UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Extending Markov Modulated Poisson processes for learning and prediction

Francesco Santoni

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Extending Markov Modulated Poisson processes for learning and prediction

Francesco Santoni

Advisor:

Prof. Enrico Vicario

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Aad Van Moorsel, *Newcastle University*
Prof. Peter Buchholz, *Technische Universität Dortmund*

XXXIII ciclo — October 2021

To my parents, to Giulio

Acknowledgments

This thesis is the result of three years of research I had the privilege of spending as a Smart Computing PhD student thanks to the Pegaso Fellowship. It is my honour to thank here a multitude of people without which this work would've stayed a dream or become a regret. The first and most important is my supervisor Prof. Enrico Vicario, who in these years not only mentored me but also enabled me to get over unavoidable hardships. I greatly benefited from his expertise, judgement and breadth of view; his help was invaluable in the completion of this thesis.

I also thank Prof.Laura Carnevali for her support, knowledge and experience and the straightforwardness in braving together with me the many bugs I introduced during my coding. I have immense gratitude for all the members of the Software and Technology Laboratory (STLab) with whom I worked at the University of Florence, not in order of importance: Dr. Fulvio Patara, Dr. Sara Fioravanti, Dr. Marco Biagi, Dr. Samuele Sampietro, Dr. Jacopo Parri, Dr.Tommaso Papini, Dr. Graziano Manduzio. I also would like to thank Miss Luyao Ye for her help and support as a fellow PhD student in Smart Computing in Florence and as a tireless guide during our trip to Beijing. I am grateful to Prof. Ezio Bartocci for his mentorship and for giving me the opportunity of spending six months as a visiting student to the T.U Wien Cyberphysical Systems Laboratory, even during the difficult Covid-19 period. I thank Prof.Paolo Frasconi for his direction of the PhD course and the great help I received. His immense knowledge of machine learning was integral in my research and the logistical steps he had to facilitate were instrumental in completing this work.

I also thank all the member of the Supervisory Committee in my three years: Prof. András Horváth, Prof.Ezio Bartocci and Prof.Peter Buchholz and Prof.Aad van Moorsel.

Apart from collaborators and scientific help, I need to mention and thank my parents and my late brother Giulio: without your love, encouragement and help i would not be here.

Abstract

The focus of this thesis is on extensions of Markov Modulated Poisson processes (MMPPs) with the aim of showing the benefits of continuous time models in approximating event-driven processes over standard techniques like machine learning. Generative models like MMPPs enable methods of quantitative evaluation supporting diagnostic and predictive analytics; their ability of generating new synthetic sequences of events permits to calculate properties both through analytic formulas and Monte Carlo simulation of sequences. The first chapter of the thesis expands on the goal of the thesis and presents preliminaries on markovian processes, respectively discrete and continuous time hidden Markov models (HMMs and CT-HMMs), Markov Arrival processes (MAPs) and MMPPs. Finally, the chapter introduces the contributions offered in this thesis. The subsequent part of the thesis details several extensions of MMPPs directed to addressing different types of events produced in real world systems. In the second chapter is described the first extension to MMPP for the efficient learning of processes that produce events with a discrete type: Marked Markov Modulated Poisson Processes are MMPPs associated with a mark at each observation; while this mark can be any type of information, focus is given to simplifications for specifically discrete types of events. In the third chapter is described the second extension, Marked Markov Modulated Compound Process (M3CPP), optimized for processes where events are marked by a combination of any number of discrete dimensions and possibly a single continuous one. An open problem of MMPPs is the restriction to exponential distributed times in the sojourn in states: to lower the impact of this restriction an extension is presented where normal states are grouped into collections with specific inter-collection transitions, effectively creating macrostates with acyclic phase-type distributed sojourn times. For all extensions the likelihood is calculated and the algorithm for parameter estimation is presented via Expectation Maximization (EM). The extensions are studied on both simulated and real world datasets. For M3CPP the comparison is made to the machine learning approach of Long Short-term Memory (LSTM) showing that the proposed extension is well performing with respect to current techniques both on simulated and real world datasets. The fourth chapter presents the conclusions and future work.

Contents

Chapter 1

Introduction and Preliminaries

1.1 Introduction

Approximations are commonly used to calculate quantitative properties of systems or processes. Every model is associated to assumptions on the system it is approximating and the correct identification of such assumptions has significant impact on the results: excessively lax assumptions may end up with the application of a model that comprises behaviour that violates the invariants of the system, on the other hand excessively restrictive ones could otherwise lead to the modelling of only a subset of the system's behaviour. Stochastic models assume that the observations of the process under study are describable by random probability distributions and allow the application of statistical techniques. When the underlying state of the process is not directly observed, but can be inferred through secondary observation processes, as is for example the case of machinery whose internal state is observed only through performance indicators, models that themselves have *hidden states* are commonly used. The approximation of processes through stochastic models where a hidden state influences an observation process has developed along different directions for *discrete* and for *continuous* abstractions of time, focusing on Hidden Markov Models (HMMs) and Markovian Arrival Processes (MAPs), respectively. Both are generative models and as such enable methods of quantitative evaluation supporting diagnostic and predictive analytics; their ability of generating new synthetic sequences of events furthermore permits the calculation of properties both through analytic formulas and Monte Carlo simulation of sequences.

On the one hand, HMMs associate observed events with the states of a hidden Discrete Time Markov Chain (DTMC) [Rabiner, 1989], emphasizing the problem of learning by maximizing likelihood in the sequencing of observation series, through Expectation Maximization [Dempster et al., 1977] or gradient methods.

Extensions beyond the limits of Markovian behavior have also been developed to permit learning of dependency of observed behavior on the age of the (discrete)

sojourn time within a state [Murphy, 2002]. Models based on HMM have encountered great success when used in several classification tasks, notably in speech recognition and synthesis [Rabiner, 1989] , though more recent advancements indicate that learning on discrete time observation sequences can achieve better performance with deep learning techniques, e.g. artificial neural networks in the architecture of Long Short Term Memory (LSTM) [Zen, 2015]. On the other hand, in Markovian Arrival Processes (MAPs) [Neuts, 1979] the arrival rate of untyped observable events is determined by the state of a hidden Continuous Time Markov Chain (CTMC), to obtain a generative model approximating stochastic behavior of arrival processes arising in performance evaluation [Lucantoni et al., 1990,Buchholz, 2003,Casale et al., 2009], usually in the steady state, through Moment Fitting [Casale et al., 2008, Kriege and Buchholz, 2010, Telek and Horváth, 2007] or also through a continuous time implementation of Expectation Maximization [Buchholz, 2003]. To reduce learning complexity and demand on the volume of training data, Markov-modulated Poisson processes (MMPPs) restrict MAPs by restraining arrivals not to change the state of the hidden CTMC [Fischer and Meier-Hellstern, 1993], with successful application in modeling network traffic [Okamura et al., 2007], self-adaptive software systems [Perez-Palacin et al., 2012] and disk I/O patterns [Verma and Anand, 2007]. In turn, marked Markov-modulated Poisson Processes (M3PP) [Casale et al., 2016] extend MMPPs by associating arrivals with class labels [He and Neuts, 1998] enabling fitting of arrival processes with typed events. As a main difference with respect to HMM, in MAP-like models, the current hidden state affects not only the type and possibly the intensity of emitted events but also their arrival time, with multiple events occurring during the sojourn in the same state, thus enabling the learned model to retain information not only about the sequencing of observed events but also about their quantitative timing. Moreover, the continuous time abstraction natively fits the case of asynchronous time series and the needs for predicting expected behavior within a time rather than within a number of events, avoiding the need of discretization of timestamps which incurs in a hard trade-off between accuracy and complexity with critical impact on the quality of the learning process [Salfner et al., 2010]. This has motivated application for diagnostic and predictive tasks in a variety of application scenarios such as activity recognition for ambient assisted living [van Kasteren et al., 2008, Ordóñez and Roggen, 2016], clinical protocols for disease progression monitoring [Liu et al., 2015] on-line failure prediction [Salfner, 2006], software aging and rejuvenation [Okamura et al., 2009], run-time verification of self-adaptive software [Calinescu et al., 2012], anomaly detection in cyber-security [Malhotra et al., 2015].

Contributions In this thesis the aim is to extend Markov Modulated Poisson processes (MMPPs) to overcome some of their restrictions, with the aim of showing the

benefits of continuous time models in approximating event-driven processes over standard techniques like machine learning.

In the second chapter of the thesis is described an extension to MMPP for the efficient learning of processes that produce events with a discrete type: Marked Markov Modulated Poisson Processes (MMMPPs or M3PPs) are MMPPs associated with a mark at each observation; while this mark can be any type of information, focus is given to simplifications for specifically discrete types of events. We use such model to abstract a system that evolves across transient states of a hidden process until reaching a final absorbing state, and produces a sequence of observable events with stochastic arrival times and types conditioned by the current state. We present an Expectation-Maximization algorithm to learn the parameters of the M3PP from a set of observation sequences acquired in repeated execution of the transient behavior of the system under analysis, as the left-to-right structure of the hidden process requires. The model is then evaluated by calculating the likelihood of different states of the hidden process at runtime based on the actual sequence of observed events, and dynamically evaluating the distribution of the remaining time to reach the final absorbing state. In so doing, we extend the EM algorithm of [Buchholz, 2003] by encompassing event types as in marked processes and by tailoring the entire approach to support online short term prediction, which in particular leads to learning a left-to-right model with transient behavior from a set of traces and to reducing the space of parameters in the MMAP to limit computational complexity and data demand. We also extend the method of [Liu et al., 2015] by learning a model where not only observation values but also arrivals are conditioned by the current system state.

Applicability of the proposed approach is illustrated with reference to the context of statistical intrusion detection, using synthetic datasets generated according to the timing of a stochastic attack tree of the literature [Kriaa et al., 2012, Arnold et al., 2014a] here enriched with a synthetic observation model that associates each visited state with an expected statistics of types and inter-arrival times of observations. Experiments are performed so as to evaluate the accuracy of prediction, evaluated in terms of precision and recall measures defined as in the formulation of online failure prediction [Salfner and Malek, 2007], under different levels of variability in the timing of the hidden process as well as in the variability of arrival times and of ambiguity of event types observed within visited states. Prediction accuracy is then compared against two variants of the approach that emulate the behavior of the methods of [Liu et al., 2015] and [Buchholz, 2003] to illustrate how the two side-channels of observation types and arrival times may have a different relevance under different conditions of variability.

In the third chapter we extend the class of M3PP by efficiently learning models where events have composite types, composed of one or more dimensions with discrete values (type) and a dimension with a numerical value of intensity and intro-

duce a phase-like structure based on Continuous Phase Types [Neuts, 1981,Horváth and Telek, 2002,Reinecke et al., 2013] that restrains the number of model parameters while capturing non-exponential sojourn times in hidden states. Complying with pre-existing models, we term this extension marked Markov-modulated compound Poisson process (M3CPP).

For the efficient implementation of the learning task on this model, we introduce an algorithm of Expectation Maximization based on the one for M3PPs, providing an explicit representation that permits to accommodate treatment of the numerical intensity of the events and the restrictions on the structure of the hidden states' connections. We then address the effectiveness of the model in a task of prediction of the remaining time to the first passage in a state of interest, comparing with LSTMs, a machine learning technique focused on long term memory. To do so we first generate datasets from generator models in the class of Semi-Markov Processes, themselves created randomly according to explicit rules about size and variability of observations, which exhibit homogeneous observable behaviour during the sojourn in a hidden state, while allowing the age of sojourn to influence remaining time and choice of the next hidden transition. This setting encompasses many real world scenarios ranging from ageing systems, where the ending state corresponds to the system breaking down, to Phased mission systems, where the process simply reaches its natural end and allow us to infer on the applicability of such models at large. We find that for such generated datasets M3CPP do indeed have an edge over LSTM, being on average better or tying.

In chapter four we further explore the potentiality of M3CPP as a model for failure prediction, applying it to two real world datasets of High-performance computing, respectively the HPC Intrepid and Mira from Argonne National Laboratory. On these datasets the models are used to predict failures from RAS data, demonstrating the feasibility of the approach. Through the comparison to LSTM, we show that in these HPC datasets and in more general in the context of event-driven processes M3CPP has comparable or better performance than LSTM, especially under data scarcity.

1.2 Preliminaries

Stochastic Process A stochastic process or random process $\{X(t), t \in \mathcal{T}\}$ is a collection of random variables indexed by some set $\mathcal{T}$ in a given probability space and measurable space. $\mathcal{T}$ is usually thought as the time. If $\mathcal{T}$ is a countable set then the stochastic process is said to be discrete-time. If $\mathcal{T}$ is the entire line, then it is said to be a continuous-time stochastic process.

$X(t)$ is called the state of the process at time t and the set of all possible values of $X(t)$ is called the state space, often denoted by S.

Markov Process Markov processes are stochastic processes that have the Markov property, that is the next value of the process depends only on the current value irrespective of the past values of the stochastic process.

$$P\{X(t_2) = j|X(t_1) = s_1, X(t_0) = s_0\} = P\{X(t_2) = j|X(t_1) = s_1\}$$

For all t_2, t_1, t_0 such that $t_2 > t_1 > t_0$.

If $P\{X(t+s) = j|X(t) = s\} = P\{X(t) = j|X(0) = s\}$ the process is said to be *time-homogeneous*.

When the state space is discrete the discrete (continuous) Markov processes are called discrete (continuous) Markov chains.

Hidden Markov Model

First introduced by Baum [Baum et al., 1970], hidden Markov models (HMM) have been increasingly used in fields like speech recognition and synthesis, finance, biology and others. A seminal paper for HMM is the tutorial by Rabiner [Rabiner, 1989].

HMMs are doubly stochastic processes $\{X_n, Y_n\}$ composed by a discrete-time Markov process X_n representing the unobservable *hidden state* and $P(Y_n \in A|X_1 = x_1, \ldots, X_n = x_n) = P(Y_n \in A|X_n = x_n)$ for every $n > 1$,$x_1, \ldots, x_n$ and a measurable set A.

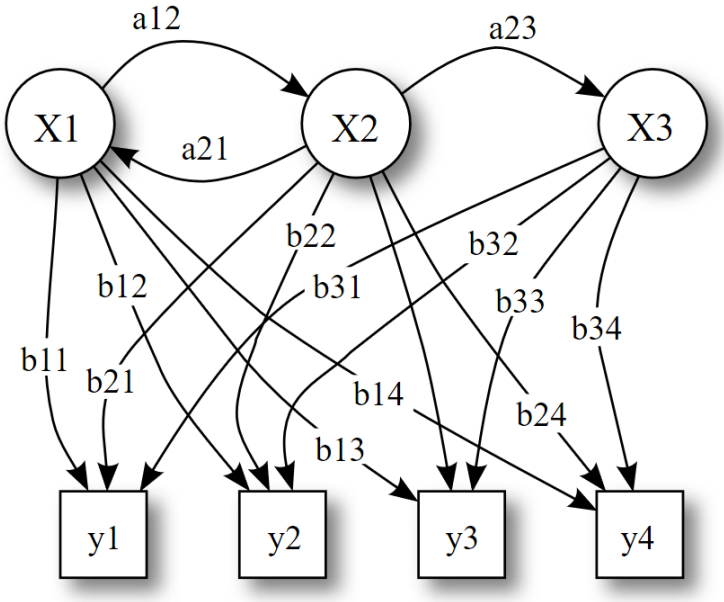

Figure 1.1: Example of an HMM, X_i are the states, Y_k are the observable events; in this image, a_{ij} are the transition probabilities and b_{ik} are the emission probabilities in each state

$P(Y_n \in A|X_n = x_n)$ is called *emission* (or *output*) *probability*.

It is assumed that the state distribution is irreducible and homogeneous leading to the existence of an unique and strictly positive stationary distribution.

Due to the inobservability of X_n the inference is based only on the information relayed by the emission process Y_n.

Inference in an HMM is typically likelihood-based since the likelihood can be written as:

$$L(\pi, P, \Theta) = \sum_{x_1 \in S} \cdots \sum_{x_n \in S} \pi_{x_1} P(Y_1 = y_1|x_1) p_{x_1,x_2} \ldots P(Y_n = y_n|x_n)$$

where π is the vector of initial probabilities of the states and Θ is the sequence of n observations. To note is that the summation is over $n^{|S|}$ permutations, so it becomes intractable with exponential speed even for small state spaces.

To remedy the problem, approaches like the ones proposed in [Baum et al., 1970] and [Rabiner, 1989] are used:

be the *forward probability* $\mathbf{a}^{(t)}(i)$ defined as the probability of being in state $x_t = i$ given the observations up to time t and can be written as:

$$\mathbf{a}^{(t)}(i) = \sum_{x_1,\ldots,x_{t-1}} \pi_{x_1} P(Y_1 = y_1|x_1) \prod_{k=2}^{t} \{p_{x_{k-1},x_k} P(Y_k = y_k|x_k) \mathbb{1}\{x_t = i\}\}$$

Where $\mathbb{1}(\cdot)$ is the indicator function.

And be the *backward probability* $\mathbf{b}^{(t)}(i)$ defined as the probability of observing the emissions from $t+1$ to the end given that at observation t the Markov Chain was in state i.

$$\mathbf{b}^{(t)}(j) = \sum_{x_{t+1},\ldots,x_n} \pi_{x_1} \mathbb{1}\{x_t = j\} \prod_{k=t+1}^{n} p_{x_{k-1},x_k} \{P(Y_k = y_k|x_k)\}$$

From this we can write the likelihood as $L(\pi, P) = \sum_{i=1}^{|S|} \mathbf{a}^{(t)}(i)\mathbf{b}^{(t)}(i)$

Since $\mathbf{a}$ and $\mathbf{b}$ can be updated recursively the problem becomes feasible.

The likelihood function can then be evaluated in multiple ways. In his tutorial Rabiner [Rabiner, 1989] proposes to use the Expectation-Maximization (EM) algorithm, an extension of the Baum-Welch algorithm.

The algorithm iteratively executes two steps: 1) create the expectation of the complete data likelihood (considering also the hidden state movements) with respect to the missing data 2) maximize the expectation. This second step can often be done in analytical form, as is the case for HMMs. Since the algorithm returns at each iteration an improved likelihood the algorithm is stable. A comprehensive description of the algorithm is given by Bilmes in [Bilmes et al., 1998].

In brief, let us have a complete log Likelihood $CL(\theta'|\mathbb{X}, \mathbb{Y})$ where θ' is the parameter vector to estimate, $\mathbb{Y}$ is the vector of observations and $\mathbb{X}$ is the vector of missing data (in our case hidden states). Then the Expectation step (E-step) requires the calculation of

$$Q(\theta'|\theta) = E_\theta(CL(\theta'|\mathbb{X}, \mathbb{Y})|\mathbb{X})$$

with θ being the actual, unseen, parameter vector to approximate. The Maximization step (M-step) requires the maximization of $Q(\theta'|\theta)$ with respect to θ. Such steps are repeated until some convergence criterion is met. To note, is that in the case of HMM multiple local maxima exist so a single execution of the algorithm only guarantees reaching a local maxima.

Continuous Time Hidden Markov Model

Hidden Markov Models encounter difficulties due to their sojourn time distributions being geometric. Many different solutions have been tested by changing the number of time units allowed in a single state to multinomial distributions or more complex discrete distributions (Hidden Semi-Markov Models [Salfner, 2006].

Expanding in a different direction, Liu et al. [Liu et al., 2015], as others, have proposed an extension of the Hidden Markov Model to account for continuous time spent in each state and continuous time delay between observations of events named Continuous Time Hidden Markov Model (CT-HMM). Given the necessity of the Markov condition the only sojourn time and emission distribution allowed is the exponential distribution.

To note is that the processes of the states' evolution and of the emission of observations become independent: it is now possible to emit multiple observations in the same state and move between multiple states before producing the subsequent observation.

The complete likelihood (as if it were fully observed) of a CT-HMM can thus be written as:

$$CL = \prod_{v'=0}^{V'-1} q_{x_{v'},x_{v'+1}} e^{-q_{x_{v'},\tau_{v'}}} \prod_{v=0}^{V} P(y_v|s(t_v))$$

where v', v are the indexes of the V' and V total changes of state and emissions, $q_{i,j}$ is the (i,j)-th element of the infinitesimal generator of the Continuous time markov chain underlying the CT-HMM and $s(t_v)$ is the state at time t_v.

The learning of CT-HMM consists in solving two challenges: estimating the posterior state probabilities and computing end-state statistics.

While we refer to [Liu et al., 2015] for the actual solution to the problems we note how resolving the first problem requires reformulating the problem as a time-inhomogeneous HMM and solving using standard HMM techniques.

Markov Arrival Process and Markov Modulated Poisson Process

Markovian Arrival Processes (MAPs) reproduce the statistics of a sequence of untyped events through a structured process where arrival rates are modulated by a hidden Continuous Time Markov Chain (CTMC) [Neuts, 1979]. This class, which includes Markov Modulated Poisson Processes (MMPPs) as a special case, has been widely investigated and applied to fit characteristics of real workload series in various processes of performance engineering [Casale et al., 2009].

Various techniques have been proposed for the identification of a MAP from an observation trace capturing dependencies in the steady-state behavior of a system [Buchholz, 2003], facing with various approaches the complexity of non-linear optimization of a large number of parameters and further afflicted by the length of the training trace needed to capture long term dependencies [Kriege and Buchholz, 2010, Casale et al., 2008, Telek and Horváth, 2007].

In particular, in [Buchholz, 2003], fitting is performed through a continuous time variant of the EM algorithm, with good results in the reproduction of autocorrelation observed in a real traffic stream.

Both Markov-modulated Poisson Process (MMPP) [Fischer and Meier-Hellstern, 1993] and Markov Arrival Processes(MAP) [Neuts,], are doubly stochastic processes $\{N(t), J(t)\}$, where $N(t)$ is a *counting process* modelling the number of arrivals (*events*) in $[0, t]$ and $J(t)$ is the instantaneous internal state of a *hidden process*. In a MMPP (or MAP) of order n, n is the size of the state space of the CTMC underlying the hidden process. Let $\mathbf{Q}$ be the $n \times n$ infinitesimal generator matrix of the CTMC. The matrix $\mathbf{Q}$ can be seen as the sum of two matrices $\mathbf{D_0}$ and $\mathbf{D_1}$, representing respectively the rates of *unobservable transitions* (i.e., transitions that do not produce an event) and the rates of *observable transitions* (i.e., transitions that produce an event), i.e., $\mathbf{Q} = \mathbf{D_0} + \mathbf{D_1}$. $\mathbf{D_0}$ and $\mathbf{D_1}$ are usually termed *transition rate matrix* and *emission rate matrix*, respectively. Both MMPPs and MAPs can then be defined as the pair of parameters $(\mathbf{D_0}, \mathbf{D_1})$, with

$$\mathbf{D_0} = \begin{pmatrix} -\mu_{1,1} & \mu_{1,2} & \cdots & \mu_{1,n} \\ \mu_{2,1} & -\mu_{2,2} & \cdots & \mu_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{n,1} & \mu_{n,2} & \cdots & -\mu_{n,n} \end{pmatrix}$$

$$\mathbf{D_1} = \begin{pmatrix} \lambda_{1,1} & \lambda_{1,2} & \cdots & \lambda_{1,n} \\ \lambda_{2,1} & \lambda_{2,2} & \cdots & \lambda_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n,1} & \lambda_{n,2} & \cdots & \lambda_{n,n} \end{pmatrix}$$

with all $\mu_{i,j}$ and $\lambda_{i,j}$ non-negative and $\mu_{i,i} = \sum_{j \neq i} \mu_{i,j} + \sum_j \lambda_{i,j}$.

