PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FITTING TECHNIQUES TO KNOWLEDGE DISCOVERY THROUGH STOCHASTIC MODELS

JOAQUIM VINICIUS CARVALHO ASSUNÇÃO

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Pontifical Catholic University of Rio Grande do Sul.

Advisor: Paulo Henrique Lemelle Fernandes
Co-advisor: Jean-Marc Vincent

Porto Alegre
2016

Ficha Catalográfica

A851f Assunção, Joaquim Vinicius Carvalho

Fitting techniques to knowledge discovery through stochastic models / Joaquim Vinicius Carvalho Assunção . – 2016.
152 f.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. paulo Henrique lemelle Fernandes.
Co-orientador: Prof. Dr. Jean-Marc Vincent.

1. Modelos estocásticos. 2. Ajuste de modelos. 3. Descoberta de conhecimento. 4. Séries temporais. 5. Formalismos estruturados. I. Fernandes, paulo Henrique lemelle. II. Vincent, Jean-Marc. III. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).

Pontifícia Universidade Católica do Rio Grande do Sul
SCHOOL OF COMPUTER SCIENCE
GRADUATE PROGRAM IN COMPUTER SCIENCE

THESIS DOCTORAL REPORT

Thesis entitled "Fitting Techniques to Knowledge Discovery Through Stochastic Models" defended by Joaquim Vinicius Carvalho Assunção in support of the degree of Doctor of Computer Science, sanctioned by the Examining Committee on August 09, 2016:

Dr. Paulo Henrique Lemelle Fernandes — PPGCC/PUCRS
Advisor

Dr. Jean-Marc Vincent — Université Joseph Fourier - Grenoble/France
Co-advisor

Dr. Felipe Rech Meneguzzi – PPGCC/PUCRS

Dr. Alberto Avritzer – Sonatype /United States

Júlio César Nievola

Dr. Júlio César Nievola – PPGIA-PUCPR

Sanctioned on 17/01/2017, as per Minutes No. 001 by the Examining Committee.

Prof. Dr. Luiz Gustavo Leão Fernandes
Chair

PUCRS PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Main Campus
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Phone: (51) 3320-3611 – Fax (51) 3320-3621
Email: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Praise be to the enlightenment,
to the knowledge and the rational thinking;
which separates us from the other living beings in the known World.

...

For all those devoted to science,
this is a tiny and humble contribution to us.

ACKNOWLEDGEMENTS

First, I would like to express my gratitude for my Advisor, Paulo Fernandes, which supported me through all these years since my Master's; which made a direct impact on this work, making it possible. Thank you, Paulo, for buying my ideas and always be receptive. The second person that has direct contribution is the quality of this work is my co-advisor, Jean-Marc Vincent, which was my host regarding my sandwich period, in Grenoble. I have had a great time learning and discussing important matters which had a direct impact on the quality of this work. Paulo and Jean-Marc, it was an honor work with great professionals and great persons like you.

I also would like to thank all the great people that had a direct participation in my work, my co-authors, and reviewers: Afonso Sales, Alan Santos, Angelika Studeny, Lucelene Lopes, Luciana Espindola, Maria Pivel, Silvio Normey Gomez, and Tiago Vier Fischer.

Others made an indirectly, yet meaningful participations in this journey. Friendly persons which I had a pleasure to share my daily life. For all my colleagues and friends from PUCRS and Inria, thank you! Others, that were important in my life in this period and shared some ideas. Those who do not have a directly importance to this work, yet a great importance to keep me strong and focused on this journey, friends and relatives, thank you! Others, from years ago, but not less important, for the ones who opened my view for the computer science world. Those who inspired me to enroll in a Master course, my graduation professors, thank you!

Finally, I would like to thank my jury, those who spent their time to read and evaluate my work. Alberto Avritzer, Júlio César Nievola, and Felipe Meneguzzi. Thank you for your comments and evaluation.

FITTING TECHNIQUES TO KNOWLEDGE DISCOVERY THROUGH STOCHASTIC MODELS

RESUMO

Modelos estocásticos podem ser úteis para representar de maneira compacta cenários não determinísticos. Além disso, simulações aplicadas em um modelo compacto são mais rápidas e demandam menos recursos computacionais do que técnicas de mineração em grandes volumes de dados. O desafio está na construção desses modelos. A acurácia, juntamente com tempo e a quantidade de recursos usados para ajustar um modelo são fatores chave para sua utilidade. Tratamos aqui de técnicas de aprendizado de máquina para ajustes de estruturas com a propriedade de Markov; especialmente formalismos complexos como Modelos Ocultos de Markov (HMM) e Redes de Automatos Estocásticos (SAN). Quanto a acurácia, levamos em consideração as atuais técnicas de ajuste, e medidas baseadas em verossimilhança. Quanto ao tempo de criação, automatizamos o processo de mapeamento de dados via séries temporais e técnicas de representação. Quanto aos recursos computacionais, usamos séries temporais e técnicas de redução de dimensionalidade, evitando assim, problemas com a explosão do espaço de estados. Tais técnicas são demonstradas em um processo que incorpora uma série de passos comuns para o ajuste de modelos com séries temporais. Algo semelhante ao que o processo de descoberta de conhecimento em banco de dados (KDD) faz; porém, tendo como componente principal, modelos estocásticos.

Palavras-chave: Modelos estocásticos, ajuste de modelos; descoberta de conhecimento; séries temporais; modelagem computacional; formalismos estruturados; soluções numéricas; cadeias de Markov; redes de autômatos estocásticos; modelos Markovianos ocultos; previsão estatística; ferramentas computacionais.

FITTING TECHNIQUES TO KNOWLEDGE DISCOVERY THROUGH STOCHASTIC MODELS

ABSTRACT

Stochastic models might be useful for creating compact representations of non-deterministic scenarios. Furthermore, simulations applied to a compact model, are faster and require fewer computational resources than the use of data mining techniques over large volumes of data. The challenge is to build such models. The accuracy as well as the time and the amount of resources used to fit such models, are the key factors related to their utility. We use machine learning techniques for the fitting of structures characterized by a Markov property; especially, complex formalisms such as Hidden Markov Models (HMM) and Stochastic Automata Networks (SAN). Regarding the accuracy, we considered the state of the art on fitting techniques and model measurements based on likelihood. Regarding the computational resources, we used time series and dimensionality reduction techniques to avoid the space state explosion. Such techniques are demonstrated in a process that embodies a set of common steps for the model fitting through time series. Similar to the knowledge discovery in databases (KDD), yet using stochastic models as a main component.

Keywords: Stochastic models, model fitting; knowledge discovery; time series; computational modeling; structured formalisms; numerical solutions; Markov chains; stochastic automata networks; hidden Markov models; statistics; computational tools.

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ALGORITHMS

LIST OF ACRONYMS

KDD	*Knowledge Discovery in Databases*
MC	*Markov Chain*
DTMC	*Discrete Time Markov Chain*
CTMC	*Continuous Time Markov Chain*
TPM	*Transition Probability Matrix*
TS	*Time Series*
SAX	*Symbolic Aggregate ApproXimation*
DTW	*Dynamic Time Warping*
HMMs	*Hidden Markov Models*
AIC	*Akaike's information criterion*
BIC	*Bayesian information criterion*
BW	*Baum-Welch*
EM	*Expectation-Maximization*
MLE	*Maximum Likelihood Estimation*
SAN	*Stochastic Automata Networks*
Ma	*Mega-annum, Million years*
SANGE	*Stochastic Automata Networks Generator*
NLP	*Natural Language Processing*
POS	*Part-Of-Speech*
TPM	*Transition Probability Matrix*

SUMMARY

Chapter 1

The need for fitting stochastic models

In the information age, knowledge emerges as a more and more valuable commodity. A commodity that is generated by science and technology; whose foundations are increasingly generating, and being based on, data. However, these amounts of data are growing in a way we can no longer analyze it by ourselves. We need to use specialized computing algorithms.

Data mining and machine learning algorithms are broadly used by computer scientists in an attempt to discover information that, due to its volume, seems invisible to a human perception. However, some situations require continuously simulation of events from large sets of data. Others that are based on events which are apparently random, yet they lie on probabilities and can be described by statistical analysis; we call these "stochastic events". In these cases, representative data can be used to create a model of the whole system. These models can also make use of stochastic techniques to better perform with stochastic events. Once a model accurately represents a system, retrieve probabilities to statistical analysis is faster and practical than perform continuous complex analysis in the whole system.

There are many modeling formalisms and they are designed for slightly different purposes, therefore, they have different structures. All these structures are filled with parameters that should adequately describe the original data (or observations). The better are the parameters to describe the original data, the better shall be the performance of the created model. The process of adapting parameters according to the input data, for a particular model structure, is called model fitting.

Fitting can be done manually or automatically. However, from now on, we shall use the word fitting always referring as an automatic fitting. The challenges involved in any fitting goes far as the complexity of a given formalism's structure. Furthermore, there are well-known problems with handling data, which are involved in fitting models; yet, they are slightly different due to the structure of each formalism.

Our objective is to generate Stochastic Automata Networks (SAN) through time series data. Our motivation is the current lack of fitting methods for this formalism, and the challenge to incorporate big data directly into a Markov model. Our contribution is the union of techniques from different domains to transform data from time series to complex model structures such as SAN.

The next Section (1.1) shows the similarities between statistical inference through model's probabilistic analysis and data mining. Section 1.2 shows a brief introduction to modeling with emphasis on Markovian models, followed by basic concepts on model fitting. Finally, we expose the questions that drive this work (Section 1.3), along with the organization of this Thesis (1.4).

1.1 Knowledge discovery and modeling

The amount of data in the world, in our lives, seems to go on and on increasing [122]. Positively and high correlated with the increase of data volume, is the common difficulty of handling these data. Among the frequently faced problems, there is the difficult to detect information among such large amounts of data. To cope with this problem, some specialized data mining algorithms which retrieve sets of informations that are practically invisible to our perception.

Due to the difficulty and time spending in modeling, some advantages provided by statistical analysis techniques are not commonly used in data mining. However, data mining techniques have much more practical approaches and its algorithms are well covered by the literature [122] [4] [17] [121] [69] [111]. When we talk about knowledge discovery, we talk about knowledge discovery on data, and especially attractive for this work is the process of knowledge discovery in databases (KDD), which is composed of important knowledge discovery steps, including data mining.

