Forward computation to reconstruct predecessor time step Proxel sets, determining the element of the most likely path in that time step and discarding all Proxel sets for later time steps. With such an approach, the Decoding task would be solvable by only storing two Proxel sets at any given time.

However, it is unlikely that an efficient inverse Forward algorithm is possible: In the Forward computation, successor Proxels are generated based on their predecessor by adjusting the age vector and –among others – resetting the age value corresponding to the activity whose completion is the cause of the new Proxel. An inverse Forward computation would have to take such a Proxel with zero-valued age vector elements and reconstruct the predecessor for which that value was not zero. However, without additional information, that age value could have had an arbitrary value and thus cannot be reconstructed unambiguously. Evaluating additional information such as all prior observations from the trace may limit the possible values for that age, but such an approach is likely to be prohibitively slow by itself, because evaluating all possible influences of prior observation on an age vector is a combinatorical task that essentially requires a Forward computation by itself to be solved.

Yet, the initial observation that no random access to the Proxels of all time steps is required still holds. And it can be exploited in a different way based on a second observation: Through the deterministic nature of the modified Forward algorithm, the set of Proxels for any time step can be reconstructed by performing the modified Forward step (cf. Algorithm 2) again for that time step (and for any of its predecessors if those have been discarded as well). Both observation together suggest that it is possible to discard the Proxel sets of arbitrary time steps in order to save memory, and to reconstruct them later when they are needed.

However, the backtracking part of the Decoding task requires a reconstruction of those time steps from last to first, while the reconstruction is only possible in the opposite order. Thus, an algorithm needs to be developed that discards Proxel sets for as many time steps as possible, but retains a well-chosen small subset of those time step Proxel sets in order to efficiently reconstruct the other time steps when needed.

5.2.1 Possible Proxel Set Discarding Policies

Many different policies for discarding time step Proxel sets are possible. A very simple one would be to discard the Proxel sets of every other time step, and to reconstruct those individually when actually needed for the backtracking. Such an approach would reduce the memory consumption for the Decoding task by about 50% and would require the reconstruction of only half of the Proxel sets, increasing the computation time by about 50%. Yet, if the Decoding for a given model and trace is practically unfeasible due to memory constraints, reducing the memory consumption by 50% barely improves this feasibility.

Another extreme would be to discard the Proxel sets for *every* time step that is not currently needed. For a trace of n observations, this would required n modified Forward computation steps to determine the Proxel set of the final time step and to determine the final Proxel of the most likely trace from that set. With that one Proxel, the approach would then need to perform the first $n-1$ modified Forward computation steps in order to reconstruct the Proxel set for the last but one time step in order to find the last-but-one Proxel of the most likely trace. This process would then be repeated, requiring the reconstruction of i time steps to determine the ith Proxel on the most likely path, and thus would require the reconstruction of $O(n^2)$ time steps. Therefore, this approach requires only the storage of a constant number of Proxel sets independent of the trace length and therefore has a feasible memory consumption for every trace length. However, the resulting computation time of $O(n^2)$ in the number of observations in the trace renders the approach practically unfeasible for longer traces.

We therefore chose to design a policy for discarding time step Proxel sets that has a more balanced trade-off between computation time and memory consumption. This approach is based on the divide and conquer paradigm [61]. For a given trace of observations, it determines the Proxel set for the middle time step of that trace while discarding all prior time steps Proxel sets. With this Proxel set, the Decoding task can be solved for the second half of the trace independent of the first half. And once the result for the second half (i.e. the second half of the most likely path) is known, the Decoding task can be performed for the first half independent of the second half. Since this division into two halfes can be applied recursively, the memory consumption can be reduced dramatically.

The algorithm is given as pseudocode in Algorithm 5. It uses the algorithms DecodingForwardStep and DecodingBacktrackingStep from the known Algorithms 2 and 3, respectively. Its input are the indices of the first and last time step, lo and hi, which are to be processed in this recursion, the set of Proxels R_{lo-1} from the time step before, the trace of observations O, state transition matrix A, set of activities TR, the Proxel on the most likely path of time step $hi+1$, ρ_{max}, and the numbers of discrete states N, observations T and activities K.

If the observation sequence is short enough (e.g. less than four observations), the algorithm simply executes the ordinary CHnMM Decoding algorithms developed in the previous section to determine the Proxels of the most likely path, and discards the Proxel set of each time step as soon as the Proxel belonging to the most likely path of internal states has been determined for that time step (lines 1–10). Otherwise it splits the trace into two halves (line 11). It then first

performs the modified Forward steps for the first half and discards the Proxel set for each time step whose successor has been computed (lines 12–17). The resulting Proxel set R_{mid} for the middle observation of the trace is kept, and the algorithm is executed recursively for the second half in order to determine all Proxels on the second half of the most likely path (line 18). With the Proxel of R_{mid} that is part of the most likely path now being known, the algorithm is finally executed recursively for the first half of the trace to determine the first half of the most likely path (line 19)[1].

Algorithm 5: DecodingRecursive

Input: $lo, hi, R_{lo-1}, O, A, TR, \rho_{max}, N, T, K$
Result: Sequence of internal states $q_0, \ldots, q_T$ that is most likely to have created the time-stamped signal sequence

```
1  if hi − lo < 4 then
2      for i = lo to hi do
3          Δt = o_i.e − o_{i−1}.e;
4          R_i = DecodingForwardStep ( R_{i−1}, o_i.v, Δt, A, TR, N, T, K);
5      for i = hi to lo − 1 do
6          if i = T then ρ_max = arg max_{ρ∈R_T} (ρ.α);
7          else ρ_max = DecodingBacktrackingStep (R_i, ρ_max) ;
8          R_i = ∅ ;
9          q_i = ρ_max.q;
10     return ρ_max
11 mid = ⌊(hi + lo)/2⌋;
12 Δt = o_lo.e − o_{lo−1}.e;
13 R_lo = DecodingForwardStep ( R_{lo−1}, o_lo.v, Δt, A, TR, N, T, K);
14 for i = lo+1 to mid do
15     Δt = o_i.e − o_{i−1}.e;
16     R_i = DecodingForwardStep ( R_{i−1}, o_i.v, Δt, A, TR, N, T, K);
17     R_{i−1} = ∅
18 ρ′ = DecodingRecursive (mid + 1, hi, R_mid, O, A, TR, ρ_max, N, T, K);
19 return DecodingRecursive (lo, mid − 1, R_{lo−1}, O, A, TR, ρ′, N, T, K);
```

Accuracy This recursive Decoding algorithm performs the same operations on the same data as the iterative algorithm: both construct the set of Proxels for a given time step based on the set of Proxels from the previous time step; and both follow the links of certain Proxels back to their unique predecessor in order to determine the Decoding discrete state sequence. Thus, both algorithms are also guaranteed to always compute exactly the same result.

The only difference between the two algorithms is that the recursive one occasionally discards Proxel sets and recomputes them later. But since the

[1] In the pseudocode, the return value of the function `DecodingRecursive` is *not* the answer to the Decoding task, but a value required to continue the algorithm, namely that Proxel of the `lo` time step that lies on the most likely path. The actual output of the algorithm, the elements of the most likely path of internal discrete states, are computed by line 9 and are stored in global variables q_i.

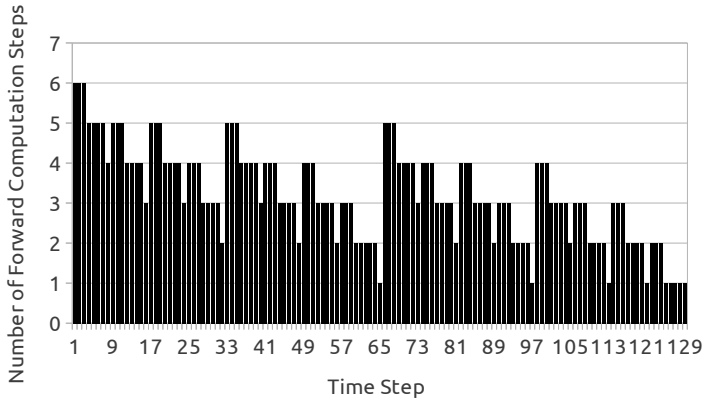

Figure 5.1: A plot of the number of times that the modified Forward step had to be executed for each time step in the recursive Decoding algorithm for a trace of about 130 observations.

recomputation is based on the same data as the original computation, this does not in any way affect the results.

Time Complexity To assess the time complexity of this algorithm in terms of the number of time steps for which the Forward computation needs to be performed it is noteworthy that this algorithm bisects the trace of observations in each recursion step. Thus, the Forward computation for a time step is possibly performed in in the first or the second half of that trace, but never in both. This means that for each *recursion depth*, the Forward computation for each time step is performed in at most a single instance of `DecodingRecursive`. Further, in each call to `DecodingRecursive`, the Forward computation is performed at most once for each of the time steps the call governs. Together with the bisection property, this means that for a trace of length n, at each *recursion depth* at most n Forward steps are performed.

Furthermore, the algorithm bisects the trace with each recursion step and terminates once the trace is short enough. This means that the algorithm has a recursion depth of at most $\lceil \log_2 n \rceil$. Together with the at most n Forward steps per recursion level, this yields an overall time complexity of $O(n \log n)$ in the number of Forward steps.

Figure 5.1 shows the number of times that the modified Forward step had to be executed for each time step of a trace of about 130 symbols in order to solve the Decoding task with this algorithm. For the iterative algorithm, the modified Forward step would have to be executed exactly once for each time step. For the recursive one this number varies from time step to time step, but – at least for this trace length – is about three on average.

Memory Complexity As mentioned for the time complexity, the recursive Decoding algorithm has a maximum recursion depth of $\lceil \log_2 n \rceil$. On each recursion depth a call stores only a constant number of time steps at the same time (two time steps for the iterative Forward computation to reach the mid-

point time step, and afterwards the midpoint time step itself, while the recursive calls are made). Thus, each recursion level requires only storage for $O(1)$ time steps, and with the maximum recursion depth of $O(log(n))$ this yields a memory consumption of $O(log(n))$ in the number of observations in the trace.

Summary As shown, the recursive algorithm for the Decoding task has a time complexity of $O(n \log n)$ and a memory complexity of $O(\log n)$ in the number of observations in the trace and thus in the number of time steps. This is only a slight increase in computational complexity from the $O(n)$ of the iterative algorithm, and a sharp reduction in memory complexity from $O(n)$ to $O(\log n)$. The slightly increased time complexity should barely impact the practical feasibility of the algorithm, while the reduction of the memory complexity should make the analysis of far longer traces feasible. Both properties are tested in the experiments of the next section.

5.3 Experiments

In this section, the feasibility of the two Decoding algorithms with respect to computation time and memory consumption will be tested experimentally.

5.3.1 Application Example

With the Decoding task it is possible to reconstruct the completion times of individual activities in a model based on a trace of observations. For example, suppose that in the production facility modelled by the Tester, both machines have been determined to cause the same high fraction of defective items (cf. first experiment in Section 4.5) and so should both be repaired. Now assume further that due to severe financial constraints the company can only repair one of the two machines. Further, some of the defective items have a characteristic defect that points to a certain easily repairable damage in one of the machines. Thus, if the company could determine whether all of these charateristic defects are indeed caused by the same machine, they could quickly and cheaply repair that one machine, while the other machine continues to produce items.

This problem can be approached with the Decoding task, which determines the most likely internal state sequence of an observation sequence (here: the production protocol). Since the CHnMM Tester was specified with two discrete states that encode, which machine produced the most recent items (cf. Section 3.5), Decoding on that model directly reconstructs the most likely sequence of completed activities as well and thus reveals the most likely machine to have produced each individual item.

For this scenario, an experiment with a synthetic trace of 1500 symbols (corresponding to about a day of observations) was conducted. In this case, the actual cause of each defect was recorded and could be compared to the Decoding result. The experiment revealed, that Decoding attributed 78.2% of all defects to the correct machine. Thus, in the application example, Decoding would show that almost 80% of the defects with a characteristic mark were caused by Machine 1 and than it is therefore likely that Machine 1 has an easily repairable damage and thus that repairing Machine 1 is the most time and cost-efficient way of reducing the number of defective items.

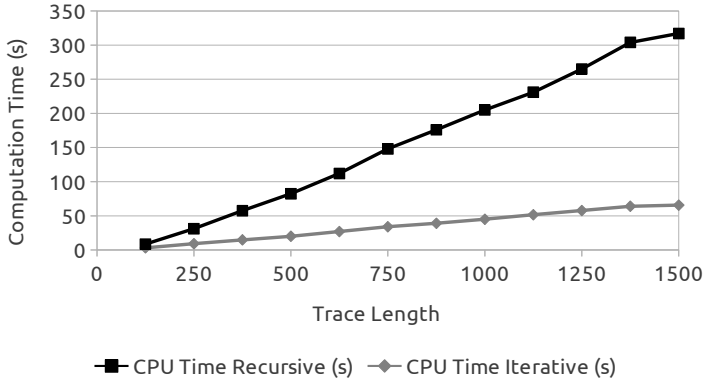

Figure 5.2: Plot of the computation time (CPU time) required by the iterative and recursive Decoding algorithms for the Car Rental Agency model for various trace lengths.

5.3.2 Comparison of Iterative and Recursive Approaches

In this Section, the iterative and recursive Decoding algorithms are tested experimentally in order to assess their similarities and differences. As argued in Section 5.2.1, both algorithms always compute exactly the same results. However, they are designed to differ in their computation time and memory consumption, and these differences are to be assessed experimentally.

For these experiments, the iterative and recursive Decoding algorithms were tested with traces of various lengths on the Car Rental Agency model. The Tester model with its two discrete states and its about ten Proxels per time step was deemed too small to obtain accurate and meaningful measurements.

Figure 5.2 shows the computation time (CPU time) for both algorithms on the Car Rental Agency model for various trace lengths. The iterative algorithm was faster than the recursive one in all instances. And the factor by which it is faster increases with increasing trace length, from about 2.8 for 125 observations to 4.8 for 1500 observations. This behavior is exactly as was expected from the theoretical time complexities of $O(n)$ for the iterative approach compared to $O(n \log(n))$ for the recursive one. Nevertheless, with the maximum computation time of about five minutes for the tested trace lengths, the slightly higher computational complexity of the recursive algorithm barely impacts its practical feasibility.

The bigger difference between the two algorithms lies in the different memory consumption patterns. Those are shown in Figure 5.3. The numbers represent the actual memory consumption of the Proxels (here 96 bytes per Proxel) stored concurrently. Furthermore, to interpret these numbers correctly, it is noteworthy that previous experiments (cf. Section 4.5) showed that the number of Proxels per time step seems to be bounded, but that the actual number varies randomly even between successive time steps.

The left hand side of Figure 5.3 shows the memory consumption of the iterative algorithm. Since the peak number of Proxels for the iterative algorithm is the sum of Proxels for all time steps, the effects of randomness are almost elimi-

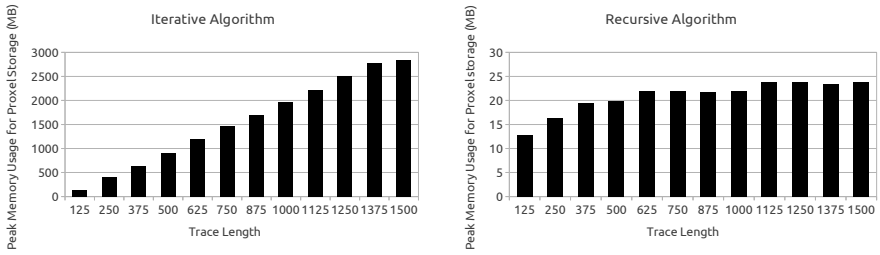

Figure 5.3: Plot of the memory consumption for storing the required Proxels for the Car Rental Agency model under different trace lengths. The diagram on the left-hand side shows the peak memory consumption for a single trace under the iterative approach. The right-hand side shows the average of the peak memory consumptions of 16 traces under the recursive Decoding algorithm. Note the different scales for the memory consumption between the two graphs.

nated through that summation and thus the graph shows a clear linear increase in the memory consumption for increasing trace length. For the tested model and maximum trace length of 1500 symbols, that peak memory consumption is already 2.8GB, an amount that barely fits in 2011 commodity hardware. Decoding longer traces with this algorithm would therefore require memory paging [78], which would slow down the computation by about two orders of magnitude and would thus render the approach practically unfeasible for longer traces.

The right hand side of Figure 5.3 shows the corresponding memory consumption for the recursive algorithm. Here, the peak memory usage is determined by only a few time steps that are held in memory concurrently and thus the effect of the random sizes of these time step Proxel sets on the actual memory usage is noticeable. Thus, the peak memory usage of the recursive algorithm for a single trace behaves erratically under increasing trace length. But with the shown average of the peak memory consumption over 16 traces, a sublinear increase in memory consumption with increasing trace length can be observed. This fits well with the theoretical $O(\log(n))$ memory usage.

Between the two algorithms it is noteworthy that the highest measured memory consumption of the recursive algorithm is less than 1% of the memory consumption for the iterative algorithm. Together with the practical computation time measurements this means that the recursive Decoding algorithm should be practically feasible for far longer traces than those analyzed in this experiment. The iterative algorithm on the other hand is slighty faster, but its much higher memory consumption limits its practical applicability on today's commodity hardware to traces about as long as those that these experiments were conducted with.

5.4 Possible Extensions

Alternative Approaches to the Recursive Decoding Algorithm In this chapter, the basic iterative Decoding algorithm has been argued to have a too high memory footprint, and the recursive divide and conquer algorithm has been developed as an alternative. Yet, several other approach are feasible to reduce

the memory footprint of the Decoding task.

First, merging of Proxels with the modified Forward step effectively discards all but one successor Proxel with a given state. Thus, some Proxels may not have successors in the next time step. Without successors these can never be part of the most likely path of internal states and can consequently be discarded. When being discarded, *their* former predecessor Proxels may in turn no longer have successors and may be discarded as well. The whole process of discarding successorless Proxels can continue up to the Proxel set of the first time step. Such an individual Proxel discarding scheme may be implemented through reference counting of the Proxels, or may occur naturally in programming languages that feature a tracing garbage collector [85]. Depending on the model structure, this approach may discard a substantial fraction of the model's Proxels, reducing the approach's memory footprint. And since relevant Proxels are never discarded (as opposed to the recursive Decoding algorithm which also discards relevant Proxels and recomputs them later), the links to predecessor Proxels could directly be implemented as pointers and need not replicate the state of the precessor Proxel, further reducing memory consumption. The downside of this approach is its increased complexity and the unpredictability of how many Proxel can actually be discarded. Furthermore, memory for Proxels needs to be allocated per Proxel (instead of using a region-based approach to memory management), further reducing the efficiency of the algorithm.

Second, other Proxel discarding schemes are possible. For example, one may decide to perform the initial modified Forward computation for all n time steps and to retain every $\sqrt{n}$th time step as a "checkpoint". The backtracking part of the Decoding path then initially requires only the reconstruction of the set of the $\sqrt{n}$ time steps between the last checkpoint and the final time step. All of those are kept in memory and the Proxels on the most likely path are determined of all those time steps. Afterwards, the Proxel sets for all these time steps can be discarded and those for the time steps between the last and the last but one checkpoint can be reconstructed in order to determine their Proxels on the most likely path, and so on for all checkpoint intervals up to the first time step. Overall, this approach requires the concurrent storage of $2\sqrt{n}$ time step Proxel sets and each of the Proxel sets has to be computed at most twice (once to generate the checkpoints, and once for the backtracking between the checkpoints). The approach is generalizable to a hierarchy of m levels of checkpoints, yielding a memory complexity of $O(m\sqrt[m]{n})$ and a time complexity of $O(mn)$ time steps.

Third, the original iterative CHnMM Decoding algorithm may be used and Proxel sets of time steps that are not currently used may simply be stored on background storage devices such as hard disks or even digital tape, which both feature abundant storage capacity. Since all Proxels of a time step can be read or written at the same time, random access is not required and even those mechanical storage systems can sustain a relatively high throughput. However, even their linear read/write throughout is usually one to two orders of magnitude lower than that of PC RAM. Furthermore, writing data to disc usually requires some kind of serialization, which further slows down the computation.

And finally, instead of requiring links to parents and backtracking each Proxel may directly store the most likely path that lead to it. With this, the most likely path can easily be found after the modified Forward computation has reached the final time step, since it is already completely stored in the

Proxel with the highest probability for that time step. Thus, no backtracking is required and time steps whose successors have been computed can be discarded as with the original Forward algorithm for the Evaluation task. The downside of this approach is that the size of Proxels grows with the trace length. Thus, for longer traces, it might be unfeasible to even keep the Proxels of a single time step in memory. And since the list of predecessor Proxels grows in size, in a naïve implementation, the time complexity of generating a Proxel will grow linearly with the number of observations in the trace.

Extensions to the Decoding Task The algorithms that solve the Decoding task may be slightly modified to solve further tasks on CHnMMs.