MMPPs differ from MAPs in the fact that arrivals of events do not affect the hidden process, that is while in MAP observations can be associated to an underlying change of state, in MMPP changing state and observing emissions are separate facts. Figure 1.2 gives an example of MMPP: the straight arrows represent transitions between states, the dotted arrows represent emissions of events. To note how no dotted arrow connects different states.

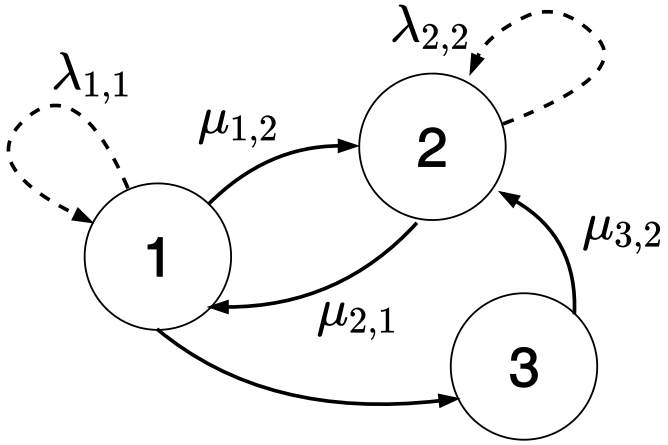

Figure 1.2: Example of an MMPP: $\lambda_{i,j}$ are the rate of emissions of observations while $\mu_{i,j}$ are the rate of transition between states.

According to this, for a MMPP of order n $\mathbf{D_1}$ is a diagonal matrix with entries $\lambda_{1,1}, \ldots, \lambda_{n,n}$, i.e., $\mathbf{D_1} = \text{diag}(\lambda_{1,1}, \ldots, \lambda_{n,n})$.

To note, if $\lambda_{i,i} = \lambda \; \forall \, i \in [1, n]$, then the MMPP is just a Poisson process with rate λ.

Learning MAP and MMPP has been generally done in two ways, 1) through the fitting of the moments, 2) by maximizing the likelihood. We will introduce only the second case and refer the reader to [Buchholz et al., 2010] and [Casale et al., 2016] for the first case.

The likelihood estimation for MAP and MMPP is similar and usually based on the forward and backward likelihoods vectors $\mathbf{a}^{(t)}$ and $\mathbf{b}^{(t)}$ containing at the i-th component (state) the value $\mathbf{a}^{(t)}(i)$ and $\mathbf{b}^{(t)}(i)$. To note that we refer with t more generally to the index of the observation sequence.

$$\mathbf{a}^{(t)} = \mathbf{a}^{(t-1)} \mathbf{D_0} [\delta_t] \mathbf{P_1}$$

with $\mathbf{a}^0 = \pi$

$$\mathbf{b}^{(t)} = \mathbf{D_0} [\delta_t] \mathbf{P_1} \mathbf{b}^{t+1}$$

with $\mathbf{b}^n = e^T$ where e^T is the vector of 1s, and δ_i is the delay between observations i and $i+1$.

Where $\mathbf{D_0}[\delta_t]$ is the transition matrix (of the hidden state) after a time δ_t has elapsed.

The calculation of this matrix involves matrix exponentials, that is the sum of a convergent power series of matrixes, due to the possibility of a potentially unlimited number of steps taken in the time δ_t.

As for HMM the likelihood can be written as $L = \sum_{i=1}^{|S|} \mathbf{a}_t(i)\mathbf{b}_t(i)$ and calculated through multiple methods.

To use Newton type algorithms requires differentiation with respect to parameters from a product of matrices and is thus complex.

Ramesh [Ramesh, 1995] uses the downhill simplex method to avoid such derivatives.

Asmussen [Asmussen et al., 1996] and Ryden [Rydén, 1996] utilized the EM algorithm considering as missing date the entire sequence of hidden state changes.

Buchholz in [Buchholz, 2003] applies the technique of uniformization [Stewart, 1994] to improve upon past EM algorithms.

Chapter 2

Marked Markov Modulated Poisson Processes

The class of Marked Markovian Arrival Processes (MMAP or MMAP[K]) [He and Neuts, 1998] extends the expressivity of MAPs by associating each observed event to a mark. In the following, for simplicity, we will always consider marks to be discrete types, although they can be of any kind, e.g. continuous or multidimensional.

MMAPs allow the distinguishing of a finite set of arrival classes (i.e., multiple types of observed events) but also further exacerbate the complexity of the fitting problem, which was thus addressed with multi-step approaches able to fit selected moments and joint-moments of data while restraining the impact of non-linear optimization [Buchholz et al., 2010].

Marked Markov modulated Poisson processes (M3PP or MMMPP) [Casale et al., 2016, Carnevali et al., 2019], akin to MMAPs, are MMPPs where events have a discrete type which depends on the state of the hidden process. Specifically, an M3PP of order n is an MMPP of order n with arrivals belonging to a set C of m discrete classes. I.e. it can be considered as an MMPP with a multinomial distribution associated to each emission rate. In [Carnevali et al., 2019] M3PPs are used to study non-stationary intervals, as such a vector of initial state probabilities was introduced to the notation, which we maintain: $\boldsymbol{\eta} \in [0,1]^n$, a n-sized row vector. An M3PP is thus defined by the tuple $(\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C})$, where

- $\boldsymbol{\eta}$ is the n-sized vector of initial state probabilities such that $\boldsymbol{\eta}(0) = 1$ and $\boldsymbol{\eta}(i) = 0 \; \forall \, i \in S$ with $i \neq 0$ (due to the fact that, as discussed in Section 2.1, for the training set we consider sequences of events observed since the entrance in the initial, less aged, state of the system);

- $\mathbf{D_0}$ is the $n \times n$ upper-triangular transition rate matrix, i.e., $\mathbf{D_0}(i,j) = 0 \; \forall \, i,j \in S$ such that $i > j$, $\mathbf{D_0}(i,j) \geq 0 \; \forall \, i,j \in S$ such that $i < j$, and $\mathbf{D_0}(i,i) \leq 0 \; \forall \, i \in S$;

- $\mathbf{D_1}$ is the $n \times n$ emission rate matrix, i.e., $\mathbf{D_1}(i,j) = 0 \; \forall \, i,j \in S$ such that $i \neq j$ and $\mathbf{D_1}(i,i) \geq 0 \; \forall \, i \in S$;
- the sum of $\mathbf{D_0}$ and $\mathbf{D_1}$ is the generator matrix of a CTMC, i.e., $\sum_{j \in S}(\mathbf{D_0}(i,j) + \mathbf{D_1}(i,j)) = 0 \; \forall \, i \in S$;
- $\mathbf{C}$ is the $n \times m$ *event probability matrix* associating each state $i \in S$ and each class $c \in C = \{0, \ldots, m-1\}$ with the probability that an event of class c is emitted in state i, i.e., $\mathbf{C}(i,c) \in [0,1] \; \forall \, i \in S, \forall \, c \in C$, and $\sum_{c \in C} \mathbf{C}(i,c) = 1 \; \forall \, i \in S$.

In particular, it turns out that $\mathbf{D_0}(n,n) = -\mathbf{D_1}(n,n)$.

$$\mathbf{C} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{m,2} & \cdots & c_{m,n} \end{pmatrix}$$

Where all $c_{i,t}$ are non-negative $\forall i, t$ and $\sum_{t=1}^{m} c_{i,t} = 1 \; \forall i$

Note that an M3PP is a marked doubly stochastic Poisson process whose arrival rate and arrival type both depend on the state of a hidden CTMC, i.e., when the CTMC is in state i, events occur according to a Poisson process with rate λ_i, each event being of class $c \in C$ with probability $\mathbf{C}(i,c)$. If $\lambda_i = \lambda \; \forall \, i \in S$, then the M3PP is a Poisson process with rate λ whose arrival type still depends on the state of a hidden CTMC according to a discrete distribution. Fig. 2.1 shows an M3PP with 3 states and 4 classes of events, defined by the vector $\boldsymbol{\eta} = (1 \; 0 \; 0)$ of initial state probabilities and by the following matrices:

$$\mathbf{D_0} = \begin{pmatrix} -10 & 3 & 2 \\ 0 & -8 & 6 \\ 0 & 0 & -7 \end{pmatrix} \quad \mathbf{D_1} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 7 \end{pmatrix}$$
$$\mathbf{C} = \begin{pmatrix} 0.6 & 0.2 & 0.18 & 0.02 \\ 0.25 & 0.5 & 0.23 & 0.02 \\ 0.01 & 0.02 & 0.02 & 0.95 \end{pmatrix} \tag{2.1}$$

Note that the following chapter contains verbatim material from Carnevali et al. [Carnevali et al., 2019], © 2019, IEEE.

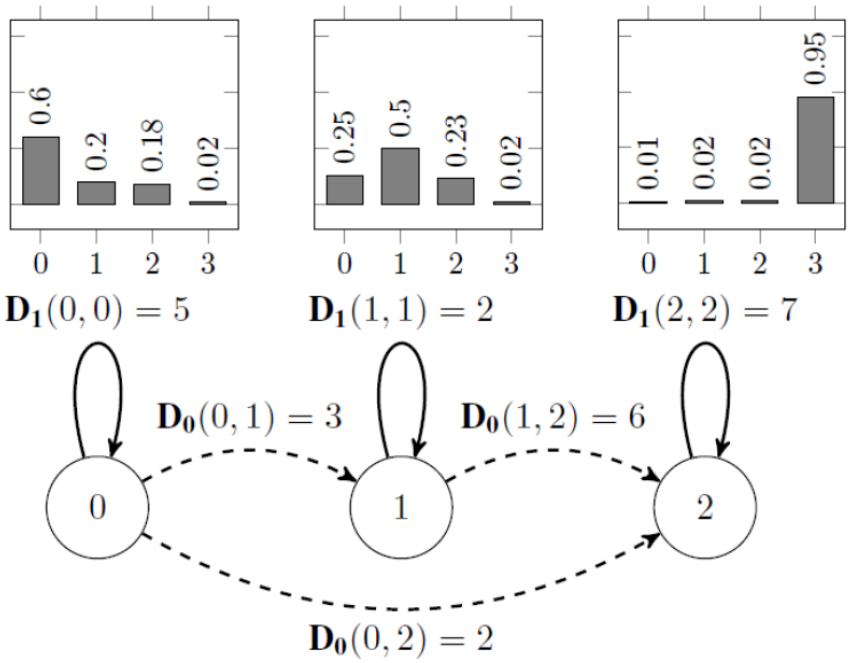

Figure 2.1: The M3PP defined by Eq. (2.1). States are represented as circles; unobservable transitions from state i to state j with $i \neq j$ are represented as dashed directed arcs labeled with the transition rate $\mathbf{D_0}(i,j)$; and, observable self-transitions from state i to state i are represented as solid directed arcs labeled with the emission rate $\mathbf{D_1}(i,i)$ and with the histogram of the probability density function of the classes of the events emitted in state i. © 2019, IEEE

An EM algorithm for M3PPs

In the following, we first define the forward and backward likelihood of a state of an M3PP, and the likelihood of a sequence of observed events. Then, we use these quantitative measures to illustrate the expectation step and the maximization step of the proposed EM algorithm.

Likelihood of an event sequence

Given an M3PP and an event sequence, we term *forward likelihood* of a state of the M3PP the *joint* probability that the event sequence is observed and that the system is in the considered state after such sequence of observations. Specifically, given an M3PP $\mathcal{M} = (\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C})$ and an event sequence $\mathcal{T} = \langle \omega_1, \ldots, \omega_K \rangle$ such that each event $\omega_k = \langle e_k, t_k \rangle$ consists of a type e_k and a time-stamp t_k $\forall\, k \in \{1, \ldots, K\}$ (as discussed in Section 2.1), we define the n-sized (row) *forward vector* $\mathbf{a}^{(k)}$ containing the forward likelihood of each state $i \in S$ with respect to the subsequence $\mathcal{T}^{1,k} = \langle \omega_1, \ldots, \omega_k \rangle$ $\forall\, k \in \{1, \ldots, K\}$, i.e. $\mathbf{a}^{(k)}(i) := P\{X(t_k) = i, \mathcal{T}^{1,k} \mid \mathcal{M}\}$ $\forall\, i \in S$. Each forward vector $\mathbf{a}^{(k)}$ can be computed as follows:

$$\mathbf{a}^{(k)} = \mathbf{a}^{(k-1)}\, \boldsymbol{\Pi}_{\tau_k}\, \mathbf{P_1} \circ \mathbf{C}(\cdot, e_k)^{\mathrm{T}} \tag{2.2}$$

where: $\mathbf{a}^{(0)} := \boldsymbol{\eta} := (1\ 0\ \cdots\ 0)$ is an n-sized row vector with the first element equal to 1 and the remaining ones equal to 0; $\tau_k := t_k - t_{k-1}$ is the time elapsed between the observations ω_{k-1} and ω_k $\forall\, k \in \{1, \ldots, K\}$, with $\tau_1 := t_1$ being the time elapsed from a fictitious observation ω_0 representing the start of system monitoring until the first observation ω_1; $\boldsymbol{\Pi}_{\tau_k}$ is the $n \times n$ matrix of transient state probabilities of the hidden CTMC, i.e., $\boldsymbol{\Pi}_{\tau_k}(i,j) := P\{X(\tau_k) = i \mid X(0) = j\}$ $\forall\, i \in S$, computed through the uniformization method [Stewart, 1994]; $\mathbf{P_1} = \mathbf{D_1}/\alpha$, with α being the uniformization rate [Stewart, 1994]; and, $\mathbf{C}(\cdot, e_k)$ is the n-sized column vector corresponding the column of the event probability matrix $\mathbf{C}$ indexed by the type e_k of the k-th event. In turn, transient state probabilities are computed by analyzing a DTMC with transition matrix $\mathbf{P} = \mathbf{I} + \mathbf{Q}/\alpha$ [Stewart, 1994], i.e., $\boldsymbol{\Pi}_{\tau_k}\mathbf{P_1} = \sum_{u=l_k}^{r_k} \psi(u, \alpha\, \tau_k)\mathbf{P_0}^{u-1}\mathbf{P_1}$, where: $\mathbf{I}$ is the $n \times n$ identity matrix; α is selected such that $\alpha \geq \max_{i \in S}\{|\mathbf{D_0}(i,i)|\}$; $\psi(u, \alpha\, \tau_k)$ is the probability that u events occur during the interval $[0, \tau_k)$ in a Poisson process with rate α; matrix $\mathbf{P}$ can be represented as $\mathbf{P} = \mathbf{P_0} + \mathbf{P_1}$ with $\mathbf{P_0} = \mathbf{I} + \mathbf{D_0}/\alpha$ and $\mathbf{P_1} = \mathbf{D_1}/\alpha$; and, l_k and r_k are the left and the right truncation points for the evaluation of the Poisson probabilities during the interval $[0, \tau_k)$, respectively [Fox and Glynn, 1988].

Furthermore, given an event sequence, we term *backward likelihood* of a state of the M3PP the probability of observing that trace starting from that state. Specifically, given an M3PP $\mathcal{M}$ and an event sequence $\mathcal{T} = \langle \omega_1, \ldots, \omega_K \rangle$, we define the n-sized (column) *backward vector* $\mathbf{b}^{(k)}$ containing the backward likelihood of each state

$i \in S$ with respect to $\mathcal{T}^{k+1,K} = \langle \omega_{k+1}, \ldots, \omega_K \rangle \; \forall k \in \{1, \ldots, K-1\}$, i.e., $\mathbf{b}^{(k)}(i) := P\{\mathcal{T}^{k+1,K} \mid X(t_k) = i, \; \mathcal{M}\}$. Each backward vector $\mathbf{b}^{(k)}$ can be computed as follows:

$$\mathbf{b}^{(k)} = \mathbf{\Pi}_{\tau_k} \mathbf{P_1} \circ \mathbf{F} \; \mathbf{b}^{(k+1)} \tag{2.3}$$

where $\mathbf{b}^{(K)} := \mathbf{1}^{\mathrm{T}}$ and $\mathbf{F}$ is an $n \times n$ matrix whose rows are all equal to the n-sized row vector $\mathbf{C}(\cdot, e_k)^{\mathrm{T}}$.

Finally, the likelihood $L(\mathcal{T}, \mathcal{M})$ of a sequence $\mathcal{T}$ of K observed events with respect to an M3PP $\mathcal{M}$ is the sum of the forward likelihoods of the states of $\mathcal{M}$:

$$L(\mathcal{T}, \mathcal{M}) := \mathbf{a}^{(K)} \, \mathbf{1}^{\mathrm{T}}. \tag{2.4}$$

E-step

Given a sequence of events $\mathcal{T} = \langle \omega_1, \ldots, \omega_k \rangle$ and the current estimates of the matrices $\mathbf{P_0}$, $\mathbf{P_1}$, and $\mathbf{C}$, the expectation step the EM algorithm evaluates the expected value of the number of: unobservable transitions between each pair of states, observable transitions between each pair of states, and event types emitted in each state. These expected values are stored in the $n \times n$, $n \times n$, and $n \times m$ matrices $\mathbf{X_0}$, $\mathbf{X_1}$, and $\mathbf{Y}$, respectively, and computed separately for each time interval of duration $\tau_k := t_k - t_{k-1}$ elapsed between the observations ω_{k-1} and $\omega_k \; \forall k \in \{1, \ldots, K\}$ (where ω_0 is a fictitious observation representing the start of system monitoring), exploiting the current forward vector $\mathbf{a}^{(k)}$ and backward vector $\mathbf{b}^{(k)}$. The evaluation yields the $n \times n$, $n \times n$, and $n \times m$ matrices $\mathbf{X_0}^{(k)}$, $\mathbf{X_1}^{(k)}$, and $\mathbf{Y}^{(k)}$, whose elements can be computed as follows for each state $i, j \in S$ of the hidden process and for each event type $c \in C$:

$$\mathbf{X_0}^{(k)}(i,j) = \sum_{u=l_k}^{r_k} \psi(u, \alpha \, \tau_k) \sum_{l=0}^{u-2} \mathbf{v}_l^{(k-1)}(i) \, \mathbf{P_0}(i,j) \, \mathbf{w}_{u-l-2}^{(k)}(j) \tag{2.5}$$

$$\begin{aligned} &\mathbf{X_1}^{(k)}(i,i) \\ &= \sum_{u=l_k}^{r_k} \psi(u, \alpha \, \tau_k) \, \mathbf{v}_{u-1}^{(k-1)}(i) \, \mathbf{P_1}(i,i) \, \mathbf{C}(i, e_k) \, \mathbf{b}^{(k)}(i) \end{aligned} \tag{2.6}$$

$$\mathbf{Y}^{(k)}(i,c) = \begin{cases} \mathbf{X_1}^{(k)}(i,i) & \text{if } e_k = c \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

where: $\mathbf{v}_u^{(k-1)} := \mathbf{a}^{(k-1)} \, \mathbf{P_0}^u$ is an n-sized row vector whose i-th element contains the conditional probability that the hidden process has performed u additional unobservable transitions during the time interval between ω_{k-1} and ω_k and that it has reached state i, given the observation of the event subsequence $\mathcal{T}^{1,k-1}$; $\mathbf{w}_u^{(k)} :=$

$\mathbf{P_0}^u \mathbf{P_1} \circ \mathbf{F}\ \mathbf{b}^{(k)}$ is an n-sized column vector whose i-th element contains the probability that the hidden process has performed u additional unobservable transitions during the time interval between ω_{k-1} and ω_k, that it has reached state i observing e_k as the type of the k-th observed event, and that the event subsequence $\mathcal{T}^{k+1,K}$ is then observed. Then, $\mathbf{X_0}$, $\mathbf{X_1}$, and $\mathbf{Y}$ are obtained summing $\mathbf{X_0}^{(k)}$, $\mathbf{X_1}^{(k)}$, and $\mathbf{Y}^{(k)}$, respectively, for each $k \in \{1, \ldots, K\}$, i.e., $\mathbf{X_0} = \sum_{k=1}^{K} \mathbf{X_0}^{(k)}$, $\mathbf{X_1} = \sum_{k=1}^{K} \mathbf{X_1}^{(k)}$, $\mathbf{Y} = \sum_{k=1}^{K} \mathbf{Y}^{(k)}$.

M-step

The maximization step of the EM algorithm derives the values of $\mathbf{P_0}$, $\mathbf{P_1}$, and $\mathbf{C}$ that maximize the likelihood of the considered trace computed by exploiting the expected values derived by the E-step. The new estimates for $\mathbf{P_0}$, $\mathbf{P_1}$, and $\mathbf{C}$ are represented by the $n \times n$, $n \times n$, and $n \times m$ matrices $\hat{\mathbf{X}}_\mathbf{0}$, $\hat{\mathbf{X}}_\mathbf{1}$, and $\hat{\mathbf{Y}}$, respectively, whose elements can be computed as follows for each state $i, j \in S$ and event type $c \in C$:

$$\hat{\mathbf{X}}_\mathbf{0}(i,j) = \frac{\mathbf{X_0}(i,j)}{\sum_{z=0}^{n-1} \mathbf{X_0}(i,z) + \mathbf{X_1}(i,i)} \tag{2.8}$$

$$\hat{\mathbf{X}}_\mathbf{1}(i,i) = \frac{\mathbf{X_1}(i,i)}{\sum_{z=0}^{n-1} \mathbf{X_0}(i,z) + \mathbf{X_1}(i,i)} \tag{2.9}$$

$$\hat{\mathbf{Y}}(i,c) = \frac{\mathbf{Y}(i,c)}{\sum_{w=0}^{m-1} \mathbf{Y}(i,w)} \tag{2.10}$$

EM algorithm

We consider a dataset of R event sequences, i.e. $T = \langle \mathcal{T}_1, \ldots, \mathcal{T}_R \rangle$. Each sequence $\mathcal{T}_r$ consists of K_r events, i.e., $\mathcal{T}_r = \langle \omega_1^r, \ldots \omega_{K_r}^r \rangle\ \forall r \in \{1, \ldots, R\}$. In turn, each event ω_k^r of $\mathcal{T}_r$ encodes a type e_k^r and a time-stamp t_k^r, i.e., $\omega_k^r = \langle e_k^r, t_k^r \rangle\ \forall k \in \{1, \ldots, K_r\}$, $\forall r \in \{1, \ldots, R\}$. The time elapsed between any two consecutive events ω_{k-1}^r and ω_k^r is denoted by $\tau_k^r := t_k^r - t_{k-1}^r\ \forall k \in \{1, \ldots, K_r\}, \forall r \in \{1, \ldots, R\}$, with $\omega_0^r = \langle e_0^r, t_0^r \rangle$ being a fictitious event representing the start of system monitoring $\forall r \in \{1, \ldots, R\}$.