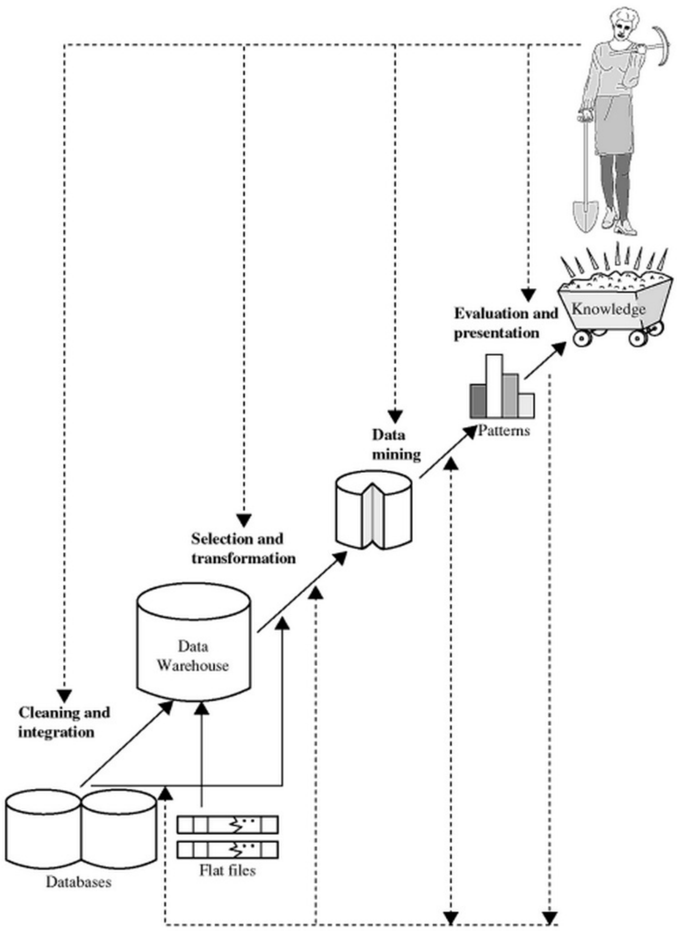


Figure 1.1: Process of KDD, according to Han *et al.* [69]

Figure 1.1 shows an interactive sequence of steps that usually drives the user to knowledge discovery through data mining. Between a database and the new knowledge to be discovered, according to Han *et al.* [69] there are basically four steps:

1. Cleaning and integration: used to remove noise and combine databases (if more than one exists).

2. Data selection and transformation: used to select relevant data and transform the format of data into a form more suitable for data mining.

3. Data mining: a necessary process that consists on run intelligent algorithms to retrieve possible patterns. There are many kinds of algorithms and different approaches to using it.

4. Evaluation: to identify patterns of interest achieved in the data mining step.

The KDD process has a well defined set of steps to handle data before execute data mining tasks. These steps aim to transform, clean and organize data in order to have a tidy (or close to) dataset. Although this process is well consolidated and often used as a guide for knowledge discovery, instead of describing the path to discovering knowledge, it seems to have been created to cover the steps that usually come before and after data mining.

Data mining and modeling have some similarities. Both fields have a goal to discover, or describe, information given some set of data. Both areas are continually improving its capabilities to handle data volume and data type. Furthermore, both are used as a core technology to discover knowledge from data. Although, unlike the data mining algorithms, stochastic modeling formalisms and its algorithms are not so widely used. In fact, these techniques are employed by a relatively small community of specialists. Generally by statisticians that are familiar with these kinds of analysis and computer scientists working with computer mathematics.

Modeling formalisms are structures (formalism) to represent a given system (model), making possible to simulate and retrieve probabilities about a given condition or state of this system. There is great potential in use these features to analyze and forecast data. However, today, stochastic modeling are not widely used as they should be. Some of the reasons are; the difficulty in understanding these formalisms and its mathematics, the time used to create a model, and the state space explosion.

Due to be created in another research field and to be focused on data mining, KDD does not reflect a process to discover knowledge through stochastic modeling. Any stochastic formalism has a structure that is made according to the rules of a stochastic process. These structures are quite specific and imply in different input data formats. Also, there are other problems involved that are common to many stochastic modeling formalisms; probably the *state space explosion* is the most well-known phenomenon. In a rough comparison, as well as KDD shows a set of steps commonly used to prepare data for data mining, here is described (Chapter 5) a process that shows useful steps to prepare data for stochastic modeling. These steps aim to decrease the time spent to develop models, handling with the state space problem and reducing the chance of human mistakes while manually developing complex models.

In the same way, that learning algorithms can predict the outcome of data by learning a piece of it, or statistical techniques can provide knowledge about a population via its sample; through a computational model, we can find knowledge through probabilistic methods by inferring how likely is a particular condition. Despite being useful to get information about data, probabilistic techniques, such as stochastic modeling, are not commonly applied to knowledge discovery in large datasets. On the other way, data mining techniques are often used to large datasets, and the KDD is widely accepted as a useful process to handle data from database beginning from the very first steps and passing through the data mining.

Frawley *et al.* [64] defines Knowledge Discovery in Databases (KDD) as "the non-trivial extraction of implicit, previously unknown, and potentially useful information from data". To do so, it relies on artificial intelligence, machine learning, and data mining techniques. In fact, all these fields are strongly related. On the other hand, stochastic modeling is commonly more related with mathematical models and probabilistic techniques. Though these research fields seem to be far from each other, in reality, they use different techniques and methods to achieve a similar purpose, knowledge discovery.

KDD is a process that aims to drive us to knowledge discovery. However, it is traditionally associated with data mining, making its structure unsuitable for structured model formalisms. Such formalisms aim to generate probabilities and statistics, which are ultimately used to knowledge discovery. Statisticians often use models to better understand some data based on data samples. Depending on the sample size, the state space explosion and the time effort in modeling can be a problem. A problem not foreseeable on the KDD representation. Here, we can pose a question: "Can we adapt KDD to a knowledge discovery process that uses stochastic modeling as its core?".

This work does not intend to go further on data mining. Here, we draw necessary comparisons focusing on the challenge to improve the state of the art in knowledge discovery through the improvement of techniques to fit stochastic models. This bridge between KDD and stochastic modeling are explained in the Chapter 5. There, we describe a knowledge discovery process that uses stochastic modeling as well as the KDD uses data mining.

1.2 Towards modeling

In the art of modeling systems, there are many techniques and formalisms available. Among such techniques and formalisms, this work focuses on those that have special characteristics to handle stochastic events. Thus, facing two great divisions, deterministic and stochastic, we shall focus on stochastic. However, in some sections, deterministic models will appear as a helper to understanding some concepts.

Stochastic Modeling can be seen as a set of techniques, methods, and formalisms to represent the behavior of a given stochastic system by describing its states and transitions. This system could be physical or mathematical, using stochastic modeling, it is often possible to represent its behavior and achieve probabilities about its next states [110].

Once a system is well modeled, through the statistical information we can efficiently predict a system behavior. In this context, a system can be the representation of a primary process or even complexes nature's events. Either way, a good model can deliver probabilities about a demanded system event; concerning both its past or its future. However, modeling is a non-trivial task that requires a specialist on the domain with expertise related to the scenario to be modeled.

Another problem frequently faced with stochastic models is the *state space explosion*. The state space of modeled systems can be very large, or even infinite. Thus, the size of a system to be modeled is directly limited to resources of time and memory. However, real world systems usually have large amounts of data; consequently, generating an enormous state space.

In few words, a model fitting (Section 1.2.2) is a technique responsible for making a model represent an input data accurately as possible. Section 1.2.1 describes the most basic Markovian structure, a Discrete Time Markov Chain. Which is an essential background knowledge for all the content from Chapter 3 to Chapter 5.

1.2.1 Markov Chains

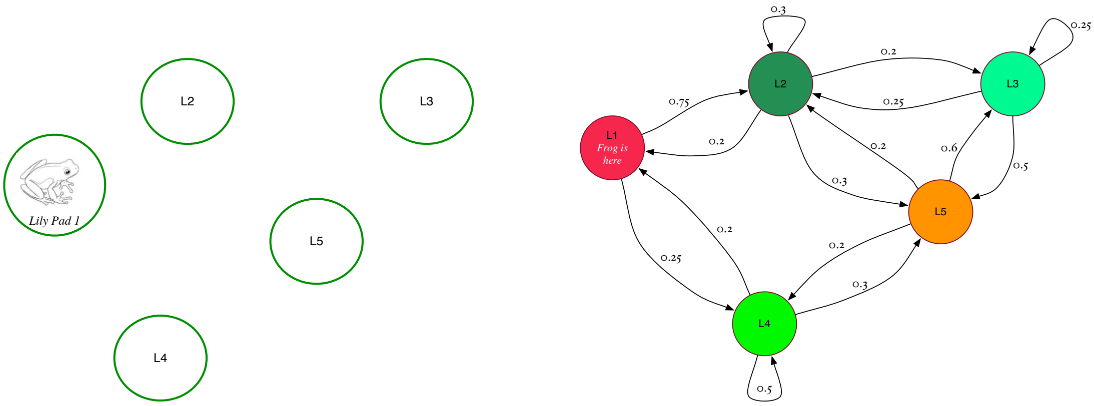
To understand Markov Chains (MCs) it is first necessary to understand the basic concepts of the Markovian process. There are two fundamental concepts: The State concept and the Transition concept. [72]. Furthermore, MCs are divided into two categories, Discrete Time Markov Chain (DTMC) and Continuous Time Markov Chain (CTMC). MCs and DTMC are usually used as synonymous and CTMC are usually, and generally, referred as a Markov process. The fundamental difference between those two types is that DTMC describes a transition between states as a probability while CTMC describes transitions between states as rates of occurrence. Here we focus on DTMC; starting by describing Markov Chains basic properties, among examples of DTMC and how it differs from a CTMC.

Markovian Process and MC

A Markovian state will return only two possibilities, occupied or free. A typical example is a frog on a pond. Each lily pad is a state that can be occupied or not by the frog. It is even possible that each lily pad has a unique characteristic; thus, we can model a system by using a set of features that are connected by transitions.