First, since the observations are caused by the completion of activities and not by remaining in a state (as with HMMs), a practitioner may be interested in the most likely sequence of completed activities instead of the most likely sequence of discrete states passed. Fortunately, the latter can easily be converted into the former: for CHnMMs, at most one activity causes the state change from one state S_i to another state S_j. Thus, if on the most likely path the nth discrete state is S_i and the $n+1$th discrete state is S_j, then the $n+1$th activity whose completion caused that state change must have been a_{ij}. This lookup can be performed for every pair of adjacent most likely states to retrieve the sequence of most likely activities.

Second, it might be of interest to not only find the one most likely path of internal system states, but to find the n most likely paths. In order to find those, it is not sufficient to find the n Proxels of the final time step with the highest probabilities and to backtrack the corresponding paths, because the modified Forward algorithm discards all Proxels that are not locally part of the most likely path and thus no Proxel in the final time step may even exist for the second most likely to nth most likely path. Instead, Proxels would have to separately store the path probabilities of the n most likely paths through them along with the predecessors' discrete state and age vector for all n paths. On Proxel merging, only those paths that do not fall into the n most likely list are discarded. With this approach, it is guaranteed that Proxels exist in the final time step for all n most likely paths, and that those paths can be backtracked. Multiple of those top n paths may end with the same Proxel, but do not need to.

Third, in addition to the most likely path itself, the Decoding algorithm may be modified to compute the conditional probability of that path given the observation sequence. This probability can be useful to assess whether the most likely path is overwhelmingly likely to have caused the observation, or whether it is just slightly more likely than many other paths. That conditional probability for a trace of n observations is formally:

$$P(Q|O) = \frac{P(Q \cap O)}{P(O)} = \frac{s^n P(Q \cap O)}{s^n P(O)} = \frac{\overline{P}(Q \cap O)}{\overline{P}(O)}$$

Here, s^n is the unknown scaling factor introduced through the virtual instantaneous state change probability of each of the n Forward computation steps (cf. Equation 4.2 on page 44). With its help the equation shows that the desired conditional probability can be computed as a fraction of two virtual probabilities even though the scaling factor used in computing these virtual probabilities

remains unknown. The virtual probability of the numerator is the one computed for the Proxel of the final time step that ends the most likely path as part of the modified Forward computation. It is thus already computed as a side effect of solving the Decoding problem. The denominator is the Evaluation probability as computed in Chapter 4. So, the probability of the most likely internal path given an observation sequence can be computed alone from (intermediate) results of the existing Decoding and Evaluation algorithms.

5.5 Conclusion

In this Chapter two algorithms that solve the CHnMM Decoding task have been developed: The iterative algorithm is a slightly modified version of the Forward algorithm used for the Evaluation task. The recursive algorithm on the other hand uses a divide and conquer approach to solve the task with a slightly higher time complexity but with a much lower memory consumption.

With that recursive approach, an exact algorithm exists for the CHnMM Decoding problem that is practically feasible even for very long traces with potentially hundreds of thousands of observations. Thus, the goal of providing an exact practically feasible CHnMM Decoding algorithm is reached.

Additionally, the chapter presented several possible alternative approaches to the time-memory tradeoff of the recursive algorithm, and several possible extensions to the Decoding algorithm in general to reconstruct further properties of the unobserved behavior.

In the next chapter, an algorithm for the CHnMM Smoothing problem will be developed based on the CHnMM Forward algorithm from Chapter 4 and the recursive CHnMM Decoding algorithm from this chapter.

Chapter 6

The Smoothing Task

6.1 Introduction

The Smoothing task is the task to determine the probability of the model to be in a particular state (i.e. discrete state and age vector) after a particular observation, given the whole observation sequence, which generally includes observations before *and after* the observation in question. Formally, it computes

$$P(q_t = S_i \cap age_t = \vec{\tau} | o_1 \dots o_t \dots o_T).$$

It is therefore similar to the results computed with the Forward algorithm, which determines the probability of the model to be in a particular state at a particular time, given only the observations from the past of the observation. However, since the Smoothing task includes more information in the form of future observations, its behavior reconstruction is usually more accurate.

The Smoothing task is also similar to the Decoding task in that the Smoothing probabilities can be used to determine the most likely model state for each time of an observation. Yet, while the Decoding task determines the most likely sequence of discrete internal states, the most likely states reconstructed by the Smoothing task need not form a valid sequence. Thus, Smoothing may reconstruct the most likely discrete state at time t to be S_i and the most likely discrete state at time $t+1$ to be S_j even though there is no possible discrete state change from S_i to S_j. Which behavior reconstruction task is more useful depends on the particular problem to be solved.

6.2 Developing a CHnMM Smoothing Algorithm

Analogous to the derivation of the Smoothing probability for HMM in Section 2.1.4, the Smoothing probability for CHnMMs can also be split into

$$P(q_t = S_i \cap age_t = \vec{\tau} | o_1 \dots o_t \dots o_T) = \frac{P(q_t = S_i \cap age_t = \vec{\tau} \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_i \cap age_t = \vec{\tau})}{\sum_{\vec{\tau}'} \sum_j P(q_t = S_j \cap age_t = \vec{\tau}' \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_j \cap age_t = \vec{\tau}')} \quad (6.1)$$

So the probability of the system to be in a particular state at a particular time given a trace of observations is the normalized product of the probability to have reached that state at that time while having emitted the trace so far, and the probability to still be able to emit the remaining trace given the state at the time.

Thus, the task of computing the probability of the model to be in a particular state given a trace of observations can be split into a fraction containing individual products of probabilities to be in a given state and having emitted a prior trace, with the corresponding probabilities of being able to emit the remaining trace given the current state. This fraction may be expanded arbitrarily by s^t, the unknown scaling factor by which the Forward probabilities have been scaled after t time steps (cf. Section 4.3.3), to yield

$$\frac{s^t P(q_t = S_i \cap age_t = \vec{\tau} \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_i \cap age_t = \vec{\tau})}{\sum_{\vec{\tau}'} \sum_j s^t P(q_t = S_j \cap age_t = \vec{\tau}' \cap o_1 \dots o_t)\ P(o_{t+1} \dots o_T | q_t = S_j \cap age_t = \vec{\tau}')}. \tag{6.2}$$

Here, the $s^t P(q_t = S_j \cap age_t = \vec{\tau}' \cap o_1 \dots o_t)$ are the virtual Forward probabilities of the model to be in a given discrete state S_j with age vector $\vec{\tau}'$ after the final observation o_t of the partial trace $o_1 \dots o_t$ has been emitted. The Forward algorithm already generates individual Proxels for all states for which this probability is non-zero, and therefore provides part of the solution to the CHnMM Smoothing task.

Furthermore, in this equation the denominator consists only of summands which are in turn products of a single virtual Forward probability and another probability for the same state. Thus, whenever a Forward probability is zero, the product is zero as well and can be omitted from the sum. Therefore, in conjunction with the fact that the Forward algorithm computes *all* existing non-zero Forward probabilities, the equation can be simplified by omitting all factors from the denominator for which the CHnMM Forward algorithm did not create a Proxel. Thus, given the set of Proxels R_t from the Forward algorithm for the tth time step, and the Proxel $\rho' \in R_t$ representing the state for which the Smoothing probability is sought, the equation to compute the Smoothing probability can be simplified to

$$\begin{aligned} &P(q_t = S_i \cap age_t = \vec{\tau} | o_1 \dots o_t \dots o_T) = \\ &\frac{\rho'.\alpha\ P(o_{t+1} \dots o_T | q_t = \rho'.q \cap age_t = \rho'.\vec{\tau})}{\sum_{\rho \in R_t} \rho.\alpha\ P(o_{t+1} \dots o_T | q_t = \rho.q \cap age_t = \rho.\vec{\tau})}. \end{aligned} \tag{6.3}$$

If no such Proxel ρ' for the state in question exists then the Forward probability for that state is zero. Consequently, the numerator of the equation and thus the whole Smoothing probability for that state is zero as well and no further computations are required to yield that result.

Otherwise, the Forward algorithm provides half of the values required to compute Smoothing probabilities. The probabilities that yet remain to be determined are of the form

$$P(o_{t+1} \dots o_T | q_t = \rho.q \cap age_t = \rho.\vec{\tau}).$$

These are the so-called Backward probabilities of the model to emit the remaining observations $o_{t+1} \dots o_T$ given that at the time of the tth symbol emission the system was in a particular discrete state with a particular age vector.

6.3 Computing Backward Probabilities

To compute those Backward probabilities, a similar approach to the Forward algorithm is possible by starting from the final time step and iterating backwards: For the final time step, all observations from the trace have already been emitted and thus the Backward probabilities to emit all remaining observations given that the model is in a particular discrete state with a particular age vector is always one, independent of the actual state and age vector.

With this induction initialization in place, the remaining Backward probabilities can be computed through backward induction in the same way as is done for HMMs [23]: the probability of a model to emit the remaining trace $o_t \dots o_T$ given the current discrete state and age vector is the sum over all possible successor states of the probabilities to change the discrete state to the successor state under the emission of the observation o_t, times the probability that the now remaining trace $o_{t+1} \dots o_T$ can still be emitted given the successor state. Here, the latter is a Backward probability for the next time step, which is already known (since the induction for the Backward algorithm starts at the final time step and works backwards). And the former is the same state change probability that was used in the Forward algorithm in Chapter 4 (cf. Section 4.3.4).

While this approach determines *how* Backward probabilities can be computed, a problem that remains is to determine for *which* states the Backward probabilities have to be computed. This problem also occured in the development of a memory-saving Decoding algorithm (cf. Section 5.2). It is related to the information loss caused by resetting the duration of an activity to zero after it has been completed, which is essential to enable Proxel merging and thus to make any Proxel-based approach feasible. In the Forward algorithm, age vector elements are set to zero when either the activity has been completed or cancelled, and thus, its prior age is no longer relevant.

But the Backward algorithm iterates from the final time step backwards to the first one. Thus, if an age element is zero in a Proxel for one time step, its value in the previous time step has to be reconstructed in order to generate the predecessor Proxel and to compute its Backward probability. But if the activity was cancelled due to the state change in question then the prior age value could have been potentially any real value inside the support of the duration probability distribution of the activity. Thus its prior value is unknown, preventing the development of a dedicated CHnMM Backward algorithm.

A look at Equation 6.1 offers a possible solution: The equation to compute a Smoothing probability depends only on products of Forward and Backward probabilities that share a common discrete state and age vector. Since a product is zero if at least one factor is zero, it follows that the Backward probabilities need only be computed for those states for which the Forward probabilities are non-zero. Consequently, the Backward computation need only be performed for those combinations of discrete state and age vector for which the Forward algorithm has created Proxels. Thus, while the CHnMM Backward computation

seems not to be possible as a dedicated algorithm (which is in contrast to what was possible for HMMs), it can be performed as an extension to the Forward algorithm by using backward induction to add Backward probabilities to those Proxels that the Forward algorithm created. Consequences of this approach are that the Forward and Backward computations for CHnMMs cannot be run in parallel (as was possible for HMMs), since the beginning of the Backward computation depends on the output of the Forward computation, and that the Proxels from the Forward computation for all time steps need to be retained (as with the Decoding algorithm) in order to compute the Backward probabilities.

A final problem in the computation of Smoothing probabilities is that the Backward probabilities are computed using the same factors as for the Forward probabilities, and thus the Backward probabilities are also *virtual* probabilities, while Equation 6.3 requires ordinary Backward probabilities. However, extending the fraction on the right-hand side of the equation with s^{T-t}, the unknown factor by which the backward probabilities of the tth time step differ from the corresponding actual unknown backward probabilities, yields

$$\begin{aligned} &P(q_t = S_i \cap age_t = \vec{\tau} | o_1 \dots o_t \dots o_T) = \\ &\frac{\rho'.\alpha \; s^{T-t} P(o_{t+1} \dots o_T | q_t = \rho'.q \cap age_t = \rho'.\vec{\tau})}{\sum_{\rho \in R_t} \rho.\alpha \; s^{T-t} P(o_{t+1} \dots o_T | q_t = \rho.q \cap age_t = \rho.\vec{\tau})} = \\ &\frac{\rho'.\alpha \; \overline{P}(o_{t+1} \dots o_T | q_t = \rho'.q \cap age_t = \rho'.\vec{\tau})}{\sum_{\rho \in R_t} \rho.\alpha \; \overline{P}(o_{t+1} \dots o_T | q_t = \rho.q \cap age_t = \rho.\vec{\tau})}. \end{aligned} \tag{6.4}$$

So the Smoothing probabilities can be computed from the *virtual* Forward and Backward probabilities alone. With this information, all problems in the development of a CHnMM Smoothing algorithm are solved.

6.4 The CHnMM Smoothing Algorithm

Since the Smoothing problem requires Forward and Backward probabilities to be computed for the same set of discrete states and age vectors, it is useful to extend the Proxel definition to contain both. We therefore define a Proxel for the Smoothing problem as $\rho(q, \vec{\tau}, \alpha, \beta, \gamma)$, where the additional element β holds the Backward probability of the state, and γ holds its Smoothing probability.

An Algorithm for the CHnMM Smoothing task then needs to perform the following steps:

- Perform an ordinary CHnMM Forward computation for every time step of the given trace. Here, the additional Proxel elements β and γ are simply set to zero
- For all Proxels of the final time step, set the Backwards probability β to one
- Perform the inductive Backward computation by determining the Backward probability of all Proxels generated by the Forward algorithm, starting from the last-but-one time step and ending with the first one. Here,

the Backward probability of a Proxel is computed as the sum over all activities in its state, where each summand is the product of

- the probability $P_{sojourn}$ that no activity is completed between the two time steps
- the virtual probability P_{change} that the activity in question is completed right at the time of the next symbol emission, given that no activity has been completed before
- the probability that the observed symbol is emitted, given that the activity in question was completed
- and the probability that the remaining trace of observations may still be emitted, i.e. the Backward probability of the successor state.

- Determine the Smoothing probability of each Proxel. With the given Proxel definition containing Forward and Backward probabilities, Equation 6.4 for the Smoothing probabilities of each Proxel ρ' of the tth time step simplifies to

$$\rho'.\gamma = \frac{\rho'.\alpha \ \rho'.\beta}{\sum_{\rho \in R_t} \rho.\alpha \ \rho.\beta}. \tag{6.5}$$

The complete Forward-Backward algorithm is given in pseudocode below in three parts: Algorithm 6 details a single Forward step. Given state transition matrix A, the set of activities TR the set of Proxels R_t for the tth time step, the symbol of the $t+1$th observation, the duration Δt between the tth and the $t+1$th time step and the numbers of discrete states N and activities K, it computes the set of Proxels R_{t+1} for the $t+1$th time step and determines their Forward probabilities. The only difference to the steps of the original CHnMM Forward algorithm (cf. Algorithm 1 on page 46) is that the additional Proxel elements β and γ are set to zero.

The corresponding Smoothing Backward step is given in Algorithm 7. Its input is the same as for the Forward step, apart from the set of Proxels R_{t+1} for time step $t+1$, for which Forward and Backward probabilities are already computed, and the set of Proxels R_t for time step t for which Forward probabilities have already been computed, and Backward probabilities *shall* be computed.

The final iterative CHnMM Forward-Backward algorithm to solve the Smoothing task is given in Algorithm 8. It uses the Forward and Backward steps from Algorithms 6 and 7 to compute the final smoothing probabilities. It assumes that the trace O begins with a dummy observation with an arbitrary symbol and the time stamp 0, so that the length of the first time step can be computed as a time stamp difference in the same way as all other time step lengths.

Its output is the sets of Proxels for all time steps, where each Proxel contains Forward, Backward and Smoothing probabilities.

With the result of this algorithm, the Smoothing probabilities for all states (discrete state and age vector) at all times of symbol emissions can be determined: If a Proxel in the given time step exists with the given state, then the desired Smoothing probability is the Smoothing probability of that Proxel, otherwise the desired Smoothing probability is zero.

Algorithm 6: SmoothingForwardStep

Input: $A, TR, R_t, v_{t+1}, \Delta t, N, K$
Output: R_{t+1}

$R_{t+1} = \varnothing$;
foreach $\rho \in R_t$ **do**
  i $= \rho.q$;
  $p_{sojourn} = \prod_{j \in \{1,\ldots,N|\ a_{ij} \neq \varnothing\}} \frac{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id} + \Delta t)}{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id})}$;
  foreach $j \in \{1, \ldots, N | a_{ij} \neq \varnothing\}$ **do**
    $Row_i = \{a_{i1}, \ldots, a_{iN}\}$;
    $Row_j = \{a_{j1}, \ldots, a_{jN}\}$;
    $\vec{\tau}' : \tau'_k =$

$$\begin{cases} \rho.\tau_k + \Delta t & \text{if } TR_k \in Row_i \wedge TR_k \neq a_{ij} \wedge \neg isExp(TR_k.dist) \wedge \\ & (TR_k \in Row_j \vee TR_k.aging) \\ \rho.\tau_k & \text{if } TR_k \notin Row_i \wedge TR_k.aging \wedge \neg isExp(TR_k.dist) \\ 0 & \text{otherwise} \end{cases}$$

    $\mu = hrf(a_{ij}.dist)$;
    $\rho' = (j, \vec{\tau}', \rho.\alpha * p_{sojourn} * \mu(\rho.\tau_{a_{ij}.id} + \Delta t) * a_{ij}.b(v_{t+1}), 0, 0)$;
    if $\rho'.\alpha = 0$ **then** continue;
    if $\exists \rho'' \in R_{t+1}$ *with* $(\rho''.q = \rho'.q \wedge \rho''.\vec{\tau} = \rho'.\vec{\tau})$ **then**
      $\rho''.\alpha += \rho'.\alpha$;
    else $R_{t+1} = R_{t+1} \cup \{\rho'\}$;
return R_{t+1}

Algorithm 7: SmoothingBackwardStep

Input: $A, TR, R_t, R_{t+1}, v_{t+1}, \Delta t, N, K$
Output: R_n updated with backward probabilities

foreach $\rho \in R_t$ **do**
  $i = \rho.q$;
  $p_{sojourn} = \prod_{j \in \{1,\ldots,N|\ a_{ij} \neq \varnothing\}} \frac{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id} + \Delta t)}{1 - cdf(a_{ij}.dist)(\rho.\tau_{a_{ij}.id})}$;
  foreach $j \in \{1, \ldots, N | a_{ij} \neq \varnothing\}$ **do**
    $Row_i = \{a_{i1}, \ldots, a_{iN}\}$;
    $Row_j = \{a_{j1}, \ldots, a_{jN}\}$;
    $\vec{\tau}' : \tau'_k =$
    $$\begin{cases} \tau_k + \Delta t & \text{if } TR_k \in Row_i \wedge TR_k \neq a_{ij} \wedge \neg isExp(TR_k.dist) \wedge \\ & (TR_k \in Row_j \vee TR_k.aging) \\ \tau_k & \text{if } TR_k \notin Row_i \wedge TR_k.aging \wedge \neg isExp(TR_k.dist) \\ 0 & \text{otherwise} \end{cases}$$
    $\mu = hrf(a_{ij}.dist)$;
    $\rho' =$ the one element of R_{t+1} with $\rho'.q = j \wedge \rho'.\vec{\tau} = \vec{\tau}'$)
    $\rho.\beta += p_{sojourn} * \mu(\rho.\tau_{a_{ij}.id} + \Delta t) * a_{ij}.b(v_{t+1}) * \rho'.\beta$
$pSum = 0$;
foreach $\rho \in R_t$ **do** $pSum += \rho.\alpha * \rho.\beta$;
foreach $\rho \in R_t$ **do** $\rho.\gamma = \frac{\rho.\alpha * \rho.\beta}{pSum}$
return R_t

Algorithm 8: SmoothingIterative

Input: A, TR, Π, O, N, K, T
Output: $\{R_0, \ldots, R_T\}$, each containing proxels with Forward, Backward and Smoothing probabilities

$R_0 = \bigcup_{i \in \{1,\ldots N | \pi_i \neq 0\}} \{(i, \vec{0}, \pi_i, 0, 0)\}$;
for $t = 1$ **to** T **do**
  $\Delta t = o_t.e - o_{t-1}.e$;
  $R_t =$ `SmoothingForwardStep` $(A, TR, R_{t-1}, o_t.v, \Delta t, N, K)$;
foreach $\rho \in R_T$ **do** $\rho.\beta = 1$;
for $t =$ *T-1* **to** *0* **do**
  $\Delta t = o_{t+1}.e - o_t.e$;
  $R_t =$ `SmoothingBackwardStep` $(A, TR, R_t, R_{t+1}, o_{t+1}.v, \Delta t, N, K)$;

Additionally, one may want to determine the Smoothing probability for a discrete state, i.e. the probability that the model is in a given discrete state S_i at the time t of a given symbol emission, given the observation sequence, but irrespective of the particular age vector. This probability can easily be determined from the Proxels of the Forward-Backward algorithm by summing up the Smoothing probabilities for all Proxels from R_t that share the discrete state S_i:

$$P(q_t = S_i|O) = \sum_{\rho \in \{R_t|\ \rho.q = S_i\}} \rho.\gamma$$

This Decoding probability can be used to determine the individually most likely discrete state at a given time, similarly to the Decoding task.