Given such a dataset T, the EM procedure in Algorithm 2 iteratively performs the E-step (lines 10–22) and the M-step (lines 24–27) on all the traces of T, until the estimates of $\mathbf{P_0}$ and $\mathbf{P_1}$ converge with an error bound ϵ, i.e., the algorithm is halted when $\max\{\|\mathbf{P_0} - \mathbf{P_0}'\|, \|\mathbf{P_1} - \mathbf{P_1}'\|, \|\mathbf{C} - \mathbf{C}'\|\} \leq \epsilon$ (line 29), where $\langle \mathbf{P_0}, \mathbf{P_1}, \mathbf{C} \rangle$ and $\langle \mathbf{P_0}', \mathbf{P_1}', \mathbf{C} \rangle$ are the estimates produced by the last and by the second to last iteration, respectively, and $\|\cdot\|$ is the matrix norm 1. Moreover, as in the approach of [Rabiner, 1989], in each iteration on trace $\mathcal{T}_r$, the expected values encoded by $\mathbf{X_0}^{(k)}$, $\mathbf{X_1}^{(k)}$,

and $\mathbf{Y}^{(k)}$ are weighted by the inverse of the likelihood $L(\mathcal{T}_r, \mathcal{M})$ of the trace when summed to $\mathbf{X_0}$, $\mathbf{X_1}$, and $\mathbf{Y}$, respectively (lines 14, 20–21).

Finally, $\mathbf{D_0}$ and $\mathbf{D_1}$ are derived by inverting the relations $\mathbf{P_0} = \mathbf{I} + \mathbf{D_0}/\alpha$ and $\mathbf{P_1} = \mathbf{D_1}/\alpha$, respectively (lines 30–31).

Algorithm 1 Expectation Maximization

Input: $T = \langle \mathcal{T}_1, \ldots, \mathcal{T}_R \rangle$
Output: $\mathcal{M} = (\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C})$

1: $\epsilon \ll 1$
2: $\boldsymbol{\eta} \leftarrow (1\ 0\ \cdots\ 0)$
3: $\alpha \leftarrow \left(\min_{r \in \{1,\ldots,R\}, k \in \{1,\ldots,K_r\}} \{\tau_k^r\} \right)^{-1}$
4: Choose $\mathbf{P_0}$ and $\mathbf{P_1}$ such that $\mathbf{P_0}$ is upper-triangular, $\mathbf{P_1}$ is diagonal, and $\mathbf{P_0} + \mathbf{P_1}$ is stochastic
5: Choose $\mathbf{C}$ such that it contains non-negative elements
6: **procedure** EM($\mathcal{T}, \epsilon, \alpha, \mathbf{P_0}, \mathbf{P_1}, \mathbf{C}$)
7: $\quad \mathbf{X_0} \leftarrow \mathbf{0}_{n \times n}$, $\mathbf{X_1} \leftarrow \mathbf{0}_{n \times n}$, $\mathbf{Y} \leftarrow \mathbf{0}_{n \times m}$
8: $\quad$ **repeat**
9: $\quad\quad$ **for** $r = 1, \ldots, R$ **do**
10: $\quad\quad\quad$ Compute $L(\mathcal{T}_r, \mathcal{M})$ according to Eq. (2.4)
11: $\quad\quad\quad$ **for** $k \in \{1, \ldots, K_r\}$ **do**
12: $\quad\quad\quad\quad$ Compute $\mathbf{a}^{(k)}$ according to Eq. (3.2)
13: $\quad\quad\quad$ **end for**
14: $\quad\quad\quad$ **for** $k = K, \ldots, 1$ **do**
15: $\quad\quad\quad\quad$ Compute $\mathbf{b}^{(k)}$ according to Eq. (3.4)
16: $\quad\quad\quad\quad$ Compute $\mathbf{X_0}^{(k)}$ according to Eq. (3.12)
17: $\quad\quad\quad\quad$ Compute $\mathbf{X_1}^{(k)}$ according to Eq. (3.13)
18: $\quad\quad\quad\quad$ Compute $\mathbf{Y}^{(k)}$ according to Eq. (3.14)
19: $\quad\quad\quad\quad$ $\mathbf{X_0} \leftarrow \mathbf{X_0} + \mathbf{X_0}^{(k)} / L(\mathcal{T}_r, \mathcal{M})$
20: $\quad\quad\quad\quad$ $\mathbf{X_1} \leftarrow \mathbf{X_1} + \mathbf{X_1}^{(k)} / L(\mathcal{T}_r, \mathcal{M})$
21: $\quad\quad\quad\quad$ $\mathbf{Y} \leftarrow \mathbf{Y} + \mathbf{Y}^{(k)} / L(\mathcal{T}_r, \mathcal{M})$
22: $\quad\quad\quad$ **end for**
23: $\quad\quad$ **end for**
24: $\quad\quad$ $\mathbf{P_0}' \leftarrow \mathbf{P_0}$, $\mathbf{P_1}' \leftarrow \mathbf{P_1}$, $\mathbf{C}' \leftarrow \mathbf{C}$
25: $\quad\quad$ Compute $\hat{\mathbf{X}}_\mathbf{0}$ according to Eq. (2.8)
26: $\quad\quad$ Compute $\hat{\mathbf{X}}_\mathbf{1}$ according to Eq. (2.9)
27: $\quad\quad$ Compute $\hat{\mathbf{Y}}$ according to Eq. (2.10)
28: $\quad\quad$ $\mathbf{P_0} \leftarrow \hat{\mathbf{X}}_\mathbf{0}$, $\mathbf{P_1} \leftarrow \hat{\mathbf{X}}_\mathbf{1}$, $\mathbf{C} \leftarrow \hat{\mathbf{Y}}$
29: $\quad$ **until** $\max\{\|\mathbf{P_0} - \mathbf{P_0}'\|, \|\mathbf{P_1} - \mathbf{P_1}'\|, \|\mathbf{C} - \mathbf{C}'\|\} \leq \epsilon$
30: $\mathbf{D_0} \leftarrow \alpha(\mathbf{P_0} - \mathrm{diag}(\mathbf{P_0}\mathbf{1}^\mathrm{T} + \mathbf{P_1}\mathbf{1}^\mathrm{T}))$
31: $\mathbf{D_1} \leftarrow \alpha\, \mathbf{P_1}$
32: **end procedure**

2.1 Inference and prediction with M3PPs

In the following, we first describe a system characterized by a hidden state process and an observation process then we define the technique with which M3PPs can be used to identify the current state of the hidden process and predict the time when the state of maximum degradation is reache, given a sequence of observed events.

Hidden process

We consider a continuous-time process $\{X(t),\ t \geq 0\}$ with state space S, where $X(t) \in S$ is the state of the process at time t, e.g., aging in software systems, disease progression in human patients, execution of security attacks in information technology systems. The states of the process are regenerations (i.e., states that satisfy the Markov condition), so that the sequence of states visited during any time interval is a Markov renewal sequence. The sojourn time in each state $s \in S$ is characterized by a non-exponential Cumulative Distribution Function (CDF) $F^s_{\text{soj}} : I_{\text{soj}} \rightarrow [0,1]$ with $I_{\text{soj}} \subseteq [0,\infty)$, i.e., $F^s_{\text{soj}}(x) \neq 1 - \exp(-\lambda\, x)\ \forall\, \lambda \in \mathbb{R}_{>0}$. According to this, the considered phenomenon behaves like a Semi-Markov Process (SMP). Transient behavior is assumed to be an almost left-to-right process, with transitions from states of minor progression to states of major progression. Right-to-left transitions are still allowed to account for rejuvenation or maintenance, provided that a unique absorbing state $s_\star$ (representing the state of maximum progression) is always reached with probability 1, e.g., failure of a software system, death of a human patient, success of a security attack.

Observed process

The considered transient process cannot be directly observed (i.e., it is hidden). However, in each state $s \in S$, it emits typed events with (exponential or non-exponential) inter-event time CDF $F^s_{\text{int}} : I_{\text{int}} \rightarrow [0,1]$, where $I_{\text{int}} \subseteq [0,\infty)$. The type and the time of the emitted events can be directly observed, comprising an SMP $\{W(t),\ \forall\, t \geq t_1\}$ with discrete state space E, where t_1 is the time of the first event and $W(t) \in E$ is the type of the last event emitted during the time interval $[t_1, t]$. Therefore, if ω_1, …, ω_K is a sequence of K events, where each event $\omega_k = \langle e_k, t_k \rangle$ consists of a type $e_k \in E$ and a time-stamp $t_k \geq t_1\ \forall\, k \in \{1,\ldots,K\}$, then $W(t) = e_k\ \forall\, t \in [t_k, t_{k+1})$ and $\forall\, k \in \{1,\ldots,K\}$, with $t_{K+1} = \infty$.

The type of the events emitted in each state $s \in S$ of the transient process is sampled from a discrete Probability Density Function (PDF) $C^s : E \rightarrow [0,1]$. A conclusive event of type $e_\star \in E$ characterizes the sojourn in the absorbing state $s_\star \in S$. Specifically, in the absorbing state, the probability of a conclusive event is significantly larger than the probability of an event of any other type, i.e., $C^{s_\star}(e_\star)\ >>$

$C^{s_\star}(e)\ \forall\, e \in E \setminus \{e_\star\}$. On the other hand, in any state except for the absorbing state, the probability of a conclusive event is significantly lower than the probability of an event of any other type, i.e., $C^s(e_\star) << C^s(e)\ \forall\, s \in S \setminus \{s_\star\}, \forall\, e \in E \setminus \{e_\star\}$.

Offline learning

We aim at learning a model of the aging process from a dataset of event sequences $\omega_1 = \langle e_1, t_1 \rangle, \ldots, \omega_K = \langle e_K, t_K \rangle$ observed from the entrance in the initial (less aged) state at least until when the absorbing state $s_\star$ is reached. On the one hand, each event $\omega_k = \langle e_k, t_k \rangle$ such that $e_k \neq e_\star$ is not likely to be emitted in $s_\star$ and thus can be considered as unlabeled data. On the other hand, each event $\omega_k = \langle e_\star, t_k \rangle$ observed at the end of an observation sequence is likely to be emitted in $s_\star$ and thus can be considered as labeled data. According to this, the problem of deriving a model of the aging process can be regarded as an instance of *semi-supervised learning* [Zhu and Goldberg, 2009].

Online prediction

Given a model of the aging process and a sequence of events observed from an arbitrary time (coincident with or subsequent to the entrance in the initial state) until a time t, we predict when the state of maximum progression $s^\star$ will be reached, which comprises a *failure* event for the system. As usual in online failure prediction [Salfner et al., 2010], the forecast emitted at time t holds during a time window $I = [t + \Delta t_\mathrm{L}, t + \Delta t_\mathrm{L} + \Delta t_\mathrm{P}]$, where Δt_L is the *lead period* representing the time interval after which prediction is valid and Δt_P is the *prediction period* representing the time window duration. As shown in Fig. 2.2, a prediction is repetitively made at times $t_1, \ldots, t_H$ with *time step* Δt_S exploiting the sequence of events observed until that time, i.e., $t_{h+1} - t_h = \Delta t_\mathrm{S}\ \forall\, h \in \{1, \ldots, H-1\}$, and it is valid during the time windows $I_1, \ldots, I_H$, respectively, i.e., $I_h = [t_h + \Delta t_\mathrm{L}, t_h + \Delta t_\mathrm{L} + \Delta t_\mathrm{P}]\ \forall\, h \in \{1, \ldots, H\}$.

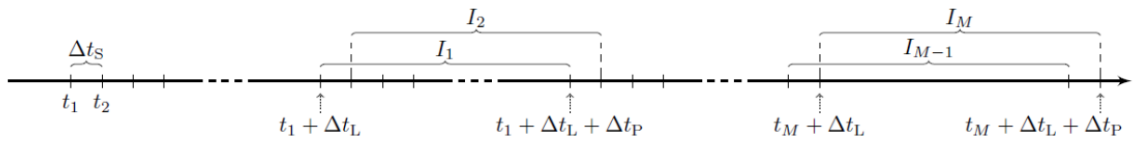

Figure 2.2: Prediction is repetitively issued at times $t_1, \ldots, t_M$ with time step Δt_S based on the event sequence observed until $t_1, \ldots, t_M$, respectively, and it is valid during the time windows $I_1 = [t_1 + \Delta t_\mathrm{L}, t_1 + \Delta t_\mathrm{L} + \Delta t_\mathrm{P}], \ldots, I_M = [t_M + \Delta t_\mathrm{L}, t_M + \Delta t_\mathrm{L} + \Delta t_\mathrm{P}]$, respectively. © 2019, IEEE

Given a sequence of observed events and the corresponding failure time $t_\star$, we say that the prediction emitted at time t_h and valid during the time window I_h, with $i \in \{1, \ldots, H\}$, is: *i*) a *true positive* if I_h contains $t_\star$ and it is actually predicted to contain $t_\star$; *ii*) a *false positive* if I_h does not contain $t_\star$ but it is predicted to contain $t_\star$; or, *iii*) a *false negative* if I_h contains $t_\star$ but it is predicted not to contain $t_\star$.

Given a data set made of N sequences of observed events, each sequence being associated with a failure time and a prediction sequence, let TP, FP, and FN be the total number of attained true positives, false positives, and false negatives, respectively. To evaluate performance, we define the following micro-averaged measures: *precision* $\mathrm{P} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ and *recall* $\mathrm{R} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$. Different P and R values can be obtained by using different thresholds to determine when an example is predicted as positive. From such values, the *precision-recall curve* can be computed and the area under that curve, termed *Area Under Precision-Recall Curve* (AUPRC), can be used to measure the prediction performance [Davis and Goadrich, 2006].

Current state classification

Given a sequence of events $\mathcal{T}_{1,k} = \langle \omega_1, \ldots, \omega_k \rangle$ observed during a time interval $[0, t]$, the objective of classification is to derive, for each hidden state $i \in S$, the probability $\gamma(i, t, \mathcal{T}^{1,k})$ that the system is in state $i \in S$ at time t given $\mathcal{T}^{1,k}$, with $\omega_h = \langle e_h, t_h \rangle$ $\forall\, h \in \{1, \ldots, k\}$ and $0 \leq t_1 \leq \ldots \leq t_k \leq t$:

$$\gamma(i, t, \mathcal{T}^{1,k}) := P\{X(t) = i \,|\, \mathcal{M},\ \mathcal{T}^{1,k}\}. \tag{2.11}$$

We assume not to know whether the start time of monitoring coincides with the beginning of the hidden process. Therefore, we consider a non-informative prior on the hidden state at the start of monitoring, estimating the initial probability of each state $i \in S$ as the ratio of the expected sojourn time in i and the sum of the expected sojourn times in each state, i.e., $\boldsymbol{\alpha}^{(0)}(i) := \mathrm{E}[\sigma_i] / \sum_{j \in S} \mathrm{E}[\sigma_j]$, where σ_i is the sojourn time in state i and $\mathrm{E}[\sigma_i]$ is its expected value. Then, the probability that the system is in state i right after the observation of event ω_k is represented by the i-th entry of the normalized forward vector $\hat{\boldsymbol{\alpha}}^{(k)}$, i.e., $\gamma(i, t_k, \mathcal{T}^{1,k}) = \hat{\boldsymbol{\alpha}}^{(k)}(i)\ \forall\, i \in \{0, \ldots, n\}$.

Finally, the n-sized row vector $\mathbf{d}_{t,\mathcal{T}^{1,k}}$ whose i-th entry contains the probability that the system is in state i at time t given the event sequence $\mathcal{T}^{1,k}$ observed during the interval $[0, t]$, i.e., $\mathbf{d}_{t,\mathcal{T}^{1,k}}(i) := \gamma(i, t, \mathcal{T}^{1,k})\ \forall\, i \in S$, can be derived as $\mathbf{d}_{t,\mathcal{T}^{1,k}} = \hat{\boldsymbol{\alpha}}^{(k)}\, \boldsymbol{\Pi}_{t-t_k} / (\mathbf{d}_{t,\mathcal{T}^{1,k}} \mathbf{1}^{\mathrm{T}})$, where $\boldsymbol{\Pi}_{t-t_k}$ is the $n \times n$ matrix of transient state probabilities of the hidden CTMC.

Failure time prediction

Given the vector $\mathbf{d}_{t,\mathcal{T}^{1,k}}$ of state probabilities at a time t conditioned to the observation of the event sequence $\mathcal{T}_{1,k}$, the objective of prediction is to determine whether the

state of maximum progression $s^\star := s_{n-1}$ will be reached at a time $t^\star$ within the time interval $I = [t + \Delta t_L, t + \Delta t_L + \Delta t_P]$, as illustrated in Section 2.1. To this end, at time t, we derive the Cumulative Distribution Function (CDF) of the remaining time $\tau^\star$ to the entrance in the failure state $s^\star$, which we denote as $\Phi(\tau) := P\{\tau^\star \leq \tau \mid \mathcal{M}, \mathcal{T}^{1,k}\}$. And, we predict that I contains $t^\star$ if the probability that $t^\star \in I$ is larger than a *prediction threshold* δ, i.e., if $\Phi(\Delta t_L + \Delta t_P) - \Phi(\Delta t_L) \geq \delta$.

The CDF $\Phi(\tau)$ can be derived from the CDFs $\Phi_i(\tau)$ of the remaining failure time conditioned to each possible state $i \in S$ at time t, which are weighted by probability of each state at that time, i.e., $\Phi(\tau) = \sum_{i \in S} \mathbf{d}_{t,\mathcal{T}^{1,k}}(i)\, \Phi_i(\tau)$, where $\Phi_i(\tau) := P\{\tau^\star \leq \tau \mid \mathcal{M}, \mathcal{T}^{1,k}, X(t) = i\}$ and $\mathbf{d}_{t,\mathcal{T}^{1,k}}(i)$ is the probability that the system is in state i at time t conditioned on the observed event sequence $\mathcal{T}^{1,k}$. In turn, given the set Z_i of the sequences of states $\langle i, \ldots, s^\star \rangle$ visited by the sequences of hidden transitions that bring the system from state i to state $s^\star$, $\Phi_i(\tau)$ can be derived $\forall\, i \in S$ from the CDF of the sojourn time in each state of Z_i, i.e., $\Phi_i(\tau) = \sum_{\rho \in Z_i} p_\rho \Phi_{i,\rho}(\tau)$, where p_ρ is the probability of the sequence of hidden transitions that visit the sequence of states ρ (obtained as the product of transition probabilities) and $\Phi_{i,\rho}(\tau)$ is the CDF of the sum of sojourn times in states of Z_i except for $s^\star$, i.e., $\Phi_{i,\rho}(\tau) := P\{\sum_{j \in Z_i \setminus \{s^\star\}} \sigma_j \leq \tau \mid \mathcal{M}, \mathcal{T}^{1,k}, X(t) = i\}$. Note that, due to the fact that the sojourn time in each hidden state is exponentially distributed, the remaining sojourn time in state s_i has an exponential distribution with the same rate of the sojourn time σ_i. In the present experimentation, $\Phi_{i,\rho}(\tau)$ is derived through Monte Carlo simulation $\forall\, i \in S, \forall\, \rho \in Z_i$.

2.2 Experimentation

The proposed approach lends itself to application in a variety of scenarios where software reliability is supported by on-line diagnosis and prediction. As a common trait, in all these cases, effectiveness of analysis results largely depends on the statistics of observable data, specifically on its diversity along different phases of operation. To characterize this dependency, we report experimental results on a synthetic dataset generated by stochastic simulation of the attack tree model of the Stuxnet case [Kriaa et al., 2012] using stochastic durations proposed in [Arnold et al., 2014b]. To this end, the model is enriched with a controlled statistics of types and inter-arrival times of observable events in each intermediate step of the attack.

To illustrate sensitivity to statistical characteristics of observed data, multiple datasets are generated by controlled variants of the model with different coefficient of variation of durations and different emission rate of events in each state (and thus different confusion among events during the steps of the attack). Each dataset is exploited to learn three different models, which exploit only the arrival times of untyped observations (as would result from the application of [Buchholz, 2003]), or the types of time-stamped observations, under the assumption that the arrival

rate of observations of any type is the same in every state (as would result from the application of [Liu et al., 2015]), or the combination of the two aspects, relaxing the assumption of state-independent arrival times of observations of any type. The effectiveness of the different learned models in predicting the completion time of the attack is evaluated in terms of false and missed alarms, as discussed in the formulation of Section 2.1.

Experiments were performed through a Java implementation of the proposed approach, exploiting the SIRIO library [SIRIO Library, 2018] of the ORIS tool [Biagi et al., 2017, ORIS Tool, 2019] for the computation of Poisson probabilities and for stochastic simulation of the attack model.

Experimental setup

Attack scenario

The attack scenario is specified through the formalism of Stochastic Time Petri Nets (STPNs) [Vicario et al., 2009], which support representation of concurrent stochastic systems. Specifically, places (depicted as circles) represent logical conditions encoded by tokens (depicted as dots); transitions (depicted as bars) represent activities; preconditions and postconditions (depicted as directed arrows from places to transitions and viceversa, respectively) control the enabling of transitions and the token moves at their firing: a transition becomes enabled when all its input places are not empty, sampling a time to fire from a given (possibly non-Markovian) distribution; then, the transition with minimum time to fire is selected as the next event, removing a token from each input place and adding a token to each output place. Enabling and update functions can be used to restrict the enabling of transitions by constraints on token counts and to perform additional updates of token counts after transition firings, respectively. A formal definition of STPN syntax and semantics is reported in [Vicario et al., 2009].