Markovian transitions are a memory-less set of changes from one state to another or even the same. It only depends on the current state instead of on a history of states. In a pond, each lily pad is a state, and these states are occupied by a frog that has a behavior. So, the behavior of the frog is a set of jumps from one lily pad to another or even the same. In a Markovian process, transitions are assumed to be instantaneous; it means that the frog jumps will consume a negligible time compared to the time that it spends sited on a lily pad [110]. The Markovian property (absence of memory) makes the change of states depending solely on the current state, e.g., as a creature with a short memory, the frog decides where to jump based only on its current position.

Figure 1.2a shows an example of a frog in a pond. In this example, we have five states, being one state is already occupied by the frog (Initial state). Next, we begin to use formal definitions regarding Markov models. The following notation might be handy as a quick guide. Still, we explain all the necessary notation through the text.



(a) Abstracted representation (b) Representation using a Markov Chain

Figure 1.2: Example of a Markov Chain representing the frog on a pond

Main notation

n	Number of states
X, X'	Time series
$\Gamma_{(t)}$	A transition probability matrix at time t
$\phi_{i,j}$	Transition probability from state i to state j
$\phi_{i,j(t)}$	Element i, j of a transition probability matrix $\Gamma_{(t)}$
c_t	A transition occurred in $\Gamma_{(t)}$
C	A sequence describing a stochastic process $C = \{c_1, c_2, ..., c_T\}$
$\varpi_{i(T)}$	The probability of having a Sojourn time of T time steps to a state i
R	A reverse Markov chain
$r_{i,j}$	Transition probability from state i to state j in R

Given an initial state, there are a set of states that can be reachable in the first transition, and once the current state changes, in the second transition and so on. For instance, the frog, cannot jump to the state $L3$, because it is too far from the initial state $L1$. Figure 1.2b shows a possible representation for the frog and its lily pond.

If our frog was immortal, and it never stop jumping, we should have an infinite number of continuous jumps $T = \{t_{-\infty} ... t_{\infty}\}$. The resultant system should be modeled as a CTMC. Although, considering that our frog is a mortal, and behave like such, we shall have a MC with limited number of jumps discrete in time: $T = \{t_1, t_2, t_3 ... t_m\}$; therefore, a discrete time markov chain (DTMC).

Entire MCs are represented by transition probability matrices (TPM). Being n the number of states, i and j identifiers for lines and columns; assuming that $i \subset I, j \subset J$ and $i <= N => j$. We shall represent a matrix $\Gamma_{(t)}$, with a set of probabilities $\phi_{i,j}$ of a given state i goes to j.

$$\Gamma_{(t)} = \begin{bmatrix} \phi_{0,0(t)} & \phi_{0,1(t)} & \cdots & \phi_{0,J(t)} \\ \phi_{1,0(t)} & \phi_{1,1(t)} & \cdots & \phi_{1,J(t)} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{I,0(t)} & \phi_{I,0(t)} & \cdots & \phi_{I,J(t)} \end{bmatrix} \tag{1.1}$$

Table 1.1: Transition probability matrix representing Figure 1.2b

$$\begin{vmatrix} 0 & 0.75 & 0 & 0.25 & 0 \\ 0.2 & 0.3 & 0.2 & 0 & 0.3 \\ 0 & 0.25 & 0.25 & 0 & 0.5 \\ 0.2 & 0 & 0 & 0.5 & 0.3 \\ 0 & 0.2 & 0.6 & 0.2 & 0 \end{vmatrix}$$

We can forecast future states by calculating the probabilities for each possible state for the next t steps [110]. This is given by the Chapman-Kolmogorov equations. Where Γ is the transition probability matrix; i, j and k are indexes for the matrix elements; and t is the time counter. Therefore, the TPM at time t is given by the cumulative sum of the products of the matrices Γ and $\Gamma_{(t-1)}$. This is detailed described on the equation 1.2.

$$\Gamma_{(t)} = \phi_{i,j(t-1)} = \sum_{all\ k} \phi_{i,k}(1)\phi_{k,j(t-1)} \qquad for\ t > 1 \tag{1.2}$$

Abstracting the elements i, j of Γ, we can represent it in a simplified form the Equation 1.2.

$$\Gamma_{(t)} = \Gamma_{(t-1)}\Gamma \tag{1.3}$$

This equation brings us the transient probability for the next jump of the frog. *i.e.*, the set of probabilities of an individual, be in other states of a system in the next move. Using the Matrix form (Table 1.1) it is possible to use the Equation 1.3 to compute the probabilities for the next step of the frog. This equation can be used recursively to find a Steady State, *i.e.*, a state which there is no change on the probabilities or the changes are too small to be considered.

The Sojourn Time

Given a TPM, if a probability of $\phi_{ij} > 0$, being $i = j$, or $\phi_{i,i} > 0$, then this system will possibly remains (with a probability $\phi_{i,i}$) at the same state, state i. The number of consecutive time periods that a MC remains in a given state is called the *sojourn time*. According to the laws of a Markovian Process (as previous described, states and transitions) the probability remains the same independently of the previous transitions. Thus, the probability of a system leaves its state i is given by $\sum \phi_{i,!i} = \sum_{i \neq j} \phi_{i,j} = 1 - \phi_{ii}$. This is known as a sequence of Bernoulli trials, which has a probability of success $1 - \phi_{ii}$ at each trial [110].

Being ϖ_i the Sojourn time to the state i and T the total time in the same state (or the total number of cumulative steps) the total probability is given by $T - 1$ consecutive Bernoulli failures followed by a successful Bernoulli trial $\phi_{i,!i}$.

$$\varpi_{i(T \mid T>1)} = \phi_{i,i}^{T-1} \ \phi_{i,!i} \tag{1.4}$$

The Sojourn Time is a concept that is present in all formalisms ruled by a Markovian behavior. Therefore, it is important to define it, because despite it is almost never mentioned, it is intrinsically present in all Markovian models.

Backward process and Reversibility

Given a sequence of observation X, a Markov chain represented by Γ and its elements $\phi_{i,j}$ draws a sequence of elements in time from i to j. This sequence of drawing elements from a Markov chain in time Γ_t can be defined by c_t. Thus we can define the sequence of drawing elements from a DTMC as a $\{c_{(t)}, t = 0, 1, 2, ..., T\}$, as a stochastic process that satisfies the Markovian property.

A MC evolution usually is given by $c_{(0)}, c_{(1)}, ..., c_{(T)}$, which is called a forward process. A backward process is nothing more than the reverse execution of a forward process. For instance, $c_{(T)}, c_{(T-1)}, ..., c_{(0)}$ or $c_{(0)}, c_{(1)}, ..., c_{(-273)}$. Usually, forward and backward processes are different from each other. Yet, in special cases both can be stochastically identical; we call this a reversible process [110].

It is important to emphasize that all MCs could be reversed (Equation 1.5), but not all MCs are reversible. This means it is always possible to reverse a chain, yet not always the reversed chain will represent the reverse flow of a system [77]. Being π the stationary distribution, $r_{i,j}$ a step transition for the reversed Markov chain R, and $\phi_{i,j}$ an observation in the probability matrix. The Reversed Markov Chain is given by the Equation 1.5 [110].

$$\sum_{all\ j} r_{ij} = \frac{1}{\pi_i} \sum_{all j} \pi_j \ \phi_{ji} = \frac{1}{\pi_i} \pi_i = 1 \quad \forall i \tag{1.5}$$

Which can be represented in a short matrix form:

$$R = diag\{\pi\}^{-1} transp\{\Gamma\} diag\{\pi\}$$

Thus, following our frog example, we use Equation 1.2 to find π. Then, we use Γ's dimensions to setup a diagonal matrix.

$$\Gamma = \begin{bmatrix} 0 & 0.75 & 0 & 0.25 & 0 \\ 0.2 & 0.3 & 0.2 & 0 & 0.3 \\ 0 & 0.25 & 0.25 & 0 & 0.5 \\ 0.2 & 0 & 0 & 0.5 & 0.3 \\ 0 & 0.2 & 0.6 & 0.2 & 0 \end{bmatrix}$$

$$\pi = (0.0799,\ 0.25423,\ 0.27118,\ 0.14124,\ 0.25423)$$

$$diag\{\pi\}^{-1} = \begin{bmatrix} 12.643 & 0.000 & 0.000 & 0.00 & 0.000 \\ 0.000 & 3.933 & 0.000 & 0.00 & 0.000 \\ 0.000 & 0.000 & 3.688 & 0.00 & 0.000 \\ 0.000 & 0.000 & 0.000 & 7.08 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.00 & 3.933 \end{bmatrix}$$

$$R = \ diag\{\pi\}^{-1} \ transp(\Gamma) \ diag\{\pi\}$$

$$R = \begin{bmatrix} 0.000 & 0.643 & 0.000 & 0.357 & 0.000 \\ 0.233 & 0.300 & 0.267 & 0.000 & 0.200 \\ 0.000 & 0.188 & 0.250 & 0.000 & 0.562 \\ 0.140 & 0.000 & 0.000 & 0.500 & 0.360 \\ 0.000 & 0.300 & 0.533 & 0.167 & 0.000 \end{bmatrix}$$

As a proof of correctness to this equation, R must be a stochastic matrix, all elements must be smaller than one and all rows sum must be equal to 1. Finally, observing the reversible matrix R we can confirm that our matrix is not reversible because $r_{i,j} \neq \phi_{i,j} \forall (i,j)$, or simple $\Gamma \neq R$. A MC reversibility can also be tested by the *detailed balance equations* [110]. It is true if it satisfy: $\pi_i \phi_{ij} = \pi_j \phi_{ji} \ \ \forall i, j$.

1.2.2 Model fitting

Model fitting is the process of adjusting model parameters according to an input data. More precisely, a technique involving a machine learning algorithm that learns the data patterns creating a model that best represents a given data sequence or system behavior.

This section aims to show the very basics of fitting for few different models. All the descriptions are basic concepts that share the same goal. Furthermore, these concepts are used here in the techniques described in the Section 3.2 and Chapter 5.