6.5 Reducing the Memory Consumption

The algorithm developed so far solves the Smoothing task, but has the same feasibility problem as the Decoding algorithm: For the algorithm to work, all Proxel sets of all time steps have to be retained. This increases the memory consumption about linearly with the number of observations in the trace (assuming that the number of Proxels per time step is bounded) and thus makes the approach unfeasible for long traces.

The remedy, too, is very similar to that of the Decoding algorithm: For Decoding, the backtracking progresses from the last time step to the first one, and the memory-efficient divide-and-conquer algorithm recomputes the Forward probabilities as needed. For Smoothing, the Backward computation progresses from the last time step to the first one, and the same divide-and-conquer approach may be used to recompute the corresponding Forward probabilities.

One additional issue for Smoothing is the size of the result set: for Decoding, the result was the sequence of the most likely discrete states with a single symbol per observation, which could easily be stored for almost arbitrary trace length. But for Smoothing, the result is the set of the Smoothing probabilities for all Proxels of all time steps; exactly the set of Proxels that is too big to make the approach feasible. Thus, if random access is required to the set of all Smoothing probabilities, then the approach is indeed limited to rather short traces (e.g. about 1000 observations for the Car Rental Agency model).

Yet, in practical applications, random access to that huge set is usually not required. For example, to determine the most likely discrete state after each observation, the Proxel sets for each time step can be processed individually, the most likely discrete state is extracted and the remaining Proxels can be discarded. And in the CHnMM Training (cf. Chapter 7) algorithm based on the Smoothing task the Smoothing results can also be processed individually for each time step.

Thus, in order to reduce the memory consumption for the Smoothing task, we propose a change in architecture: Instead of providing the user with the whole set of Smoothing Proxels for all time steps at once, he is provided with the Proxel set containing Forward, Backward and Smoothing probabilities for each two consecutive time steps[1] in turn, and can perform arbitrary data analysis

[1] How many time step Proxel sets should be provided to the user at the same time depends on the application of the Smoothing task. To determine the most likely discrete state of

on those Smoothing probabilities. Afterwards, the set of Proxels from the later time step is regarded as obsolete and can be discarded to save memory.

Algorithm 9: SmoothingRecursive

Input: A, TR, O, lo, hi, R_{lo} (containing Forward probabilities, but no Backward or Smoothing probabilities), R_{hi+1} (containing both Forward, Backward and Smoothing probabilities), $EvFunc, N, T, K$

Result: Proxels of time step $hi+1$ have been deleted (if they existed), Forward and Backward algorithms have been executed for time steps lo to hi and the results of these time steps have been passed to EvFunc for analysis. Backward and Smoothing variables have been added to time step lo

```
1  if hi − lo < 4 then
2      for i = lo + 1 to hi do
3          Δt = o_i.e − o_{i−1}.e;
4          R_i = SmoothingForwardStep (A, TR, R_{i−1}, o_i.v, Δt, N, K);
5      for i = hi to lo do
6          if i = T then
7              foreach ρ ∈ R_T do ρ.β = 1;
               pSum = Σ_{ρ∈R_t} ρ.α ∗ ρ.β;
8
9              foreach ρ ∈ R_T do ρ.γ = ρ.α∗ρ.β / pSum;
10             EvFunc (A, TR, R_hi, ∅, O, hi, N, T, K);
11             continue;
12         Δt = o_i.e − o_{i−1}.e;
13         R_i = SmoothingBackwardStep (A, TR, R_i, R_{i+1}, o_i.v, Δt, N, K) ;
14         EvFunc (A, TR, R_i, R_{i+1}, O, i, N, T, K);
15         R_{i+1} = ∅ ;
16     return R_lo;
17 mid = ⌊(hi + lo)/2⌋;
18 for i = lo+1 to mid do
19     R_i = SmoothingForwardStep (A, TR, R_{i−1}, o_i.v, o_i.e − o_{i−1}.e);
20     if i − 1 > lo then R_{i−1} = ∅;
21 R_mid = SmoothingRecursive (A, TR, O, mid, hi, R_mid, R_{hi+1}, EvFunc);
22 return SmoothingRecursive (A, TR, O, lo, mid − 1, R_lo, R_mid, EvFunc);
```

The whole recursive divide-and-conquer Smoothing algorithm is given in Algorithm 9. It uses the Smoothing Forward and Backward steps as given in the Algorithms 6 and 7. As with the iterative Smoothing algorithm, it assumes that a dummy symbol with time stamp zero has been introduced at the beginning of the trace, in order to not require special code to determine the duration of the first time step.

each time step, providing each set individually is sufficient. Here, the decision to provide the user with two consecutive time steps at once was made with regard to the Training task (cf. Chapter 7), which requires the Proxel sets for two consecutive time steps to be provided at the same time.

The whole algorithm is very similar to the recursive Decoding algorithm (cf. Algorithm 5 on page 66). The major difference between the two is that for the Smoothing algorithm, the user provides an evaluation function `EvFunc`. This function is called for each pair of consecutive time step Proxel sets and performs the data analysis on those Proxels that is required in the respective application area.

Algorithm 10: ExampleEvalFunc

Input: $A, TR, R_t, R_{t+1}, O, t, N, T, K$

Result: Determines the Probability $P(q_5 = S_1|O)$ to have been in state S_1 after the emission of the fifth symbol, and stores it in the global variable $globalProb$.

1 **if** $t \neq 5$ **then return**;

2 $globalProb = \sum_{\rho \in R_t \text{ s.t. } \rho.q = S_1} \rho.\gamma$;

A very simple example of such an evaluation function is given in Algorithm 10. It uses the Smoothing task to determine the probability that the model is in the discrete state S_1 right after the fifth observation. Note that this function does not access most of the parameters passed to it. Those are present for alternative evaluation functions that may need those values.

With different evaluation functions for the recursive Smoothing algorithm, different aspects of unobserved behavior can be reconstructed. In the next Experiments section, one additional example of such an evaluation function is given, and the recursive and iterative algorithms are tested with respect to computation time and memory consumption.

6.6 Experiments

6.6.1 Application Example

For an application example we resort to the example from used for Decoding in Chapter 5: In the Tester model (cf. Figure 3.1 on page 32), both machines produce items with the same defective probability. The individual defective items have to be attributed to the originating machine in order to better plan the repairs. We again use the same synthetic trace of 1500 symbols (about a day of observations) for this task, but this time we use the Smoothing task to determine the most likely source for each individual observation. This is done with the Smoothing evaluation function for the recursive Smoothing algorithm that is given in Algorithm 11. It simply sums up the Smoothing probability of Proxels having the same discrete state, and then uses a majority vote to determine the most likely source.

The results of this experiment are shown in Table 6.1. Using this Smoothing-based approach 83.8% of the produced items could be attributed to the correct machine, whereas the Decoding-based approach (cf. Section 5.3) only attributed 78.2% of the items correctly. This difference was to be expected, since the Smoothing approach indeed attempts to identify the most likely correct source of each observation, while Decoding only finds the most likely *sequence* of sources,

Algorithm 11: MostLikelyStateEvalFunc

Input: $A, TR, R_t, R_{t+1}, O, t, N, T, K$
Result: The most likely discrete state after the tth symbol emission q_t^*

1 **if** $t = 0$ **then return**;
2 $p_{M1} = \sum_{\rho \in R_t \text{ s.t. } \rho.q = S_{prev\_from\_1}} \rho.\gamma;$
3 $p_{M2} = \sum_{\rho \in R_t \text{ s.t. } \rho.q = S_{prev\_from\_2}} \rho.\gamma;$
4 **if** $p_{M1} \geq p_{M2}$ **then**
5 $\quad q_t^* = S_{prev\_from\_1};$
6 **else**
7 $\quad q_t^* = S_{prev\_from\_2};$

	Decoding	Smoothing
Correctly Classified	78.2%	83.8%
Incorrectly Classified	21.8%	16.2%

Table 6.1: Results for the attribution of produced items to the correct source for a trace of 1500 observations of the Tester model.

even if some of the elements of this sequence are rather unlikely. The downside of the more accurate Smoothing approach is that its reconstructed sources may not form a valid sequence of internal states. For example, the Smoothing approach may attribute four consecutive observations to the same machine, even though it is virtually impossible in this setting that the other machine has not produced a single item in this time interval. Whether the more accurate Smoothing or the guaranteed consistent Decoding is preferable depends on the actual application scenario.

6.6.2 Comparison of Iterative and Recursive Approaches

To assess the feasibility of the iterative and the recursive Smoothing algorithms their computation time and memory consumption are measured. Here, again, the Car Rental Agency model is used for all experiments, since the Tester model is too small and is solved too quickly for accurate measurements.

In the first experiment the computation time under increasing trace length is assessed. Figure 6.1 shows the results for both algorithms, and the corresponding values for the Decoding algorithms for comparison. For Decoding and Smoothing, the iterative algorithm is faster than the recursive one, since it does not have to recompute the Forward probabilities for discarded time steps.

For each type of algorithm (recursive and iterative), the Smoothing algorithm is slightly slower than the corresponding Decoding algorithm. The explanation is that for Decoding, an algorithm has to perform a certain number of Forward steps; the additional backtracking involves only locating a single Proxel per time step, which barely impacts the computation time. For Smoothing on the other hand, in addition to the Forward steps a Backward computation step

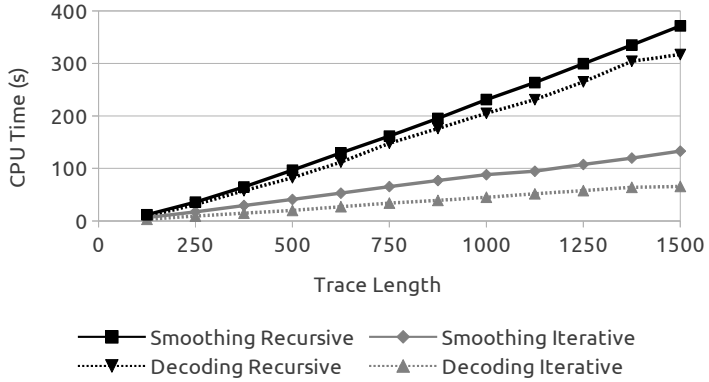

Figure 6.1: Plot of the computation time (CPU time) required by the iterative and recursive Smoothing algorithms for the Car Rental Agency model for different trace lengths. The computation time of the corresponding Decoding algorithms is shown for comparison.

with essentially the same computation time as the Forward step has to be performed for every time step. This also explains why the computation times of the Smoothing algorithms differ by a much smaller factor (about three vs. five) than those of the Decoding algorithms: For Smoothing, both algorithm have perform the same number of Forward steps as their Decoding counterparts. But the Smoothing algorithms both additionally have to perform the same number of additional Backward steps, which brings their relative computation times closer together.

Overall, the computation times of the Smoothing algorithms are very similar to those of the Decoding algorithm. And as with the Decoding algorithms, the factor by which the iterative algorithm is faster than the recursive one increases along with the trace length, from 1.8 for 125 observations to 2.8 for 1500 observations. This behavior was to be expected, since the Smoothing algorithm perform as many Forward computation steps as the corresponding Decoding algorithms, and additionally perform at most as many additional (and equally computationally expensive) Backward as Forward steps. Thus, the number of Forward computation steps is the dominant factor on the time complexity, which should thus be equal to that of the Decoding algorithms: $O(n)$ for the iterative approach and $O(n \log(n))$ for the recursive one in the trace length n. Therefore, judging by the computation time alone both approaches should be practically feasible even for vastly longer traces (e.g. 100.000 observations and more).

However, the memory consumption of the algorithms limits the general feasibility. Figure 6.2 shows the peak memory consumption of the two algorithms under different trace lengths. They are very similar to the same measurements for the Decoding algorithm (cf. Figure 5.3 on page 70). Indeed, the memory consumption for Smoothing is smaller than that for Decoding of the same trace length by a constant factor, because Smoothing does not need to store the age vector of a parent Proxel in each Proxel.

Since the memory consumption and computation time of the Smoothing and

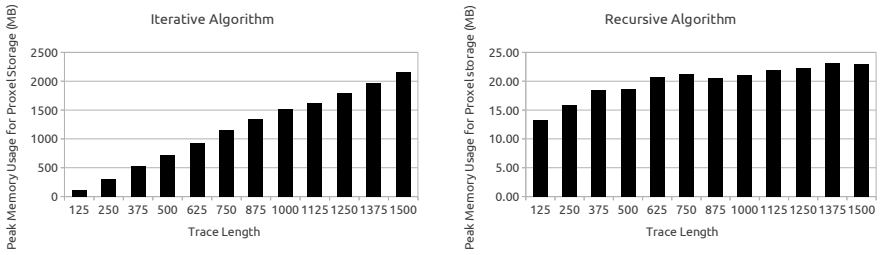

Figure 6.2: Plot of the memory consumption for storing the required Proxels for the Car Rental Agency model under different trace lengths. The diagram on the left-hand side shows the peak memory consumption for a single trace under the iterative approach. The right-hand side shows the average of the peak memory consumptions of 10 traces under the recursive Smoothing algorithm. Note the vastly different scales between the two graphs.

Decoding algorithms are similar, their limits of practical feasibility are close as well: The iterative algorithms have a linear increase in memory consumption with increasing trace length and reached the limit of physical memory present in 2011 commodity hardware at traces with about 1500 observations. For the recursive algorithms on the other hand the memory consumption increases only with the logarithm of the trace length, so that even far longer traces can be processed without a prohibitive increase in memory consumption. The drawback of the recursive algorithms is their slightly increased computational complexity compared to the iterative ones. But since their computational complexity is only $(O(n \log(n)))$ in the number of observations (compared to $O(n)$ of the iterative approaches), this barely negatively impacts the practical feasibility.

6.7 Conclusion

In this chapter, the CHnMM equivalent of the HMM Backward computation was developed and was used to develop an iterative algorithm to the CHnMM Smoothing task. Based on experience from the development of the recursive Decoding algorithm a recursive variant of the Smoothing algorithm was developed as well. It has a slightly higher time complexity than the iterative approach, but a dramatically reduced memory consumption. However, in contrast to the Decoding task, the recursive Smoothing algorithm is only usable in application areas where the computed Smoothing probabilities may be processed time step by time step and no random access over all time steps is required.

Since the structure of the Smoothing algorithm is similar to that of the Decoding algorithm, several of the alternative approaches to reduce the memory consumption of the Decoding algorithm would also be applicable to the Smoothing task as well. In particular, storing the Proxel set of time steps currently not processed to secondary storage, and the checkpointing discarding scheme (cf. Section 5.4) should work for the Smoothing task as well. In contrast, storing symbol sequences in Proxels to be able to omit the backtracking step is not applicable, since the result of the Smoothing task is not a simple sequence of discrete states, but the complete set of Proxels of all time steps. For the

same reason, discarding individual Proxels that have become irrelevant is not an option.

Still, the recursive algorithm has been shown to exactly solve the CHnMM Smoothing task, and to be practically feasible even for big models (e.g. 5000 discrete states in the Car Rental Agency model) and traces longer than 1000 observations. Thus the goals of this work with respect to the Smoothing task have been solved.

In the next chapter, the results of the Smoothing algorithm are used to train conversive hidden non-Markovian models, i.e. to adapt existing models to better explain given observation sequences.

Chapter 7

The Training Task

The algorithms for all CHnMM algorithms so far (Evaluation, Decoding and Smoothing) assumed that a complete model of the system whose behavior is to be reconstructed exists. In practical applications, this may not always be the case: The behavior of the real system may have changed since the model was built (e.g. machines age and thus fail more often or work slower) and thus the model does not match the real system anymore. Or the real system may be a *black box* whose behavior was never observable and thus could not be analyzed to build a model (e.g. machine that performs a sequence of processing steps, which must not be opened and so duration of individual steps cannot be measured). In these cases no correct model exists of the real system and the CHnMM behavior reconstruction algorithms would not be usable.

The solution is to develop a Training algorithm that can refine an existing approximate model to better explain given observation sequences. Formally, the Training task given an initial Model $\lambda = (A, \Pi)$ and an observation sequence O is to find an alternate model $\lambda' = (A', \Pi')$ with $P(O|\lambda') \geq P(O|\lambda)$.

For HMMs, the most widely used Training algorithm is the Baum-Welch algorithm [23, 62, 66]: (cf. Section 2.1), which is an instance of the Expectation-Maximization (EM) approach[19]. It is thus a local optimization algorithm that finds a better matching model, but not necessarily the best model to fit the observation.

Since the goal of this work is to provide algorithms for CHnMMs to solve the same tasks as are solved for HMMs, we will develop a CHnMM Training algorithm based on the EM-based HMM Baum-Welch algorithm. Additionally, in the second part of this chapter, we will develop an algorithm to directly optimize parts of the model specification based on the Maximum Likelihood principle [24].

7.1 Developing an EM Algorithm for CHnMM Training

The EM-based CHnMM Training algorithm shall train the same model parameters as the Baum-Welch algorithm for HMMs. Since CHnMMs and HMMs are specified differently, it is necessary to determine the correspondence between the

HMM parameters trained by the Baum-Welch algorithm, and the corresponding CHnMM model parameters:

1. The Baum-Welch algorithm for HMMs trains the initial state probabilities. The description of a CHnMMs also contains an initial probability vector, and thus this vector also needs to be trained for CHnMMs.

2. The Baum-Welch algorithm for HMMs trains the symbol emission probabilities of all states. In CHnMMs, the symbols are emitted by and specified for the activities and not states and thus a Training algorithm for CHnMMs needs to train the activity symbol emission probabilities.

3. Finally, the Baum-Welch algorithm for HMMs trains the state transition probabilities, which have a direct relationship to the mean time until a given state change occurs. In CHnMMs, the state change behavior is not specified through constant probabilities, but through parametric or arbitrary probability distributions describing the durations of activities. Thus, to solve the equivalent task of training HMM state transition probabilities, we will train the mean durations of all activities of an CHnMM.

4. In addition to CHnMM parameters that are equivalent to trained HMM parameters we will train the sample variance of the activity durations. For the parametric probability distributions of CHnMMs the trained mean duration would be insufficient to determine updated distribution parameters. But for a given parametric probability distribution the mean and standard deviation are often sufficient to uniquely determine the corresponding distribution parameters [30, 31].

7.1.1 Quantities to be Determined

Thus, to train a CHnMM the following quantities need to be determined, based on a given trace:

1. the initial discrete state probabilities $\overline{\pi_i}$[1] for all discrete states S_i
2. the symbol emission probabilities $\overline{TR_k.b(v)}$ of all activities and all symbols
3. the mean activity durations $\overline{E(TR_k.dist)}$ of all activities
4. the sample standard deviation $\overline{stdev(TR_k.dist)}$ of all activities.

The first quantity is a Smoothing probability for $t = 0$ and can thus be computed with the CHnMM Smoothing algorithm (cf. Chapter 6). To determine the other three, the HMM Baum-Welch algorithm will be modified.

The Baum-Welch algorithm is based on the following general principle: First, given the initial model, it executes the HMM Forward-Backward algorithm to determine the Forward, Backward and Smoothing probabilities. These probability are then used for *path counting*, i.e. to compute the expected number of times that the model took a given path of internal discrete states while emitting

[1] Note that the overbar in the expressions $\overline{\pi_i}$, $\overline{TR_k.b(v)}$ and $\overline{E(TR_k.dist)}$ does *not* mean that those are virtual probabilities. It is simply used to differentiate those updated values based on a given trace of observations from the original π_i, $TR_k.b(v)$ and $E(TR_k.dist)$ from the model specification.

the observed trace. Ratios of these path probabilities are then computed to determine the updated model parameters.

For example, the HMM Training algorithm uses the Smoothing probabilities to compute the expected number of times that the model was in a particular discrete state at all, and to compute the expected number of times that it was in that particular state *while* a particular symbol was emitted. The ratio of the latter divided by the former is then the updated probability that the particular symbol was emitted given that the model was in the particular state, i.e. the updated symbol emission probability.

For CHnMMs, we decided to follow the same approach. Some modifications to the algorithm are necessary though, because in CHnMMs, a single activity can be responsible for multiple different discrete state changes from and to different discrete states. Thus, path counting for CHnMMs needs to be performed not for each discrete state, but for each activity. A positive side effect is that since multiple state changes can be caused by a single activity, there are generally fewer activites to be trained in a CHnMM than state probabilities would need to be trained in a corresponding HMM and thus less training data (shorter traces of observations) should be required to train a CHnMM.

Expected Number of Activity Completions All quantities to be computed will be shown to be dependent on the expected number of times N_k that activity TR_k was completed, given the current trace of observations.

$$N_k = E(TR_k \text{ completed } |O)$$

This can be broken down to the sum over all time steps that the activity TR_k is completed at the end of that time step

$$N_k = \sum_{t=1}^{T} P(TR_k \text{ at } o_t.e|O)$$

and further to the sum over all time steps and all possible ages of the activity TR_k that it is completed at the end of that time step with that age:

$$N_k = \sum_{t=1}^{T} \sum_{\tau_k} P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$$

At first glance, this sum over all possible ages of an activity appears to require the evaluation of an infinite number of different age values. But since the Proxels of the Smoothing task for each time step record all possibly occuring age values, this sum can be computed based on these Proxel sets.