Fig. 2.3 shows the STPN model of the attack by the StuxNet malicious computer worm [Kriaa et al., 2012], targeting Supervisory Control and Data Acquisition (SCADA) systems that use Microsoft Windows and Programmable Logic Controllers (PLCs) [Kriaa et al., 2012]. In the first of two phases, the worm infiltrates the business corporate network through an infected removable drive (in Fig. 2.3, this step is represented by transition `injectViaUsb`), and it updates itself to the last version either by establishing a peer-to-peer (P2P) communication with a Remote Procedure Call (RPC) server (`p2pComm`) or by directly connecting to a Control and Command (C&C) server (`ccServerCmd`). Then, the worm propagates through the network infecting workstations by means of removable media (`removableMedia`), network shared resources (`networkShares`), the print spooler service exploit (`printServerVuln`), the Windows server service exploit (`serviceRpcVuln`), or WinCC

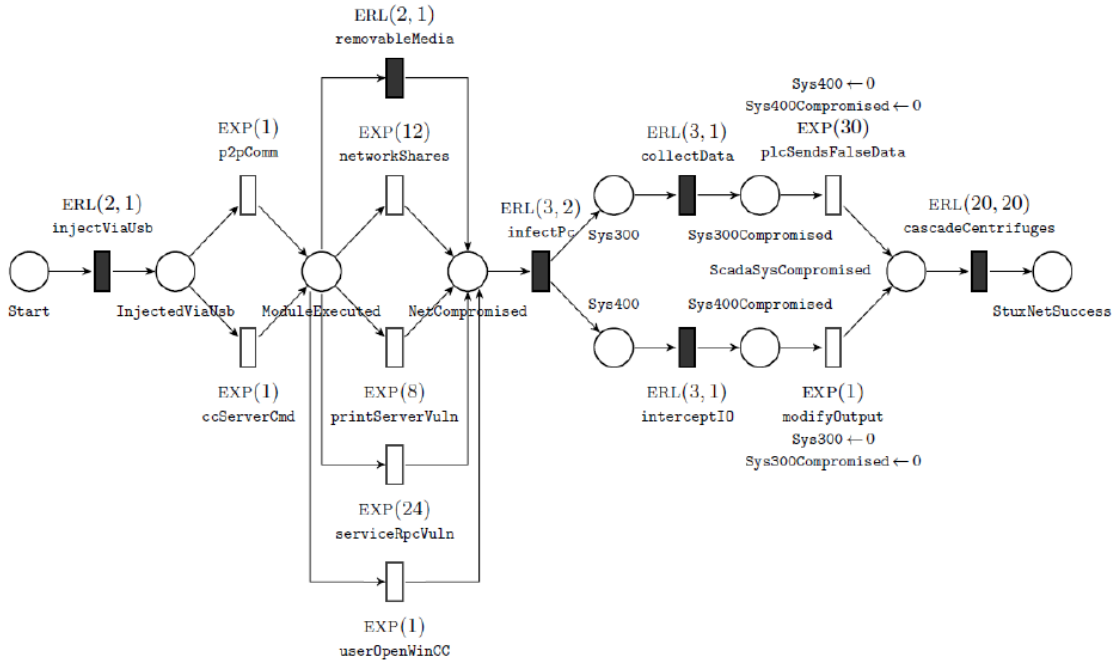

Figure 2.3: STPN model of the StuxNet attack. Thick white bars and thick black bars represent transitions with exponential distribution and with general (i.e., non-exponential) distribution, respectively. Specifically, $\text{EXP}(\lambda)$ and $\text{ERL}(k, \lambda)$ denote an exponential distribution with rate λ and an Erlang distribution with shape k and rate λ, respectively. Update functions are annotated next to transitions as "`place` $\leftarrow$ *expression*". Times are expressed in days. © 2019, IEEE

project files (`userOpenWinCC`). In the second phase of the attack, the worm waits until it infects a pc in the process control network (`infectPc`), typically via a removable drive, and then it compromises the SCADA system by modifying the PLC function blocks, with different steps depending on the CPU type: for 300–series systems, the worm collects data for a long time (`collectData`) before sending false data to the physical infrastructure (`plcSendsFalseData`); for 400–series systems, the worm intercepts input and output signals of the PLC (`interceptIO`) and modifies the output signals by sending false data (`modifyOutput`). Finally, the worm compromises the physical infrastructure (`cascadeCentrifuges`), which consisted of centrifuges for separation of nuclear material in the attack to Iranian nuclear enrichment facilities.

The model of the StuxNet attack shown in Fig. 2.3 assumes the stochastic temporal parameters reported in [Arnold et al., 2014b], except for the rate of the Erlang distribution of transitions `infectPc`, `collectData`, and `interceptIO`, which has been increased from 0.2 to 2, from 0.1 to 1, and from 0.1 to 1, respectively, not to make the final state too easily identifiable.

The model is enriched with a set of observable events detailed in Table 2.1, de-

rived from the description of the StuxNet attack in [Kriaa et al., 2012]. Each event type (column 1) can be observed in one or more steps of the attack, identified by an enabling condition referred to the STPN model of Fig. 2.3 (column 2), and it is associated with an exponential arrival time CDF (column 3), defined on the basis of the structure of the attack and the sojourn times in its steps not to make the attack too easily detectable. On the one hand, the SCADA system recurrently emits events, in any step of the attack. On the other hand, the StuxNet worm also emits events, which can be of the same type as those emitted by the SCADA system, and which can be either recurrent or one-shot. For instance, `port80Comm` is emitted by the SCADA system with rate 0.1 in any step of the attack, and by the StuxNet worm with rate 4 in the second step only (i.e., after infiltrating the network through an infected removable media, before updating to the last version).

Note that a one-shot event could be represented through an extension of M3PPs as an observable transition from a state where such event can be observed to a counterpart state where such event cannot be observed. Nevertheless, experimental results discussed in Section 3.2 show that the present model achieves sufficient accuracy for the context of use.

Table 2.1: Observable events enriching the attack model of Fig. 2.3. Rates of exponential distributions are expressed in days. © 2019, IEEE

System events		
Recurrent events		
Event type	**Enabling condition**	**Inter-arrival time CDF**
port80Comm	TRUE	$\text{EXP}(0.1)$
rpcOp	TRUE	$\text{EXP}(0.5)$
newLib	TRUE	$\text{EXP}(0.05)$
degradation	TRUE	$\text{EXP}(0.2)$
sendData	TRUE	$\text{EXP}(0.1)$
checkWindow	TRUE	$\text{EXP}(0.1)$
checkAdmin	TRUE	$\text{EXP}(0.1)$
databaseConn	TRUE	$\text{EXP}(0.3)$
StuxNet events		
Recurrent events		
Event type	**Enabling condition**	**Inter-arrival time CDF**
rpcOp	InjectedViaUsb $== 1$	$\text{EXP}(3)$
port80Comm	InjectedViaUsb $== 1$	$\text{EXP}(4)$
rpcOp	ModuleExecuted $== 1$	$\text{EXP}(5)$
newLib	NetCompromised $== 1$	$\text{EXP}(0.5)$
sendData	Sys300Compromised $== 1$ $\|\|$ Sys400Compromised $== 1$	$\text{EXP}(10)$
degradation	ScadaSysCompromised $== 1$	$\text{EXP}(2)$
attackSuccess	StuxNetSuccess $== 1$	$\text{EXP}(50)$
One-shot events		
Event type	**Enabling condition**	**Single-arrival time CDF**
checkWindow	Start $== 1$	$\text{EXP}(4)$
checkAdmin	Start $== 1$	$\text{EXP}(4)$
databaseConn	ModuleExecuted $== 1$	$\text{EXP}(10)$

Synthetic datasets

We generate datasets of event sequences observed during the attack scenario of Section 2.2. Specifically, stochastic simulation of the attack model of Fig. 2.3 is performed, yielding the sequence of states visited in each simulation run and the corresponding sojourn times. For each state, a sample is extracted from the arrival time CDF of each event specified in Table 2.1 that can be observed in that state, selecting the event with minimum arrival time as the next observed event. This step is iterated, under the assumption that one-shot events can be observed only once, until the sum of the observed inter-event times is larger then the sojourn time. In so doing, a sequence of typed and time-stamped events is obtained for each performed simulation run.

We generate three datasets to evaluate sensitivity of prediction accuracy to stochastic parameters of the attack scenario: *i*) a dataset aimed at showing performance

prediction on the StuxNet attack illustrated in Section 2.2, which we term *SN-dataset*; *ii*) a variant of the SN-dataset aimed at showing sensitivity to the statistics of sojourn times in hidden states, which we term *ST-dataset*; and, *iii*) a variant of the SN-dataset aimed at showing sensitivity to the statistics of event emission rates, which we term *ER-dataset*. Specifically:

- **SN-dataset.** The dataset is generated exploiting the attack model of Fig. 2.3 and the observable events of Table 2.1.
- **ST-dataset.** The dataset is generated exploiting a variant of the model of Fig. 2.3 obtained by halving the coefficient of variation of the sojourn time distribution in each state, while maintaining the same expected value. According to this, each Erlang CDF with shape k and rate λ is replaced by an Erlang CDF with shape $4k$ e rate 4λ (note that an exponential CDF is regarded as an Erlang CDF with shape 1 and the same rate, thus being replaced by an Erlang CDF with shape 4 and quadrupled rate). By compacting sojourn time distributions around their respective mean value, this dataset is expected to make the different steps of the attack more easily detectable.
- **ER-dataset.** The dataset is generated exploiting a variant of the observable events of Table 2.1 obtained by making the sum of rates of the StuxNet events equal to 4 in every step of the attack (e.g., in the attack step identified by the enabling condition `InjectedViaUsb` $==$ 1, the rates of `rpcInstall` and `port80Comm` are both turned into 2). In so doing, this dataset is expected to make the time elapsed between consecutive observed events less informative.

Each generated dataset consists of 1100 traces with average duration equal to 7.6 days and average number of observed events equal to 25. Each dataset is split into a training set of 1000 traces and a test set of 100 traces. Each training trace starts at the beginning of the attack and terminates with its success. Each test trace terminates with the attack success as well, but it starts at a random time after the beginning of the attack. In so doing, assuming a prior distribution based on expected sojourn times is non-informative. In the applicative perspective, this assumption corresponds to considering a system that is able to detect ongoing attacks, not necessarily at their beginning.

Learned models

For each generated dataset, we learn three models: *i*) an M3PP; *ii*) a variant of M3PPs that does not exploit observed event types, which we term *No Event Type M3PP* (NET-M3PP); and, *iii*) a variant of M3PPs that does not exploit the time elapsed between consecutive observed events of any type, which we term *No Inter-Event Time M3PP* (NIET-M3PP). Specifically:

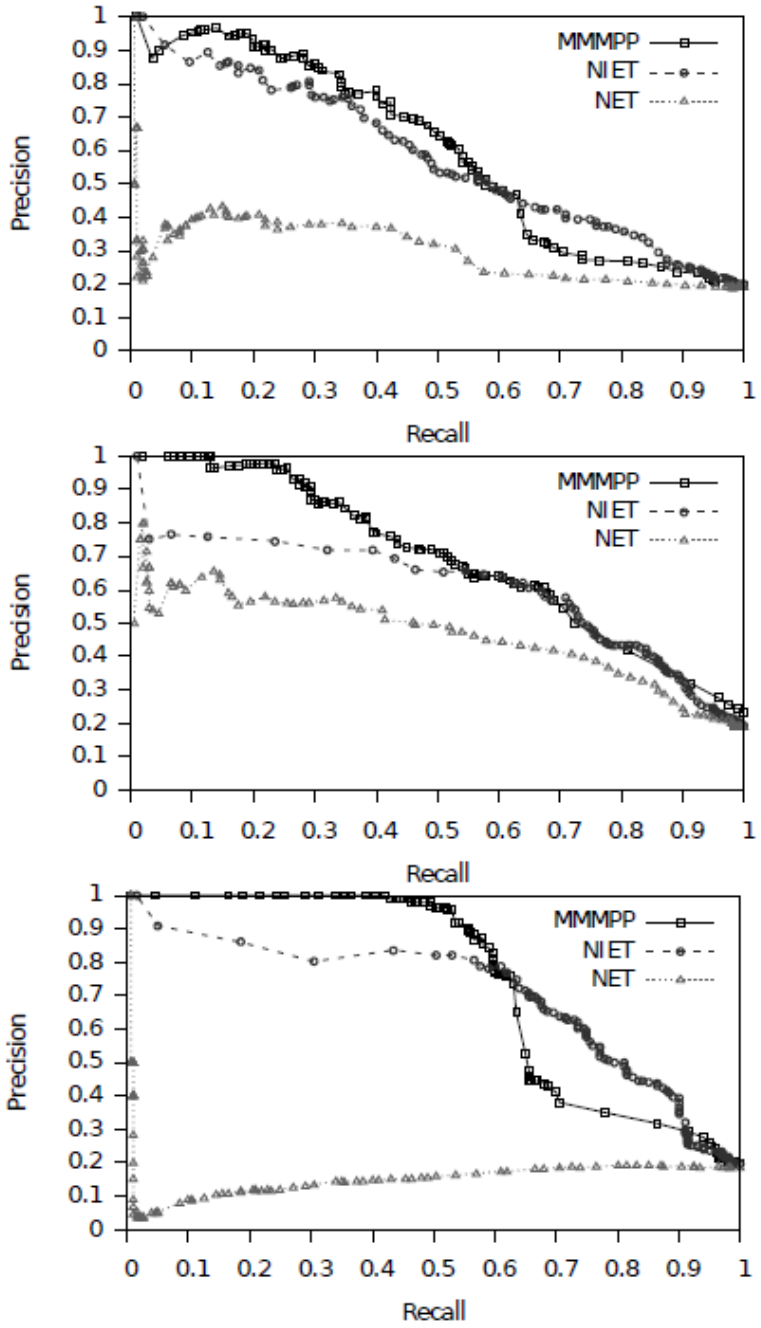

Figure 2.4: Prediction performance of the learned models on the SN-dataset (left), the ST-dataset (center), and the ER-dataset (right). © 2019, IEEE

- **M3PP.** The model is learned through the EM procedure of Algorithm 2. In particular, $\mathbf{P_0}$ and $\mathbf{P_1}$ are selected randomly so that the overall average sojourn time in the hidden states, except for the absorbing state $s^\star$, is equal to the average duration of the event sequences observed in the training set until when $s^\star$ is reached. And, $\mathbf{C}$ is initialized with the observed probability (i.e., computed over the training set) that each event type is emitted by the SCADA system in each state.

- **NET-M3PP.** The model is an M3PP variant where the set of event types consists of the conclusive event $e^\star$ and a non-conclusive event e. The model is learned by a variant of Algorithm 2: $\mathbf{P_0}$ and $\mathbf{P_1}$ are initialized as for M3PPs; $\mathbf{C}$ is initialized so that *i*) in every state except for $s^\star$, the probability of e and $e^\star$ is 1 and 0, respectively, and *ii*) in $s^\star$, the probability of e is equal to the observed probability p that a non-conclusive event is emitted by the SCADA system in $s^\star$, and the probability of $e^\star$ is equal to $1 - p$; and, $\mathbf{C}$ is never modified after initialization (matrices $\mathbf{Y}^{(k)}$, $\mathbf{Y}$, and $\hat{\mathbf{Y}}$ are not computed).

  Note that distinguishing the conclusive event facilitates failure prediction with respect to a direct application of the method of [Buchholz, 2003] which does not exploit event types.

- **NIET-M3PP.** The model is an M3PP variant where the diagonal elements of $\mathbf{D_1}$ are equal to each other, except for the one in the last row. According to this, while events of different type can be emitted with different rate in each state, their sum is constrained to be the same in every state, except for $s^\star$. The model is learned by a variant of Algorithm 2: $\mathbf{P_0}$, $\mathbf{P_1}$, and $\mathbf{C}$ are initialized as for M3PPs, under the assumption that $\mathbf{P_1} = \text{diag}(p, \ldots, p, q)$; $\mathbf{P_1}$ is never modified after initialization ($\mathbf{X_1}$ and $\hat{\mathbf{X}}_\mathbf{1}$ are not computed); and, $\hat{\mathbf{X}}_\mathbf{0}(i,j)$ is computed as $\hat{\mathbf{X}}_\mathbf{0}(i,j) = (1 - \mathbf{P_1}(i,j))\mathbf{X_0}(i,j) / \sum_{z=0}^{n-1} \mathbf{X_0}(i,z)$ $\forall\, i, j \in S$ (see line 24 of Algorithm 2).

  Note that assigning a different inter-arrival rate to the conclusive event with respect to the other events facilitates failure prediction with respect to a direct application of the method of [Liu et al., 2015] which does not exploit the information encoded by the inter-arrival times of events of any type.

Experimental results

Each training set is used to learn the three models defined in Section 2.2, while the corresponding test set is used to asses the model accuracy in predicting the completion time of the attack, as illustrated in Section 2.1. In particular, we consider the following parameter values: $n = 10$ hidden states, time step $\Delta t_\text{S} = 0.5$, lead period $\Delta t_\text{L} = 0.1$, and prediction period $\Delta t_\text{P} = 1.0$, where times are expressed in days.

Moreover, for each model learned on each dataset, we compute the minimum and the maximum value of the output probability (i.e. the probability that the state of maximum progression is reached at a time contained in the prediction window), and we discretize such interval in 100 slots to obtain the precision-recall curves.

Fig. 2.4-left shows the results obtained on the SN-dataset. The M3PP approach largely outperforms the NET-M3PP approach while performing slightly better than the NIET-M3PP approach, showing that the information encoded in event types is far more important than the one included in event timing on this specific dataset. Specifically, the M3PP approach achieves an AUPRC equal to 0.602, slightly larger than the one obtained by the NIET-M3PP approach (0.588), and far above that of the NET-M3PP (0.307).

Fig. 2.4-center shows the results obtained on the ST-dataset. On this dataset, the M3PP approach largely outperforms both the competitors, showing that reducing the CV of the sojourn times in the steps of the attack improves the performance of the approaches that heavily rely on the information encoded by event timing. In fact, the gain with respect to the SN-dataset is much more evident for the M3PP and the NET-M3PP approaches. Overall, the M3PP approach achieves an AUPRC equal to 0.704, slightly larger than the one obtained by the NIET-M3PP approach (0.616), and far above that of the NET-M3PP (0.469).

Fig. 2.4-right shows the results on the ER-dataset. In this setting, the performance of the NET-approach heavily drops, due to the fact that the sum of the event rates is the same in each state, which makes it hard to identify the steps of the attack based on event timing only. As in the previous cases, the M3PP-approach achieves the best result. Specifically, the M3PP approach achieves an AUPRC equal to 0.746, slightly larger than the one obtained by the NIET-M3PP approach (0.717), and far above that of the NET-M3PP (0.156).

Chapter 3

Marked Markov Modulated Compound Poisson Processes

In this chapter we introduce a model that extends marked MMPPs along two different directions: we introduce the combination of a MMPP with an ordinary compound Poisson process, which we refer to as Marked Markov-modulated Compound Poisson Processes (M3CPPs), and we define restrictions to the structure of the infinitesimal generator to model non-markovian sojourn times with little added complexity.

We report notes on Long short-term Memory Recurrent Neural Networks (Chapter 3), which are used to obtain a comparison for the experiments.

In the following, we denote vectors and matrices with bold lowercase letters and bold uppercase letters, respectively.

M3CPP

M3CPPs extend M3PPs by adding to the arrivals a continuous component, the *magnitude* of the events, which depends on the discrete type of the arrival and on the state of the hidden process.

M3CPPs are likewise extensions of Markov-modulated Compound Processes (MMCPP) [Okamura et al., 2007] where discrete types are added to the arrivals, which already have a magnitude.

An M3CPP is thus defined by the tuple $(\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C}, \mathbf{G}(x))$, where $\mathbf{G}(x)$ is the $n \times m$ *probability density function matrix* associating each state $i \in [1, n]$ and each event type $t \in C$ with a probability density function (p.d.f) that an event of type t and emitted in state i has magnitude $x \in \mathbb{R}$.

$$\mathbf{G}(x) = \begin{pmatrix} g_{1,1}(x) & \cdots & g_{1,m}(x) \\ \vdots & \ddots & \vdots \\ g_{n,1}(x) & \cdots & g_{n,m}(x) \end{pmatrix}$$

It is important to note that while marked MMPP technically already cover any combination of types, the restriction to only events characterised by one or more discrete dimensions and a single continuous dimension allows for the efficient learning of the parameters through the separation in $\mathbf{D_0}, \mathbf{D_1}$ and G and permits the possible "sharing" of parameters between different states akin to Hierarchical Hidden Markov Models (HHMM) [Bui et al., 2004].

It is well known that an exponential distribution cannot approximate well a non-markovian one. If the hidden process we want to learn has a state space of size n and contains non-markovian sojourn times then an infinitesimal generator of size $n \times n$ will not be able to approximate it well. It is intuitive that increasing the dimension of $\mathbf{D_0}$ may produce better results, yet it brings a problem: increasing the size of the $n \times n$ matrix of a single unit results in an increase of $2n$ free parameters, with the resulting higher computation costs and data volume requirements. Continuous phase-type distributions (CPHs) [Neuts, 1981] are able, given a sufficient number of states, to approximate any distribution and can be exploited to limit the exponential increase in complexity. We substitute the exponential sojourn times in $\mathbf{D_0}$ for Acyclic CPHs (ACPH) in Canonical Form 1 [Cumani, 1982]: such change can be reached by restricting $\mathbf{D_0}$ to the following structure:

$$\mathbf{D_0} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ & \boldsymbol{\alpha}_2 * \mu_{1,2} & & \boldsymbol{\alpha}_n * \mu_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}_n \\ \boldsymbol{\alpha}_1 * \mu_{n,1} & \boldsymbol{\alpha}_2 * \mu_{n,2} & & \end{pmatrix}$$

$$\mathbf{A}_k = \begin{pmatrix} -\mu^k_{1,1} & \mu^k_{1,2} & 0 & \cdots & 0 \\ 0 & -\mu^k_{2,2} & \mu^k_{2,3} & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu^k_{r-1,r-1} & \mu^k_{r-1,r} \\ 0 & 0 & \cdots & 0 & -\mu^k_{r,r} \end{pmatrix}$$

The resulting $\mathbf{D_0}$ is an $(n * r) \times (n * r)$ matrix where n is the number of CPHs or *macrostates*, r is the number of *inner states* in each CPH and $\boldsymbol{\alpha_i}$ is a r-sized row vector of initial probabilities of the i-th CPH. To note is that no changes happen to the arrival process: each phase of the ACPH A_i is still associated to the same emission rate λ_i and $\mathbf{D_1}$ is still a $n \times n$ matrix. All $\mu_{i,j}$ and $\mu^k_{i,j}$ are non-negative, $\mu^k_{i,i} = \mu^k_{i,i+1} + \lambda_k \; \forall i \neq r$ and $\mu^k_{r,r} = \sum_{j \neq k} \mu_{k,j} + \lambda_k$. With this structure, adding an inner state to better fit a

sojourn distribution requires only 2 additional parameters, one for the connection and one for the initial probability of the CPH. The partitioning of $\mathbf{D_0}$ can also be seen from the point of view of an infinitesimal generator of size $n * r \times n * r$ with restrictions imposed to reduce the complexity: with regards to $\mathbf{D_0}$ only, the number of free parameters goes from $(n * r - 1)^2$ to $n * (n * r - 1)$.

Background on Long Short-Term Memory

In Section 3.2 we report results of M3CPPs applied to diagnosis and prediction; to provide a benchmark we compare them to a successful machine learning technique used for time series, Long short-term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], a Recurrent Neural Network (RNNs), we report here some details.