As a toy example, let us consider a sequence X containing T observations, $X = \{x_1, x_2, ..., x_T\}$. Which MC better describe this sequence? To answer this question, we first need to define a formalism. *I.e.*, the kind of model technique we will use. Supposing a Markov Chain as a formalism and X a sequence that always generates ones followed by zeroes and *vice-versa*. $X = \{0, 1, 0, 1, 0, 1, 0, 1, ...\}$ as a input. A perfect model to describe this sequence is a two-state Markov Chain (Represented by Γ and starting by its first state) as follows:

$$\Gamma = \begin{bmatrix} \phi_{0,0} = 0 & \phi_{0,1} = 1 \\ \phi_{1,0} = 1 & \phi_{1,1} = 0 \end{bmatrix} \tag{1.6}$$

We consider this model the "perfect" because it has a perfectly accuracy, it means that each time the first state $\phi_{0,0}$ is active, the second state $\phi_{0,1}$ will be active in the next transition. Therefore, the probability P of Γ be in the state $S_1 = \{\phi_{0,1}, \phi_{1,1}\}$ given $S(0)$ is 100%, formally $P(S_1|S_0) = 1$ and $P(S_0|S_1) = 1$. Thus, given the initial state S_0 this model will represent exactly the same records of X using a minimum number of states (2); an optimal maximum likelihood. We discuss the likelihood details in Section 3.1.2.

As a second example, consider one of the most common models for data, linear models. Linear models, or linear regression, are relatively simple to generate and analyze. Given a sequence S of T observations, $S = \{s_1, s_2, ..., s_T\}$ we can draw a regression line that represents the tendency of the data. This is also called, regression analysis. In the Figure 1.3 a linear model representing the price of diamonds according to its weight.

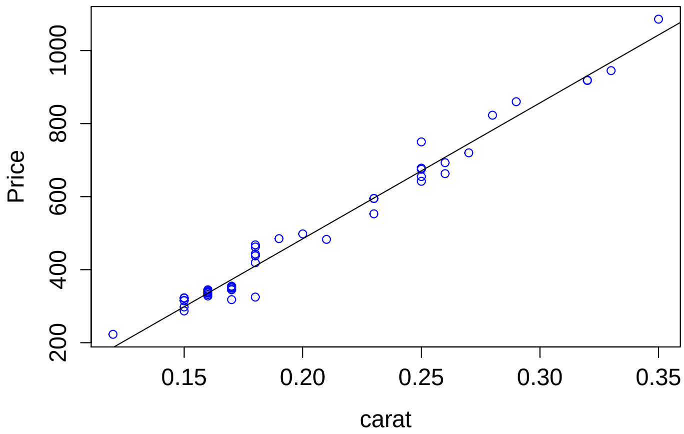


Figure 1.3: Basic linear fitting; diamonds price

In this case, few values were collected to best describe a line between the data. *I.e.*, for any new value, we expect it to be near this line. In this case, only 1 parameter are needed to describe this model, a vector of coordinates in Cartesian plan: $Xi, Yi; Xf, Yf$.

However, in other cases, we might find values that do not follow a linear pattern or are not ruled by a deterministic law. *E.g.*, the number of earthquakes per month in the last 200 years. This will generate a set $X = \{x_1, x_2, ..., x_{2400}\}$ with a non linear pattern or deterministic rule. Filters can be applied, such as, "focus on the earthquakes with intensity bigger than 7.0" or "just take a few records as a training set".

Figure 1.4 shows a simple representation of the number of earthquakes with intensity bigger than 7.0. However, a model like this needs too many parameters and fails to represent the behavior of the elements due two main reasons. First, the range of the values is too far from the modeled ones. Second, the behavior of the data is not deterministic. *I.e.*, earthquakes follow complex rules, making then an apparently random phenomenon. Among other reasons, these characteristics lead this earthquake model to a poor accuracy.

In cases, such, stochastic models might have a better accuracy due to its lack of bias and bases on probabilistic techniques. Furthermore, sometimes a model with few parameters can have

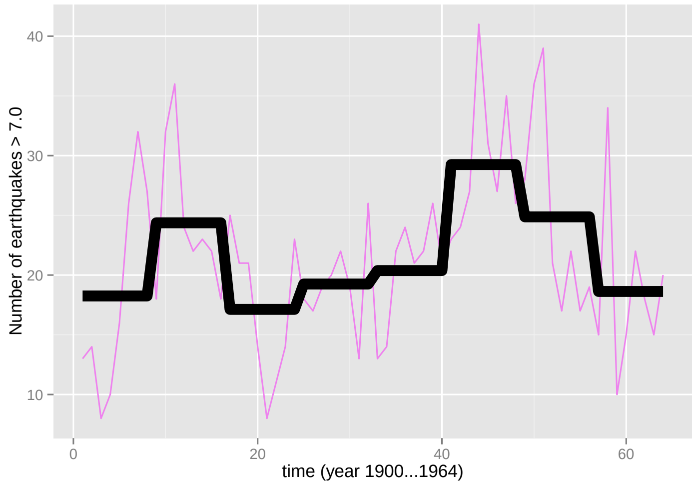


Figure 1.4: Earthquake, time series modeling representation

a performance near to the ones described by a bigger set of parameters. This leads us to the basis of Markovian models and numeric solutions [109] (Section 1.2.1). In fact, stochastic formalisms such as Hidden Markov Models are broadly used for time series data where the stochasticity are present [127]. They allow us to represent a large set of possibilities generated by a stochastic pattern in ways that some forecast driven through the model should be close to some real value. Note that there are other formalisms that could be considered, yet we focus on stochastic models, more specifically our examples are driven for model fitting in Markovian models.

Model fitting is a fundamental characteristic of models used in real time. indeed, an automated fitting means less effort from the modelers, thus allowing a better use of time, which leads to a better performance.

In these examples, a simple MC was fitted to represent a sequence of zeros followed by ones. In the next section, we change to a more robust formalism, going deeply into a more complex structure and fitting techniques. We also illustrate a representation using earthquake data X. Data that are commonly treated as a time series, because they are a sequence of values in time. In fact, we can see that the diamonds data S has the same structure, only indexed by its carats instead of time; therefore, the solution applied to X might be useful for data structured like S. The use of time series methods is described at Chapter 2 and its applications in fitting methods is discussed along Section 3.2 and Chapter 5.

1.3 Research questions

Studer *et al.* [111] says that “every model is only an approximation of the reality". Like writing a thesis, modeling the reality always can be improved; thus, a perfect model of the reality does not exist, it always can be enhanced. Now, let us suppose that we have a set of algorithms. These algorithms follow a structured process and can save time from handmade steps; but to do this, they sacrifice a small percentage of precision. Thus, a question that drives this work is: *“Can* *we effectively automate some steps, making the time saved compensate the loss of precision?"*. Automation such as the ones that are possible with the fittings demonstrated in the Section 1.2.2. Yet, not a toy example like MC 1.6, or a simple ordered data like in Figure 1.3, or poor accurate as Figure 1.4. The goal is to use more advanced formalisms with general tidy data. Data which its variables are related and indexed. For instance, an earthquake dataset, containing data such as Figure 1.4 and other related such as earthquakes' intensity, area, location.

The more complex is the scenario to be modeled, the more difficult to create a stochastic model; therefore more time to achieve a proper fitness. Considering the lifetime and accuracy of a modeling process, we can pose the question. *When the time spending on improving a model becomes too expensive?* In fact, "the modeling process is a cyclic process. New observations may lead to a refinement, modification or completion of the already built-up model." [111]. Thus, the time spent to create a model and the amount of data that a model can represent are both key features to improve the modeling process.

Usually, data are carefully (thus slowly) analyzed by a human, or humans, that has some expertise about the data and the necessary modeling techniques. The slow process of analyze some data improve the accuracy of a model and give to a modeler some knowledge about the data, allowing him to represent the system with a smaller model.

In modeling, size control is important to keep the model useful concerning computational cost. Therefore, usually the bigger the number of states in a model, the bigger is the computational cost to solve it. The number of states in a model can easily reach millions, and this is well known and explained by a phenomenon called *state space explosion*.

Since controlling the size is crucial, and a human analysis takes larger amounts of time; it is challenging to solve this problem; "*How can the creation of Stochastic models deviated from large amounts of data be more orthodox?*". And here we speak "orthodox" in the sense of practice, a common way to be created, without the need of crafting performed by specialists working in it for many days or weeks. In a few words, a process of flexible, generic (in the sense of not restricted to some specific data) but structured steps. These are characteristics of KDD, a process used for a similar purpose. Thus, the question can be posed as "How can stochastic models benefit from the KDD steps?", or "how can we adapt KDD steps be useful to modeling application?".

To make these questions clear and precise, we enter in the area of time series, which is particularly helpful with distinct representation methods for the original data. So, *"How can time series be helpful to select and transform data for stochastic models?"*. This question is discussed along the next Chapters (from 2 to 5) starting by a short literature review about time series and how it can be useful, especially to reduce data dimensionality and to represent it.

1.4 Volume organization

This Thesis is organized according to its main contributions. Those are based on the questions posed in the Section 1.3 and most chapters focus on a specific contribution. Also, the chapters' sequence are given according to the three formalisms discussed in this volume. Markov chains, Hidden Markov Models, and Stochastic Automata Networks. Chapter (2) is an exception, it contains the necessary background knowledge to the understanding of the next chapters.

Chapter 2

This chapter shows a literature review on the time series methods, for purposes of both, representing data, and measure distances between two or more different series. The first use of these methods appears in Chapter 3. Our purpose is clarified along the chapters, especially in Chapter 5, where the general idea is explained.

Chapter 3

This chapter is dedicated to Hidden Markov Models. The contribution is introduced in Section 3.2, which shows a pre-fitting method for faster fitting Hidden Markov Models. It is a sequence of the Section 1.2.1 that explains the most basic structure of a Markovian model. Also, in Section 3.1.2, we describe the most common measures that can be applied to stochastic models. This chapter also introduces the Hidden Markov Models' most known fitting algorithm.

Chapter 4

This chapter describes the Stochastic Automata Networks formalism (SAN) along with some SAN models, including a model for a set of geological phenomena, which exemplifies the use of SANs to model large non-deterministic systems. The model described in Section 4.2, estimates the behavior of sediment strata formation in continental margins resulting from interactions of geological phenomena over the last 130 million years. Our model is a pioneer in the use of SANs for such types of phenomena.

Chapter 5

This chapter describes the proposed knowledge discovery process through stochastic modeling. This process uses techniques, first introduced in Chapter refchap:TimeSeries, to automatically represent and reduce the dimensionality of a given data, thus reducing the effect of the common problems involved in stochastic modeling, such as the state space explosion and the considerable modeling efforts to create models. Furthermore, this process is successfully used along the Chapters 3 and 6. Finally, in Section 5.5, we propose to use TS measurements to evaluate models' simulations.