The actual formula for $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$ will be derived in the next section, after it is shown that all model parameters to be trained can be computed based on mathematical expressions very similar to that for N_k.

Symbol Emission Probabilities The emission probability of a particular symbol caused by the completion of a particular activity given a trace is the fraction of the number of completions of that activity that actually emitted the symbol, given the trace:

$$\overline{TR_k.b(v)} = \frac{\mathrm{E}(TR_k \text{ completed and emitting symbol } v|O)}{\mathrm{E}(TR_k \text{ completed } |O)} \tag{7.1}$$

Here, the denominator is N_k, and the numerator can be broken down similarly to N_k to yield

$$\overline{TR_k.b(v)} = \frac{1}{N_k} \sum_{t \in \{1 \ldots T | o_t.v = v\}} \sum_{\tau_k} P(TR_k \text{ at } o_t.e \text{ with } \tau_k | O).$$

Thus, in order to determine updated symbol emission probabilities, one only needs to be able to compute the probability that a particular activity TR_k is completed at a particular time $o_t.e$ after a particular duration τ_k. This quantity will be determined in the next section.

Expected Activity Duration The mean duration of an activity can be estimated by dividing its expected total duration (the sum of time intervals during which the activity occurred and which ended with the completion of the activity) by the number of times that the activity was completed:

$$\overline{E(TR_k.dist)} = \frac{\text{E(Total duration } TR_k \text{ was active and later completed } |O)}{\text{E}(TR_k \text{ completed } |O)}$$

Here again, the denominator is N_k, and the numerator can be broken down to a similar sum as was N_k. But in this case, the sum in not just computed over the probabilities of the activity to be completed in each time step with each possible age, but the weighted sum of those ages τ_k, weighted by the corresponding probability:

$$\overline{E(TR_k.dist)} = \frac{1}{N_k} \sum_{t=1}^{T} \sum_{\tau_k} P(TR_k \text{ at } o_t.e \text{ with } \tau_k | O)\ \tau_k \tag{7.2}$$

The only quantity that cannot yet be computed in this formula is $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$, which is also the only yet unknown quantity in the computation of the symbol emission probabilities.

Standard Deviation of Activity Duration Equation 7.2 follows the ordinary formula for the computation of a sample mean for weighted samples: The τ_k are the samples, and the $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$ are the corresponding weights. The corresponding weighted sample standard deviation can be computed based on the sum of weighted squared samples

$$samples_{ws} = \sum_{t=1}^{T} \sum_{\tau_k} P(TR_k \text{ at } o_t.e \text{ with } \tau_k | O)\ \tau_k^2 \tag{7.3}$$

as [1]

$$\overline{stdev(TR_k.dist)} = \sqrt{\frac{samples_{ws} - N_k\ \overline{E(TR_k.dist)}^2}{N_k - 1}}. \tag{7.4}$$

Thus, the only quantity required for a CHnMM Training algorithm that is yet unknown is $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$. Its derivation is the subject of the next section.

7.1.2 Conditional Probability of Activity Completion

So in order to train CHnMMs, the probability $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$ needs to be computed for all observation times $o_t.e$, all activities TR_k and all age values τ_k that are possible for that activity at that time.

To compute this conditional probability, we adopt the standard HMM approach of first computing the (for CHnMMs virtual) corresponding joint probabilities

$$\overline{P}(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)$$

for all activities and all possible values for τ_k, and then normalizing them using the law of total probability to obtain the conditional probability. Here again, the computation of a conditional probability from the fraction of two virtual joint probabilities is possible, because two joint probabilities for the same time step are in the same ratio as the corresponding virtual probabilities (cf. Equation 4.3 on page 44) and thus

$$\begin{aligned} P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O) &= \frac{P(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)}{\sum_i \sum_{\tau_i} P(TR_i \text{ at } o_t.e \text{ with } \tau_i \cap O)} \\ &= \frac{\overline{P}(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)}{\sum_i \sum_{\tau_i} \overline{P}(TR_i \text{ at } o_t.e \text{ with } \tau_i \cap O)}. \end{aligned} \quad (7.5)$$

So, only the virtual probabilities that an activity TR_k is completed at time $o_t.e$ after going on for τ_k time, $\overline{P}(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)$, need to be computed. For this case (activity completion at a given time with a given age) to occur, the following must all hold:

1. After the $t-1$th state change $o_{t-1}.e$, the model may be any state (S_i, τ') but must have emitted the trace so far, i.e. it must have emitted the partial trace $o_1 \dots o_{t-1}$. Additionally, since the age τ'_k of activity TR_k must reach τ_k at time $o_t.e$, at time $o_{t-1}.e$ it must have been

$$\tau'_k = \tau_k - (o_t.e - o_{t-1}.e). \quad (7.6)$$

2. At exactly the time of the tth symbol emission $o_t.e$, the model must change its discrete state to an arbitrary new state (S_j, τ'') through the completion of activity TR_k and must emit the symbol $o_t.v$.

3. After that disrete state change to state (S_j, τ''), the model must still be able to emit the remaining trace $o_{t+1} \dots o_T$.

For a single pair of states, (S_i, τ') and (S_j, τ'') this virtual path probability can be computed from the Smoothing results: the first probability is the Forward probability of the Proxel for (S_i, τ'), the second is the state change probability as used in the forward and backward computations (the product of $P_{sojourn}$, P_{change} and the symbol emission probability $TR_k.b(o_t.v)$), and the third is the Backward probability of the reached state (S_j, τ'').

In contrast, the whole probability $\overline{P}(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)$ allows for arbitrary discrete states before and after the activity completion and almost arbitrary age vectors τ' and τ'' (the kth element of τ' must conform to Equation 7.6 and τ'' must be the result of TR_k being completed at time $o_t.e$). Thus, its

value is the sum over all possible path probabilities for all (S_i, τ') and (S_j, τ'') for which τ' conforms with Eqaution 7.6, for which Proxels exist from the Smoothing algorithm and which are connected by the activity completion of TR_k. This summation can easily be done in the following way:

1. Select all Smoothing Proxels from R_{t-1} whose age vectors conform with Equation 7.6.

2. For each, determine the discrete state and age vector for the corresponding successor Proxel caused by the completion of TR_k at time $o_t.e$. This can be done in the same way as detailed in the CHnMM Forward algorithm (cf. Algorithm 1), where the successor Proxels are generated. With this discrete state and age vector, find the corresponding successor Proxel in R_t.

3. For each pair of Proxel and successor Proxel, compute the virtual path probability as the product the Proxel's Forward probability, the state change probability between the two states and the successor Proxel's Backward probability.

4. Sum up all of those path probabilities.

This way, the joint activity completion probability $\overline{P}(TR_k \text{ at } o_t.e \text{ with } \tau_k \cap O)$ is determined. With all these virtual joint path probabilities known the corresponding conditional probabilities $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$ can be computed (cf. Equation 7.5). And with those in turn, all updated model parameters for the Training task can be computed. The next section summarizes all steps required for the CHnMM Training.

7.1.3 Summary of CHnMM Training Steps

With the formulas derived in the previous section, the whole approach for CHnMM Training is to

1. Execute the Smoothing algorithm (cf. Chapter 6) to provide Forward, Backward and Smoothing probabilities for all states in all time steps.

2. Use the Smoothing probabilities for $t = 0$ as the new initial state probabilities, i.e.

$$\overline{\pi_i} = \rho.\gamma \text{ s.t. } (\rho \in R_0 \wedge \rho.\vec{\tau} = \vec{0} \wedge \rho.q = S_i).$$

3. For each time step t use the probability computation from the previous section to compute all individual conditional probabilities that the model was in a given state since the $t-1$th state change, changed its state through the completion of some activty TR_k at the time of the tth state change after going on for some τ_k time, and altogether emits the whole trace O.

4. For each time step t sum up these probabilities $P(TR_k \text{ at } o_t.e \text{ with } \tau_k|O)$ over all occuring values for τ_k (those for which Proxels exist) to yield $P(TR_k \text{ at } o_t.e)$.

5. Interpret these conditional probabilities as the expected number of times that activity TR_k is completed during the tth symbol emission. Then, sum up these expected values of an activity over
   (a) all time steps to yield E(# state changes by $TR_k|O) = N_k$
   (b) all time steps that were reached under emission of symbol v to yield E(# state changes by TR_k with symbol $v|O$)
6. Compute the new symbol emission probability $\overline{TR_k.b(v)}$ of the activity TR_k as the quotient of those two quantities (cf. Equation 7.1). This completes the training of CHnMM symbol emission probabilities.
7. Use the computed $P(TR_k$ at $o_t.e$ with $\tau_k|O)$ with Equation 7.2 to determine the mean duration of each activity $\overline{E(TR_k.dist)}$.
8. Similarly, use the computed $P(TR_k$ at $o_t.e$ with $\tau_k|O)$ and $\overline{E(TR_k.dist)}$ in Equations 7.3 and 7.4 to determine the standard deviation of each activity duration $\overline{stdev(TR_k.dist)}$

With this algorithm, the individual model parameters can be trained, as long as random access to all Proxels of all time steps is possible. However, requiring random access to all Proxels of all time steps is often practically unfeasible due to the high memory consumption. Therefore, in the next section these computations are reordered to yield an efficient algorithm for the Training of all relevant model parameters without requiring random access to the Proxels of all time steps.

7.1.4 Efficient Computation of Training Probabilities

The CHnMM Training algorithm as given in Section 7.1.3 works, but has some disadvantages: First, as shown in the Smoothing chapter the set of Proxels of all time steps that this algorithm needs access to may require too much memory to fit into the RAM of commodity hardware, rendering the approach unfeasible for longer traces. This can be mitigated by noting that in order to compute the updated model parameters, one in one case needs access only to the Proxels of the first time step (for $\overline{\Pi}_i$) and on all other cases needs to compute a sum on an expression over all time steps (for $\overline{TR_k.b(v)}, \overline{E(TR_k.dist}$ and $\overline{stdev(TR_k.dist)}$). Thus, the computations can be serialized to access only the Smoothing Proxel sets time step by time step. This allows for the Training algorithm to be implemented as a evaluation function to the Smoothing algorithms, where only $O(log(T))$ time step Proxel sets need to be stored in parallel. This is the approach we follow in this section.

Second, following the computations detailed above to the letter would mean to repeatedly filter the set of Proxels for each time step to find those in which a given activity is going on for a given duration. Instead, we process each Proxel only once, note the age of each activity at the time of the next symbol emission, and directly add the corresponding joint probability to the counters for the total activity completion probability as well as for the weighted sum of activity durations and weighted squared sum of activity durations. The normalization needed to convert the computed joint probabilities is deferred until the end of the processing of all Proxels for that time step, since at that time the normalization

constant is naturally known as the sum of all individual path probabilities for that time step (cf. Equation 7.5).

The complete CHnMM Training algorithm that accounts for all of these changes is given in Algorithm 12 as a Smoothing evaluation function. The Smoothing algorithm calls this function individually for each time step, but always also provides the Proxels for the next time step, since their Backward probabilities are required for this computation. Consequently, the algorithm cannot be executed for the Proxel set of the final time step (line 1).

While processing the Proxels for the first time step the updated initial state probabilities are extracted from the corresponding Smoothing probabilities (line 2). Then, for all time steps but the last one, for each Proxel all successor Proxels in the next time step are determined (lines 3–15) in the same way as this is done in the Forward algorithm (cf. Algorithm 1 on Page 46). But in the case of the Training task, those successor Proxels already exist from the Backward part of the Smoothing algorithm. So, each existing successor is located (line 16). And its Backward probabilities is used along with the Proxel's Forward probability and the state change probability between their states to determine the virtual joint path probability (line 17). This probability is then used in turn in the running sum of all path probabilities (line 18), the probability that the corresponding activity caused the state change (line 19) as well as the weighted sum of activity durations and squared activity durations (lines 20 and 21). Once this process has been completed for all Proxels of a given time step, *pAnyActivityCompleted* holds the sum of all path probabilities and thus the normalizing factor for this time step, and so the computed running sums for this time step are normalized in order to convert the computed joint probabilities to the corresponding conditional probabilities, and those are added to the corresponding running sums over all time steps (lines 18–19).

To work, the algorithm assumes that all variables whose names start with *global* are initialized to zero before the Smoothing algorithm is executed, and that those variables are kept in global (i.e. not function-level) memory locations and are thus shared between all calls to the evaluation function.

After the Smoothing algorithm with the Training evaluation function has finished, the quantities to be trained can be computed from those *global* variables as follows:

- $\overline{\pi}_i = globalInitialProb_i$
- $\overline{TR_k.b(v)} = \frac{globalActivityCompleted_{k,v}}{globalN_k}$
- $\overline{E(TR_k.dist)} = \frac{globalActivityDurationSum_k}{globalN_k}$
- $\overline{stdev(TR_k.dist)} = \sqrt{\frac{globalActivityDurationSquared_k - globalN_k \, \overline{E(TR_k.dist)}}{globalN_k - 1}}$

The next section evaluates how these quantities can be used to update the model specification and thus solve the Training task.

7.1.5 Usage of Training Results

To train a model to adapt its parameters, one would repeatedly execute this CHnMM Training algorithm with a single trace of observations, and after each

Algorithm 12: TrainingEvalFunc

Input: $A, TR, R_t, R_{t+1}, O, t, N, T, K$
Output: Updated values for all variables prefixed with *global*

if $t = T$ **then return**;
if $t = 0$ **then foreach** $\rho \in R_0$ **do** $globalInitialProb_{\rho.q} += \rho.\gamma$;
$symbol = o_i.v$;
$\Delta t = o_{i+1}.e - o_i.e$;
foreach $\rho \in R_i$ **do**
  $i = \rho.q$;
  $$P_{sojourn} = \prod_{j \in \{1,\ldots,N|\ a_{ij} \neq \varnothing\}} \frac{1 - cdf(a_{ij}.dist)(\rho.\vec{\tau}_{a_{ij}.id} + \Delta t)}{1 - cdf(a_{ij}.dist)(\rho.\vec{\tau}_{a_{ij}.id})};$$
  foreach $j \in \{1, \ldots, N | a_{ij} \neq \varnothing\}$ **do**
    $\mu = hrf(a_{ij}.dist)$;
    $k = a_{ij}.id$;
    $transProb = P_{sojourn} * \mu(\rho.\vec{\tau}_k + \Delta t) * TR_k.b(symbol)$;
    if $transProb = 0$ **then** continue;
    $Row_i = \{a_{i1}, \ldots, a_{iN}\}$;
    $Row_j = \{a_{j1}, \ldots, a_{jN}\}$;
    $\vec{\tau}' : \tau'_k =$
    $$\begin{cases} \rho.\tau_k + \Delta t & \text{if } TR_k \in Row_i \wedge TR_k \neq a_{ij} \wedge \neg isExp(TR_k.dist) \wedge \\ & (TR_k \in Row_j \vee TR_k.aging) \\ \rho.\tau_k & \text{if } TR_k \notin Row_i \wedge TR_k.aging \wedge \neg isExp(TR_k.dist) \\ 0 & \text{otherwise} \end{cases}$$
    $successor =$ only element $\rho' \in R_{t+1}$ with $\rho'.q = j \wedge \rho'.\vec{\tau} = \vec{\tau}'$;
    $pathProb = \rho.\alpha * transProb * successor.\beta$;
    $pAnyActivityCompleted += pathProb$;
    $pActivityCompleted_k += pathProb$;
    $pActivityDurationSum_k += pathProb * (\rho.\tau_k + \Delta t)$;
    $pActivityDurationSquared_k += pathProb * (\rho.\tau_k + \Delta t)^2$;
for $k = 1$ **to** K **do**
  $globalActivityCompleted_{k,symbol} += \frac{pActivityCompleted_k}{pAnyActivityCompleted}$;
  $globalActivityDurationSum_k += \frac{pActivityDurationSum_k}{pAnyActivityCompleted}$;
  $globalActivityDurationSquared_k += \frac{pActivityDurationSquared_k}{pAnyActivityCompleted}$;
  $globalN_k += \frac{pActivityCompleted_k}{pAnyActivityCompleted}$;

completion of the algorithm would update the model with the parameters obtained through the Training.

However, as noted in the introducton to this chapter there is not a direct correspondence between the quantities computed be the CHnMM Training algorithm and the model specification: only the trained symbol emission probabilities and initial probability vector can be used directly to update the model specification based on a trace of observations. But the mean and standard deviation of activity durations have no direct counterpart in the specification, since CHnMM activities are not specified by their mean duration, but as parametric or arbitrary continuous probability distributions. It is thus necessary to assess under what conditions and in which way those activity duration statistics can be used to update a model.

The mean activity duration was computed by measuring the total time that an activity was going on until it was completed, and dividing it by the number of times that the activity was completed during that time. But this computed value is an estimate for the actual mean duration of the activity only if each beginning of an activity is eventually followed by a completion. This estimate is thus biased when activities can be cancelled and the corresponding durations are thus not recorded. Therefore, the computed mean duration until completion is an estimate for the actual mean of the probability distribution of the activity only if the activity can never be cancelled, either because the activity is of type RACE_AGE, or due to a model structure where the only way that the model can change its discrete state from a state in which an activity is active to a state in which it is not active is through the completion of that activity itself. In all other cases the computed mean duration is of little use for model training. And since the standard deviation of the activity durations are computed for the same samples, the same limitation holds for the applicability of the computed standard deviation as well.

And even for those activities were the computed mean durations and sample standard deviations can be used as an estimate for the mean duration and standard deviations of the activities, their usage depends on the type of probability distribution that the activities follow. If the activity duration is known to be exponentially distributed then the only parameter of that distribution is its rate, the reciprocal of the computed mean value, and thus the computed mean duration fully specifies the updated activity duration distribution function. Many other well-known probability distributions such as the Erlang, Gamma, Lognormal, Normal, Uniform and Weibull distributions are parameterized by exactly two parameters which are directly related to the mean and standard deviation of the distribution [1]. Thus the updated mean and sample standard deviations determined by the Training algorithm can be used to determine updated distribution parameters for those distributions.

But, if activity durations do not follow a well-known probability distribution but are arbitrarily distributed or if the probability distribution function is unknown, then there is no known way to use the trained mean activity duration to update the model. Additionally, since this approach is EM-based it only finds a local optimum of the model parameters and not the global optimum. Thus, the Training approach described here is only a partial solution to the CHnMM Training problem, even though it covers all model parameter equivalents to those trained for HMM with the Baum-Welch algorithm.

While there is currently no solution for the first problem that the Training

algorithm cannot train all aspects of a model, the next section details an alternative approach to CHnMM Training that in some instances solves the second problem of finding the globally most likely parameter set.

7.2 Optimal Model Parameters through Maximum Likelihood Estimation

The EM-based CHnMM Training algorithm requires a fully-specified initial model and modifies it to better fit the observations. This approach may not be desirable if a fully-specified model cannot be supplied since some model parameters are unknown, and randomly guessing values for the unspecified parameters could cause the Training to reach a suboptimal local minimum.

In this section we therefore develop an alternate approach that leaves unknown parameter values unspecified and find their truely globally optimal value. This means that the approach finds those values for the unspecified parameters that result in the highest Evaluation probability, i.e. the model with the highest probability to have caused the trace used for training.

The approach is based on the maximum likelihood estimation (MLE, cf. [24]) principle: It attempts to describe the relevant optimization criterion (here: the likelihood of the model to have created the trace) as a function of the unknown model parameters. The position of the global maximum of that function then corresponds to the most likely unknown parameter values.

The key difference between the classic MLE approach and the situation for CHnMMs is that the classic MLE assumes that the individual observations are statistically independent so that the likelihood function can be created as the product of the individual observation probabilities (or the sum of the log-likelihoods). In CHnMMs, however, the observations are not independent: Here, the completion of an activity determines the next discrete state, which in turn determine the next activity to be completed. Thus, there is a correlation between subsequent activities, and since the completion of activities causes the observable symbol emissions, subsequent observations are correlated as well.

Consequently, the classic MLE approach of multiplying individual observation probabilities is not applicable to CHnMMs. So there are two challenges for a MLE-based CHnMM Training algorithm: First, the likelihood function for correlated observations has to be found. And second, the position of the maximum of that likelihood function has to be determined in order to find the most likely values for the unknown parameters. Both challenges are the subject of the next two sections.

7.2.1 A Likelihood Function for Incomplete CHnMMs

In order to apply the MLE approach to CHnMMs, one needs to determine a mathematical expression that describes the Evaluation probability (cf. Chapter 4) as a function of model parameters with unknown values. Using the Forward algorithm is an obvious starting point, since the algorithm already computes the Evaluation probability for a completely specified model.

The naïve extension of the Forward algorithm to determine a likelihood functions would be to leave all unknown parameters of the model as symbolic

variables, and to perform all computation of the Forward algorithm symbolically. The resulting virtual Evaluation probability would then be a symbolic mathematical expression in those symbolic variables and would therefore be a likelihood function. And the position of the global maximum of that expression would determine the most likely values for the unknown parameters.