LSTMs are noteworthy due to their ability to maintain long term memory, overcoming the vanishing gradient problem of RNNs and consequently have been useful for learning sequences containing long term correlations of unknown length. LSTMs are well suited to the tasks of classification and prediction on discrete time series and have obtained success in several fields other than HPC e.g. activity recognition [Ordóñez and Roggen, 2016], anomaly detection [Malhotra et al., 2015], neural machine translation [Wu et al., 2016]. A common LSTM memory block is composed of four units: a *cell*, which maintains the dependencies between the observed events; an *input gate* I_G which controls the intensity of changes to the cell due to new values; an *output gate* O_G which controls the extent to which the cell state participates to the output activation; a *forget gate* F_G that determines how much of the cell state is preserved for the next time step (the forget gate was introduced in [Gers et al., 1999]). The three gates prevent the cell from being affected by irrelevant inputs. In Figure 3.1 the scheme of an LSTM memory block can be seen. The cell state is marked in red, the crosses represent multiplications.

The equations for the *forward pass* (production of an output vector from an input vector) of an LSTM block with a forget gate are:

- $f_G(t) = \sigma_g(W_f x(t) + U_f h(t-1) + b_f$
- $i_G(t) = \sigma_g(W_i x(t) + U_i h(t-1) + b_i$
- $o_G(t) = \sigma_g(W_o x(t) + U_o h(t-1) + b_o$
- $\tilde{C}(t) = \sigma_h(W_c x(t) + U_c h(t-1) + b_c$
- $C(t) = f_G(t) \circ C(t-1) + i_G(t) \circ \tilde{C}(t)$
- $h(t) = o_G(t) \circ \sigma_h(C(t))$

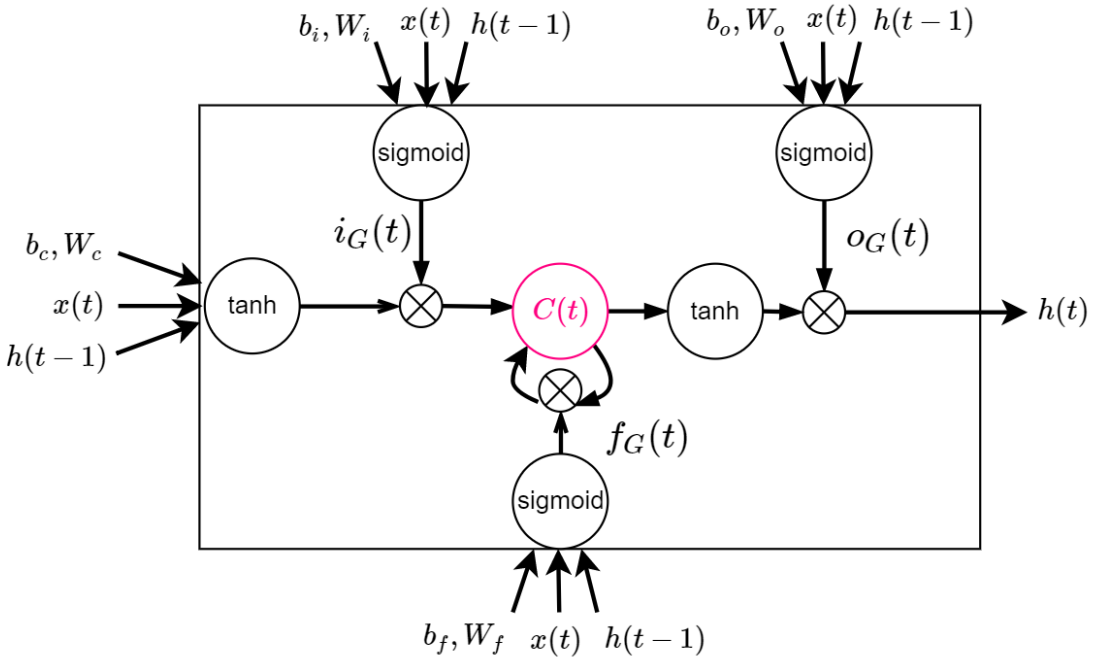

Figure 3.1: Overview of a memory block in a LSTM-RNN.

Where $\circ$ is the Hadamard product, σ_g, σ_h are the sigmoid and hyperbolic tangent activation functions and W_-, U_- and b_- are respectively the weight and biases vectors, which need to be learned during training, the rest are:

- $x(t) \in \mathbb{R}^d$: input vector to the LSTM unit at time t
- $f_G(t) \in \mathbb{R}^h$: forget gate's activation vector at time t
- $i_G(t) \in \mathbb{R}^h$: input gate's activation vector at time t
- $o_G(t) \in \mathbb{R}^h$: output gate's activation vector at time t
- $h(t) \in \mathbb{R}^h$: hidden state (output) vector at time t
- $\tilde{C}(t) \in \mathbb{R}^h$: cell input activation vector at time t
- $c(t) \in \mathbb{R}^h$: cell state vector at time t

Given a time series, at the t-th iteration each gate activation function applies a non-linear transformation (sigmoid or tanh) to the linear combination of the current input $x(t)$ and previous hidden state $h(t-1)$ with the weight and bias vectors; the hidden state $h(t)$ is emitted as output. A *backward pass* can be applied to modify the weights and biases based on the output to conduct learning.

Typically multiple memory blocks are organized in a set, termed *layer*, where they all receive the same input $x(t)$. Layers can then be stacked, receiving as an input the outputs of the blocks of the preceding layer, to learn higher level temporal features.

Time discretization Deep learning models generally deal with time series whose values occur equidistantly [Box et al., 2015] that is, from period-based approaches. However applications like HPC, where data is not sampled at discrete intervals, but rather produced irregularly are also common. It is known that simply adding the time of an event as a feature in input to the model does not produce optimal results.

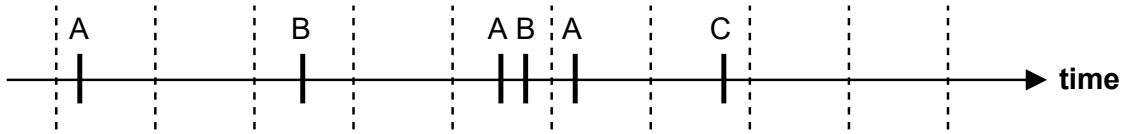

Figure 3.2: Discretization of event-driven time series into fixed length time slots.

A straightforward approach to incorporate an event's continuous timestamp into a discrete time series is to identify a fixed length with which divide the continuous time in intervals [Wei et al., 2002]. Figure 3.2 shows a time series discretized in time slots. If no events are present in a time slot, a special event representing "silence" is considered. This approach hides some problems:

- there is always a non-null probability that more than one event are contained in the same time slot
- time resolution is weakened since it is no longer known when in the time slot the event happened

While a simple solution would be to reduce the discretization step (and so the slots widths) to minimize co-occurrence, small time steps incur problems when long sequences of "silences" are present. Even LSTM, which is able to maintain a long memory, eventually suffers from the vanishing gradient problem. In general, if the inter-event times vary greatly, for example when the hidden process modes have different emission rates as is the case in failure prediction, identifying the correct discretization step has a significant impact on the results.

3.1 Parameter Estimation for M3CPP

Expectation Maximization Algorithm

In this section we present the parameter estimation procedures for M3CPPs. We first remind the Expectation Maximization (EM) principles and then develop the concrete EM algorithm. As prerequisite, we define the forward and backward likelihood of a state of an M3CPP. While by themselves each step is not novel, being largely based on the M3PP one, we are effectively introducing in the formulation of our EM algorithm the key details to create hierarchies of states and of event types needed for the construction of a model that is continuous time and hierarchical in both fields.

Consider the sequence of events $\mathcal{T} = \langle \omega_1, \ldots, \omega_K \rangle$ such that each event $\omega_k = \langle t_k, e_k, x_k \rangle$ consists of a continuous time-stamp t_k, a type e_k and a intensity $x_k \; \forall k \in \{1, \ldots, K\}$. Given an M3CPP $\mathcal{M} = (\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C}, \mathbf{G}(x))$ with n macrostates and r inner states per macrostate we want to find the parameters that maximize the likelihood of producing the sequence $\mathcal{T}$.

Following the uniformization method we discretize the underlying CTMC and compute $\mathbf{P_0} = \mathbf{I} + \mathbf{D_0}/\alpha$ and $\mathbf{P_1} = \mathbf{D_1}/\alpha$, with α being the uniformization rate such that $\alpha \geq \max_{i \in S}\{|\mathbf{D_0}(i,i)|\}$ and $\mathbf{I}$ is the identity matrix. Respectively $\mathbf{P_0}$ is an $(n * r) \times (n * r)$ matrix with elements $p^0_{i,j}$ (by construction it preserves the partitioning in CPH of $\mathbf{D_0}$) and $\mathbf{P_1}$ is a $n * n$ matrix with elements p^1_i. As a result $\sum_i (\sum_j p^0_{i,j}) + p^1_i = 1$; the components $p^0_{i,j}, p^1_i$ are the transition probabilities of the resulting uniformized DTMC. Obtaining the parameters that maximize the likelihood of the uniformized DTMC is equivalent to maximizing the likelihood of the original CTMC. $\mathbf{D_0}$ and $\mathbf{D_1}$ can be obtained through the following equations $\mathbf{D_0} = \alpha(\mathbf{P_0} - \mathrm{diag}(\mathbf{P_0}\mathbf{1}^{\mathrm{T}} + \mathbf{P_1^E}\mathbf{1}^{\mathrm{T}}))$, $\mathbf{D_1} = \alpha\,\mathbf{P_1}$. Where $\mathbf{P_1^E}$ is the $(n * r) \times (n * r)$ diagonal matrix with $p^E_i = p^1_{\delta(i)} \; \forall i \in [1, n * r]$, with $\delta(i)$ function that returns the index of the corresponding macrostate i.e. expands $\mathbf{P_1}$ so that it has the same dimension of $\mathbf{P_0}$.

Given $i, j \in [1, n * r]$ and $t \in [1, m]$ we define the following unobserved variables:

- $\hat{\mathbf{X}}^{(k)}_{\mathbf{0}}(i,j)$: number of transitions from state i to j in the interval (t_{k-1}, t_k)
- $\hat{\mathbf{X}}^{(k)}_{\mathbf{1}}(i)$: number of emissions in state i in the interval (t_{k-1}, t_k)
- $\hat{\mathbf{Y}}^{(k)}(i,t)$: number of emissions in state i of type t in the interval (t_{k-1}, t_k)

The complete log-likelihood on the uniformized DTMC is then given by:

$$CLL = \sum_{i=1}^{n*r} I(J(0)=i)\log\boldsymbol{\eta}_i + \sum_{k=1}^{K}\sum_{i=1}^{n*r}\sum_{j=1}^{n*r}\hat{\mathbf{X}}_{\mathbf{0}}^{(k)}(i,j)\log p_{i,j}^{0} + \sum_{k=1}^{K}\sum_{i=1}^{n*r}\hat{\mathbf{X}}_{\mathbf{1}}^{(k)}(i)\log p_i^E + \sum_{k=1}^{K}\sum_{i=1}^{n*r}\sum_{t=1}^{m}\hat{\mathbf{Y}}^{(k)}(i,t)\log c_{\delta(i),t} + \sum_{k=1}^{K}\sum_{i=1}^{n*r}\sum_{t=1}^{m}\hat{\mathbf{Y}}^{(k)}(i,t)\log g_{\delta(i),t}(\mathbf{x}_k;\boldsymbol{\theta}_{\delta(i),t}) \quad (3.1)$$

where $I(\cdot)$ is the indicator function and $\boldsymbol{\theta}_{p,t}$ are parameters of the distribution $g_{p,t}(\cdot)$. In the following, we present the cases where $g_{p,t}(\cdot)$ is either an exponential output, with only the rate $\beta_{p,t}$ to maximize (in Eq. 3.10) and the case where $g_{p,t}(\cdot)$ is a multinomial distribution of (fixed and known) values $[g_{p,t}^1,...,g_{p,t}^w]$ (in Eq. 3.11); an extension to the continuous case for Gaussian mixture can be easily achieved by incorporating the Maximization steps described in the EM algorithm for Gaussian Mixture fitting of [Bilmes et al., 1998]. Let O be a random variable corresponding to the observed data and $\boldsymbol{\theta}$ the previous parameter set.

To average over the possible observation sequences we define the following quantities $\mathbf{X_0}^{(k)} := E[\hat{\mathbf{X}}_{\mathbf{0}}^{(k)}O|\boldsymbol{\theta}]$, $\mathbf{X_1}^{(k)} := E[\hat{\mathbf{X}}_{\mathbf{1}}^{(k)}O|\boldsymbol{\theta}]$, $\mathbf{Y}^{(k)} := E[\hat{\mathbf{Y}}^{(k)}O|\boldsymbol{\theta}]$.

To calculate the previous expected values we need to define the forward and backward likelihood of a state. Let $\mathbf{a}^{(k)}$ be a $(n*r)$-sized row vector that contains in position i the joint likelihood of being in state i at time t_k and observing the first k events. When normalized over i, $\mathbf{a}^{(k)}(i)$ is equal to $P\{J(t_k)=i, \mathcal{T}^{1,k} \mid \mathcal{M}\}$, where $\mathcal{T}^{1,k} = \langle\omega_1,\ldots,\omega_k\rangle$ is the subsequence of $\mathcal{T}$ containing the observations from 1 to k included.

$$\mathbf{a}^{(k)} = \mathbf{a}^{(k-1)}\,\boldsymbol{\Pi}_{\tau_k}\,\mathbf{P_1^E} \circ \mathbf{f}(e_k,x_k) \quad (3.2)$$

where $\mathbf{a}^{(0)} := \boldsymbol{\eta}$, $\tau_k := t_k - t_{k-1}$, $\circ$ denotes the Hadamard product, $\boldsymbol{\Pi}_{\tau_k}$ is the $n \times n$ matrix of transient state probabilities of the hidden CTMC, i.e., $\boldsymbol{\Pi}_{\tau_k}(i,j) := P\{J(\tau_k)=i \mid J(0)=j\}$ and, $\mathbf{f}(e_k,x_k)$ is defined as a $(n*r)$-sized row vector containing at the i-th place $\mathbf{C}(\delta(i),e_k) * \mathbf{G}(\delta(i),e_k)$. In turn

$$\boldsymbol{\Pi}_{\tau_k}\mathbf{P_1^E} = \sum_{u=l_k}^{r_k}\psi(u,\alpha\,\tau_k)\mathbf{P_0}^{u-1}\mathbf{P_1^E} \quad (3.3)$$

where $\psi(u,\alpha\,\tau_k)$ is the probability that u events occur during the interval $[0,\tau_k)$ in a Poisson process with rate α; l_k and r_k are respectively the left and right truncation points for the evaluation of the Poisson probabilities during the interval $[0,\tau_k)$ [Fox

and Glynn, 1988]. Let $\mathbf{b}^{(k)}$ be a $(n * r)$-sized column vector that contains in position i the likelihood of observing the events $k+1$ to K, given that the state at time t_k is i.

Likewise $\mathbf{b}^{(k)}(i)$ is proportional to $P\{\mathcal{T}^{k+1,K} \mid J(t_k) = i,\ \mathcal{M}\}$.

$$\mathbf{b}^{(k)} = \mathbf{\Pi}_{\tau_{k+1}}\ \mathbf{P}_{\mathbf{1}}^{\mathbf{E}} \circ \mathbf{F}(e_k, x_k)\ \mathbf{b}^{(k+1)} \tag{3.4}$$

where $\mathbf{b}^{(K)} := \mathbf{1}^{\mathrm{T}}$ and $\mathbf{F}(e_k, x_k)$ is an $(n * r) \times (n * r)$ matrix whose rows are all equal to $\mathbf{f}(e_k, x_k)$.

Finally, the parameters that maximize the expected CLL are given by:

$$\boldsymbol{\eta}_i = \frac{\mathbf{a}^{(0)}(i) * \mathbf{b}^{0}(i)}{\sum_{j=1}^{n*r} \mathbf{a}^{(0)}(j) * \mathbf{b}^{0}(j)} \tag{3.5}$$

Where $\mathbf{a}^{(0)}(i) * \mathbf{b}^{(0)}(i)$ gives the probability of being in i at time 0 given O.

$$p_v^1 = \frac{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} {\mathbf{X_1}}^k(i)}{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} ({\mathbf{X_1}}^k(i) + \sum_{j=1}^{n*r} {\mathbf{X_0}}^k(i,j))} \tag{3.6}$$

Given that the emission probability in a macrostate is the same for all inner states, all expected emissions of the macrostate are considered (i.e. sum over all the inner states) .

$$\boldsymbol{\alpha}_v(p) = \frac{\sum_{k=1}^{K} \sum_{i=1,\delta(i)\neq v}^{n*r} {\mathbf{X_0}}^{(k)}(i, (v-1) * r + p)}{\sum_{k=1}^{K} \sum_{i=1,\delta(i)\neq v}^{n*r} \sum_{p'=1}^{r} {\mathbf{X_0}}^{(k)}(i, (v-1) * r + p')} \tag{3.7}$$

The initial probability in a CPH is obtained from the expected arrivals in it from the other CPHs, which can be exited only from the last inner state.

$$p_{i,j}^0 \qquad = \qquad \begin{cases} \frac{\sum_{k=1}^{K} {\mathbf{X_0}}^k(i,j)}{\sum_{k=1}^{K} \sum_{v=1}^{n*r} {\mathbf{X_0}}^k(i,v))} * (1 - p_{\delta(i)}^1) & \text{if } \delta(i) = \delta(j). \\ \\ \frac{\sum_{k=1}^{K} \sum_{p=1}^{r} {\mathbf{X_0}}^k(i,\delta(j)+p)}{\sum_{k=1}^{K} \sum_{v=1}^{n*r} {\mathbf{X_0}}^k(i,v))} * \\ (1 - p_{\delta(i)}^1) * \alpha_{\delta(j)}(j - (\delta(j) - 1) * r) & \text{otherwise.} \end{cases} \tag{3.8}$$

For simplicity, $p_{i,j}^0$ is determined after $\mathbf{P_1}$ since the same emission probabilities are shared in $\mathbf{P_0}$. The value of $p_{i,j}^0$ depends on if i and j belong to the same macrostate, when they are in different macrostates the expected emission from i to the $\delta(j)$-th macrostate are collected and divided based on $\alpha_\delta(j)$. By construction, only from the last inner state of a macrostate can a different macrostate be reached.

$$c_{v,t} = \frac{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} \mathbf{Y}^{(k)}(i,t)}{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} \sum_{t'=1}^{m} \mathbf{Y}^{(k)}(i,t')} \tag{3.9}$$

$$\beta_{v,t} = \frac{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} x_k \mathbf{Y}^{k}(i,t)}{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} \mathbf{Y}^{k}(i,t)} \tag{3.10}$$

$$g_{v,t}(g^i_{v,t}) = \frac{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} I(x_k = g^i_{v,t}) \mathbf{Y}^{(k)}(i,t)}{\sum_{k=1}^{K} \sum_{i,\delta(i)=v} \sum_{t'=1}^{m} \mathbf{Y}^{(k)}(i,t')} \tag{3.11}$$

In eq. 3.9, 3.10 and 3.11 the expected emissions in all inner states belonging to the same macrostate need to be collected.

The matrices $\mathbf{X_0}^{(k)}$, $\mathbf{X_1}^{(k)}$, and $\mathbf{Y}^{(k)}$ are computed as follows:

$$\mathbf{X_0}^{(k)}(i,j) = \sum_{u=l_k}^{r_k} \psi(u, \alpha\, \tau_k) * \sum_{l=0}^{u-2} (\mathbf{a}^{(k-1)}\, \mathbf{P_0}^{l})(i)\, \mathbf{P_0}(i,j)\, (\mathbf{P_0}^{u-l-2}\, \mathbf{P_1^E} \circ \mathbf{F}(e_k, x_k)\ \mathbf{b}^{(k)})(j) \tag{3.12}$$

$$\mathbf{X_1}^{(k)}(i) = \mathbf{a}^{(k)}(i)\mathbf{b}^{(k)}(i) \tag{3.13}$$

$$\mathbf{Y}^{(k)}(i,t) = \begin{cases} \mathbf{X_1}^{(k)}(i) & \text{if } e_k = t \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

Of these formulas, equations 2, 4 and 11 to 13 correspond to the Expectation step of the EM algorithm; equations 5 to 10 correspond to the Maximization step. Algorithm 2 summarizes the procedure. It iteratively performs the E-step (lines 7–12) and the M-step (lines 14–17) until a termination condition is reached. Common termination conditions are reaching a minimum relative change in the dataset likelihood, a maximum number of iterations or a minimum change in the norm of the matrices.

Multiple observation sequences

If the matrix $\mathbf{D_0}$ is reducible, i.e. there exist an absorbing state or a Bottom Strongly Connected Component (BSCC), the transient nature of the model only allows a finite, usually small, number of observations before the process falls into the BSCC. As such, to have reliable estimates for all parameters, a single long sequence of observations is insufficient and multiple observation sequences are needed.

Algorithm 2 Expectation Maximization Algorithm

Input: $T = \langle \mathcal{T}_1, \ldots, \mathcal{T} \rangle, n, r$
Output: $\mathcal{M} = (\boldsymbol{\eta}, \mathbf{D_0}, \mathbf{D_1}, \mathbf{C}, \mathbf{G}(x))$

Initialize $\boldsymbol{\eta}$ and $\mathbf{C}$ such that they are stochastic
Initialize $\mathbf{G}(x)$
$\alpha \leftarrow \left(\min_{r \in \{1,\ldots,R\}, k \in \{1,\ldots,K_r\}} \{\tau_k^r\} \right)^{-1}$
Initialize $\mathbf{P_1}$ such that $\mathbf{P_1}$ is diagonal
Initialize $\mathbf{P_0}$ such that it is partitioned in n CPHs and $\mathbf{P_0} + \mathbf{P_1^E}$ is stochastic
procedure EM($\mathcal{T}, \alpha, \mathbf{P_0}, \mathbf{P_1}, \mathbf{C}, \mathbf{G}(x)$)
    repeat
        for $k \in \{1, \ldots, K\}$ **do**
            Compute $\mathbf{a}^{(k)}$ and $\mathbf{b}^{(k)}$ according to Eq. (3.2) and Eq. (3.4)
        end for
        for $k = K, \ldots, 1$ **do**
            Compute $\mathbf{X_0}^{(k)}$, $\mathbf{X_1}^{(k)}$ and $\mathbf{Y}^{(k)}$ according to Eq.(11), (12) and (13)
        end for
        Compute $\mathbf{P_1}$ according to Eq. (3.6)
        Compute $\mathbf{P_0}$ according to Eq. (3.8)
        Compute $\mathbf{C}$ according to Eq. (3.9)
        Compute $\mathbf{G}(x)$ according to Eq. (3.10)
    until ending condition satisfied
$\mathbf{D_0} \leftarrow \alpha(\mathbf{P_0} - \text{diag}(\mathbf{P_0}\mathbf{1}^{\mathrm{T}} + \mathbf{P_1^E}\mathbf{1}^{\mathrm{T}}))$
$\mathbf{D_1} \leftarrow \alpha\, \mathbf{P_1}$
end procedure

Cases where the $\mathbf{D_0}$ matrix may be desired to be reducible are when the underlying hidden process contains failure modes or ages, rendering the process left-to-right.