Chapters 6 and 7

In this chapter, we emphasize some applications based on the proposed process. There are multiple contributions in this chapter. Section 6.2 shows an applied version of the proposed process for the model described in the Section 4.2. Section 6.3 shows the first fitting method for SAN models. An applied version of the process, which uses the likelihood estimation to create SAN models directly from generic serial data. Section 6.4 shows an example of how stochastic methods can be used with non-numerical data. Although we use external tools to classify and train our models, the basic steps of the process remains the same. Finally, the final Chapter is a short review of the contributions and the perspectives regarding this work.

Chapter 2

Time Series to organize and measure data

As described in the Section 1.2.2 time series are a common way to represent and handle data dimensionality. As discussed in Section 1.3, our goal lies on improving the use of stochastic models through automating steps and controlling dimensionality. Furthermore, generated models must be analyzed in regard to its quality; here we discuss time series and methods that are useful for such purposes. Dividing it into two classes, distance measures and a brief overview of representation methods.

Originally used by mathematicians, time series (TS) became popular in many sub-fields of sciences such as statistics, economy, biology, geography, engineering, communication, machine learning *etc.* TS is characterized by a collection of information in time. Due to its simple structure to modify, compare and analyze data; TS stands among the most popular form to represent and analyze data [32] [90] [87].

Data, usually, is represented by fixed periods of time and can represent many dimensions of a system. Let X a time series; p, an observation point in space; T, the maximum size (time) of X; and t some time belonging to T. Formally, TS can be defined as:

$$X = [(p_1, t_1), (p_2, t_2), ..., (p_k, t_k), ...(p_T, t_T)]$$
$$(t_1 < t_2 < ... < t_k < ... < t_T) \tag{2.1}$$

Or in a simplified description: $X = \{x_1, x_2, ..., x_T\}$. The visualization of TS is usually given by the broadly used two-axis scheme, y, x, with a line plot representing data. Although, TS lies in the simplicity of a two axis representation; TS can easily represent high dimensional data [69]. In fact, we can represent many dimensions with TS. It is even possible to represent shapes like fossils, geological materials *etc.* [124] [81].

This chapter contains a short literature review of TS, its representation methods (Section2.1) and distance measurements (Section 2.2). These features allow us to perform useful things, such as dimensionality reduction while manipulating data to the desired format. It also makes it possible to measure how far, considering some criteria, one data curve is from another.

2.1 Representation Methods

Considering a time series (TS) as a compact representation of multidimensional data. The primary goal of representing data as TS is to reduce its dimensionality. Furthermore; being TS usually streaming data, it is often necessary to reduce its dimensionality to improve the performance of a given system. There are many techniques in the literature to handle this problem using different methods to represent it; Figure 2.1 shows a hierarchy of time series representation methods available in the literature.

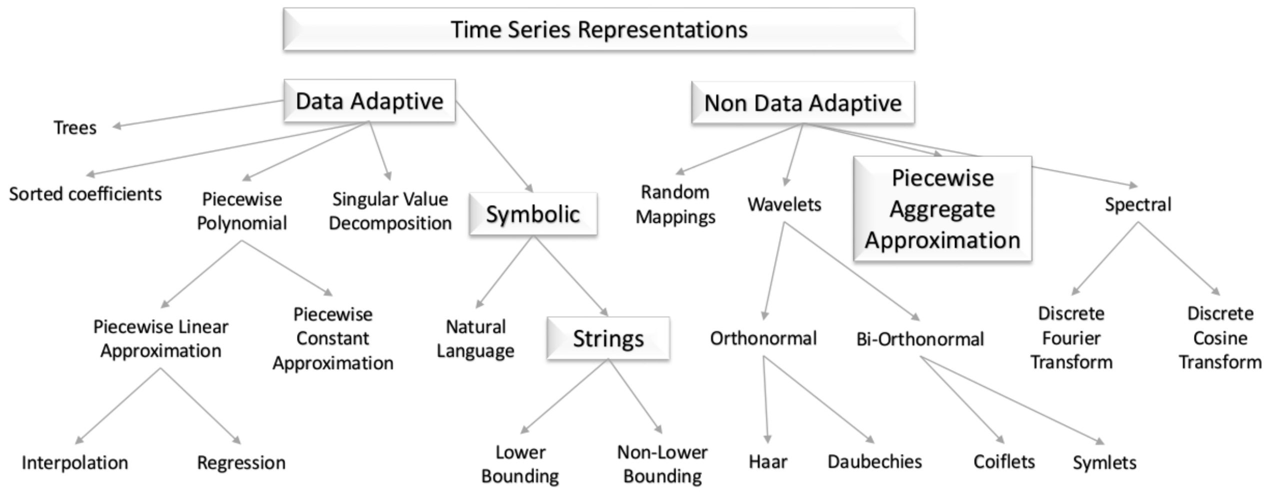


Figure 2.1: Hierarchy of time series representation methods [90]

In the Figure 2.1 different approaches to representing TS are shown. Between the most highlighted methods we can cite: Discrete Fourier Transform (DFT) [2], Discrete Wavelet Transform (DWT) [33] (Wavelets), Discrete Cosine Transformation (DCT) [82], Piecewise Aggregate Approximation (PAA) [32], Symbolic Aggregate ApproXimation SAX [90] (Lower Bounding), 1D-SAX [97] (Lower Bounding).

Despite all the representation methods and techniques available to represent TS; *symbolic* and *string* representations are especially useful for automatically generate automata from data. Data adaptive because it is possible to define a standard representation for all series to be read. And symbolic strings, because it is useful to handle and convert to models. The exception is PAA [32], a not data-adaptive method that can be changed to well fits our needs. In SAX [90], for example, PAA is internally used to create a symbolic method (Symbolic, Strings, Lower Bounding). Thus, it can be merged or modified as a *string* representation. We discuss more the reasons to choose these methods along Chapters 3 and 5.

A clear, and broadly used, dimensionality reduction example, is the signal conversion from analogical to digital. An analogical input is received as a continuous time series and its dimensions are reduced. The output is a digital signal, which has a discrete number of states. An example is a signal conversion from an electric current to a binary output. The input is analogic and continuous; the output is a two-dimensional signal that is used as the basis for all computers.

To ease the understanding our algorithm, we keep the description of PAA to the Section 3.2. Furthermore, in Section 5 we used representation methods in some phases of the process. All the description of these methods follows the order of its usage.

2.2 Distance Measures

There are vast amounts of works to measure distance between time series [52, 88] [37, 116] [38] [18, 39]. So far, the most known distance measurements are the Euclidian Distance (ED) and the Manhatan Distance. Although, there are many other techniques that could have a better performance depending on the scenario [117] [51].

A good measure technique is important to achieve good results about correlation and differences between two TS. Many of the representation methods can be used to reduce data dimensionality and prepare it to prediction or description techniques.

Let us consider two time series, X and X^I, with the same length T, then:

$$X = x_1, x_2, ..., x_T$$
$$X^I = x^I_1, x^I_2, ..., x^I_T$$

Considering the shape of an observation in time, in order to choose a distance measure we have to consider how important is the time for our time scale. For instance, if x_5 is an entirely independent data from x_6 and we are sure about its temporal position (*i.e.*, $6th$ element), we should consider a technique that lies on the values only; such as the Euclidean Distance. Otherwise, if our values are susceptible to be shifted in time, which is common in regard of a Markovian simulations, we should consider techniques that treat time as well; such as Dynamic Time Warping.

There are many techniques to measure distances in TS, among them: Piecewise slope distance (PSD) [88], Pattern Distance, (PatDist) [52], Included Angle Distance (IADist) [52], Radian Distance (RadDist) [52], Distance based on Longest Common SubSequence (LCSS) [116], Distance with Real Penalty (ERP) [37], Edit Distance on Real sequence (EDR) [38], Spatial Assembling Distance (SpADe) [39], Similarity search based on Threshold Queries (TQuEST) [1], Dynamic Time Warping (DTW). Among all these measurements, we focus on the two most advantageous techniques according to the time series community [117] [51] [78] [95] [50]. ED, due to it simplicity and performance; and DTW due to its time alignment feature, which is especially useful for our purposes. Although our purpose is not to compare all these techniques, this list might be useful for a future work since our process is flexible enough to support other methods with symbolic representation capabilities.

2.2.1 Euclidean Distance

There are clear characteristics about Euclidean Distance (ED). ED is easy to understand, ED is easy to implement and ED is efficient as a basic measurement; cases when there is no concern about temporal anomalies in data.

Formally ED can be defined as:

$$ED = |X^I - X| = \sqrt{\sum_{t=1}^{T}(x_t^I - x_t)^2} \tag{2.2}$$

Where, n is the number of dimensions. Two for a plane, three for a three-dimensional space and so on. Although this is the canonical ED measurement, there are cases when we wish to put penalties on data that are farther apart. In such cases, the distance penalty can be enforced by the elimination of the square root; formally a squared ED:

$$ED^2 = |X^I - X| = \sum_{t=1}^{T}(x_t^I - x_t)^2 \tag{2.3}$$

2.2.2 Dynamic Time Warping

Dynamic Time Warping (DTW) [1] is a technique that allows a non-linear mapping of one time series to another, minimizing the distance between then. Being used for a long time by the speech recognition community, even on nowadays DTW still useful and some authors claim to have a better precision using DTW instead of ED and other measurement techniques [78].

The non-linear mapping performed by DTW allows us to get the distance between two TS considering their alignment. In other words, the time warping is considered to get the best relation between distance and alignment, *i.e.*, the alignment between points shifted in time. Consider two TS, X=$\{1,1,2,1,2,1,2\}$ and X^I=$\{1,2,1,2,1,2,1\}$. Using the traditional Euclidean distance, these TS are very distant from each other. However they both have a similar behavior, X is just one bit shifted from X^I, thus an alignment would group the points $X^I_{(t)}$ and $X_{(t+1)}$, retrieving a lower distance through DTW.