Yet, while this approach is theoretically sound, it is not practically feasible: When Proxels are merged, their probabilities are added. For symbolic probabilities this requires a concatenation of the probability expressions of the source Proxels, at least in the general case. Thus, whenever two Proxels are merged the length of the resulting probability expression about doubles. And since Proxel merging occurs in each time step, this causes an exponential increase in the length of the Proxel probability expressions with increasing trace length, rendering the approach practically unfeasible for all but the shortest traces. On the other hand simply not merging Proxels would result in an exponential growth of the number of Proxels per time step [29], rendering the approach unfeasible as well.

Our solution to this feasibility problem is to restrict the mathematical expression representing the Proxel probabilities to a class that allows automatic simplification of the concatenated expressions [12]. Such a class of mathematical expressions would have to fulfill the following requirements:

1. When two expressions of that class are merged, then the resulting expression must not be substantially longer than either one of the two expressions. This is required to limit the expression length caused by Proxel merging and thus to keep Proxel merging feasible.

2. The length of a Proxel probability expression must not increase substantially when multiplied by the state change probability (the product of $P_{sojourn}$, P_{change} and the symbol emission probability) used in the inductive Forward computation in order to keep it feasible.

3. The class needs to be closed under addition (for Proxel merging), multiplication (for the inductive Forward computation), computation of $P_{sojourn}$ (cf. Equation 4.1 on page 41) and computation of P_{change} (cf. Equation 4.2 on Page 44) for all probability distributions. This means that when a Proxel probability expression or the expression for a unknown model parameter is a member of the selected class, then all of these operations have to yield expressions in that class as well. Otherwise, Proxel probability expressions could "escape" the class, and would thereby violate the first two requirements.

Unfortunately, no class of mathematical expression is known that fulfills *all* three requirements, and it is unlikely that such a class exists: To be closed under the $P_{sojourn}$ and P_{change} of all conceivable continuous probability distributions would require a rather extensive, expressive class of mathematical expression. The ability to simplify expressions to limit their length under addition and multiplication on the other hand requires a rather simple, limited class of expressions.

To resolve this contradiction we decided to choose univariate polynomials as our class of mathematical expressions, and to limit the applicability of the approach in order to make the class of those polynomials closed under all required

operations. Since polynomials are not closed under the computation of $P_{sojourn}$ and P_{change} (if a distribution parameter is unknown and thus is replaced by a symbolic variable or a polynomial in that variable, computed values for $P_{sojourn}$ and P_{change} will generally not by polynomials), the approach will not be applicable to the training of distribution parameters. One notable exception is the Training of multiple exponentially distributed activities that always occur concurrently and whose overall rate (i.e. the number of the times that any of the activities is completed per time unit) is known (cf. Appendix B). Those are closed and thus trainable using symbolic polynomials.

Additionally, univariate symbolic polynomials are closed under multiplication and addition only if the input polynomials are polynomials in the same variable, i.e. when two univariate polynomials in different variables are added or multiplied, the result is not a univariate polynomial, but a bivariate one. Thus, the restriction to univariate polynomials further limits the applicability of this MLE-based approach to models where all unknown parameter values can be expressed as polynomials in a single common variable.

With these limitations univariate polynomials are closed under all required operations, fulfilling the third requirement[2]. When adding the polynomials of two Proxel probabilities during Proxel merging, all that needs to be done is adding the corresponding coefficients of both polynomials. Thus, the degree of a probability polynomial after Proxel merging is not higher than that of any of the input polynomials, fulfilling the first requirement. And when multiplying the probability polynomial of a Proxel with the state change probability polynomial (whose length depends only on the model parameters and is thus independent of length of the given trace) during the actual inductive Forward computation, the degree of the Proxel probability polynomial increases only by the constant size of the state change probability polynomial, fulfilling the second requirement.

Thus, with all three requirements fulfilled the following algorithm should be a practically feasible approach to construct a likelihood function:

1. Express all allowed unknown model parameters (symbol emission probabilities, initial state probabilities, rates of concurrent exponentially distributed activities with known total rate) as polynomials in a single free variable.

2. Execute the CHnMM Forward algorithm (cf. Algorithm 1 on Page 46) with this model; Perform all computations of the algorithm that involve the free variable symbolically. Thus, all Proxel probabilities will no longer be numbers, but polynomials in the free variable.

3. Sum up all Proxel probabilities of the final time step to yield the Evaluation probability. Since the Proxel probabilities are symbolic polynomials, the Evaluation probability will be a symbolic polynomial in the free variable as well.

4. Interpret the Evaluation probability polynomial as a likelihood function in the free variable.

[2]Allowing arbitrary *multi*variate polynomials would also have fulfilled this requirement. But the length of a multivariate polynomial can increase substantially under multiplication, violating the second requirement.

The next section explains how this likelihood function can be used to determine updated model parameters.

7.2.2 Evaluation of the Likelihood Polynomial

The main goal of evaluating the likelihood polynomial is to extract the most likely values of the unknown model parameters, and to that end to determine the position of the valid maximum of that polynomial.

The accurate solution to this problem would be to symbolically compute the first and second derivatives of the polynomial, to find the extrema using numerical root-finding approaches [33] on the first derivative, and to select the maxima from these roots using the second derivative. Additionally, potential global maxima are interval boundaries for the valid intervals of the free variable, those intervals in which all unknown model parameters expressed as polynomials in the free variable have valid values (values in the interval $[0, 1]$ for symbol emission probabilities and initial state probabilities, values greater than zero for rates of exponential distributions). For all of those potential global maxima, i.e. the local maxima and the interval boundaries, the likelihood of the corresponding parameter combination has to be determined by evaluating the likelihood polynomial at their positions. The position (=the value of the free variable) with the highest likelihood is then the global maximum.

An alternate less accurate solution that has practically been shown to usually be sufficient is to sample the likelihood polynomial in small steps and chose the one valid position with the highest likelihood of these samples as a sufficient approximation of the most likely value of the free variable.

In both cases, the found most likely value of the free variable needs to be inserted into all polynomials describing the unknown model parameters in order to determine their most likely values.

Thus, this MLE-based CHnMM Training algorithm finds the *most likely* values of some unknown model parameters with a single iteration. The previously developed EM-based Training approach on the other hand only finds a *more* likely set of model parameters, and several iterations of that algorithm with the same trace are required for the algorithm to at least converge to a locally optimal parameter set. However, the symbolic computations of the MLE-based algorithm introduce some difficulties for practical implementations that are described in the next section.

7.2.3 Implementation Considerations

If symbol emission probabilities or distribution parameters are unknown then the overall state change probability as the product of $P_{sojourn}$, P_{change} and the symbol emission probability is a polynomial in the free variable. Thus, in each time step the Proxel probabilities are multiplied by such a polynomial, increasing the degree of the Proxel probability polynomial by at least one. The likelihood polynomial as the sum of all Proxel polynomials of the final time step may thus be a polynomial of very high order (e.g. of degree ~ 1500 for the traces used in the experiments of previous chapters).

This in turn causes the polynomial to be numerically unstable and thus requires its coefficients to be stored with a high numerical accuracy in order to obtain accurate results. As the next Section will show, the required precision is

higher than the IEEE 754 `double` precision available in many programming languages and implemented in many hardware architectures. An implementation of this algorithm on those platforms thus requires arbitrary precision arithmetic [34] as provided by many software libraries such as GMP [26].

Furthermore, adding and multiplying Proxel probabilities are core operations of this algorithm. The way of performing those operations may thus impact the computation time and memory consumption of the algorithm.

For adding polynomials the naive approach of individually adding corresponding coefficients (those for the same power) of both source polynomials is already asymptotically optimal: Since each coefficient of the two source operands can potentially impact the result, they all have to be read at least once, resulting in lower bound of $O(n)$ for the time complexity of adding two polynomials of degree n. In the naïve approach, each coefficient of one polynomial is read exactly once, added to the corresponding coefficient of the other polynomial, and the result needs to be stored. This thus results in a time complexity of $O(n)$ in the degree of the polynomials.

For the multiplication of two polynomials the naïve approach is to multiply each coefficient of one polynomial to each coefficient of the other polynomial. The product of the ith degree coefficient of the first polynomial and the jth degree coefficient of the second polynomial is then added to the $(i+j)$th coefficient of the result. For two polynomials of degrees n and m this thus requires $O(nm)$ operations. More efficient algorithms with a lower time complexity exist [65], but those are beneficial only for multiplying polynomials of similar degree. For this Training algorithm, however, it is only necessary to multiply high degree polynomials (the Proxel probabilities) with low degree polynomials (the specified symbol emission probabilities or rates of exponential distributions). Here, the advanced algorithms have no practical advantage over the naïve implementation.

The algorithm implemented with these considerations as well as the initial EM-based Training algorithm are tested experimentally in the next Section.

7.3 Experiments

In this chapter, two algorithms to solve the CHnMM Training task have been developed. Those will now be tested in this section. First, an application example for both algorithms is given. Then, it is argued why the computation time and memory consumption of the EM-based approach are essentially identical to the CHnMM Smoothing algorithm, and the computation time and memory consumption of the MLE-based approach are tested. Finally, characteristics specific to each algorithm are tested. The EM-based algorithm will usually applied iteratively to successively improve a model; its results are thought to converge to a local optimum. This convergence behavior is tested experimentally. For the MLE-based approach, finding the global optimum is guaranteed by the algorithm. But this approach requires a high numerical accuracy. How high this accuracy needs to be is also tested in this section.

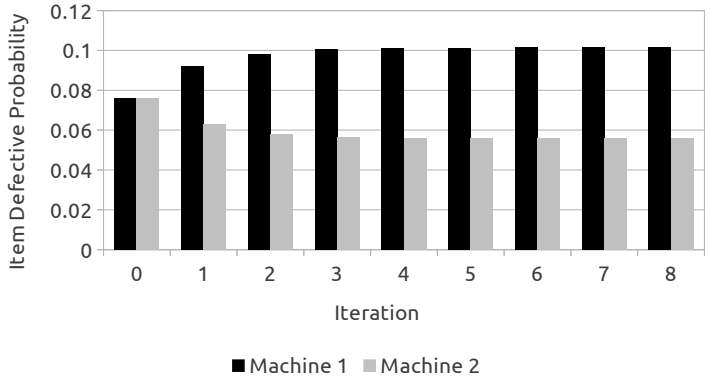

Figure 7.1: Plot of the symbol emission probabilities for the symbol "Defective" in the Tester model for several iterations until convergence. Iteration 0 is the initial parameter set as estimated from the trace alone.

7.3.1 Application Example

As an application example we return to the Tester model and the initial task of determining the probabilities with which each of the two machines produces defective items. So far, this problem has been solved using the Evaluation task for a special case, where only three values for the defect probabilities were possible. Using the Training task it is possible to solve this problem even without such limitations.

For the EM-based Training algorithm, solving this task means that arbitrary values must first be used for the unknown parameter values. In this scenario, the symbol emission probabilities ("ok" or "defective") of the two machines are unknown. In order to provide initial values for these, we determined the overall fraction of defective items produced by both machines together as recorded in the provided trace. This overall defective probability is then used as the initial defective probability for both machines, and the probability to produce working items is adjusted accordingly.

With this preliminary model the EM-based Training algorithm can be used to determine more likely model parameters. Since in this scenario the actual activity durations are known, the trained mean activity durations are ignored and only the trained symbol emission probabilities are used to update the model.

Figure 7.1 shows the Training results for this model. Here, several iterations of the Training task were performed until the model parameters converged to a locally most-likely model. In this most likely model, the relevant defective probabilities converged to about 0.1 for the first machine and to about 0.05 for the second machine. These values can be taken as a good estimate for the defective probabilities of the two machines, and can be used to make management decisions regarding maintenance or replacement priorities.

The same problem can be solved with the MLE-based training algorithm as well. Here, the challenge is to express all four unknown quantities (the symbol emission probabilities for "ok" and "defective" of both machines, respectively) as polynomials in a single variable. This can be done by estimating the total

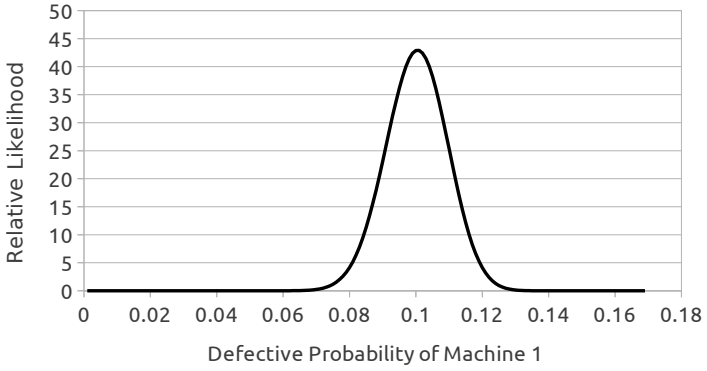

Figure 7.2: Plot of the normalized likelihood polynomial of the Tester model. The independent variable is the "defective" probability of Machine 1.

number of produced items of each machine based on its mean activity duration and the length of the observation time interval. With this estimate, the defect probabilities of both machines can be relating to each other, since the expected sum of produced defective items of both machines (expected number of produced items times the defect probability) has to match the recorded number of defective items in the trace. Rearranging this equation shows that the defective probability of one machine can be expressed as a polynomial in the defective probability of the other machine. And since each machine has to emit either the "defective" or the "ok" symbol, the "ok" probabilities are simply the complementary probabilities of the respective "defective" probability. This way, all four unknown symbol emission probabilities are expressed as polynomials in a single "defective" probabilities. Consequently, all requirements for the execution of the MLE-based Training algorithm are fulfilled.

Executing the MLE-based Training algorithm computes a likelihood polynomial. To be better comprehensible, Figure 7.2 does not show the original likelihood polynomial (whose values would be in the range of 10^{-3000}), but a scaled version where the polynomial has been multiplied by a constant value in order for the area under the graph to yield 1. Such a scaling is permissible since we are only interested in the position of the maximum of this polynomial, and this position is independent of any actual scaling.

The position of the maximum of this polynomial - here about 0.1 - is then the most likely value for the defective probability of Machine 1. And inserting this value in the equations for the "defective" probability of Machine 2 yields the corresponding most likely value for this parameter as well.

Thus, in this scenario both Training approaches compute very similar values for the unknown model parameters.

7.3.2 Computation Time and Memory Consumption

As with all developed algorithms, the practical feasibility of the Training algorithms is a primary concern. To that end, their computation time and memory

consumption are to be evaluated.

For the EM-based Training approach, these properties can directly be derived from the CHnMM Smoothing algorithm, since the EM-based Training algorithm is implemented as an evaluation function for the Smoothing algorithm. And that Evaluation function is almost identical to the CHnMM Backward computation step (cf. algorithms 7 and 12 on pages 81 and 97, respectively). So, for a trace of n observations the CHnMM Smoothing algorithm needs to perform about $nlog(n)$ Forward computation steps (or n if the iterative Smoothing algorithm is chosen) and n Backward computation steps. The EM-based Training algorithm needs to additionally execute the evaluation function n times, which performs almost the same operations on the same data as the Backward computation, but retains only a constant amount of data. Thus, while the EM-based Training algorithm increases the memory consumption by a small constant value over the Smoothing algorithm and increases the computation time by a small constant factor (< 2), its practical feasibility is virtually identical to that of the Smoothing algorithm: Traces with over 1000 observations can feasibly be processesed for models with thousands of discrete states as long as the model has fewer than six concurrent activities.

The feasibility of the MLE-based Training approach is more difficult to assess. On the face of it, it seems to have the same constraints to feasibility as the CHnMM Forward algorithm developed to solve the Evaluation task, since the core of the MLE Training algorithm is to perform the CHnMM Evaluation task once using symbolical computations. However, the details of this approach increase both computation time and memory consumption: The symbolic computations require all probabilities to be stored as symbolic polynomials as opposed to simple floating-point numbers. The degree of these polynomials increases with each time step, increasing the memory consumption of individual Proxel probabilities with each time step. Furthermore, due to the numerical instability of the approach, each polynomial coefficient needs to be stored as a high-precision number, which are bigger than normal floating-point numbers. And finally, the computation time of mathematical operations on those polynomials increases at least linearly with their length and thus is also heavily increased. All of these effects cause a vast increase in computation time and memory consumption for the MLE-based Training approach over the CHnMM Forward algorithm.

To assess the actual computation time and memory consumption, experiments on the Car Rental Agency model were conducted (the Tester model was shown to be too small to accurately measure computation time and memory consumption). In the Car Rental Agency model (cf. Figure 2.2 on page 21) the symbol emission probabilities need not be trained, since only one symbol exists and thus the completion of each activity is certain to emit that symbol. Thus, the only model properties that can be trained with the MLE-based approach are the rates of the exponentially distributed arrival times. To train those parameters, the algorithm needs to know the combined rate of those activities (i.e. the rate with which customers arrive independent of whether they are premium or standard customers). This combined rate can be estimated quite reliably from the trace, since on average half of the observations must be due to arrivals (while the other half is due to the completion of the service, after which a customer leaves).

With this estimate, the likelihood polynomial and the resulting most likely

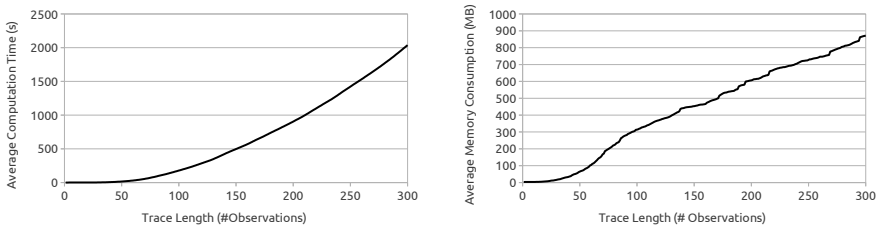

Figure 7.3: Plot of the average memory consumption and average computation time over 20 traces for the Car Rental Agency model under different trace lengths for the MLE-based Training algorithm using symbolic polynomials. The numerical precision of the Proxel polynomial coefficients was fixed at 256 bits for all trace lengths.

parameter value for the arrival rates were computed for several traces of observations. Initial experiments showed that the memory consumption of the algorithm is too high to process the usual trace lengths of about 1500 symbol, and so traces with only 300 symbols were used.

Figure 7.3 shows the results for the cumulative computation time and memory consumption averaged over 20 traces for different trace lengths. The computation time increases quadratically (a quadratic regression of the plotted data yields a coefficient of determination of $R^2 = 0.9993$) with increasing trace length and already reaches about 30 minutes for a trace of 300 observations. This quadratic increase was to be expected, since the length of the Proxel probability polynomials and thus the computation time per time step increases linearly in the trace length (and the cumulative computation time is the sum of the computation times over all those time steps). Thus, doubling the trace length would quadruple the computation time. This behavior severely limits the feasibility of the approach to process longer traces.

The memory consumptions of the approach (cf. right-hand side of Figure 7.3) increases linearly with increasing trace length, reaching a memory consumption of almost 900 MB for traces containing 300 observations. Here, the reader is reminded that the MLE-based Training approach is in essence a single execution of the algorithm for the Evaluation task, which needs to store the Proxels for at most two time steps concurrently. Thus, the linear increase in the memory consumption with respect to the trace length is caused by the linear increase of the memory consumption of the Proxels for a single time step. This, too, is explained by the linear increase of the degree of the Proxel probability polynomials with increasing trace length. Since the algorithm already stores only Proxels for the two necessary time steps, there is no simple way of further decreasing the memory consumption of the approach.

The linear increase in memory consumption along with the quadratic increase in computation time severely limits the practical feasibility of the MLE-based Training approach compared to all other developed CHnMM behavior reconstruction algorithms. The tested Car Rental Agency model along with the used traces are close to the limits of practical feasibility: bigger models (more concurrent activities or bigger discrete state space) would quickly exhaust the available memory in today's commodity hardware, and longer traces would ad-

ditionally also quadratically increase the computation time beyond the limits of most practical applications.

Yet, especially the ability to processing long traces is very desirable, because it counteracts the problem known as overfitting [86]: when only little data (a short trace of observations) is used to train a complex model, the model tends to become just a memorization of the observations instead of representing the general behavior of the real system, of which the trace of observations is just one realization.

7.3.3 Convergence Behavior for the EM-Based Training Algorithm

The EM-based Training algorithm for CHnMMs was developed based on the Baum-Welch algorithm[62], which is used to train HMMs. For the original Baum-Welch algorithm, it has been proven that Training never results in a worse model with respect to the Evaluation probability of a given trace. This property is desirable for the CHnMM Training approach as well.

To test the convergence behavior we repeatedly randomly parameterized our model and then iteratively used the EM-based training algorithm with a single trace with 1500 observations in order to train the model. In particual, we used 100 different parameterizations of the model and for each parameterization performed 60 iterations of the CHnMM Training task. Since those 6000 iterations of the Training task a very time intensive, we performed them on the smaller Tester model[3]. Here, the model parameters to be trained are the parameters of the duration normal distributions of each machine as well as their defect probabilities. The initial defect probabilities were chosen randomly from the interval $[0, 1]$, the production duration means from the interval $[0, 500]$ and the corresponding standard deviations from $[0, 50]$.