The modifications to the equations 3.5 to 3.10 are straightforward: the numerators and denominators are calculated separately for each sequence, scaled by the inverse of the sequence probability and added together. The same procedure can be seen applied to HMMs in [Rabiner, 1989] and to M3PPs in [Carnevali et al., 2019].

Eq. 3.15 is an example of the changes to eq. 3.5.

$$\boldsymbol{\eta}_i = \frac{\sum_{r=1}^{R} \frac{1}{P_r} \mathbf{a}_r^{(0)}(i) * \mathbf{b}_r^{(0)}(i)}{\sum_{r=1}^{R} \frac{1}{P_r} \sum_{j=1}^{n*r} \mathbf{a}_r^{(0)}(j) * \mathbf{b}_r^{(0)}(j)} \tag{3.15}$$

Where R is the number of sequences, $\mathbf{a}_r$ and $\mathbf{b}_r$ are respectively the forward and backward likelihood of the r-th sequence and P_r is the probability of the r-th sequence. Where:

$$P_r = \mathbf{a}^{(K_r)} \mathbf{1}^{\mathrm{T}}$$

with K_r number of observations of the r-th sequence.

First passage time to a state of interest

Given a sequence of events $\mathcal{T}_{1,k} = \langle \omega_1, \ldots, \omega_k \rangle$ and an M3CPP $\mathcal{M}$ we may want to compute the CDF of the first passage time to some state of interest from instant t_k: a practical example is if the state represents a failure in a failure prediction scenario or more generally an absorbing state in a left-to-right model [Carnevali et al., 2019]. To do so we need to first calculate $P\{J(t_k) = i \,|\, \mathcal{M},\ \mathcal{T}^{1,k}\}$ for each hidden state i, that is to diagnose the current state at time t_k: it is the i-th entry of the normalized forward vector $\hat{\mathbf{a}}^{(k)}$. Depending on the sequence start we may need to use a prior for the initial probability that differs from $\boldsymbol{\eta}$. If we are interested in the distribution at a time $t > t_k$ then $P\{J(t) = i \,|\, t > t_k, \mathcal{M},\ \mathcal{T}^{1,k}\}$ is obtained by normalizing $\mathbf{a}^{(k)}\mathbf{\Pi}_{t-t_k}$. Let $\tau^\star$ be the remaining time to the entrance in the state of interest $s^\star$. We term as $\Phi(\tau) := P\{\tau^\star \leq \tau \,|\, \mathcal{M}, \mathcal{T}^{1,k}\}$ the CDF of $\tau^\star$ and $\Phi_i(\tau) := P\{\tau^\star \leq \tau \,|\, \mathcal{M}, \mathcal{T}^{1,k}, J(t) = i\}$ the CDFs of the remaining time conditioned to being in the hidden state i at time t. $\Phi(\tau)$ can then be computed from the $\Phi_i(\tau)$ by their sum weighted by the probability of being in state i at that time. $\Phi_i(\tau)$ can be computed either analytically, by enumerating and calculating the probability of the possible paths to the state of interest, if finite, or by simulation. In the present experimentation, the $\Phi_i(\tau)$ are derived through Monte Carlo simulation.

3.2 Experimentation

In this section we evaluate the effectiveness of the proposed EM algorithm for M3CPPs by comparing the performance of the learned model to LSTMs.

Specifically we aim to learn the behavior of a Semi-Markov Process (SMPs) enhanced with an Observation Model, i.e. a process made of a set of Modes, each characterized by a sojourn time distribution and by an emission process producing observations characterized by a discrete type, a continuous space intensity, and a continuous timestamp. In doing so we evaluate the degree to which a M3CPP model can approximate the more general class of Continuous time SMPs with observations. Given a set of sequences of events generated from the SMP, we first train the models and then examine their performance in the online evaluation of the remaining time to a chosen Mode of interest given a partial sequence of events; the task is described in Subsection 3.2. The quality of the fitting jointly depends on the capacity (expressivity) of the class of M3CPPs and LSTMs models in encompassing the behavior of the generator model, on the optimization approach implemented in the learning process, and on inherent diagnosability and predictability of the model that generates the dataset. In order to marginalize the impact of the latter factor,

we identify 3 classes of SMPs from which we generate a suite of synthetic datasets (Subsection 3.2) so as to have controlled variation of the factors that affect the recognizability of the generator model. In Subsection 3.2 we subsequently introduce the results through Precision-Recall curves and conduct a statistical hypothesis test to see if the models are statistically equivalent.

Application scenario

We consider a hidden continuous-time process $\{J(t),\ t \geq 0\}$ with state space S, where $J(t) \in S$ is the state of the process at time t which contains a state or Mode of interest $s^\star$. The hidden, unobservable, process is enhanced by an Observation Process such that each state $s \in S$ emits events, associated to a discrete type and a continuous intensity. This kind of processes can be used as abstractions of aging in software systems [Cotroneo et al., 2014], disease progression in human patients [Liu et al., 2015], execution of security attacks in information technology systems [Carnevali et al., 2019]. We consider then the application scenario where the observations in a sequence of events produced by such an hidden process are used by a Monitor to learn a M3CPP model fitting the observed statistics and then to predict the arrival in some absorbing state.

Failure prediction Following the standard formulated by Salfner and Malek [Salfner and Malek, 2007] for online failure prediction, the chosen task in our experimentation is to predict if the absorbing state is reached inside of a specific window of prediction.

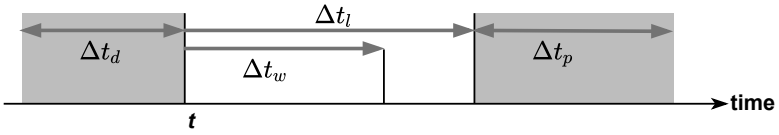

Figure 3.3: Time relations in online failure prediction:
t – present time; Δt_l – lead time; Δt_w – warning time;
Δt_p – prediction period; Δt_d – data window size

A visualization of the time relations in the task is given in Figure 3.3 . Specifically, given a time t at which we perform the prediction, we want to know whether the state of interest will be reached in the window $[t + \Delta t_l, t + \Delta t_l + \Delta t_p]$, where Δt_l is the *lead time*, Δt_p is the *prediction period* and describes the lenght of time over which the prediction holds. Δt_w is the *warning time*, a lower bound over the lead time, needed so that responses to the prediction can be enacted e.g. time needed for preventive

shutdowns. Δt_d is the *data window size* and corresponds to length of the data window considered when making the prediction (in our case all data is considered).

To note, the prediction period width requires careful balance: it needs to be sufficiently wide so that the probability of containing the event of interest (arrival in the absorbing state) is significant while small enough to maintain precision of the prediction.

It is worth nothing that the task considers only the reach of a particular absorbing state inside a window of prediction: in the learning phase we only need to learn when the state is reached; the sojourn time or distribution of event types (or intensities) of the absorbing state are of interest only to identify a posteriori when the state has been reached. As such, to facilitate the fitting we relabel all the events emitted in the absorbing state as *conclusive* events $e^\star$, a type that is produced only by the absorbing state. As a remark, seeing such ending event in the sequence allows our model to know with certainty that the absorbing state has been reached but it does not eliminate the uncertainty about when the state was first entered. Conversely seeing any other event type is proof that the absorbing event has not yet been reached. This special treatment for the ending events lets us consider the task as a problem in the class of semi-supervised learning [Zhu and Goldberg, 2009], since only some events are associated to a "label". As a consequence of the task, we require that with probability 1 the ending state is eventually reached in the future. As such, the infinitesimal generator of the M3CPP is reducible and thus the learning phase cannot be accomplished by a single, sufficiently long, sequence of events: to learn the model multiple sequences (that not necessarily reach the absorbing state) are needed.

Generator Models

We consider the case where the hidden process' states are regenerations (i.e. satisfy the Markov condition) and the sojourn time in each state $s \in S$ and the time between emission are characterized by a non-exponential Cumulative Distribution Function (CDF). Accordingly the hidden process under consideration behaves like a Semi-Markov Process.

Since the intent of our experimentation is to evaluate the effectiveness of M3CPP and LSTM in learning the generator model, we desire to perform a sensitivity analysis by experimenting on a suite of synthetic datasets where variations on the key features that determine mode recognizability are exercised (i.e. modes sojourn times, frequency and range of types and intensities of observed events).

We thus introduce a way to generate Semi-Markov Processes with the required variations to produce the datasets.

In Figure 3.4 we have a general description of our SMPs. Each generator model is associated with a molteplicity of Modes, each of which is described by a sojourn time spent in the Mode, an Observation Process and a number of outgoing Tran-

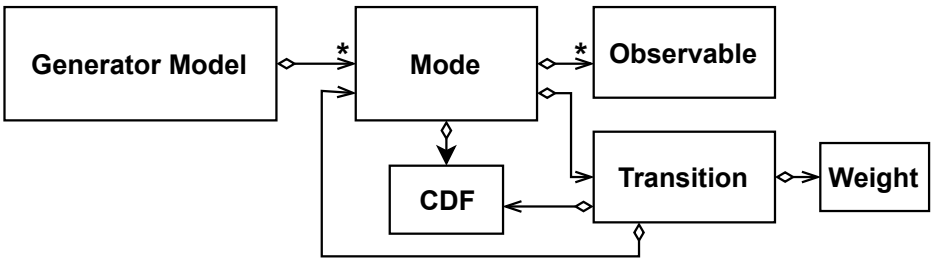

Figure 3.4: Structure of the Semi-Markov Process generator

sitions. The Observation Process is defined by an inter-event time distribution, a set of emission types and a probability and a distribution of intensity for each event type associated with the process. Each Transition is associated with an arrival Mode and with CDF and a weight. While not explicitly drawn nor technically required, to simplify the model we allow at most a single transition between each couple of modes.

As required by Section 3.2, there exists a special absorbing Mode $s^\star$ capable of emitting only events of type $e^\star$, which are exclusive to this Mode. A single Mode is the initial Mode of the model.

The generator model starts in the initial Mode; at the arrival in each mode a transition, outgoing from the mode, is selected with probability dependent on their associated weight and a sojourn time is sampled from the correspondent CDF. In parallel, an observation time is sampled from the Observation Process associated with the current Mode. If the observation time precedes the remaining sojourn time then the sojourn time is decreased by the corresponding value, the event is observed (its type and intensities are sampled) and a new observation firing time is sampled.

In general the generator models are (Continuous Time) Hidden Semi-Markov Processes and as such described by:

- $S :=$ a set of states with cardinality N
- $G_{ij}(t) := P(t_1 \leq t \wedge S(t_1) = j | S(0) = i)$ a global kernel, where t_1 is the time of first transition and $S(t)$ is the state at time t
- $O_i :=$ observation process in state i

The observation process describes the events observed in a state and their arrival process. Like the sojourn time, the observation process can have different degrees of expressivity: arrivals of events (consisting in the type and intensity) may be dependent and the inter-events time can have memory.

While many of the parameters of the generator model can (and are) randomly sampled at the start (e.g. transition weights and firing time distributions), to satisfy

required invariants (e.g. existance of a path from initial to ending Mode) some restrictions have been applied to define what transitions are permitted. Depending on the connections between modes the corresponding reachability graph could result in not being connected (presence of multiple cliques) or present multiple Bottom Strongly Connected Components (BSCCs) other than the absorbing mode of interest. While the former does not introduce problems (the actual model being the connected component containing both initial and ending mode), we decide to disallow the possibility of multiple BSCC. Although not required by the theory, for the sake of simplicity and to reduce the amount of data required for learning, we further restrict ourselves to left-to-right models. As such, given the number of Modes in the SMP, the connections between sequential modes (i.e. from mode i to mode $i+1$) are fixed while we randomly sample, given a probability of connection p, the existence of the other left-to-right connections. Consequently the adjacency matrix of the Semi-Markov Process is an upper diagonal matrix where the first super-diagonal has only non-zero elements.

We consider the sojourn time distributions and emission distributions as Erlang distributions, where for each transition and observation process we randomly sample the rate λ given a fixed shape k. We fix the Erlang distributions' shape to given values as a way to determine their coefficient of variation and ease of approximation. For the intensity of the events in the observation process we utilize Gaussian distributions where we randomly sample the mean and variance.

Specifically we consider 3 classes of generator models:

- **Default Model (DM)**: a "small" model with "clearly defined Modes"
- **Undisciplined Model (UM)**: a small model with higher ambiguity between Modes
- **Larger Model (LM)**: a "bigger" model, with clearly defined Modes.

Behind our choice of generator models is the desire of comparing the performance of M3CPP and LSTM on a default model and then study their reactions to variations that branch both on the dimension of the model and on the ease of recognition of the component modes.

Default Model

- 5 modes
- probability of having connections between modes (non-first super-diagonal of the adjacency matrix) is $p = 0.4$ for an average of 6.4 total transitions (4 fixed on the first super-diagonal and 6*0.4=2.4 on the rest)

- CDF of inter-event and sojourn times are Erlang distributions with shape 4 and rate λ, with λ chosen such that k/λ is a random variable with Uniform distribution $\in (0, 10]$
- each mode can emit 3 types of events out of 10
- intensity has mean and variance Uniform $\in (0, 10]$
- window of prediction is set at $[t + 1, t + 5]$

Undisciplined Model Differs from DM in:

- CDF of inter-event and sojourn times are Erlang distributions with shape 2 and rate λ such that e k/λ is Uniform $\in (0, 10]$
- each mode can emit 5 types of events out of 10

Larger Model Differs from DM in:

- 10 modes
- probability of having connections between modes (non-first super-diagonal of the adjacency matrix) is $p = 0.4$ for an average of 23.4 total transitions (9 fixed on the first super-diagonal and 36*0.4=14.4 on the rest)

We consider the models as disciplined when the Erlang distributions have shape 4 due to the lower coefficient of variation ($1/2$ versus $1/\sqrt{2}$) and the lower probability of overlapping event types.

For each of these 3 classes of generator models we produce 20 instances to marginalize the inherent diagnosability and predictability of the model that generates the dataset due to parameters selection. From each instance we generate 3 datasets: all the testsets are composed of 1000 sequences while the training sets are respectively composed by 1000, 300, and 100 sequences to evaluate the models performance under different quantities of data. The training set sequences start all on the initial mode and terminate on the ending mode of interest. The test set sequences starts on the initial mode and end after a random percentage of the time needed to reach the absorbing state, possibly not simultaneously with an event; the test sequences are all associated to the sampled remaining time to reach the absorbing state: this time is compared to the window of prediction to determine if the state is reached inside of it.

The dimension of the window of prediction is the primary factor in determining if the absorbing state has been reached inside of it; this promotes differences between the 3 generator classes: DM and UM, requiring a lower average time to reach the

absorbing state will likely have a higher number of True Positives since the window of prediction covers an higher percentage of the transient behavior, this results in an higher number of test cases that correspond to a positive event(absorption inside the window); conversely for LM the same is true for True Negatives.

Setting the model hyperparameters Before learning the models multiple hyper-parameters need to be set: in the case of LSTM the parameters are the number of cells in each layer, the number of layers in the stack and the discretization step; in the case of M3CPP the hyper-parameters refer to the number of macrostates and the number of local states inside each macrostate.

Specifically, for each of the 3 generator models, we first identify the hyper-parameters that optimize the first of the 20 instance-datasets with 1000 sequences for both techniques and then directly apply those values of the parameters for the other 19 datasets. While the 20 datasets for each generator model are created such that different conditions for the hidden states recognizability are presented, they still maintain the properties shared by the generator class which are the number of states and the overall ambiguity or discipline of the datasets: we consider these shared characteristics as sufficient to capture the degree of complexity of the models and so are enough to identify the correct number of macrostates and local states for each generator model. We assume that the optimal number of states for the M3CPP will be comparable to the number of states of the generator model while the number of local states will depend on the degree of uncertainty in the sojourn time: we assume that in the disciplined models the higher certainty will actually require more local states to lower the coefficient of variations of the M3CPP to be closer to that of the generator model's hidden states.

Specifically, the hyper-parameters values we considered for the LSTM are 10 and 30 cells in a configuration of either 1 or 2 layers. We also considered the discretized time steps corresponding to 0.5,1 and 2 units of time.

For the M3CPPs the number of macrostates tested ranged from 5 to 8 states for the smaller models and 10 to 15 states for the larger one. Regarding the number of local states per macrostate considered, instead of optimizing the number in a range, all experiments were done with both 1 and 3 local states to show the differences in the introduction of non-markovian behavior for sojourn times. In all cases we considered only normal distributions to learn the intensity of the events.

In the case of the M3CPP the model is learned through the EM procedure described in Algorithm 2. In particular the initial values of the infinitesimal generator $\mathbf{D_0}$ and $\mathbf{D_1}$ are selected randomly so that the overall average sojourn time in the hidden states, except for the absorbing state, is equal to the average duration of the event sequences observed in the training set until when the absorbing state is reached. The event probability matrix $\mathbf{C}$ is initialized with the observed probabil-

ity (collected over the training set) that each event type is emitted by the generator model in each state. Likewise, the initial means and variances of the gaussian distributions are set to the means and variances calculated over all the events of the same type in the dataset.

For the LSTM model, the chosen loss is binary crossentropy while the chosen optimizer is ADAM, no optimization was done on the rate.

Experimental results

After determining the probability of reaching the absorption state inside the window of prediction, as described in Section 3.1, to determine if our prediction is positive or negative we need to compare the calculated probability to a threshold. If the model produces a probability that is higher than a given threshold then we say that the model has predicted the arrival in the absorbing state. If the model has predicted the arrival and the window of prediction actually contains it then the prediction is a *true positive* else if the window does not actually contain the arrival it is a *false positive*. If the model has predicted that there will not be an arrival and the window actually contains it, it is a *false negative* otherwise a *true negative*.

Given a testset let TP, FP, and FN be the total number of attained true positives, false positives, and false negatives, respectively. To evaluate performance, we define the following measures: *precision* $\mathrm{P} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ and *recall* $\mathrm{R} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$. Different P and R values can be obtained by using different thresholds to determine when an example is predicted as positive. With these values we can compute the *Precision-Recall curve* and the area under the curve, termed *Area Under Precision-Recall Curve* (AUPRC), which can be used to measure the prediction performance [Davis and Goadrich, 2006]. To note that given the unbalance of the testcases accuracy would not have been a good measure.

Each training set (for each of 3 classes, 3 data dimensions and 20 instances) is used to learn the M3CPP (with both 1 and 3 local states) and LSTM models while the corresponding test set is used to asses the model performance in predicting the arrival to the absorbing state.

For each model learned on each dataset, we discretize the Recall in 100 slots and compute the corresponding Precision by gradually lowering the threshold.

After fixing the generator class and the data dimension, we compute the mean Precision-Recall curve by averaging over the values of the Precision at each Recall slot in the 20 instances.

Figures 3.5-3.7 contain the mean Precision-Recall graphs, respectively of the Default Model, Undisciplined Model and Large Model. At the left the models are trained over 1000 sequences, in the center over 300 sequences and in the right they are trained over 100 sequences.

To corroborate the results, we conducted an hypothesis test based on the work of [Hanley and McNeil, 1983] and [Hajian-Tilaki, 2013]. Specifically we want to determine from the AUPRC if the M3CPP-3 and LSTM are equivalent: the null hypothesis is that they are equivalent, the alternative is that the differences in the Precision-Recall curve are statistically significant. In order for the null hypothesis to be rejected at the 95% confidence level $|Z_{obs}| > 1.96$, where Z_{obs} is the observed value of the Z-score. The score is calculated from the formula 3.16

$$Z = \frac{A\hat{U}C_M - A\hat{U}C_L}{\sqrt{Var(A\hat{U}C_M) + Var(A\hat{U}C_L) - 2Cov(AUC_M - AUC_L)}} \tag{3.16}$$

Where $Var(\cdot)$ is the variance and $Cov(\cdot)$ is the covariance, calculated from [DeLong et al., 1988].

In Table 3.1 are contained the AUPRC values and the Z-score between M3CPP with 3 local states and LSTM.

In Figure 3.5 are shown respectively the results, averaged over the 20 instances for the 3 dimension of the datasets for the Default Model.

It can be seen how the methods are comparable with 1000 sequences, then the results tend to degrade as the sequence number lowers and that LSTMs clearly is more sensitive to the data available.

In Figure 3.6 are shown the results for the Undisciplined Model.

In this case we can see that respect to the more recognizable Default Model the techniques find greater difficulty in predicting the arrival in the absorbing state.

In Figure 3.7 are shown the results for the Larger Model.

In this setting, the performance of the approaches drops since the underlying generator model is more complex and so harder to learn, particularly for the increase of connections between states (from 6.4 to 23.4 on average). Overall M3CPP with 3 local states appears to be better performing than both the 1 local state version and LSTM: as predicted the increase of performance over the 1 local state is more evident in the DM and LM models, whose Erlang distributions have more phases. There is a clear trend in losing performance when the number of training sequences lowers, specially for LSTM for which it is common knowledge that it require a large amount of data to perform. On the other hand it has been proven that deep learning models with a sufficiently large dimension can approximate any distribution [Lee et al., 2017], as such, with virtually infinite data and time of computation LSTM would certainly outperform M3CPP, by perfectly approximating the generator model.