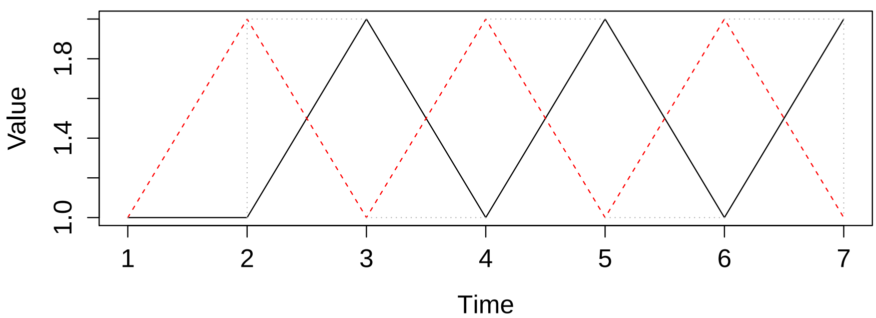


Figure 2.2: Alignment between two time series using DTW

To calculate the minimum distance between two points X_1^I, X_2 a matrix $n * m$ stores the squared distances between the points of X^I and X, $d(X_i^I, X_j) = (X_i^I, X_j)^2$. This retrieves possible alignments for these time series. Then, the next step is to retrieve a path through the matrix that minimizes the total cumulative distance between them, in other words, the optimal path to minimizes the warping cost [78] [79] [67] .

$$DTW(X^I, X) = \sqrt{\sum_{t=1}^{T} \Xi_t} \tag{2.4}$$

Thus, Ξ_t is the matrix element $(i,j)_t$ that belongs to the tth element of a warping path that represent a mapping between X^I and X [78].

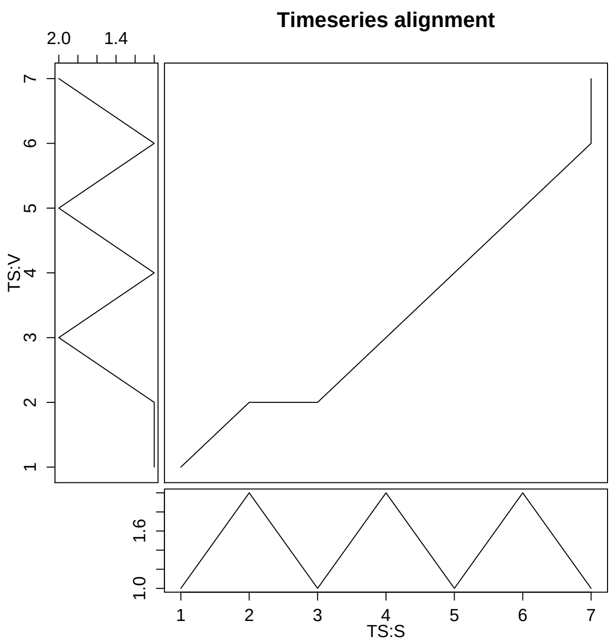


Figure 2.3: DTW alignment path

Few good examples for the use of DTW alignment are given by Chotirat Ann *et al.* [79]. Video retrieval, image retrieval, handwriting and text mining are among then. More than matches the better alignment for two TS, the DTW algorithm has a distance measurement, which is based on the optimal alignment. When data are simulated in big time periods, it is normal to get time-shifted results, which would be misinterpreted if compared without the time warping feature. It means that we can have a metric for how close two TS are. Therefore, we can use this metric as a measurement to check the quality of the generated data and improve it by a regularly stochastic generation. This idea is clearly described in Section 3.1.2 and used in Chapter 5.

Chapter 3

A Baum-Welch based algorithm for faster HMM fitting

Due to their relative simplicity and power to represent complex systems, Hidden Markov Models (HMM) are one of the most widely used stochastic formalism for time series data [127]. HMMs owe their flexibility and their ease of application to a well-developed methodological framework, including a model fitting algorithm. The Baum-Welch (BW) algorithm can be considered a particular case of the Expectation-Maximization (EM) algorithm. In essence, it derives the maximum likelihood estimates for the parameters of a given model. *I.e.*, the best fit for the input data in the sense that the probability to observe the data given the model parameters is maximal. This is achieved by an iterative procedure guaranteed to converge to a local maximum of the likelihood function and thus offers a model solution even when the likelihood is too costly to maximize directly. However, BW as well as general EM algorithms, tends to be very sensitive to the input parameters [31].

Several extensions have been suggested to overcome the flaws of EM, [31, 49, 55, 73], based on combinations of algorithms and techniques such as classification, randomization or more complex stochastic additions. However, these extensions focus mostly on general mixture models.

Our approach is an improvement of the original BW via parameter pre-selection. The idea is to use a deterministic discretization for time series data, before the actual model fitting. From this approximate description of the original observations, we then derive initial parameters to feed the BW. These parameters are in general reasonably close to the maximum values sought for. Thus, the combination of this parameter selection step with the BW algorithm reduces the number of iterations to maximize (Maximization step) the parameters (Figure 3.4) and is more likely to find the local maximum.

Focusing on HMMs, this chapter continues from the basics of Markovian structures and fitting models (Section 1.2.2) and then get deeper into fitting techniques by describing the formal definition of HMMs and the Baum-Welch algorithm (Sections 3.1.1 and 3.1.3). Finally, we describe how a pre-fitting, deterministic step, can be useful to reduce the time to fitting HMMs (Section 3.2).

3.1 Hidden Markov Models

“Hidden Markov Models are models in which the distribution that generates an observation, depends on the state of an underlying and unobserved Markov process" [127]. The hidden part is an ordinary Markov chain that generates transient and steady states to another stochastic layer that generates the observations. Thus, an HMM can be described as a two level stochastic model. The first one is an unobservable DTMC (See Section 1.2.1); and the second one is a state dependent, visible set of states, which generates a sequence of tokens according to the probabilistic output of the hidden part.

A simple way to visually describe an HMM is via a Trellis diagram. In the Figure 3.1 are shown outputs from hidden chain states (c_t) and a set of observable states (y_t) that are generated based on probabilities related to the hidden output.

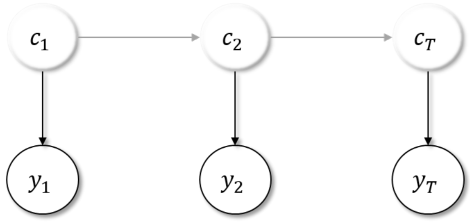


Figure 3.1: HMM trellis diagram

Using hidden states as a condition and trigger to the visible chain, HMMs can be especially useful to describe stochastic systems. It differs from other Markovian formalisms by its generated observations (Y). Unlike most of the Markovian formalisms where an output is directly generated, in HMMs, the output are made by inference. Each hidden state has a probability distribution over the possible output.

The first applications of HMM were focused on speech recognition [11, 12, 74]. However, in the last decades, they are intensively used in applications for bioinformatics, environment analysis, finance, etc. In [127], many applications are shown by using together HMM and time series. In most of its approaches, time series are used to describe data, possibly an HMM output. We can also find models for many applications; such as financial series, medical indicators, wind direction, animal behavior, etc. [11, 76, 126, 127].

3.1.1 Formal definition

Notation

θ	A set of input parameters
δ	Initial distribution
Γ	Transition probability matrix
$\phi_{i,j}$	A specific cell in Γ, representing the state i to state j
s_i	A hidden state i of Γ
c_t	A hidden state at time t
n, m	Number of states
λ	Emission probability matrix
T	A time series length
X	Input data $[x_t, x_{t+1}, ..., x_T]$
x_t	A single observation of X
$\alpha_{(t)}, \beta_{(t)}$	The tth vector of forward and backward probabilities.
$P(x)$	A probability to find an element x
P	An identity matrix (forward-backward notations)
y_t	An emitted observation at time t
$1'$	A vector of ones, usually used to describe the creation of a vector

At time t, an output Y_t is generated by an emission probability of $\lambda_i(Y_t)$, which is associated to a hidden state i, with a probability: $P_i(Y_t) = P(Y_t|c_t = i)$.

Figure 3.2 represents how an HMM model generate its observations y_t. In the hidden part a probability transition matrix Γ is represented by a set $\phi_{i,j}$ representing the probability of a state s_i goes to a state c_j. Considering T the maximum time, the output will be a sequence of hidden emissions $c_1,\ c_2, ...,\ c_T$. Also hidden is the emission probability matrix λ that can have different dimensions than Γ, m, and will generate a sequence of observations in time y_t. In a few words, each output (c_t) from the hidden matrix has M possible outputs, which relies on different probabilities according to the hidden state.

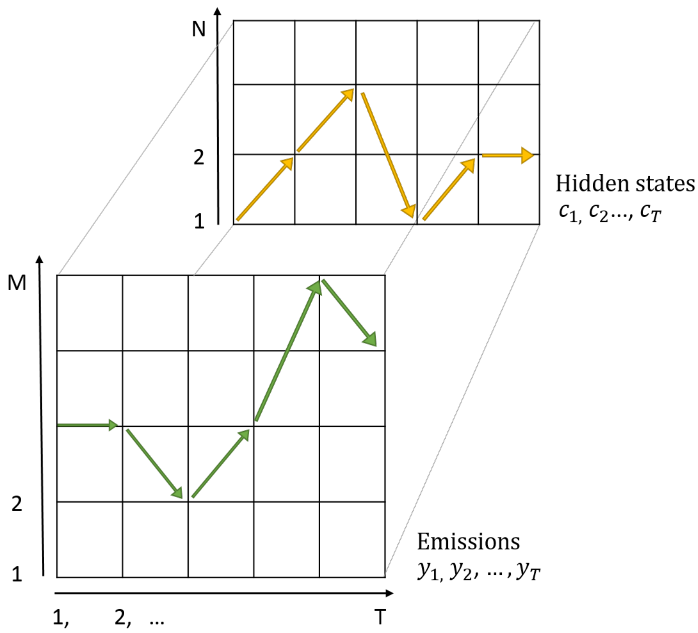


Figure 3.2: HMM emissions scheme

3.1.2 Model Measurement and Selection

This section describes some measurements that can be applied to stochastic models. Since that are no single correct model for a system, it is important to have formal methods to measure the quality of the models and its results.

Model likelihood

A model likelihood is a measurement that describes how likely is the observed data to be derived from this model. Formally, likelihood LH_T is a function of T parameters of a statistical model, where T is a consecutive sequence of parameters. Its computation consists in a sum of m^T terms; being each one, a product of $2T$ factors, making it require $O(Tm^T)$ operations, which is considered by many authors as too costly for practical operations. However, in [127] they showed an HMM's applied likelihood with a cost of $O(Tm^2)$ operations. Since this work is focused on Markovian models, we use

DTMCs and HMMs as references for our description. DTMCs because it is the simplest formalism; therefore, the easiest example to understand. Then, HMM because the likelihood is broadly applied to fit HMM.