The Training of the differently parameterized models converged to two different final models: of the 100 initial models, 64 converged to a model very close to the actual model specification from which the used trace was generated. 12 models converged to a very different model with a far lower Evaluation probability ($\sim 3 * 10^{-3436}$ vs. $\sim 6 * 10^{-3092}$). And for the remaining 24 models, Training was not possible, because the random initial model had zero Evaluation probability of generating the trace. Thus, no path of internal states in that model had non-zero probability to generate the trace and so the path-counting which forms the basis of the CHnMM Training algorithm was not possible. For the same reason, the same issue occurs in the Training of HMMs. Thus, this limitation that the initial model has to have a non-zero Evaluation probability with respect to the given trace in order for the model to be trainable is shared between the EM-based Training algorithms for HMMs and CHnMMs.

The convergence behavior for the 64 models that converged to the actual model specifications is shown in Figure 7.4, with a linearly scaled axis for the probability on the left-hand side and a log-scaled axis on the right-hand side. Both plots confirm that the Evaluation probability is monotonically increasing

[3]This is only to say that we thought it impractical to use the bigger Car Rental Agency model for the experimental validation. To train the model for a practical application one would only use very few or even just a single initial parameterization and thus would need far fewer total iterations. Thus, in practical applications, Training of the Car Rental Agency model would still be feasible.

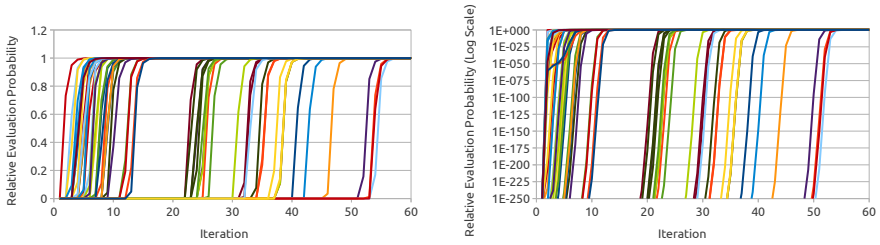

Figure 7.4: Convergence of the Training results for those randomly parameterized Tester models whose Evaluation probability converges to $6.606 * 10^{-3092}$ (in both plots normalized to one). Both plots show the same data, but the one on the right-hand side has a log-scaled axis for the probability.

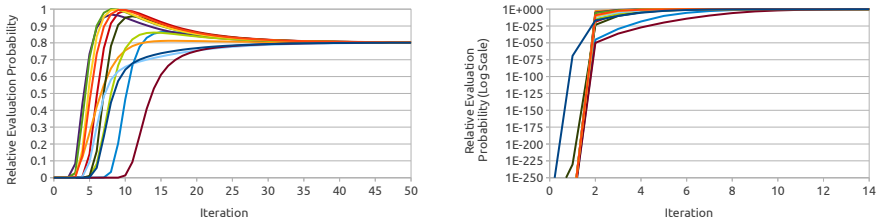

Figure 7.5: Convergence of the Training results for those randomly parameterized Tester models whose Evaluation probability converges to $3.301 * 10^{-3436}$ (in both plots normalized to one). Both plots show data from the same experiment. But the plot on the right-hand side has a log-scaled axis for the probability and only shows the results for the first 14 iterations of each Training task, as later iterations would be indistinguishable on a log scale.

with each iteration and thus the EM-based CHnMM Training algorithm here always finds a better model.

Figure 7.5 shows the corresponding results for those initial models that converged to a common alternative model. Here, the convergence behavior was not monotone. While Training converged to the same model in all cases eventually, for some initial models a more likely model was found in between and in these cases further Training actually reduced the Evaluation probability and thus the model quality. So the developed EM-based CHnMM Training algorithm cannot guarantee to always find a better model.

Further evaluation of the data at hand, however, suggests that the algorithm may still reliably be used to find models that fit the observations well. Both plots in Figure 7.5 show that the reduction of model Evaluation probability only occurs when the model used in Training is already very close to the optimal model. So, Training *never* made a bad model worse and thereby converged to arbitrary model irrespective of the data. It only made the locally optimal model slightly worse. In particular, for all of these initial models that eventually converged to the alternative model, the Evaluation probability of the final model is only about 20% lower than that of the locally optimal model. This is in contrast to the hundreds or even thousands of orders of magnitude [sic] by

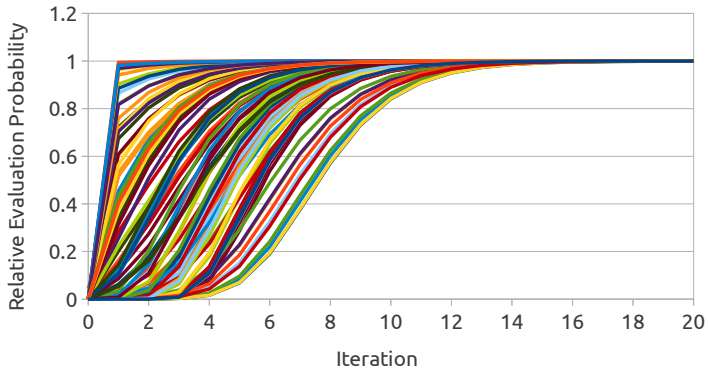

Figure 7.6: Convergence of the Training results for 100 random models where probability distribution parameters were fixed and only symbol emission probabilities were trained.

which the random initial models were less likely to have generated the trace than the model that they eventually converged to during Training. Thus, while the CHnMM Training algorithm cannot guarantee to find the locally optimal model, it has experimentally been shown to always find models that are very close to that optimum.

A likely explaination for that discrepancy between the locally most likely model and the model that training may converge to is that the sample statistics (mean and standard deviation) computed to train the activity durations are good, but not optimal values for the distribution parameters. The reason for this discrepancy is that the set of random samples from which the statistics are computed is finite and thus the shape of the distribution given by those samples does not perfectly match that of the specified continuous probability distribution. Since those shapes do not match, the computed mean and standard deviation of the samples are not the best explanation for the mean and standard deviation of the continuous probability distribution. Using them as such (for lack of better estimates) can lead to the observed behavior subsequent Training iterations of a near-optimal model lead to a slightly worse model.

To test this hypothesis that the discrepancy between optimal distribution parameters and those derived from sample statistics cause the non-monotonous Training behavior, we performed an additional experiment. We recorded the distribution parameters of the suboptimal model ($\sim N(300.4, 36.75)$ and $\sim N(85.64, 40.09)$, respectively) that these Training instances converged to. Additional Training experiments were then conducted where the activity duration distribution parameters were fixed to these values and only the symbol emission probabilities were trained. If the hypothesis holds then Training should now always lead to more accurate models.

Figure 7.6 shows the results and confirms the hypothesis. Even though the probability distribution parameters were fixed to those values to which some traces converged after initially having found a better model, when only the symbol emission probabilities are trained, then the Training convergence behavior

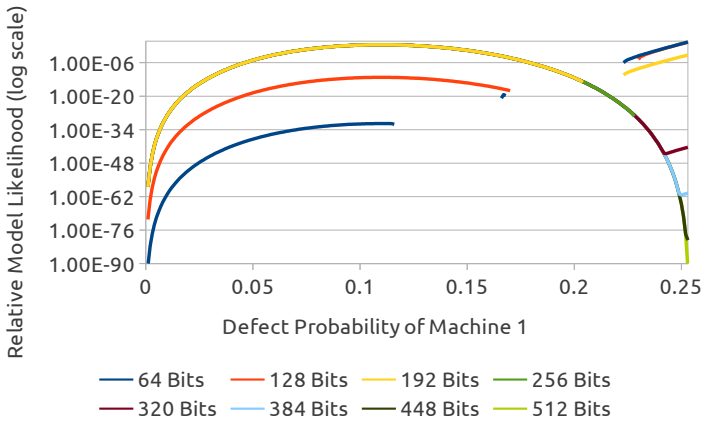

Figure 7.7: Plot of the normalized likelihood polynomial of the Tester model for different numerical precisions. The independent variable is the "defective" probability of Machine 1. The vertical axis is log-scaled to visualize values of vastly different magnitudes. Gaps in the graphs occur when the computed likelihoods were negative.

to that model is always monotonic.

Thus, while the EM-based CHnMM Training algorithm cannot guarantee to find the locally most likely model, it has been shown in all experiments to find a model that is very close to that local optimum and overwhelmingly closer to the optimum that the random initial models that the Training started with.

7.3.4 Numerical Stability of the MLE-based Training Approach

The MLE-based CHnMM Training algorithm generates polynomials of very high degree. It is thus numerically unstable and requires a high numerical precision to compute accurate results.

To analytically assess the numerical precision required to accurate results we performed the Training task from the application example above using various numerical accuracies for the computation of the probability polynomial coefficients and the sample of the resulting model likelihood polynomial.

Figure 7.7 shows the results for this Training of the Tester model with a trace of 1500 observations. The horizontal axis specifies the possible values for the unknown value from the model specification, the defect probability of the first machine. Its range contains all valid probabilities for this parameter, i.e. all those values for which the defect probability of the first machine *and* the dependent defect probability of the second machine are in the range $[0, 1]$. The vertical axis specifies the corresponding normalized model likelihood (scaled by a constant factor so that the area under the graph is one). It is log-scaled to better visualize the likelihoods which cover several orders of magnitude. If this axis were linearly scaled, the graphs for all precisions of at least 192 bits would look like the one in Figure 7.2.

Since the graph for 512 bits has the highest precision tested, it is likely to

be the most accurate. Using this as the baseline of accuracy, it is clear from the graph that the results for 64 and 128 bits differ from the correct results by several orders of magnitude. Furthermore, both graphs have gaps where the likelihood was computed to be negative. Starting with a precision of 192, the graph agrees with the correct result at least for most of the displayed value interval. The higher the numerical accuracy is, the longer is the section of the graph with correct results. And only the graph for 512 bits seems to be correct over the whole interval.

Thus, selecting a suitable numerical precision is crucial for the accurate performance of the MLE-based Training algorithm. However, currently there is no known way to determine the necessary numerical precision of the algorithm beforehand. The only existing viable approach is to perform the computations with different precision levels and to judge from the graphed results whether the precision is high enough to provide reliable results.

In the experiment conducted, 512 bits of precision for the computations of a polynomial of degree 1500 seems to be sufficient. Thus, a preliminary heuristic for selecting the precision would be to use $n/3$ bits of precision for a computation that results in a polynomial of degree n.

7.3.5 Summary

Overall, the MLE-based CHnMM Training algorithm was developed to always find the most likely model parameters, and will do so if a numerical accuracy sufficient for the degree of the generated polynomial is chosen. Its practical feasibility is limited by its memory consumption and computation time to models no bigger than the used Car Rental Agency model and to comparatively short traces of about 500 observations.

The EM-based algorithm on the other hand was developed to only find a better matching model with each iteration, and to converge to a locally most likely model eventually. It has been shown that this behavior cannot be guaranteed and that EM-based Training may converge to a slightly sub-optimal model when symbol emission probabilities and parametric probability distributions are trained. The great advantage of the EM-based algorithm are its speed and – when based on the recursive CHnMM Smoothing algorithm – the low memory consumption, which both make the algorithm feasible for bigger models and longer traces than the MLE-based approach.

7.4 Differences between the EM and MLE CHnMM Training Algorithms

As two different algorithms for very similar tasks were introduced in this chapter it is useful to discuss their differences.

The expectation-maximization algorithm similar to the HMM Baum-Welch algorithm uses a fully specified model and a trace to update the model to better explain the emission of the trace. It thus requires at least estimates for all model parameters. The maximum likelihood algorithm on the other hand can naturally deal with missing values and does not require estimates.

Further, the EM-based algorithm determines only a – usually – more likely model to explain the observations. It requires iterations to let the model con-

verge to a final likely model, and is not guaranteed to ever locate the globally most likely model. The MLE-based algorithm offers a way of directly computing the values for the unknown parameters that are certain to be optimal with respect to explaining the trace.

Additionally, each iteration of the EM-based algorithm requires the execution of the CHnMM Smoothing algorithm. For n time steps it thus requires a total of at least $2n$ Forward and Backward computation steps (or about $n+n\log_2 n$ steps if the recursive Smoothing algorithm is used to reduce memory consumption). The MLE-based algorithm only requires a single Forward computation consisting of only n Forward computation steps.

However, the MLE-based algorithm has some severe drawbacks itself: All probabilities computations have to be performed on polynomials instead of single floating point numbers. And those polynomials grow in size (i.e. in the number of coefficients they consist of increases) with each time step. Thus, the probability computations for each time step are not only more costly for the MLE approach than for the EM approach, their cost even increases with each time step.

Furthermore, with the increasing degree of the polynomials with each time step, their numerical instability increases as well. Thus, the longer the trace used for Training is the higher the precision of the polygon coefficients has to be in order to obtain accurate results. And there is currently no known way of determining the required precision beforehand. So one has to either have domain expert knowledge available or has to experiment with different numerical precisions to find the sufficient level of numerical precision; or has to resort to computationally expensive and complex approaches like floating point filters [25] (in order to dynamically adjust the numerical accuracy during the computation) or interval arithmetic [56] (in order to determine the need for higher numerical precision).

Thus, the EM-based Training algorithm is an easy-to-implement algorithm to improve an existing complete model to *better* explain a trace of observations. The MLE-based approach on the other hand is suitable for finding the best values for one or few interdependent model parameters to explain a trace of observations, but does not modify the already specified model parameters and is complicated to implement and potentially prohibitively computationally expensive.

7.5 Possible Extensions

So far, the EM-based CHnMM Training algorithm only determines the mean and standard deviation of the duration of activities, but offers no help in determining the type of the probability distribution. But the approach to determine those two sample statistics may be extended to also determine higher moments [30] such as the skewness and kurtosis. Those may then be used to identify the most accurately matching parametric probability distribution. Thus, if higher order moments were be determined, the EM-based Training algorithm could be used to train more aspects of the duration of activities.

For the MLE-based CHnMM Training algorithm, several extensions are conceivable. First, while the probability polynomials of the Proxels are of very high order, the shape of the polynomial in the relevant range (cf. Section 7.2.2)

is often relatively regular and may be reasonably approximated by a lower-order polynomial, for example through Lagrange interpolation [21]. If such an approach succeeds it may reduce the degree of the polynomials used in this approach to a point were IEEE 754 floating-point numbers are sufficient for the storage of polynomial coefficients and would thus eliminate the need for arbitrary-precision arithmetic.

Furthermore, the likelihood polynomial of the MLE-based Training approach may be used to better characterize the accuracy of the determined parameter values, i.e. the likelihood that the reconstructed *most likely* parameter value is indeed the *actual* parameter value. The value that the likelihood polynomial evaluates to at a given position is proportional to the Evaluation probability of the model to have caused the trace of observations used for the Training and is thus similar to a continuous probability distribution function. The likelihood polynomial may indeed be converted to a *pdf* by determining the area under its graph (using symbolic computation of its antiderivative) over the valid range, and normalizing the area under the graph to one by dividing the polynomial by the computed area. From this true pdf a cumulative distribution function may then be derived through symbolic integration, and this cdf can be used to directly compute confidence intervals around the position of the most likely parameter values to characterize the accuracy of the parameter reconstruction.

Finally, the MLE-based CHnMM Training approach might be extended to train probability distribution parameters of activity durations. So far, this was not directly possible using symbolic polynomials, as the quantities $P_{sojourn}$ and P_{change} cannot generally be expressed as polynomials in the distribution parameters. But it is possible to perform the computations for those two quantities for several different values in a given range, and to construct an interpolation polynomial to approximate their behavior between those values. The interpolation polynomial could then be used in the normal MLE-based Training approach using symbolic polynomials. One difficulty for this approach is to determine a good number of samples and a good set of sampling points to sufficiently closely approximate the actual behavior. And a severe drawback is that the length of the Proxel probability polynomials increases by sum of the lengths of the $P_{sojourn}$ and P_{change} polynomials with each time step, since the probability of a successor Proxel is computed by multiplying - among other things - the probability of the predecessor Proxel with the corresponding $P_{sojourn}$ and P_{change} polynomials. Thus, the higher the degree of the interpolation polynomials the faster increase the Proxel probability polynomials their degree and thus the higher does the precision of the Proxel coefficients have to be (thereby increasing the computational complexity) in order to ensure accurate results.

7.6 Conclusion

In this chapter, two partial solutions for the Training of CHnMMs were developed and tested: The Training algorithm based on the expectation-maximization paradigm updates an existing completely specified model based on a trace of observations in order to make the model better explain the observations. The second Training algorithm based on the maximum likelihood approach uses a trace of observations to find the most likely values of unknown parameters in an incomplete model.

The practical feasibility of the EM-based approach is identical to that of the CHnMM Smoothing algorithm (cf. Chapter 6), since the algorithm has been shown to be implementable as an evaluation function to the recursive (or even to the ordinary iterative) CHnMM Smoothing algorithm, and this particular evaluation function barely increases the memory consumption or computation time.

Both algorithms can be used to train initial state and symbol emission probabilities. The EM-based algorithm can additionally train the parameters of many well-known probability distributions. However, while the similar Baum-Welch algorithm used to train HMMs has been proven to never result in a worse-fitting model, this behavior cannot be guaranteed for the EM-based CHnMM Training algorithm. Yet, it has been argued that the sometimes slightly suboptimal result of the EM-based approach will usually be acceptable in practice. Overall, the goal of this Chapter to provide a CHnMM Training algorithm with the same power as the HMM Baum-Welch Training algorithm has largely been reached.

Still, no Training algorithm currently exist to train the *type* of parametric probability distribution used to specify activity durations, or how to train arbitrary distribution functions. The EM-based CHnMM Training approach is not guaranteed to result in a model that is more likely to explain the trace of observations than the input model. Thus, while the newly developed algorithms enable the Training of certain aspects of CHnMMs for the very first time, the overall problem of completely training CHnMMs is not yet solved completely.

In this and the previous three chapters, algorithms for all four basic behavior reconstruct tasks derived from HMMs have been developed for the more expressive model class of CHnMMs. The next chapter concludes this work by assessing success or failure of completing the goals of this work. Furthermore, the possible impact of its findings with respect to other research areas and potential applications will be discussed, and an outlook on possible further research will be given.

Chapter 8

Conclusion

In this work, Conversive Hidden non-Markovian Models as an extensive class of partially observable discrete stochastic models have been introduced and defined. CHnMMs are more expressive than the previous state-of-the-art in PODS models, and are in fact a superset of it (cf. Table 2.1 on page 16). The main benefit of CHnMMs over existing approaches is that they can model systems that at the same time are continuous in time with arbitrarilily distributed activity durations *and* contain concurrent activities. They can thus be used to accurately model more real-life systems than was previously possible.

PODS models are usually used to solve one of four well known behavior reconstruction tasks: Evaluation, Decoding, Smoothing and Training. For each of the four tasks algorithms have been developed in this work that can solve them for CHnMMs. The algorithms solve the tasks exactly without having to resort to approximations where applicable (i.e. for Evaluation, Decoding and Smoothing). And for each task, at least one developed algorithm has been shown experimentally to be efficient enough to be practically feasible in real-life applications.

CHnMMs thus extend the state-of-the-art in behavior reconstruction of partially observable discrete stochastic sytems by expanding the set of practical problems whose behavior can be reconstructed.

8.1 Assessment of Goal Completion

The goal for this work (cf. Section 1.3) was to enable practitioners to use the four behavior reconstruction tasks Evaluation, Decoding, Smoothing and Training in partially observable discrete stochastic systems that are continuous in time and contain concurrent activities. Those systems can be modelled as CHnMMs, which have been introduced in Chapter 3, and algorithms for the four basic tasks have been developed in Chapters 4, 5, 6 and 7. Only the CHnMM Training algorithms have been show to be somewhat limited in their effectiveness. Thus, the goals of this work have generally been reached.

Furthermore, two success criteria on those goals where defined that should ensure that the developed algorithms are not only of theoretical interest, but are actually practically applicable to real-life problems. To that end, the first criterion demanded that - where applicable - the algorithms *must* compute ex-

act results and not approximations. For Evaluation, Decoding and Smoothing algorithms have been developed that do not resort to any kind of approximations[1] and are thus exact, fulfilling this success criterion. And to the Training task the exactness requirement is not applicable, since the Training task is only to find a *better* matching model, but does not require any particular model to be returned. Thus, this mandatory success criterion has been met for the Training task as well.

The second success criterion is a soft criterion that demands the developed algorithms to be as efficient as possible with respect to memory consumption and computation time. This criterion was established in order for the algorithms to be computationally feasible for practical problems on commodity computer hardware so that the cost savings realized through the behavior reconstruction with CHnMMs are not eaten up by the additional costs of extensive computing hardware. To that end, algorithms for all four tasks have been shown to be practically feasible with respect to memory consumption and computation time. This feasibility has been shown for a range of models that differ in the number of concurrent activities and the number as well as the connectivity of discrete states, and for long traces of observations (e.g. sequences of 10000 observations). Furthermore, for each task the memory and time complexity of at least one developed algorithm has been shown to be at most $O(n \log(n))$ in the length of the observation sequence whose behavior is reconstructed. Thus, with increasing computing power, the length of an observation sequence whose behavior can be reconstructed in a given amount of time increases by almost the same factor. So, the soft criterion of allowing *efficient* behavior reconstruction on CHnMMs has been met for a wide range of models and observation data.