While the mean AUPRC gain over LSTM would appear to be significant, the Z-score shows that even with low sequences (and thus relatively high difference in performance between M3CPP-3 and LSTM) the null hypothesis cannot be rejected: it is due to the very high variance of the prediction over the different instances, that is not completely countered by the covariance. By studying the results, while most

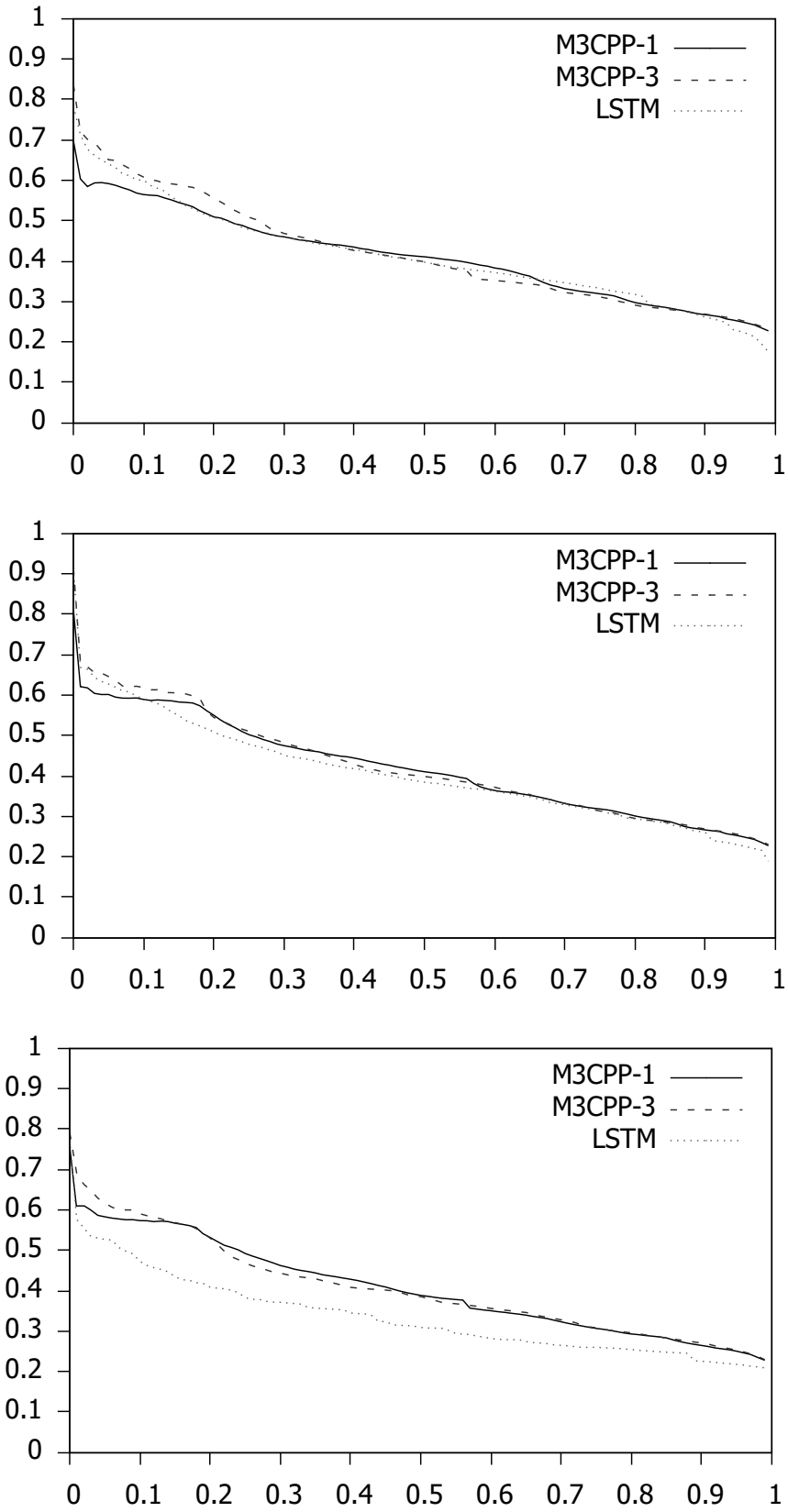

Figure 3.5: Mean Precision-Recall graphs over the DM, with models trained on 1000 (left), 300 (center) and 100 (right) sequences

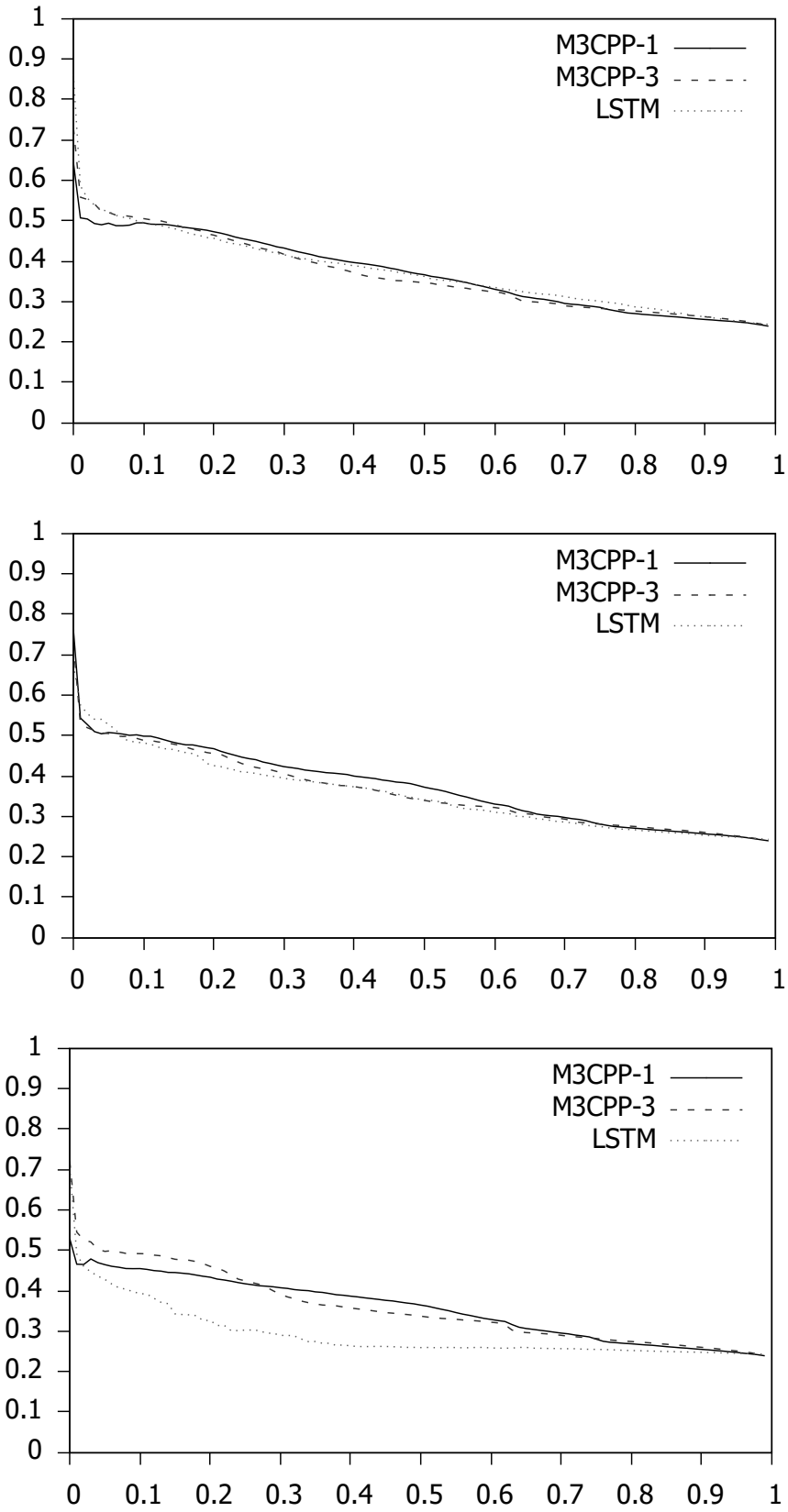

Figure 3.6: Mean Precision-Recall graphs over the UM, with models trained on 1000 (left), 300 (center) and 100 (right) sequences

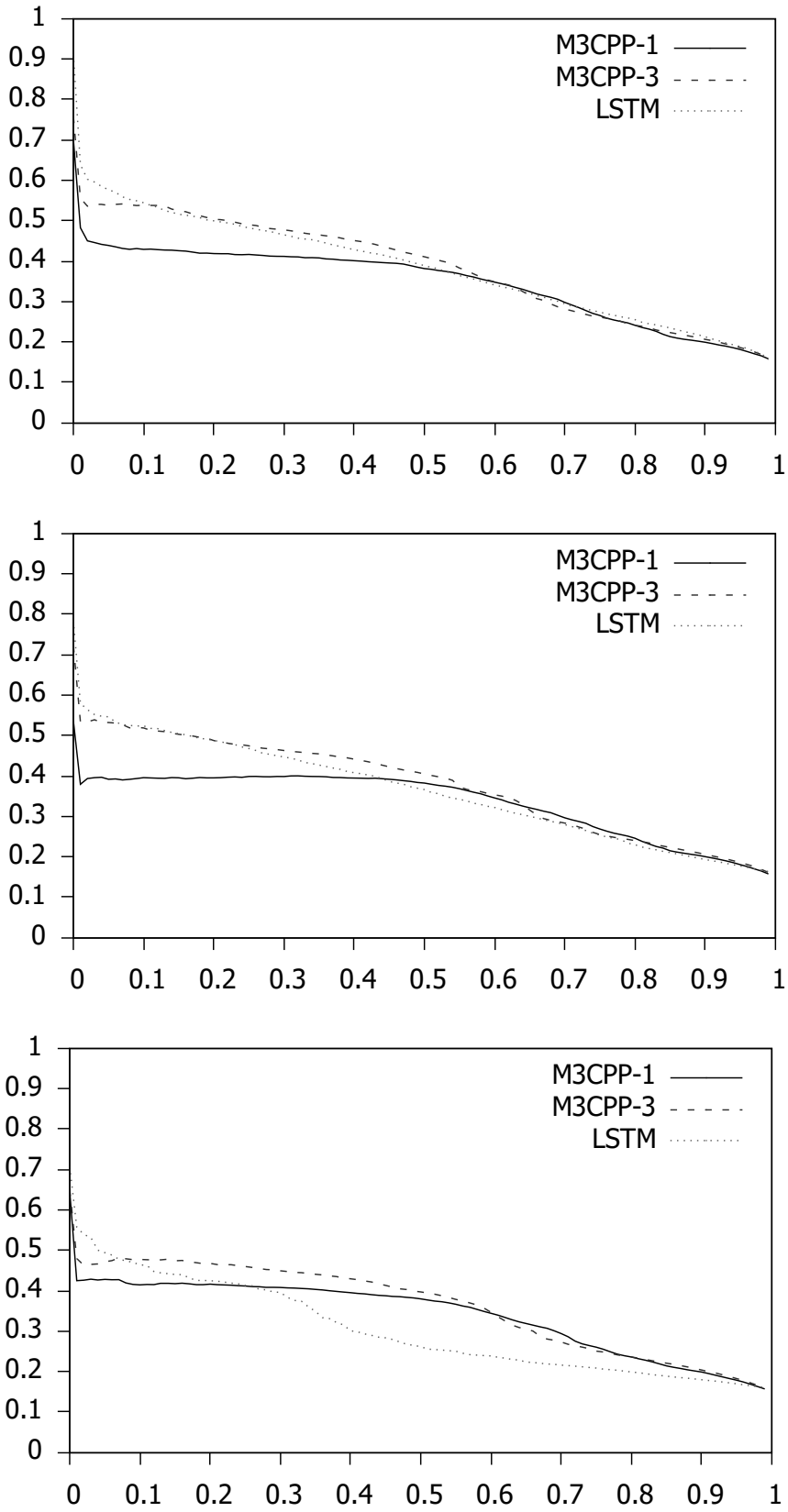

Figure 3.7: Mean Precision-Recall graphs over the LM, with models trained on 1000 (left), 300 (center) and 100 (right) sequences

Table 3.1: Prediction performance of the learned models

	AUPRC			
Model - size	M3CPP-1	M3CPP-3	LSTM	Z
DM - 1000	0.409	0.419	0.413	0.71
DM - 300	0.418	0.423	0.405	0.42
DM - 100	0.405	0.406	0.334	0.71
UM - 1000	0.368	0.365	0.373	0.31
UM - 300	0.370	0.359	0.354	0.45
UM - 100	0.353	0.356	0.290	0.74
LM - 1000	0.346	0.384	0.385	0.02
LM - 300	0.333	0.376	0.364	0.31
LM - 100	0.339	0.361	0.304	0.82

instances show high values of AUPRC some have extremely poor results, likely due to peculiar sampling of the generator models, lowering the mean and introducing high variance.

In conclusion we find that M3PCC is always at least comparable to LSTM and shows marked benefits when data becomes scarce, as one would expect from the intrinsic machine learning weakness.

Chapter 4

Application of M3CPP to High-performance computing

In this chapter we apply the M3CPP model described in Chapter 3 to real world datasets in the field of High-performance computing, respectively the HPC Intrpid and Mira from Argonne National Laboratory. The comparison to LSTM allows us to show that the previous results, although obtained on a broad spectrum of models, are not simply theoretical but find confirmation in a practical application.

4.1 Background on High-performance computing

We give a brief overview of the application context of HPC clusters, describing RAS and Job logs from the Intrepid Blue Gene/P system at Argonne National Lab., and finally formulate the failure prediction task.

RAS Log and Job Log from Intrepid

Intrepid was a 40-rack Blue Gene/P system that operated at Argonne National Laboratory for the U.S. Department of Energy from 2008 until 2013 when it was dismissed after his natural life-cycle ended [Collins, 2013]. A more in-depth overview of the HPC framework can be had from [Almasi et al., 2008]. In the HPC a Core Monitoring and Control System (CMCS) reported anomalous events in the hardware components as Reliability, Availability, and Serviceability (RAS) event messages.

Logs have been made publicly available by the Argonne Leadership Computing Facility [Data,] and by [Zheng et al., 2011]. In this work we exploit *RAS logs* and *Job logs* spanning 273 days, from 2009-01-05 to 2009-08-31, respectively of 1.1 GB and 55MB.

An example of event record from Intrepid RAS log is shown in Table 4.1. The major fields are explained below.

RECID	26123930
MSG_ID	KERN_0802
COMPONENT	KERNEL
SUBCOMPONENT	_bgp_unit_ddr
ERRCODE	_bgp_err_ddr_single_symbol_error
SEVERITY	WARN
EVENT_TIME	2009-01-05-00.02.51.162211
FLAGS	-
LOCATION	ANL-R46-M0-512
SERIAL_NUMBER	44V3575YL12M80156ZH
MESSAGE	ECC-correctable single symbol error..

Table 4.1: Example of RAS event message

- *RECID* is the sequence number of the message. record
- *MSG_ID* is the source of the message.
- *COMPONENT* is the software component that has detected and reported the event, the possibilities are : *MMCS*, *KERNEL*, *MC*, *APPLICATION*, *BAREMETAL*, *CARD*, or *DIAGS*.
- *SUBCOMPONENT* is the functional area of the component that generated the message. There exists a total of 40 subcomponents.
- *ERRCODE* identifies the event type information. There exists a total of 182 unique error codes.
- *SEVERITY* can be *INFO*, *WARN* or *FATAL* in order of increasing severity level. INFO provide information about progress of system software. WARN events represent recoverable "soft" errors. FATAL events likely produce interruptions in the Job or Jobs in execution.
- *EVENT_TIME* identifies the time of detection.
- *LOCATION* identifies where the event has occurred: can be a node, midplane, rack or multiple racks.
- *MESSAGE* is a summarization of the event condition and can contain information from other fields.

Besides, Job Logs collected by the scheduler include:

- *Execution File* is the executable path.
- *Start Time* is the time when the job started to run.

- *End Time* is the time when the job exits whether by finishing or by interruption.
- *Location* is the location of the execution with a minimum of one midplane.

Smaller jobs tend to take 1 rack and larger jobs with up to 16 racks were common. An administrator was not required for any job with less than 32 racks.

In this kind of context where any failure in multiple nodes can cause an interruption in a job execution, the prediction of the FATAL severity events has a central importance for failure tolerance approaches to mitigate damage.

Failure Prediction Described in section 3.2, online failure prediction methods can generally be classified into two groups depending on the definition of the data window size Δt_d: the period-based approach and the event-driven approach which differ in their trigger mechanism [Salfner, 2006].

Period-based approaches rely on constant monitoring of some resource e.g. the amount of free memory in a RAM to identify symptoms of some underlying anomaly that has not yet manifested itself as a failure. As such their data window size is fixed to some length usually multiple of the window of prediction. Some of their kind tend to approximate a function of the failure probability to make predictions. Event-driven approaches instead rely on information from logged events, like the RAS logs. As such their data window is ideally the entire sequence of events observed. In practice due to space and computation constraints and thanks to the less usefulness of distant events for prediction, even the event-driven data size is limited and can assume fixed or flexible lengths.

In our case RAS message logs naturally lend themselves to event-driven approaches, as it has been shown in [Yu et al., 2011].

4.2 Experimentation

In this section we fully describe our specific task of failure prediction and how we use M3CPP and LSTM to predict failure times from RAS message logs (Subsection 4.2). We then evaluate the proposed approach on the Intrepid 2009 RAS log dataset and analyze the experiment results (Subsection 4.2).

Approach

Our workflow is described in Fig. 4.1. The task we seek to resolve is to predict if a specific job will fail in a given window of prediction basing our information on all and only the messages recorded in the RAS event log in the racks, midplanes and nodes assigned to the job. In doing so we intentionally disregard messages being produced in different locations as it has been shown in [Zheng et al., 2011]

that they rarely propagate between jobs. We consider a failure the case where a FATAL severity event is produced by a location pertaining to the job under analysis although FATAL events not necessarily always produce interruptions.

Specifically to fit our models we create for each job in the Job Log a sequence of events corresponding to all and only the events reported by locations overlapping with the job, as we assume that any event originating from overlapping locations is produced by a shared node. Likewise, when an event is recorded with missing location information (some typology of events do not have it) we assume that it refers to all the location in Intrepid and as such are included in all concurrent sequences.

Specifically we are concerned with the problem of failure prediction to optimize checkpointing. As such we consider only sequences that correspond to jobs not scheduled to already failing or failed nodes: between the job start and the FATAL event at least another message has to be recorded. We justify this by considering that in such cases checkpointing would be ineffective as too little work would have been executed for a checkpoint to be of worth. Li et al. [Li et al., 2008] have shown that only with sufficient accuracy run-time fault management can be effective. Due to that we consider the worst case scenario: all the task we consider are doomed to fail during their execution, where their natural execution end to be at a sufficiently distant future time. As such our models are trained (and tested) only on sequences of events that originate from jobs that have observed a FATAL event during their execution (but not necessarily have been interrupted by them). Such FATAL events corresponds to the end of each sequence under consideration.

As described in Section 4.1, our dataset is composed of RAS message logs and Job logs collected on the Intrepid BlueGene/P system over the period from 2009-01-05 to 2009-08-31. During the 237 days 68,794 jobs and 2,084,392 RAS records, with 33,370 FATAL messages, have been logged. In case of the RAS events, records are extremely redundant:the events corresponding to the same failure can be reported multiple times by different nodes or by the same monitoring service.

It is common knowledge that HPC RAS logs contain unnecessary records: it has been shown that such records can be compressed by over 99% without loss of information [Liang et al., 2005]. Filtering RAS logs has been an active area of research with other works such as [Oliner and Stearley, 2007, Zheng et al., 2009, Zheng et al., 2012]. Main techniques applied are spatial-temporal filters: temporal filters use thresholds to remove multiple events being reported in the same location while spatial filters remove the same type of event being reported in multiple locations. On our very dataset, [Zheng et al., 2011] also apply a causality-related filter to identify and filter sets of fatal events co-occurring together. Through these operations Zheng at al. have been able to reduce the FATAL severity records from 33,370 to 549 events (compression ratio of 98.35%).

Likewise we apply some crude filtering to reduce the amount of records and in-

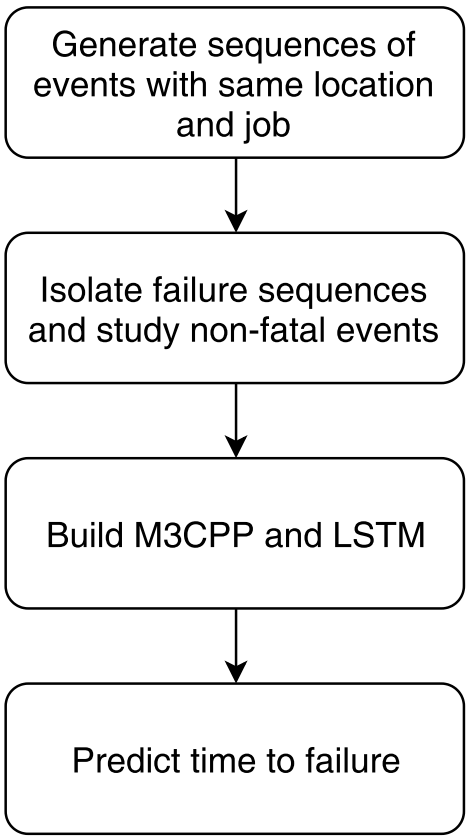

Figure 4.1: The Prediction Workflow

crease prediction accuracy. Comparatively we do not need to apply spatial filters as we consider each job's location separately: the redundant information is instead useful to us as it informs us about which locations are affected by the unobserved anomaly. The filter has 2 phases: first we disregard any non-FATAL message that is produced before 1 second has elapsed from the last, (effectively we can be considered to be discretizing the time in a time series for non-FATAL events), then we limit the messages with the same component, subcomponent and error code (and location) to be repeated only once each 5 seconds.

After this operation our dataset results to have a total of 610 sequences ending in FATAL events. To limit computation time we bounded the maximum number of events preceding the FATAL message to 60. We assume that older events do not introduce ulterior information and are far enough in time to not see the FATAL event in their prediction window. After this passage the average of messages per sequence is 14.7 . In these sequences only 46 error codes appear of the original 182.

We note that the resulting dataset contains sequences that are identical as concurrent jobs can be terminated by the same FATAL error and observe only events without a specified location.

This dataset is still composed of raw RAS logs, which we cannot directly input in our models.

In [Liang et al., 2005] clustering RAS event data was first introduced to compress the logs and improve prediction accuracy. Following the approach of [Chen et al., 2013] we apply a frequency analysis to the error codes to cluster them in more manageable and generalizable groups. We consider the distribution of the mean time to failure of the non-FATAL events in our sequences and define as belonging to the same cluster all the events falling in specific bounds. In Fig. 4.2 we can see the distribution of the mean times to failure for the 46 error codes versus the number of actual observed events. The vertical lines represents bounds for the specific case of STEP=30 seconds.

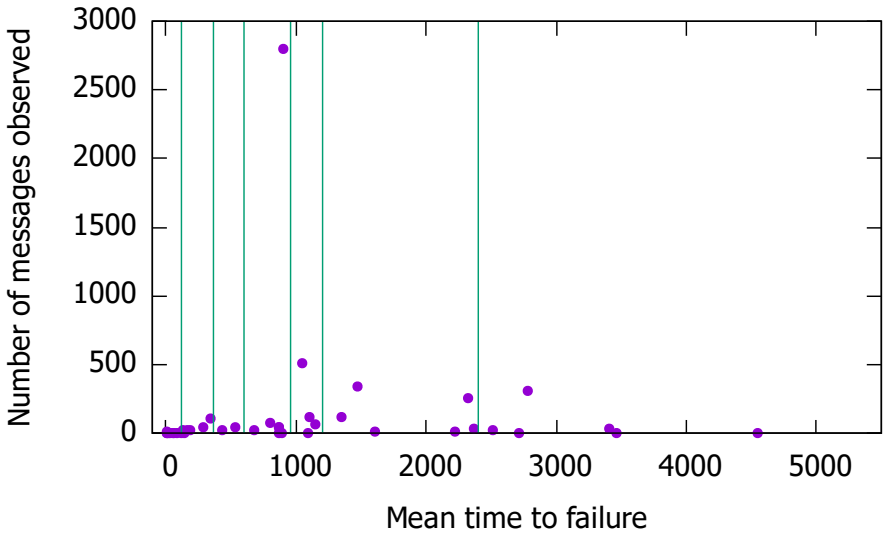

Figure 4.2: Average time to failure and frequency for each message type, bounds of the 120 seconds case.