Finally, there are specific model likelihood measurements for specific kinds of data format and randomly missing data. We consider the overall likelihood, usually applied with HMMs having more than 2 states and applied to poisson distributed observations. Yet, this root concept of likelihood can usually be applied for other purposes.

The likelihood equation can be summarized as a probability LH_T of observing a sequence of T observations $O_1 ... O_T$. In HMMs, this sequence is calculated under an n-state HMM which has initial distribution δ and a set of transition probabilities Γ for the Markov chain. Thus, the likelihood is given by the Equation 3.1.

$$LH_T = \delta P(O_1) \prod_{t=2}^{T} \Gamma P(O_t) \tag{3.1}$$

Although, if the initial distribution of C_1 is the MC's stationary distribution, then the likelihood is given by the Equation 3.2.

$$LH_T = \delta \Gamma P(O_1) \prod_{t=2}^{T} \Gamma P(O_t) \tag{3.2}$$

To ease of understanding, we can calculate the likelihood in three steps. First, we fill the first element of a vector α with the probability to find the first element O_1 using the initial distribution; thus, $\alpha_{(1)} = \delta P(O_1)$. Then, for each index $t = 2$ until T, $\alpha_{(t)} = \alpha_{(t-1)} \Gamma P(O_t)$; thus, the likelihood is $LH_T = \alpha_{(T)}$.

One may realize the only difference between the equations 3.1 and 3.2 is the addition of Γ in the first observation. Thus, in our 3 steps, we can formally describe it by changing just the first element to $\alpha_{(0)} = \delta$. Divide the equation is algorithmic convenient because we can explicitly show the element α (forward probabilities [127], used along this work).

Likelihood maximization

Following the likelihood equations (Eq. 3.1 and eq. 3.2), we can notice that the constant multiplication of parameters will inevitably make the value be progressively smaller until it is rounded to zero. This is easy to imagine, because considering a perfect match, 1 is returned; otherwise, the probability will be represented as $0 >= P < 1$, thus reducing the value with the consecutive iterations. This is a well known and an already solved problem. The first attempts to perform likelihood maximization on HMMs date from earlier 70s [15]. After, there was some important improvements in this method, most due to the speech recognition techniques [101].

Here, we consider the method *inter alia* proposed by Durblin *et al.* [54]. This approach was applied to biological sequence analysis and it relies on a maximization and an approximation for the log likelihood. As explained by Zucchini *et al.* [127], a good strategy to compute the logarithm of LH_T is by using a vector of forward probabilities a_t, where: $\phi_{(t-1)} = \alpha_{(t)}/\alpha_{(t)} * 1'$, being $1'$ a vector of ones, with $length = t$.

$$LLH_T = \sum_{t=1}^{T} log(\phi_{t-1}\Gamma P(O_t)) \tag{3.3}$$

The equivalent algorithm is 1. However, the vector of forward probabilities can be replaced with an occurrence matrix; this brings us a more straightforward approach when the whole model represents the data using only one matrix, *e.g.*, MCs. In these cases, we can make a directly log-likelihood computation using such matrix.

So far we can observe that all the formalisms can generate a unique matrix representation. However, given a structure with multiple stochastic layers, such as an HMM, this is costly to perform; thus, the next equation advantage lies on MCs, especially the ones that are fitted using a Maximum Likelihood approach. In this approach all the original observations are stored into the TPM, Γ, which is given by the forward probabilities and generated through a frequency matrix $f\gamma$. Therefore, we can achieve the same result through these matrices. Thus, the T elements can be accomplished by getting the sum of $f\Gamma$ elements.

$$LogLLH = \sum_{i,j}(f\Gamma_{i,j} * log(\Gamma_{i,j})) \tag{3.4}$$

Data: A initial distribution: δ
Two N by N matrices: Γ and $P(x_t)$
Two Vectors of length N: v and ϕ_t
Output: A scalar of accumulated log-likelihood $maxLLH$

```
1) φ0 ← δ;
2) maxLL ← 0;
3) for t ← 1 to T do
4)     v ← φ_{t−1} Γ P(x_t);
5)     u ← v1′;
6)     maxLLH ← maxLLH + log(u);
7)     φ ← v/u;
8) end
9) return (maxLL);
```

Algorithm 1: Model log-likelihood

Despite the likelihood being a well known statistic metric, when considering a Markovian model, it is important to consider other variables such as the number of states. Thus, the bigger the system, the larger the modeling possibilities. So, “how to choose a better model?" is an important question to be considered. In HMMs, an important concern is to determine an ideal number of states, also referenced as “order". Although this concern is characteristic of HMM's researchers, the same problem can be applied to other formalisms. Although, even for HMMs, the question of order estimation is not settled [26]. According to Zucchini *et al.* [127], “The goal of model selection is to identify the model which is in some sense the best". However, the “sense" is given by some balance between the likelihood and the model size. There are two major measurements that ponder the model size and its likelihood. Akaike's information criterion (AIC) and Bayesian information criterion (BIC) .

Selection via AIC and BIC

Akaike's information criterion is a relative quality measure of a statistical model for a given set of observations [3]. Focusing on statistical model identification, AIC's equation still being used for measuring a statistical models fitness. Concerning HMMs, AIC is used as a measure to select some model simply by applying penalties related to the total number of parameters.

$$AIC = 2p - 2logL \tag{3.5}$$

At AIC's equation, p denotes the number of parameters in a model, in Markovian models, the number of states. L stands for the log likelihood of the fitted model. Thus, AIC rewards goodness of fit, but applies a penalty according to the number of parameters.

Just like AIC, Bayesian Information Criterion (BIC), also known as Schwarz criterion, provides us a relative quality measure of a statistical model. However, BIC favors models with fewer parameters by applying more penalty for models with more parameters. Being T the total number of observation in the time series, BIC apply penalties not only according to the number of parameters, but also according to the time series size.

$$BIC = p(logT) - 2logL \tag{3.6}$$

Although these measurements are often related in the literature. They only concern to apply a penalty to generation of a large state space. Thus, while the likelihood is an indicator of a fitness accuracy, these measures are also concern about the performance in general. However, these criteria can be subjective depending on the user requirements regarding memory or execution time.

3.1.3 Baum-Welch algorithm

Section 1.2.2 briefly described the aim of a model fitting procedure. Such bases are used in other chapters as well. Therefore it was placed in the first Chapter of this volume. Now we go further by describing the classical model fitting algorithm for HMMs.

The Baum-Welch algorithm (BW) is a version of Expectation-Maximization (EM) algorithm for HMMs [15]. The goal is to estimate the parameters of an HMM given an input data [127], in the statistics literature, commonly described as observations. This goal implies that unlike the traditional modeling approach, the initial distribution δ, and other model parameters, are not estimated by the modeler observations but by the model itself.

The General EM algorithm, EM or GEM, works interactively by successive maximizing the local approximations of the likelihood function. GEM has slightly different implementations and interpretations of itself. The EM algorithm is divided into two steps. In the Expectation step the conditional expectations of the missing data are computed according to a set of observations and the current state of the input parameters θ. These computations are based on the current complete log-likelihood. In the Maximization step, the likelihood is maximized by improving the fitting according to the input parameters.

Among the deterministic versions of EM there are variants such as Classification EM (CEM) [31], Accelerated EM (AEM) [73], Aitken's accleration (AA) [49], [103], Expectation Conditional Maximization (ECM) [55], ECM Either (ECME) [92], Space-Alternating Generalized EM (SAGE) [62], Parameter-Expanded EM (PX-EM) [93] etc. Among stochastic versions the main variants are: Stochastic EM (SEM) [30], Stochastic Approximation type EM (SAEM) [29], Data Augmentation algorithm (DA) [113] and Monte Carlo EM (MCEM) [119]. All these variants are based on General EM (GEM) that is characterized by having two interactive steps; approximation or expectation, and maximization. Thus, almost all of its variants have the steps that give the name of the algorithm, expectation and maximization.

Figure 3.3 shows a concise list of EM variants and its classifications. The purpose is to present all the related algorithms and highlight their specific characteristics, which is necessary to understand the relevance and contribution of this chapter.

The vast major derivations have a deterministic approach. Among then, CEM is a special case because he is the only variant that does not use the M step to maximize the likelihood. However, CEM has a fast convergence, and it is often used in practice, in particular through its famous instance, the k-means algorithm.

A stochastic approach was first introduced via SEM algorithm. SEM is an EM derivation with a stochastic layer between expectation and maximization. This layer helps the algorithm to get a faster convergence through random generations; coping with the parameter sensitiveness of EM.

Until now, was described a series of GEM-based algorithms, they all have in common the use of an iterative Maximum Likelihood Estimation (MLE) or Recursive MLE (RMLE)[1]. However, not all fitting algorithms are based on MLE. Some are based on Minimum Model Divergence (MMD) and Minimum Prediction Error (MPE), which can be extended to Recursive Prediction Error (RPE) as a general recursive stochastic algorithm [5]. Minimum Model Divergence (MMD), which in few words can be described as a combination of MLE and the minimization of parameter's divergence using entropy measures. Also, Minimum Prediction Error (MPE) which consists of measuring an HMM

[1] To avoid confusion, we refer MLE for all recursive and non-recursive MLEs.