Overall, the goals of this work have been reached and all success criteria have been adhered to. The research was thus successful.

8.2 Limitations

While the goals set for this work have been reached the developed approaches still have some limitations with regard to their applicability.

First, Conversive Hidden non-Markovian Models were specified as a subset of the more general Hidden non-Markovian Models. This reduced expressiveness made the development of efficient algorithm possible. However, its downside is that CHnMMs can only model PODS systems in which the completion of every activity causes the emission of an observable symbol. Real-life systems in which the completion of some activities may go unnoticed cannot be modeled as CHnMMs and their behavior cannot be analyzed with the algorithms developed.

Second, the CHnMM Training algorithms cannot train all characteristics of a model: They cannot train the connectivity between states (i.e. determine the completion of which activity causes what state change), the aging policy (RACE_AGE or RACE_ENABLE) or the type of parametric probability distribution that describes the duration of an activity. Thus, the developed Training algorithms are not sufficient to build a complete CHnMM model of a real-life system. Expert knowledge and manual modelling of the system is still required.

[1] The reader is again reminded that while the algorithms themselves are indeed exact, practical implementations may deviate from the exact results due to their limited numerical accuracy.

Furthermore, the EM-based Training algorithm has been shown to not always generate more likely models with each iteration. So, in practical applications each model updated by that Training algorithm has to be validated using the Evaluation algorithm in order to ensure that it indeed explains the data better than the previous model. This barely increases the computation time, since the Forward variables (which sum up to that Evaluation probability) are computed by the Smoothing part of the next Training iteration in any case. Still, it cannot be guaranteed that the EM-based CHnMM Training algorithm finds even a locally most likely model – a clear disadvantage over the EM-based HMM Training algorithm for which the convergence to a locally most likely model has been proven [62].

Also, the derivation of the P_{change} probability that is used in every algorithm depends on the fact that the probability of a continuous probability distribution to take on any discrete value is infinitessimal. This property holds for all continuous probability distributions that are also functions, but is notably violated by the deterministic distribution. This means that CHnMMs cannot directly model systems with deterministic behavior. Practically though, this limitation should be of little relevance, since deterministic behavior can be reconstructed without CHnMMs and deterministic distributions may be reasonably approximated by very narrow normal distributions.

Additionally, the developed algorithms have been shown to exhibit a time complexity of at most $O(n \log(n))$ in the length of the input observation sequence and the size of the discrete state space, and exponentially with the number of concurrent activities. So, the model complexity that is practically feasible is severely limited by the number of concurrent activities. Experiments showed that the behavior reconstruction of systems with more than six concurrent activities is not practically feasible on commodity hardware. The time complexity of $O(n \log(n))$ with respect to the size of the discrete state space and the length of the observation sequence means that the algorithms are far more allowing of a bigger discrete state spaces and longer observation sequences. Experiments have been conduction for trace lengths of about 1500 observations, and should still be feasible for traces of about 100000 observations. Similarly, experiments have demonstrated feasible behavior reconstruction for systems with about 5000 discrete states, and systems with up to 100000 discrete states should still be practically feasible. It should be noted though that the size of the discrete state space increases combinatorically: For example, if the system contains four queues that are filled and emptied independently of each other, and each queue can hold up to 100 items, then a discrete state has to individually encode the number of items present in each queue, generating a discrete state space of 100 million (100^4) discrete states. Thus, even some conceptually simple CHnMMs may exceed the number of feasible discrete states.

And finally, while the developed algorithms are the most efficient behavior reconstruction algorithms known for CHnMM, they are less efficient than the existing algorithms for less expressive classes of PODS models. Thus, if a real system can be modelled without concurrent activities, behavior reconstruction algorithms for GHSMMs [70] will be more efficient and easier to implement. And if the system can be modelled with Markovian activity durations alone, CVDHMMs [48] will be more efficient and easier to implement. In both special cases, the CHnMM algorithms will also be far more efficient than for general CHnMMs (since in both cases no age vectors are necessary), but will still be

slower than the existing specialized algorithms.

8.3 Applicability of Findings Beyond CHnMMs

While the research in this work was specifically targeted at developing algorithms for the behavior reconstruction of CHnMMs, some of its findings are also applicable to and may further other research topics.

Reducing the Memory Consumption of Hidden Markov Models Most behavior reconstruction algorithms for Hidden Markov Models assume that the probabilities of the model to be in any state at any given time are concurrently held in memory. For CHnMMs with their vastly increased state space this was no longer feasible and recursive divide-and-conquer algorithms have been developed that solve these tasks while only holding a small fraction of the set of probabilities in memory at any given time. These approaches can directly be used for Hidden Markov Models as well and may become relevant for HMMs whose discrete state space is very extensive and thus memory consumption is an issue.

Applicability of Findings to General HnMMs In this work, the class of CHnMMs was deliberately chosen as a subclass of the more general Hidden non-Markovian Models in order to reduce the complexity of the underlying behavior reconstruction algorithms. CHnMMs require that the completion of every internal activity of the system is externally observable, while general HnMMs do not have that limitation and thus allow for the completion of an arbitrary number of activities in between two observed activity completions. For example, the Tester model used throughout this work models a real system were two machines produce indistinguishable items, but where the completion of each items test is detected. If this system was modelled as a general HnMM, its behavior could even be reconstructed if the completion of some items went unnoticed due to imperfect sensor equipment. Thus, fewer real systems can be modelled as CHnMMs than could as general HnMMs. On the other hand, no feasible behavior reconstruction algorithms for general HnMMs exist as of yet.

While the algorithms developed in this work cannot directly be applied to general HnMMs, they may be extended to develop theoretical behavior reconstruction algorithms for general HnMMs. The basic approach for this extension is to split the time domain into the instants of time at which the completion of activities was observed, and the time intervals between those observations in which an arbitrary number of additional activities may have been completed. For the instants of time of the recorded observations the CHnMM algorithms may be used for the behavior reconstruction, and in the time intervals between them the possible internal behavior can be simulated with the original Proxel method [29] that the CHnMM algorithms are based on.

While this approach may yield theoretically sound algorithms for the behavior reconstruction of general HnMMs, it is unlikely to yield practically feasible algorithms: As discussed in Chapter 4, CHnMMs have some properties that slow them down compared to the Proxel simulation method, and some that make them more efficient than said method. But the behavior reconstruction algorithms for general HnMMs would inherit the disadvantages of both approaches

and thus render the behavior reconstruction of even tiny HnMMs practically unfeasible. A small comparison experiment on the Tester model (cf. Appendix A) showed that the CHnMM Evaluation algorithm takes 0.041 seconds for a trace of 1000 symbols, while the general HnMM algorithm requires 0.096 seconds for the same problem, but about five hours for the equivalent HnMM Tester model where 10% of all item completions go unnoticed.

Thus, the developed CHnMM algorithms may be adapted to reconstruct the behavior of general HnMMs, but the resulting algorithms would only be of theoretical interest and would not be practically feasible.

MultiProxels for the Proxel Method MultiProxels [11] (cf. Section 4.6) were initially developed to eliminate redundancies and thereby speed up the computation of earlier versions of the CHnMM Training algorithms [12]. In these algorithms the model itself contained counters that inflated the state space, but did not change the behavior of the model.

However, the same type of redundancies are present in the other CHnMM algorithms, and even in the original Proxel simulation method. For example the computation time for the Proxel simulation of the "Warranty" model used in the initial publications on the Proxel method [45] has been shown to be reduced by about 50% using MultiProxels [11].

Thus, a technique developed for the behavior reconstruction of CHnMMs can help to improve speed and therefore increase the practical model size for state space-based simulation with the Proxel method.

8.3.1 Direct Probability Computation for the Proxel Method

A common point between the CHnMM behavior reconstruction algorithm and the Proxel simulation method is the need to determine probabilities of state changes. The Proxel method used a set of coupled ordinary differential equations of the type

$$\frac{d\Pi_i}{dt} = \Pi_{sojourn}(t) * \mu_i(t + \tau_i)$$

to describe the state change behavior. The crude Euler integration method or other expensive numerical integration methods are then used to solve these ODEs and compute the desired probabilities [10].

For CHnMMs the same underlying ODEs have been used as a starting point, but it has been shown that certain probabilities of this state change behavior can be computed directly and exactly. These direct formulas developed for CHnMMs can now be inserted into the ODEs and with their help those ODEs can be simplified[2] to integrals of type

$$\Pi_i(t) = \int_0^t \left(\prod_j \frac{1 - F_j(t + \tau_j)}{1 - F_j(\tau_j)} \right) \mu_i(t + \tau_i)\, dt$$

Thus, formulas developed for the behavior reconstruction of CHnMMs can be used to simplify the computation of state change probabilities in the Proxel

[2] "Simplify" here refers to the effort needed to solve the equation accurately, it does not refer to the structure of the mathematical expression.

method from numerical solution of ODEs to numerical solution of an integral, which can be done by comparatively simple methods such as Simpson's Rule [33].

The potential benefit of computing those state change probabilities using integrals have been recognized before [41], but those older attempts used incorrect formulas. The immediate practical impact of these new findings on the Proxel method are low, because the method has at least three sources of errors [10] and numerical integration is only one of them. And at least one of the other error sources is of the same order as the numerical Euler integration [43] meaning that a more accurate computation of state change probabilities will not automatically yield more accurate results. However, if other error sources could be reduced as well (e.g. through a procedure akin to DTMC uniformization [6, 76]), these finding would be a first step in noticably reducing the approximation error of the Proxel method.

8.4 Benefits and Applications of Findings

With the class of CHnMMs that were developed in this work, hidden behavior reconstruction can be applied to additional real-world systems. Thus, it is now possible for additional real-life systems to reconstruct their behavior from already existing data, giving insights into these systems and saving the money of introducing additional sensory equipment for data measurements. In particular, behavior reconstruction is now possible for system with all of the following properties

1. The system can be model with a discrete state space, i.e. all quantities in the model can be represented as integers, and those quantities are only changed through the completion of activities
2. Activity durations are random and follow arbitrary continuous probability distributions. In particular, those durations need not be Markovian.
3. Activities can take place concurrently
4. The end times of all activities *must* be detectable. However, it need not be detectable *which* activity ended, only that one did.

Prior to this work it was only possible to reconstruct the behavior of systems that contain concurrent activities *or* arbitrarily distributed non-Markovian activities. But behavior reconstruction was not possible for systems that contain both characteristics at the same time.

This limitation severely restricted the practical application of prior approaches. Lifting it with CHnMMs is therefore a major step in increasing the practical applicability of the behavior reconstruction of partially observable discrete stochastic systems, especially since many real-life systems contain concurrent non-Markovian activities:

For example, speech recognition of a single individual is often performed using HMM-derivatives that model non-Markovian durations for individual phonemes [62]. But recognizing speech of multiple speakers speaking at the same time would requires the explicit modelling of the concurrent non-Markovian speech activities, which was not possible until now.

Similarly, touch-screen devices may allow for the user to interact with the system by using movement gestures, i.e. by drawing certain shapes on the screen with their fingers. The time needed to drawing characteristics parts of the shape may be random, following a non-Markovian distribution, and a recorded sequence of these times may be used to recognize the shape and therefore the gesture. However, when this interaction is extended to bimanual gestures, then two non-Markovian activities occur concurrently, requiring CHnMMs.

Beyond pattern recognition, most other real-life systems also contain concurrent non-Markovian activities. All systems containing independent agents (i.e. people, cars, animals, machines or molecules) inherently perform activities independently. And most real-life activities are non-Markovian by nature: Mechanical wear is usually Weibull-distributed, many empirically-determined quantities follow a lognormal distribution and due to the central limit theorem many compound activities are normally-distributed [1]. So, a significant number of real-life systems contain concurrent non-Markovian activities, and using CHnMMs, their behavior may now be reconstructed for the very first time.

In particular, it is now possible for the first time to model and reconstruct the behavior of partially observable systems with non-Markovian concurrency process characteristics, such as *fork, join, concurrency, competition, synchronization* and *limited ressources*.

And finally, CHnMMs are thought to provide the basis for the development of virtual stochastic sensors [38], measurement devices that use an easily measurable property of a system in order to estimate another system property that is linked stochastically to the first one.

8.5 Possible Extensions and Future Research

The research in this work primarily led to the development of behavior reconstruction algorithms for CHnMMs. Yet, it also found new questions and research opportunities. Many of those have already been presented at the conclusion of the individual chapters in which CHnMM algorithms were developed. Those will not be repeated in this section. Instead, this section concludes this work by detailing additional, more general ideas for further research.

CHnMMs with Spurious Observations The class of CHnMMs was carefuly chosen to be broad enough for practical application, but at the same time limited enough to allow for efficient behavior reconstruction algorithms. Still, one opportunity for further research is to find model classes that are more expressive than CHnMMs, but do not make behavior reconstruction much more computationally expensive.

Lifting the requirement of CHnMMs that the end of every activity must be detectable would lead to systems with "cause without observable effect", i.e. to general HnMMs, whose behavior reconstruction seems not to be feasible with approaches similar to those developed for CHnMMs. But the opposite direction, the behavior reconstruction of systems with "observable effect without actual cause" may be feasible. This would allow behavior reconstruction of real systems were observations are sometimes caused by false alarms (e.g. a laser barrier in a factory that is triggered not by a passing item, but by dust particles in the

air; or pattern recognition systems where observations are sometimes caused by sensor noise and not by actual behavior).

In models of these systems changes of the internal discrete state are still only possible at the times of symbol observations. Thus, the behavior reconstruction should be similarly efficient to that of CHnMMs, which also adhere to this limitation. The main extension over CHnMM behavior reconstruction is that in these new class of models observations may not have a cause (at least none that the model accounts for) and thus for each observation the possibility that no activity has been completed needs to be taken into account.

Multi-Threaded Proxel-based Algorithms So far, the Proxel simulation method as well as the Proxel-based CHnMM algorithms developed in this work operate purely sequentially. Thus, implementations of these algorithm will only utilize a single CPU core. Extending the existing algorithms to allow for true multi-threading could greatly improve throughput of these algorithms on modern multi-core CPUs and can help to scale the algorithms further, even to high performance computing (HPC) clusters.

All Proxel simulation and CHnMM behavior reconstruction algorithms are ostensibly easy to parallelize: All algorithms iterate over the set of Proxels for one time step and generate successor Proxels for the next time step. Splitting that set of Proxels and letting each available CPU core generate the Proxels for one of the subsets would easily parallelize the algorithms.

The core problem is that Proxel merging is essential to all of these algorithms: duplicate elements in the Proxel set for the next time step need to be found and merged. Currently, all active CPU cores would thus need access to a shared data structure to find those duplicates, and access to this data structure poses a bottleneck. The key challenge for parallelized Proxel-based algorithms is to find a good locking strategy [78] to that data structure that allows for efficient concurrent access while still guaranteeing data integrity.

One step further to scaling Proxel-based algorithms would be to use the massively parallel architecture of modern GPUs with hundreds of cores [60]. While GPUs can potentially offer an order-of-magnitude increase in data throughput [49], optimizing algorithms to take advantage of the unique capabilities of GPUs is complex and not possible for all algorithms [46].

Behavior Reconstruction under Uncertainty with CHnMMs In all developed CHnMM algorithms the provided model specification and observation traces are assumed to be exact. In practical applications, however, the model specifications may be outdated, or the model specifications and trace information may contain errors or inaccuracies based on inaccurate measurements.

One resulting research question that is yet unanswered is how accurate the CHnMM behavior reconstruction results of the Evaluation, Decoding and Smoothing algorithms are when the provided data is inaccurate within known error margins.

CHnMMs with Correlated Behavior In this work, the durations of successively completed activities as well as the symbol emission probabilities of successive observations are independent of each other. In practical applications, however, this may not always be the case. For example, if a machine in the

"Tester" model produces a defective item, leftovers from the defective item may stay in the machine and may be likely to damage the next item as well.

It is yet unclear whether the developed algorithms are still valid when correlations exist in the system behavior, under which circumstances information on the degree of correlation can be integrated into CHnMMs under the current definition, and whether correlated behavior may even increase the information content of the provided data and thus make the behavior reconstruction more accurate.

Information-Theoretical Analysis of CHnMM Models In this work algorithms for the exact behavior reconstruction of CHnMM were developed. Yet, the question whether those results are statistically significant has not been answered. For example, the Decoding algorithm determines the most likely sequence of unobserved internal behavior. But it does not determine whether this most likely behavior is indeed very likely, or whether it is just marginally more likely that any other out of a thousand different behaviors.

A useful tool for practitioners would be an approach to quantify the amount of information present in a trace and thus the level of certainty with which the behavior can be reconstructed. Such an information-theoretical approach may be able to assess whether behavior reconstruction is viable for a given system, or how long a trace on observations would need to be in order to accurately reconstruct the behavior. Ideally, such an approach would answer those questions even before the observation data is ever collected.

Appendix A

Feasibility of HnMM Behavior Reconstruction

The purpose of this appendix is to provide data on how much more costly computations on general HnMMs are compared to CHnMMs. To that end, experiments will be conducted with variants of the Tester model (cf. Section 3.5 on page 31). To make the application of general HnMM algorithms necessary in the first place, the Tester model needs to be modified so that not every completion of a production step emits an observable symbol. We generated such a general HnMM by assuming that the quality tester in the Tester model is faulty and thus fails to record test results for 10% of the tests. Thus, not every completion of an activity causes a symbol emission and consequently activities may have been completed between observations - a system behavior that CHnMMs cannot reconstruct and thus behavior reconstruction algorithms for general HnMMs are required.

To assess the different algorithm computation times, the Evaluation task was performed with a trace for the Tester model with 1500 observations covering a time period of 100000s. With this trace the following behavior reconstructions experiments were conducted:

- The CHnMM Evaluation algorithm was used for the behavior reconstruction of the original CHnMM Tester model in order to provide a baseline against which the HnMM algorithm can be compared
- The general HnMM Evaluation algorithm (cf. Section 8.3) was used to reconstruct the behavior of the original CHnMM Tester model in order to determine the computational overhead of the general HnMM algorithm on CHnMM models.
- Finally, the general HnMM Evaluation algorithm was used to reconstruct the behavior of the HnMM Tester model modified as noted above. This experiment determines the computation time of the HnMM on true HnMM models and thus provides realistic insights into the computational complexity of general HnMM algorithms.

Of these three scenarios, the first one directly computes an exact result, since the CHnMM computations are exact. The usage of general HnMM algorithms on CHnMMs in the second experiment also provides exact results. For

Algorithm	Mean Computation Time (10 Replications)
CHnMM	0.041s
General HnMM	0.096s

Table A.1: Computation times for the CHnMM and general HnMM Evaluation algorithms on the CHnMM Tester model.

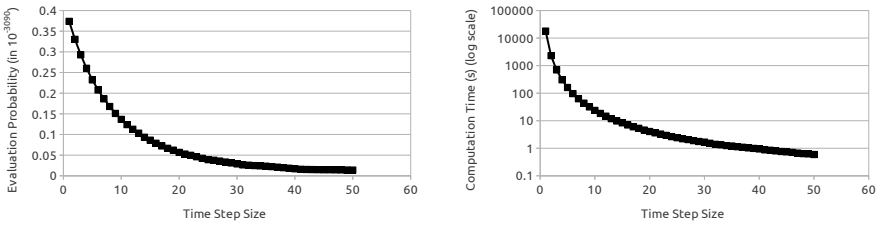

Figure A.1: Plots of the results of the HnMM Evaluation task and the corresponding computation time for different time step sizes. The vertical axis in the right-hand side plot is log-scaled.

the second experiment, the probability computations for activity completions *during* time steps that the general HnMM algorithm takes directly from the Proxel method are indeed only approximations, since they are numerical solutions of either ODEs or of integrals. However, as long as the model is a CHnMM, activities are never completed *during* time steps and thus the inaccurate computations are not used, yielding exact results in the second experiment as well. Only in the third experiment where the general HnMM algorithm is used on general HnMMs do the approximations have an effect. Here, one thus has to select a time step size for the algorithm and this will determine the trade-off between result accuracy and computation time.

The results for the first two experiments are summarized in Table A.1. On a CHnMM model, both algorithms are similarly computationally expensive. The dedicated CHnMM Evaluation algorithm is about twice as fast as the general HnMM one, but both are feasible for the behavior reconstruction of traces of this length.

The results for the HnMM Evaluation task on the general HnMM Tester model are more difficult to assess. Here the computed Evaluation virtual probability is only an approximation and its accuracy depends on the selected time step size. Smaller time steps will result in a smaller error, but also in higher computation times.