In the real world it is not possible to choose the prediction window and the frequency of failure predictions based only on the best result from the prediction method: similarly we aim to show how M3CPP and LSTM perform under differing conditions of prediction window size and lead time. The chosen prediction window sizes and lead times are multiples of the prediction *step*, i.e. the time after which we must make another prediction. Respectively we tested the configurations of windows [1**step*,3**step*],[1**step*,6**step*],[2**step*,6**step*]. With *step* assuming the values of 30,60,90,120 and 150 seconds. The first window configuration for *step* 30 seconds would come to have a lead time of 30 seconds and a width of 60 seconds.

The cluster bounds are fixed as multiples of the prediction *step*: respectively [0,3**step*],[3**step*,5**step*],[8**step*,10**step*] and greater than 10**step*. Events not observed in sequences ending with FATAL events are either unassigned, if assent from the

entire dataset or placed in the last class if present in non-FATAL sequences. We furnish to M3CPP and LSTM both the Component and the cluster information of each message: in case of unassigned events(although they do not happen in our experimentation) the Component part of the message can still be exploited to obtain information based on collected statistics.

After the clustering of the events, to limit the instability of the initial learning phases, we bounded the maximum inter-event times to 300 seconds, a value that we assume sufficiently high to still "hide" the preceding events while sufficiently low to not impact too much the computation of either M3CPP and LSTM.

Experimental Results

We evaluate the effectiveness of the proposed EM algorithm for M3CPPs by comparing the performance of the learned model to LSTMs. The models are compared in a variety of datasets which vary in their predictability. We further study the effect of the dimension of training set on the models by fitting the models only on half and a quarter of the training set (note that the entire training set was used to determine the clustering).

The dataset was split into 80% training and 20% test set.

The test set sequences are prematurely ended after a random percentage of the entire sequence. The models are required to produce a prediction of failure every time step from the start of the sequence up to the premature end.

The M3CPP model was tested under different numbers of macrostates and the chosen configuration is 10 macrostates, we show the differences in results by learning with both one and three local inner states. For the LSTM model we use a 2-layer stack of 20 LSTM cells. 10 seconds was identified as a good discretization step. The input to the LSTM models were bounded to 100 discretization steps. For the LSTM model, the chosen loss is binary cross-entropy while the chosen optimizer is ADAM, no optimization was done on the rate.

After the model has determined the probability of reaching the absorption state (for M3CPP, Section 3.1) or a measure of confidence of the failure (for LSTM) inside the window of prediction; to determine if our prediction is positive or negative we need to compare the calculated value to a threshold. If the model produces a measure that is higher than a given threshold then we say that the model has predicted the failure in the window. If the model has predicted the failure and the window of prediction actually contains it then the prediction is a *true positive* else if the window does not actually contain the failure it is a *false positive*. If the model has predicted that there will not be a failure and the window actually contains it, it is a *false negative* otherwise a *true negative*.

By gradually increasing(or decreasing) the values of the two thresholds, different precision and recall values can be obtained as the windows are increasingly

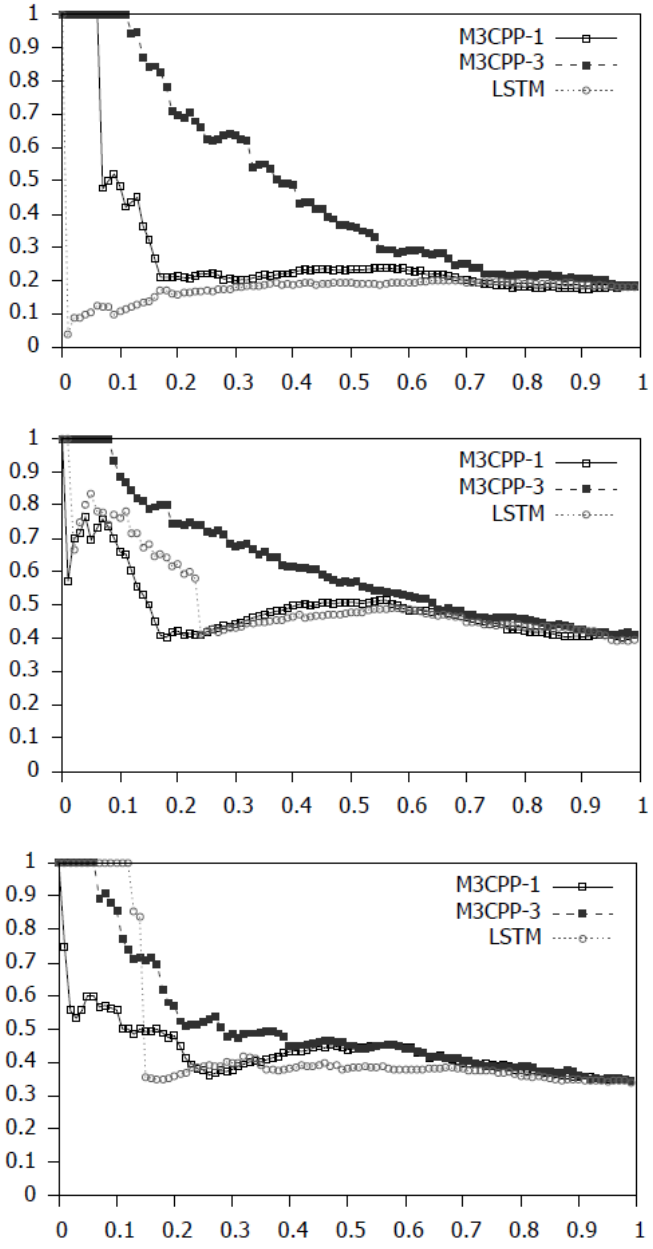

Figure 4.3: Mean Precision-Recall graphs for different window of predictions and lead times, top [1,3] center [1,6] bottom [2,6] with the same step of 120 seconds, for different learning techniques (straight - M3CPP with one local state, dashed - M3CPP with three local states, dotted - LSTM)

(decreasingly) classified as not containing failures. These values can be used to graph a *Precision-Recall curve* and obtain the area under the curve, termed *Area Under Precision-Recall Curve* (AUPRC), which can be used to measure the prediction performance [Hanley and McNeil, 1983, Davis and Goadrich, 2006]. To note that given the unbalance of the classes in the test set (multiple non-failure windows precede

Table 4.2: Prediction performance (AUPRC) of models (M3CPP with one and three local states, LSTM) for different configurations: step 120 and 180 seconds seconds [1,3],[1,6] and [2,6] prediction windows over the entire and halved datasets

Config		M3PC-1		M3PC-3		LSTM	
step	start, end	full	halve	full	halve	full	halve
120s	1 / 3	0.2840	0.2867	0.4731	0.3280	0.1830	0.1614
	1 / 6	0.4895	0.4955	0.4731	0.5072	0.5151	0.5292
	2 / 6	0.4354	0.4421	0.5135	0.4593	0.4646	0.4496
180s	1 / 3	0.2380	0.2785	0.3608	0.2634	0.3042	0.1954
	1 / 6	0.5743	0.5657	0.6127	0.5908	0.5580	0.4482
	2 / 6	0.4675	0.4994	0.5396	0.4969	0.5246	0.4174

the failure ones), accuracy would not have been a good measure. For each model learned on each dataset, we discretize the $[0, 1]$ range of the Recall in 100 slots and compute the corresponding Precision by gradually lowering the threshold.

Figure 4.3 contains the mean Precision-Recall graphs for selected configurations of the window of prediction given the fixed step of 120 seconds to compare visually. All the AUPRC of the different configurations for step, window of prediction and size of the dataset, are summarized in Table 4.2. In each graph the straight line represents M3CPP with a single local state per macrostate; the dashed line represents M3CPP with three local states per macrostate; and the dotted line represents LSTM.

The graphs in figure 4.3 shows that the three models are comparable with M3CPP-3 outperforming the others. It is clear that changing window sizes and lead times has significant effect on the results.

Overall M3CPP with three local states is superior or comparable to both the single local state version and LSTM. The single innerstate model, more simple, works better when we have little data as it overfits less.

M3CPP models tend to degrade less than LSTM when there is data scarcity, they are more robust in this sense: this is predictable since machine learning approaches are known to require large amount of data to perform.

In conclusion, as with the previous chapter, we find that M3PCC models are always at least comparable to our implementation of LSTM and are affected less by data scarcity.

Chapter 5

Conclusions

5.1 Concluding remarks

In this thesis extensions of MMPPs have been studied and tested on synthetic and real world event-driven datasets against appropriate techniques. In particular we presented EM procedures to learn the parameters of a marked MMPP with left-to-right structure and a marked MMPP enhanced with a Compound Poisson Process and extended to comprise non-markovian sojourn times. Such models have been used to represent degradation processes emitting events with stochastic type and arrival time depending on the current state. The model have been used at runtime to infer the current state of the process from a sequence of observed events and to evaluate the remaining time to the final absorbing state. The approaches have been applied to three different application contexts: 1) predictive analysis of an attack scenario (StuxNet) where the model is compared with variants that do not exploit the information carried either by event types or by the average emission rate in each state; 2) predictive analysis of absorbing Semi-Markov Processes where multiple datasets have been clinically constructed varying key properties to test the validity of the claim that M3PCC are useful even when compared to relatively current techniques like Long Short-Term Recurrent Neural Networks; 3) predictive analysis of failures on two real world case studies of High Performance computing where the M3PCC approach was again tested against LSTM.

Overall, the approach comprises a fully data-driven method that automatically exploits the information available in data, being it in the timing of events and/or in their type, the latter of which can comprise multiple dimensions.

5.2 Questions and future studies

We have found that M3CPPs are able to predict failures but are still limited in the information they can accept: when subsequent events are dependent on one an-

other for example the when we have linked events like opening and closing a signal no structuring of the transition graph is enough for the M3CPP to learn the relation. This completely cuts off the model from fields like Activity Recognition, where most events belong to a coupling. While the markov property implicitly leads to the absence of memory and thus relative independence of events allowing the model to capture such information would allow its application to a greater number of fields of study.

Finding ways for the model to capture such faucet is thus an important challenge that future studies may tackle.

On the other side, M3CPPs have been proposed along machine learning algorithms as a way to better approximate event-driven processes where the time scale results in difficult discretization. Pressure from enhancements in machine learning algorithms, like the current Attention Mechanisms, requires further study in better M3CPPs or extensions so that their usefulness may not "die in the cradle".

Appendix A

Publications

Journal papers

1. Laura Carnevali, Reinhard German, **Francesco Santoni**, and Enrico Vicario "Compositional Analysis of Hierarchical UML Statecharts", *Transactions on Software Engineering*, 2021. **Candidate's contributions**: Designed part of algorithms and part of proofs,partial design of experiments, implementation and execution of experiments, some writing and editing of paper.

Peer reviewed conference papers

1. Biagi Marco, Carnevali Laura, **Santoni Francesco**, and Vicario Enrico "Hospital Inventory Management through Markov Decision Processes @runtime", *International Conference on Quantitative Evaluation of SysTems* (*QEST*), 2018. **Candidate's contributions**: Design, implementation and execution of all experiments, writing of relative part in the article.

2. Carnevali Laura, **Santoni Francesco**, and Vicario Enrico "Hospital Inventory Management through Markov Decision Processes @runtime", *Learning marked Markov modulated Poisson processes for online predictive analysis of attack scenarios*, 2019. **Candidate's contributions**: Designed algorithms and proofs,partial design of experiments, implementation and execution of experiments, some writing and editing of paper.

Bibliography

[Almasi et al., 2008] Almasi, G., Asaad, S., Bellofatto, R. E., Bickford, H. R., Blumrich, M. A., Brezzo, B., Bright, A. A., Brunheroto, J. R., Castanos, J. G., Chen, D., et al. (2008). Overview of the ibm blue gene/p project. *IBM Journal of Research and Development*, 52(1-2):199–220.

[Arnold et al., 2014a] Arnold, F., Hermanns, H., Pulungan, R., and Stoelinga, M. (2014a). Time-dependent analysis of attacks. In *Principles of Security and Trust*, pages 285–305, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Arnold et al., 2014b] Arnold, F., Hermanns, H., Pulungan, R., and Stoelinga, M. (2014b). Time-dependent analysis of attacks. In *International Conference on Principles of Security and Trust*, pages 285–305. Springer.

[Asmussen et al., 1996] Asmussen, S., Nerman, O., and Olsson, M. (1996). Fitting phase-type distributions via the em algorithm. *Scandinavian Journal of Statistics*, pages 419–441.

[Baum et al., 1970] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171.

[Biagi et al., 2017] Biagi, M., Carnevali, L., Paolieri, M., and Vicario, E. (2017). An introduction to the ORIS tool. In *VALUETOOLS*, pages 9–11. ACM.

[Bilmes et al., 1998] Bilmes, J. A. et al. (1998). A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126.

[Box et al., 2015] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

[Buchholz, 2003] Buchholz, P. (2003). An EM-Algorithm for MAP Fitting from Real Traffic Data. In Kemper, P. and Sanders, W. H., editors, *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 218–236, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Buchholz et al., 2010] Buchholz, P., Kemper, P., and Kriege, J. (2010). Multi-class Markovian arrival processes and their parameter fitting. *Performance Evaluation*, 67(11):1092–1106.

[Bui et al., 2004] Bui, H. H., Phung, D. Q., and Venkatesh, S. (2004). Hierarchical hidden markov models with general state hierarchy. In *Proceedings of the national conference on artificial intelligence*, pages 324–329. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[Calinescu et al., 2012] Calinescu, R., Ghezzi, C., Kwiatkowska, M., and Mirandola, R. (2012). Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77.

[Carnevali et al., 2019] Carnevali, L., Santoni, F., and Vicario, E. (2019). Learning marked markov modulated poisson processes for online predictive analysis of attack scenarios. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 195–205. IEEE.

[Casale et al., 2009] Casale, G., Mi, N., and Smirni, E. (2009). Model-driven system capacity planning under workload burstiness. *IEEE Transactions on Computers*, 59(1):66–80.

[Casale et al., 2016] Casale, G., Sansottera, A., and Cremonesi, P. (2016). Compact markov-modulated models for multiclass trace fitting. *European Journal of Operational Research*, 255(3):822–833.

[Casale et al., 2008] Casale, G., Zhang, E. Z., and Smirni, E. (2008). KPC-toolbox: Simple yet effective trace fitting using markovian arrival processes. In *Int. Conf. on Quantitative Evaluation of Systems*, pages 83–92. IEEE.

[Chen et al., 2013] Chen, X., Lu, C.-D., and Pattabiraman, K. (2013). Predicting job completion times using system logs in supercomputing clusters. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–8. IEEE.

[Collins, 2013] Collins, J. (2013). Passing the torch from intrepid to mira.

[Cotroneo et al., 2014] Cotroneo, D., Natella, R., Pietrantuono, R., and Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems*, 10(1):8.

[Cumani, 1982] Cumani, A. (1982). On the canonical representation of homogeneous markov processes modelling failure-time distributions. *Microelectronics Reliability*, 22(3):583–602.

[Data,] Data, A. L. C. F. P.

[Davis and Goadrich, 2006] Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

[DeLong et al., 1988] DeLong, E. R., DeLong, D. M., and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

[Fischer and Meier-Hellstern, 1993] Fischer, W. and Meier-Hellstern, K. (1993). The Markov-modulated Poisson process (MMPP) cookbook. *Performance evaluation*, 18(2):149–171.

[Fox and Glynn, 1988] Fox, B. L. and Glynn, P. W. (1988). Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445.

[Gers et al., 1999] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[Hajian-Tilaki, 2013] Hajian-Tilaki, K. (2013). Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian journal of internal medicine*, 4(2):627.

[Hanley and McNeil, 1983] Hanley, J. A. and McNeil, B. J. (1983). A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148(3):839–843.

[He and Neuts, 1998] He, Q.-M. and Neuts, M. F. (1998). Markov chains with marked transitions. *Stochastic proc. and their applications*, 74(1):37–52.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Horváth and Telek, 2002] Horváth, A. and Telek, M. (2002). Phfit: A general phase-type fitting tool. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 82–91. Springer.

[Kriaa et al., 2012] Kriaa, S., Bouissou, M., and Pièvre-Cambacédès, L. (2012). Modeling the Stuxnet attack with BDMP: Towards more formal risk assessments. In *Int. Conf. on Risks and Security of Internet and Systems*, pages 1–8.

[Kriege and Buchholz, 2010] Kriege, J. and Buchholz, P. (2010). An empirical comparison of map fitting algorithms. In *Int. GI/ITG Conf. on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 259–273. Springer.

[Lee et al., 2017] Lee, H., Ge, R., Ma, T., Risteski, A., and Arora, S. (2017). On the ability of neural nets to express distributions. *arXiv preprint arXiv:1702.07028*.

[Li et al., 2008] Li, Y., Lan, Z., Gujrati, P., and Sun, X.-H. (2008). Fault-aware runtime strategies for high-performance computing. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):460–473.

[Liang et al., 2005] Liang, Y., Zhang, Y., Sivasubramaniam, A., Sahoo, R. K., Moreira, J., and Gupta, M. (2005). Filtering failure logs for a bluegene/l prototype. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 476–485. IEEE.

[Liu et al., 2015] Liu, Y.-Y., Li, S., Li, F., Song, L., and Rehg, J. M. (2015). Efficient learning of continuous-time hidden markov models for disease progression. In *Advances in Neural Information Processing Systems 28*, pages 3600–3608. Curran Associates, Inc.

[Lucantoni et al., 1990] Lucantoni, D. M., Meier-Hellstern, K. S., and Neuts, M. F. (1990). A single-server queue with server vacations and a class of non-renewal arrival processes. *Advances in Applied Probability*, 22(3):676–705.

[Malhotra et al., 2015] Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain.

[Murphy, 2002] Murphy, K. P. (2002). *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, Univ. of California, Berkeley.

[Neuts,] Neuts, M. F. *Structured stochastic matrices of M/G/1 type and their applications. 1989*.

[Neuts, 1979] Neuts, M. F. (1979). A versatile markovian point process. *Journal of Applied Probability*, 16(4):764–779.

[Neuts, 1981] Neuts, M. F. (1981). Matrix-geometric solutions in stochastic models (johns hopkins ser. math. sci. 2).

[Okamura et al., 2009] Okamura, H., Dohi, T., and Trivedi, K. S. (2009). Markovian arrival process parameter estimation with group data. *IEEE/ACM Transactions on networking*, 17(4):1326–1339.

[Okamura et al., 2007] Okamura, H., Kamahara, Y., and Dohi, T. (2007). Estimating markov-modulated compound poisson processes. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–8.

[Oliner and Stearley, 2007] Oliner, A. and Stearley, J. (2007). What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584. IEEE.

[Ordóñez and Roggen, 2016] Ordóñez, F. J. and Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115.

[ORIS Tool, 2019] ORIS Tool (2019). Homepage. `http://www.oris-tool.org`.

[Perez-Palacin et al., 2012] Perez-Palacin, D., Merseguer, J., and Mirandola, R. (2012). Analysis of bursty workload-aware self-adaptive systems. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 75–84.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286.

[Ramesh, 1995] Ramesh, N. (1995). Statistical analysis on markov-modulated poisson processes. *Environmetrics*, 6(2):165–179.

[Reinecke et al., 2013] Reinecke, P., Krauß, T., and Wolter, K. (2013). Phase-type fitting using hyperstar. In *European Workshop on Performance Engineering*, pages 164–175. Springer.

[Rydén, 1996] Rydén, T. (1996). An em algorithm for estimation in markov-modulated poisson processes. *Computational Statistics & Data Analysis*, 21(4):431–447.

[Salfner, 2006] Salfner, F. (2006). *Modeling event-driven time series with generalized hidden semi-Markov models*. Techical Repport 208. Department of Computer Science, Humboldt-Universität zu Berlin, Berlin, Germany.

[Salfner et al., 2010] Salfner, F., Lenk, M., and Malek, M. (2010). A survey of online failure prediction methods. *ACM Comp. Surv.*, 42(3):10:1–10:42.

[Salfner and Malek, 2007] Salfner, F. and Malek, M. (2007). Using hidden semi-Markov models for effective online failure prediction. In *Int. Symposium on Reliable Distributed Systems*, pages 161–174. IEEE.

[SIRIO Library, 2018] SIRIO Library (2018). `https://github.com/oris-tool/sirio`.

[Stewart, 1994] Stewart, W. J. (1994). *Introduction to the numerical solution of Markov chains*. Princeton University Press.

[Telek and Horváth, 2007] Telek, M. and Horváth, G. (2007). A minimal representation of Markov arrival processes and a moments matching method. *Performance Evaluation*, 64(9-12):1153–1168.

[van Kasteren et al., 2008] van Kasteren, T., Noulas, A., Englebienne, G., and Kröse, B. (2008). Accurate activity recognition in a home setting. In *Proc. Int. Conf. on Ubiquitous Computing*, UbiComp '08, pages 1–9. ACM.

[Verma and Anand, 2007] Verma, A. and Anand, A. (2007). General store placement for response time minimization in parallel disks. *Journal of Parallel and Distributed Computing*, 67(12):1286–1300.

[Vicario et al., 2009] Vicario, E., Sassoli, L., and Carnevali, L. (2009). Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. Softw. Eng.*, 35(5):703–719.

[Wei et al., 2002] Wei, W., Wang, B., and Towsley, D. (2002). Continuous-time hidden Markov models for network performance evaluation. *Performance Evaluation*, 49(1):129–146.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Yu et al., 2011] Yu, L., Zheng, Z., Lan, Z., and Coghlan, S. (2011). Practical online failure prediction for blue gene/p: Period-based vs event-driven. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 259–264. IEEE.

[Zen, 2015] Zen, H. (2015). Acoustic modeling in statistical parametric speech synthesis-from hmm to lstm-rnn.

[Zheng et al., 2009] Zheng, Z., Lan, Z., Park, B. H., and Geist, A. (2009). System log pre-processing to improve failure prediction. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 572–577. IEEE.

[Zheng et al., 2012] Zheng, Z., Yu, L., Lan, Z., and Jones, T. (2012). 3-dimensional root cause diagnosis via co-analysis. In *Proceedings of the 9th international conference on Autonomic computing*, pages 181–190.

[Zheng et al., 2011] Zheng, Z., Yu, L., Tang, W., Lan, Z., Gupta, R., Desai, N., Coghlan, S., and Buettner, D. (2011). Co-analysis of ras log and job log on blue gene/p. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 840–851. IEEE.

[Zhu and Goldberg, 2009] Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.