The results for different time step sizes are given in Figure A.1. The most accurate computed value is that for the smallest time step size of $1s$. As the plot on the left-hand side shows, the relationship between decreasing step size and the Evaluation result is non-linear, and it is therefore difficult to assess what the correct Evaluation result for the theoretical step size of 0 would be and thus also how accurate the computed results are. Nevertheless, the difference between the most accurate and the second most accurate result is about 13% and it is likely that the difference between the most accurate computed result

and the exact result is at least as high. Thus, for a practical application where the Evaluation probabilities of two models are to be compared to determine the most likely model to cause an observation, even the most accurate computed result would likely be too unreliable.

As the right-hand side of Figure A.1 shows decreasing the time step size leads to an exponential increase in computation time. For the most accurate result the computation time was about 5 hours. In contrast, the computation time for the same algorithm and the same trace for the corresponding *CHnMM* model was less than 0.1s. So in this case the behavior reconstruction of a general HnMM is more than 100000 times more costly than that of the corresponding CHnMM. With 5 hours of computation time it is barely viable for this very small model containing only two concurrent activities and a single discrete state. With this approach, the behavior reconstruction of bigger general HnMMs such as the Car Rental Agency model with more than 5000 discrete states would not be practical feasible for most application scenarios.

Appendix B

Training Concurrent Exponential Activities with MLE

In the derivation of the MLE-based CHnMM Training algorithm (cf. Section 7.2) it was explained that the parameters of probability distributions cannot generally be trained with this approach, because the resulting mathematical expressions for $P_{sojourn}$ and P_{change} would not be polynomials. It was further noted that Training of those parameters is possible at least in one special case: when some activities are always active together, their durations are all exponentially distributed with unknown rates λ_i, and the total rate of all activities together is known.

In this appendix it is shown *why* it is possible to train those parameters. This will be done by separately showing that under these circumstances the mathematical expressions for $P_{sojourn}$ and P_{change} yield polynomials in the unknown parameters.

Sojourn Probability Given a set of n exponentially-distributed activities that fulfill the requirements above, then the sojourn probability is computed based on these activities and all m other concurrently occurring activities as (cf. Equation 4.1 on page 41):

$$P_{sojourn} = \prod_{i=1}^{n+m} \frac{1 - F_i(\tau_i + \Delta t)}{1 - F_i(\tau_i)}$$

This product can be split into two separate products, one for the n exponentially-distributed activities, and one for the remaining m arbitrarily-distributed activities:

$$P_{sojourn} = \prod_{i=1}^{n} \frac{1 - F_i(\tau_i + \Delta t)}{1 - F_i(\tau_i)} \prod_{i=n+1}^{n+m} \frac{1 - F_i(\tau_i + \Delta t)}{1 - F_i(\tau_i)}$$

The second product depends only on the known parameters of the m arbitrarily-distributed activities, and can therefore be evaluated to a definite number, subsequently replaced by the constant c. And the first product can be evaluted

by using the cumulative distribution function of the n exponentially distributed activities with unknown parameters $\lambda_1, \ldots, \lambda_n$:

$$P_{sojourn} = c \prod_{i=1}^{n} \frac{1 - (1 - e^{-\lambda_i(\tau_i + \Delta t)})}{1 - (1 - e^{-\lambda_i \tau_i})} = c \prod_{i=1}^{n} \frac{e^{-\lambda_i(\tau_i + \Delta t)}}{e^{-\lambda_i \tau_i}} = c \prod_{i=1}^{n} e^{-\lambda_i \Delta t}$$

This can be further simplified to

$$P_{sojourn} = c\, e^{(\sum_{i=1}^{n} -\lambda_i \Delta t)} = c\, e^{-\Delta t (\sum_{i=1}^{n} \lambda_i)}$$

Now since the total rate λ of all those exponentially distributed activities together is known, the sum of all unknown rates yields this total rate:

$$\lambda = \sum_{i=1}^{n} \lambda_i$$

Using this equality, the equation for $P_{sojourn}$ can further be simplified to

$$P_{sojourn} = c\, e^{-\Delta t \lambda}$$

This final equation does not depend on any of the unknown rate parameters, but only on the known total rate. Therefore, $P_{sojourn}$ can be computed even if the individual rate parameters are unknown, and the result is always a zeroth degree polynomial (i.e. a constant) in the unknown parameters.

State Change Probability The instantaneous state change probability of the ith activity is computed as (cf. Equation 4.2 on page 44):

$$P_{change_i} = \mu_i(\tau_i + \Delta t) = \frac{f_i(\tau_i + \Delta t)}{1 - F_i(\tau_i + \Delta t)}$$

For an exponential distribution with unknown rate λ_i this yields:

$$P_{change_i} = \frac{\lambda_i\, e^{-\lambda_i(\tau_i + \Delta t)}}{1 - (1 - e^{-\lambda_i(\tau_i + \Delta t)})} = \lambda_i$$

So, for activities with exponentially distributed durations the corresponding virtual probability P_{change} always equals the unknown parameter λ_i and therefore is a first degree polynomial in that parameter.

Conclusion It has thus been shown that for the special Training case of exponentially distributed concurrent activities with unknown individual rate parameters but known total rate the probabilities $P_{sojourn}$ and P_{change} can be expressed as polynomials in the unknown rates. Thus, the MLE-based CHnMM Training algorithm is applicable to this special class even though it is not applicable to the general case of activities with unknown parameters for their duration probability distribution.

Appendix C

Symbols used in this Work

- N ... number of discrete states
- $\{S_1, \dots S_N\}$ ... the set of discrete states
- M ... number of emittable symbols
- $\{V_1, \dots, V_M\}$ ... the set of emittable symbols
- K ... number of unique state transitions
- $TR = \{TR_1, \dots, TR_K\}$ the set of state transitions, with each state transtion being a tuple $(dist, id, b(m), aging)$, with
  - $dist$ ... the probability distribution of the state transition's duration
  - id ... a unique identifier, defined as $TR_k.id = k$
  - $b(m)$ ... the function of symbol emission probabilities, maps each symbol V_m to the probability that the state transition emits symbol V_m while changing the discrete state
  - $aging$ ... a boolean value ($\in \{true, false\}$) specifying whether a state transition memorizes how long it has been active before it was deactivated by a state change ($= true$). If so, it will continue with that stored age when it next becomes activated. Otherwise ($= false$) it will start with age zero.
- $pdf(dist), cdf(dist), hrf(dist)$: The probability density function, cumulative distribution function and hazard rate function of a probability distribution, respectively
- $isExp(dist)$: returns whether the distribution $dist$ is an exponential distribution and thus memoryless (=Markovian)
- A ... matrix of state transitions, with each matrix element $a_{ij} \in TR$ representing the state transition from state S_i to S_j
- Π ... the initial state probability vector, with element π_i being the probability of the model to be in discrete state S_i at time $t = 0$
- λ ... the complete model $\lambda = (A, \Pi)$

- o ... a symbol emission, given as the tuple $o = (e, v)$, with
  - v ... the symbol emitted, $v \in \{V_1, \ldots, V_M\}$
  - e ... the time stamp of the signal **e**mission
- T ... the number of symbol emissions in a trace
- $O = o_1 o_2 \ldots o_T$ ... a trace (=sequence) of T symbol emissions;
- ρ ... a Proxel with $\rho = (q, \vec{\tau}, \alpha[, \beta, \gamma])$:
  - q ... the discrete state of the Proxel
  - $\vec{\tau}$ ... the age vector, containing durations of activities for all non-Markovian state transitions
  - α ... the Forward probability, i.e. the probability to be in the current state (discrete state + age vector) after having emitted the symbol trace so far
  - β ... (optional), the Backward probability, i.e. the probability to still emit the remainder of the trace given that the model is in the current state
  - γ ... (optional), the Smoothing probability, i.e. the probability of the model to be in the current state at the current time, given the observation of the whole trace
- R_t ... The set of Proxels describing all reachable model states after the tth symbol emission
- q_t the discrete state of the model after the tth symbol emission
- age_t the age vector after the tth symbol emission, containing the durations that each activities has already been active since it was last cancelled or completed

Bibliography

[1] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall international series in industrial and systems engineering. Prentice-Hall, 3rd edition, 2000.

[2] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[3] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Reconsidering custom memory allocation. In *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '02, pages 1–12, New York, NY, USA, 2002. ACM.

[4] A. Bobbio, A. Horváth, M. Scarpa, and M. Telek. Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Performance Evaluation*, 54(1):1 – 32, 2003.

[5] A. Bobbio, A. Puliafito, M. Telek, and K. S. Trivedi. Recent developments in non-markovian stochastic petri nets. *Journal of Systems Circuits and Computers*, 8(1):119–158, 1998.

[6] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*, chapter 5, pages 184–185. Wiley, 1998.

[7] Christian Borgelt and Rudolf Kruse. *Graphical Models: Methods for Data Analysis and Mining*, chapter 6, pages 151–158. Wiley, 2002.

[8] Andrei Borshchev and Alexei Filippov. From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools. In *Proceedings of 22nd International Conference of the System Dynamics Society, Oxford, England*, July 2004.

[9] Sascha Bosse. Vergleich klassischer maschineller lernverfahren mit hidden non-markovian models anhand ausgewählter anwendungsbeispiele. Master's thesis, Otto-von-Guericke University, 2008.

[10] Robert Buchholz. Improving the efficiency of the proxel method by using variable time steps. Master's thesis, Otto-von-Guericke University, 2008.

[11] Robert Buchholz, Claudia Krull, and Graham Horton. Efficient state space-based simulation: Avoiding redundancies in the proxel method. In *The 24th annual European Simulation and Modelling Conference, ESM 2010. 25th-27th Oktober 2010, Hasselt, Belgium*, 2010.

[12] Robert Buchholz, Claudia Krull, and Graham Horton. Reconstructing model parameters in partially-observable discrete stochastic systems. In Khalid Al-Begain, editor, *Analytical and Stochastic Modeling Techniques and Applications*, volume 6751 of *Lecture Notes in Computer Science*, pages 159–174. Springer-Verlag, 2011.

[13] Robert Buchholz, Claudia Krull, Thomas Strigl, and Graham Horton. Using hidden non-markovian models to reconstruct system behaviour in partially-observable systems. In *Third International Conference on Simulation Tools and Techniques (SIMUTools)*, 2010.

[14] Robert Buchholz, Christian Krätzer, and Jana Dittmann. Microphone classification using fourier coefficients. In *Information Hiding 2011 (Darmstadt)*, volume 5806 of *Lecture notes in computer science*, pages 235–246, 2009.

[15] J.N. Coleman, C.I. Softley, J. Kadlec, R. Matousek, M. Tichy, Z. Pohl, A. Hermanek, and N.F. Benschop. The european logarithmic microprocesor. *Computers, IEEE Transactions on*, 57(4):532 –546, april 2008.

[16] D. R. Cox. The analysis of non-markovian stochastic processes by the inclusion of supplementary variables. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:433–441, 1955.

[17] Raimund Dachselt and Robert Buchholz. Throw and tilt - seamless interaction across devices using mobile phone gestures. In *Proceedings of the 2nd Workshop on Mobile and Embedded Interactive Systems*, pages 272–278, 2008.

[18] Raimund Dachselt and Robert Buchholz. Natural throw and tilt interaction between mobile phones and distant displays. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, 2009.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[20] Thomas G. Dietterich. Machine learning for sequential data: A review. In T. Caelli, editor, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 15–30. Springer-Verlag, 2002.

[21] Gwynne A. Evans. *Practical Numerical Analysis*. Wiley, 1995.

[22] J. D. Ferguson. Variable duration models for speech. In *Proceedings of the Symposium on the Application of HMMs to Text and Speech*, pages 143–179, 1980.

[23] Gernot A. Fink. *Markov Models for Pattern Recognition - From Theory to Applictions*. Springer Verlag, Berlin, Heidelberg, 2008.

[24] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society*, (222):309–368, 1922.

[25] Steven Fortune and Christopher J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the ninth annual symposium on Computational geometry*, SCG '93, pages 163–172, New York, NY, USA, 1993. ACM.

[26] Inc. Free Software Foundation. The gnu mp bignum library. `http://gmplib.org`, 2011. [Online; accessed 02-November-2011].

[27] Reinhard German and Christoph Lindemann. Analysis of stochastic petri nets by the method of supplementary variables. *Performance Evaluation*, 20(1-3):317 – 335, 1994. Performance '93.

[28] Zoubin Ghahramani. Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures*, volume 1387 of *Lecture Notes in Computer Science*, pages 168–197. Springer-Verlag, 1998.

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

[30] Norman L. Johnson, Samuel Kotz, and N. Kalakrishnan. *Continuous Univariate Distributions*, volume 1. Wiley, 1994.

[31] Norman L. Johnson, Samuel Kotz, and N. Kalakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, 1995.

[32] N.G. Kingsbury and P.J.W. Rayner. Digital filtering using logarithmic arithmetic. *Electronics Letters*, 7(2):56 –58, 28 1971.

[33] Michael Knorrenschild. *Numerische Mathematik*. Fachbuchverlag Leipzig, 2005.

[34] Donald E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms of *Addison-Wesley series in computer science and information processing*. Addison-Wesley, 3rd edition, 1997.

[35] Donald E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching of *Addison-Wesley series in computer science and information processing*. Addison-Wesley, 2nd edition, 2007.

[36] Claudia Krull. *Discrete-Time Markov Chains: Advanced Applications in Simulation*. PhD thesis, Otto-von-Guericke Universität Magdeburg, 2008.

[37] Claudia Krull, Robert Buchholz, and Graham Horton. Improving the efficiency of the proxel method by using individual time steps. In K. Al-Begain, D. Fiems, and G. Horváth, editors, *The 16th International Conference on ANALYTICAL and STOCHASTIC MODELLING TECHNIQUES and APPLICATIONS, 9th-12th June 2009, LNCS 5513*, pages pp. 116–130, 2009.

[38] Claudia Krull, Robert Buchholz, and Graham Horton. Virtual stochastic sensors: How to gain insight into partially observable discrete stochastic systems. In *The 30th IASTED International Conference on Modelling, Identification and Control*. IASTED, 2011.

[39] Claudia Krull and Graham Horton. Hidden non-markovian models: Formalization and solution approaches. In *Proceedings of 6th Vienna International Conference on Mathematical Modelling*, February 2009.

[40] Claudia Krull and Graham Horton. Solving hidden non-markovian models: How to compute conditional state change probabilities. In *21st European Modeling and Simulation Symposium*, 2009.

[41] Hannes Köberle. Analyse und implementierung von strategien zur bestimmung geeigneter schrittweiten für die proxelbasierte simulation von stochastischen modellen. Master's thesis, Otto-von-Guericke University, 2006.

[42] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001.

[43] Sanja Lazarova-Molnar. *The proxel-based method: formalisation, analysis and applications*. PhD thesis, Otto-von-Guericke Universität, 2005.

[44] Sanja Lazarova-Molnar and Graham Horton. An experimental study of the behaviour of the proxel-based simulation algorithm. In *Proceedings of Simulation und Visualisierung 2003*. Simulation und Visualisierung, SCS European Publishing House, 2003.

[45] Sanja Lazarova-Molnar and Graham Horton. Proxel-based simulation of a warranty model. In *European Simulation multiconference 2004*. SCS European Publishing House, 2004.

[46] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. *SIGARCH Comput. Archit. News*, 38:451–460, June 2010.

[47] S. Levinson. Continuously variable duration hidden markov models for speech analysis. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.*, volume 11, pages 1241 – 1244, apr 1986.

[48] S.E. Levinson. Continuously variable duration hidden markov models for automatic speech recognition. *Computer Speech & Language*, 1(1):29 – 45, 1986.

[49] Peter J. Lu, Hidekazu Oki, Catherine A. Frey, Gregory E. Chamitoff, Leroy Chiao, Edward M. Fincke, C. Michael Foale, Sandra H. Magnus, William S. McArthur Jr., Daniel M. Tani, Peggy A. Whitson, Jeffrey N. Williams,

William V. Meyer, Ronald J. Sicker, Brion J. Au, Mark Christiansen, Andrew B. Schofield, and David A. Weitz. Orders-of-magnitude performance increases in gpu-accelerated correlation of images from the international space station. *Journal of Real-Time Image Processing*, 2009.

[50] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Series in Parallel Computing. Wiley, 1995.

[51] Andrew Mccallum and Dayne Freitag. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598. Morgan Kaufmann, 2000.

[52] William Q. Meeker and Luis A. Escobar. *Statistical Methods for Reliability Data*, chapter 2, page 28. Wiley, 1998.

[53] C.D. Mitchell and L.H. Jamieson. Modeling duration in a hidden markov model with the exponential family. In *In Proc. ICASSP*, pages 331–334, 1993.

[54] Otto Moeschlin, Eugen Grycko, Carsten Poppinga, and Frank Steinert. *Discrete Stochastics*. Springer-Verlag, 2003.

[55] M.K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 31:913–917, 1982.

[56] Ramon E. Moore. *Methods and applications of interval analysis*. Society for Industrial and Applied Mathematics, 1979.

[57] Jens Müller. Ausarbeitung und realisierung eines standards für simulationsmodelle zur langfristigen unternehmensplanung in der produktionssteuerung der bmw ag. Master's thesis, Otto-von-Guericke University, 2009.

[58] Michael Neike. Analyse und bewertung der geplanten kapazitätserhöhung einer bestehenden dünnschichtsolar-produktionslinie. Master's thesis, Otto-von-Guericke University, 2009.

[59] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: an Algorithmic Approach*, chapter Chapter 2: Probability Distributions of Phase Type. Dover Publications Inc., 1981.

[60] NVIDIA Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi (Whitepaper)*, 2009.

[61] Paul Walton Purdom and Cynthia A. Brown. *The Analysis of Algorithms*. Holt, Rinehart and Winston, Inc., 1984.

[62] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 –286, feb 1989.

[63] Padma Ramesh and Jay G. Wilpon. Modeling state durations in hidden markov models for automatic speech recognition. In *Proceedings of the 1992 IEEE international conference on Acoustics, speech and signal processing - Volume 1*, ICASSP'92, pages 381–384, Washington, DC, USA, 1992. IEEE Computer Society.

[64] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[65] Daniel S. Roche. Space- and time-efficient polynomial multiplication. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, ISSAC '09, pages 295–302, New York, NY, USA, 2009. ACM.

[66] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[67] M. J. Russell and A. E. Cook. Experimental evaluation of duration modeling techniques for automatic speech recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 2376–2379, 1987.

[68] Gunter Saake and Kai-Uwe Sattler. *Algorithmen und Datenstrukturen: Eine Einführung mit Java*. dpunkt-Verlag, 2010.

[69] Felix Salfner. Modeling event-driven time series with generalized hidden semi-markov models. Technical report, Humboldt-Universität zu Berlin, 2006.

[70] Felix Salfner. *Event-based Failure Prediction: An Extended Hidden Markov Model Approach*. PhD thesis, Humboldt-Universität zu Berlin, 2008.

[71] Felix Salfner and Miroslaw Malek. Using hidden semi-markov models for effective online failure prediction. In *2007 26th IEEE International Symposium on Reliable Distributed Systems SRDS 2007*, pages 161–174. IEEE, 2007.

[72] J. Sansom and P. Thomson. Fitting hidden semi-markov models to breakpoint rainfall data. *Journal of Applied Probability*, (38A):142–157, 2001.

[73] Robert Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. Wiley, 1992.

[74] Sebastian Schönfeld. Entwicklung und bewertung von optimierungskonzepten für eine solarzellen-produktionsanlage auf basis einer materialflusssimulation. Master's thesis, Otto-von-Guericke University, 2008.

[75] Norman Siemer. Simulationsgestützte bewertung von planungsszenarien für ein werksübergreifendes produktionssystem am beispiel der bmw werke regensburg und leipzig. Master's thesis, Otto-von-Guericke University, 2011.

[76] William J. Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*, page 361. Princeton University Press, 2009.

[77] David Stirzaker. *Elementary Probability*, page 244. Cambridge University Press, 1994.

[78] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2nd edition, 2001.

[79] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

[80] Hanna M. Wallach. Conditional random fields: An introduction. Technical report, University of Pennsylvania, 2004.

[81] Wei Wei, Bing Wang, and Don Towsley. Continuous-time hidden markov models for network performance evaluation. *Performance Evaluation*, 49(1-4):129–146, 2002.

[82] Fabian Wickborn and Graham Horton. Feasible state space simulation: Variable time steps for the proxel method. In *2nd Balkan Conference in Informatics, 17th-19th November 2005, Ohrid, Institute of Informatics, Faculty of Natural Sciences and Mathematics, Skopje, Macedonia*, 2005.

[83] Fabian Wickborn and Graham Horton. Reducing the effect of stiffness for a state space-based simulation method using adaptive time steps. In *6th International Workshop on Rare Event Simulation*, 2006.

[84] Fabian Wickborn, Claudia Isensee, Thomas Simon, Sanja Lazarova-Molnar, and Graham Horton. A new approach for computing conditional probabilities of general stochastic processes. In *39th Annual Simulation Symposium 2006*, April 2006.

[85] Paul Wilson. Uniprocessor garbage collection techniques. In Yves Bekkers and Jacques Cohen, editors, *Memory Management*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1992.

[86] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 2nd edition, 2005.