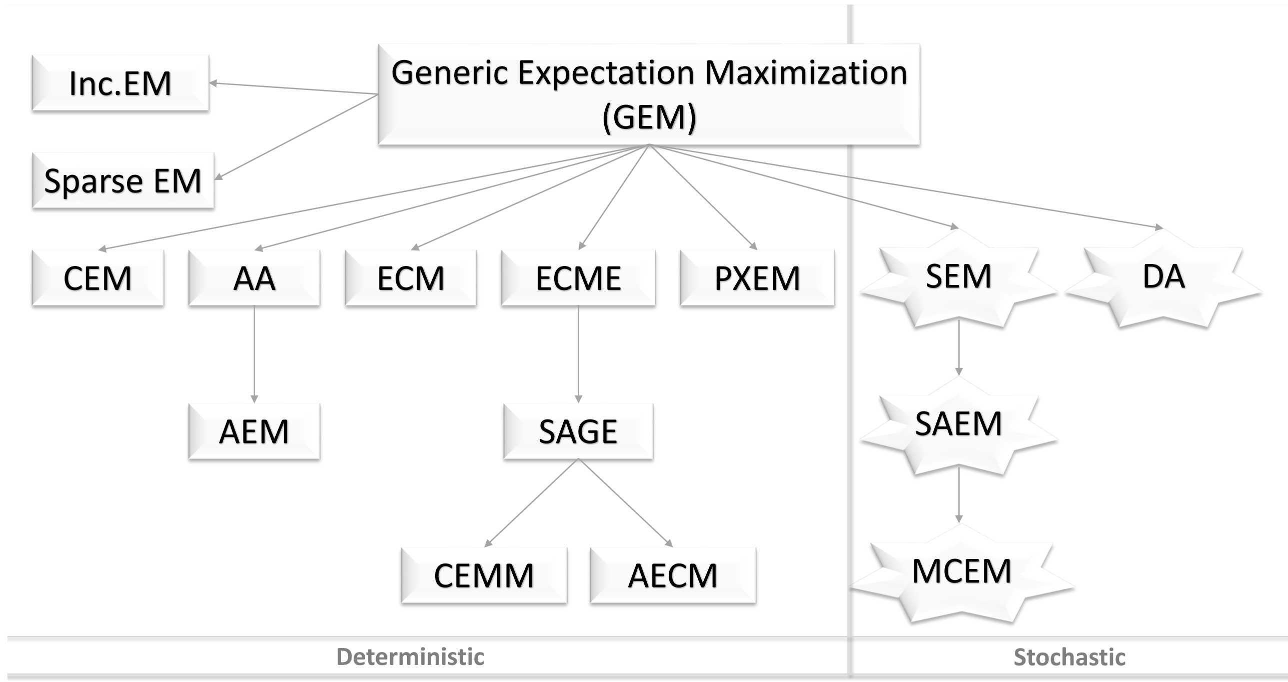


Figure 3.3: EM and MLE based algorithms and its extensions

error output prediction and provide an updated estimation for the HMM parameters [40]. Among the algorithms using MPE we can emphasize Collings *et al.* [40] and LeGland and Mevel [86]. Using MMD we can emphasize Garg and Warmuth [65].

Despite the uses of MMD and MPE, we focus on a classic MLE, which is commonly used for HMM. So far, the related works, using MLE, are: [13, 25, 63, 83, 108]. However, our approach is different because we do not intend to create an entirely new algorithm nor improve within itself. Instead, we perform a pre-fitting to avoid the BW sensitiveness (an approach used by the EM variants) and get an initial higher model likelihood. However, we do not intent to create an optimal algorithm, but a better version of the traditional BW, which aims to be a practical option that do not suffer from the same flaws of GEM. GEM as well as BW, is strongly dependent on its initial parameters [31]. Therefore, the convergence is directly dependent on how good are those initial conditions. Furthermore, the achieved local likelihood is not always the best one. Although each iteration is proved to generate equal or better parameters than the previous one, commonly the algorithm fails to get the global maximum likelihood.

Next, is shown the use of forward and backward probabilities to estimate parameters and/or observations of an HMM though the Baum-Welch algorithm. More specifically, we described the properties of the Forward-Backward algorithm [34] that is part of the EM algorithm (E-phase).

Forward probabilities

As components of the BW; forward and backward procedures are standard for HMMs, as well as the mainly responsible for the likelihood maximization at each iteration. The BW computational cost is directly linked to the EM iterations, composed by forward and backward procedures.

The forward probabilities α are achieved by the cumulative product of the probabilities in the transition matrix (Γ) of representing an element x in a time t. [2] The first element (x_1) is represented outside of the cumulative product, to isolate the initial distribution δ. Thus, the probability of the first element x_1 is achieved by the product of the initial distribution (δ), the identity probability matrix (P) and the element itself. This will be joined to the cumulative probabilities for $t = \{2, ..., T\}$.

In a short description, we can define the forward probabilities ($\alpha_{(t)}$) as the cumulative product of the probabilities to find an element x in time ($\alpha_{(t)} = \delta P(x_1)\Gamma P(x_2), ..., \Gamma P(x_t)$). More specifically, $\alpha_{(t)}$ is a vector of probabilities at time t, retrieved from a matrix $\alpha = \{\alpha_{(1)}, \alpha_{(2)}, ..., \alpha_{(t)}\}$ of dimensions $size(\delta)$ by $size(X)$. For a more elegant representation, we can define the forward vector as an equation 3.7. In this equation, $\alpha_{(t)}$ represents the probability of an observation x at time t.

$$\alpha_{(t)} = \left(\delta P(x_1) \prod_{w=2}^{t} \Gamma P(x_w) \right) \tag{3.7}$$

Thus, once we have a, we can calculate the joint probabilities from the forward and backward procedures. Before describing the backward procedure, it is important to define that a_j as the j_{th} component of a_t is a joint of probabilities of $P(X_1 = x_1, X_2 = x_2, ..., X_t = x_t)$ given a set of input parameters θ at time t equals j. Following the equation 3.7, it is possible to derive a simpler equation for the next step of α, Eq. 3.8.

$$\alpha_{(t+1)} = \alpha_{(t)} \Gamma P(x_{t+1}) \tag{3.8}$$

More precisely, considering the lines and columns of the probability matrix and the vector α, we can add another dimension J, as a Jth component of the vector α. Furthermore, considering the other component of an a_t as $a_t(J)$ a scalar form can be defined from the previous equation, where m is the length of the observations Eq. 3.9.

$$\alpha_{(t+1)} = \left(\sum_{i=1}^{n} \alpha_{(t)}(i) \Gamma_{ij} \right) P_j(x_{t+1}) \tag{3.9}$$

Backward probabilities

Similar to the forward procedure, backward probabilities can be roughly described as a cumulative matrices product. $\beta_{(t)} = \Gamma P(x_{t+1}) \Gamma P(x_{t+2}) ... \Gamma P(x_T)$. However, unlike the forward probabilities, backward's are achieved by a reverse analysis. In a standard description the time is used as a forward measure $t = \{1, 2, ..., T\}$. Thus, concerning the current time t, backward probabilities are achieved from the cumulative product of the transitions matrix probabilities for an element x in time w, $\Gamma P(x_w)$.

$$\beta_{(t)} = \left(\prod_{w=t+1}^{T} \Gamma P(x_w) \right) \tag{3.10}$$

[2]In these equations, we used w instead of t to not cause confusion with the main time counter.

To ease of understanding, we can rearrange the timescale. Thus, concerning the time scale $t = \{1, 2, ..., T\}$ we can establish a non usual notation, yet more straight forward to computer scientists, due to be closer to the algorithmic form. This equation (3.10) basically describes how the probabilities are accumulated in the vector $\beta_{(t)}$. It can be seen in the algorithm 2.

$$\beta_{(t)} = \left(\prod_{w=T-1}^{1} \Gamma P(x_w) \right) \tag{3.11}$$

To define the difference from forward and backward procedures, we can simplistically define the forward procedures as a basically cumulative product in time a_t that counts the probability of have an observation x in an initial distribution δ and, for $t > 1$, its transient probability matrix, Γ. Thus $\alpha_{(t)} = \delta P(x_1) \Gamma P(x_2) ... \Gamma P(x_t)$. Concerning the backward probabilities, the difference lies in the order and consequently in the first element δ.

Having the forward and backward probabilities, it is possible to update the temporary variables. *I.e.*, Γ, δ and P. BW as well as its variations uses these equations interactively to maximize the fit of a model. To ease the understanding, here follows the equations 3.7 and 3.10 described in an algorithm form; the Forward-Backward algorithm (Alg. 2). Notice that these equations do not use the log-likelihood, which is used to avoid the values underflow.

Data: A sequence of observations: X
A TPM: Γ
The TPM matrix order: n
The initial probabilities: P
The Initial/Steady distribution: δ
Output: Two probability matrices α and β, with a [n,length(X)] dimension

1) $m \leftarrow length(X)$;
2) $\alpha \leftarrow matrix(n,m) \leftarrow 0$;
3) $\beta \leftarrow matrix(n,m) \leftarrow 0$;
4) **if** δ=*NULL* **then**
5) $\quad \delta \leftarrow$ ChapmanKolmogorov(Γ);
6) probs $\leftarrow$ distribution(X,P);
7) probs $\leftarrow$ normalize(probs);
8) $\alpha_{[,1]} \leftarrow \delta * probs[1,]$;
9) **for** *i* $\leftarrow 2$ **to** *m* **do**
10) $\quad \alpha_{[,i]} \leftarrow \Gamma * probs[i,]$;
11) **for** *i* $\leftarrow$ *m-1* **to** *1* **do**
12) $\quad \beta_{[,i]} \leftarrow \Gamma * probs[i,]$;
13) **return** (α,β);

Algorithm 2: Forward-Backward

In the lines 2 and 3 we create two matrices, α and β. These matrices are responsible for holding the probabilities for each observation in X according to the TPM dimensions. Next, if the initial distribution δ was not given, the TPM is used to estimate the steady state through the Chapman-

Kolmogorov equation (Eq: 1.2). At line 6, a distribution, *i.e.*Poisson, is applied to the vector X and the probability vector P. Then (line 8) the vector α receives the product of δ by the normalized distribution of probabilities. Then, the fist loop (line 9) fills the next elements of α. To achieve β (line 11) are performed an inverted loop starting from $m-1$ going to the first element. In this case, the MC reversibility proprieties are not considered.

Having the main component for the *Expectation* step (Forward-Backward) we can describe the complete EM for HMMs, the Baum-Welch algorithm. Due to be mainly used by statisticians; this algorithm is usually described with sets of equations. Here, we describe it using an algorithm shape, more close to an implementation (Alg. 3). Yet, first, we start via an unstructured description of the EM steps.

On the **Expectation Step** is computed the expectations for the missing parameters. As input, a set of observations and the current estimation for the initial parameters. In other words, are computed the provisional expectations of the missing data according to the complete data log-likelihood. On the **Maximization Step** are maximized the complete data likelihood through achieving a better fit for the model according to the input data and the initial parameters [127] (Alg. 3).

First, we temporarily set the values of the next iteration parameters equal as the current ones (lines 2..4). Then, at line 5, we start a loop until the desired number of iterations. At line 6 we use a function to get the log distribution [3] of the vector X based on the initial probabilities P, also described as λ.

At line 7, the forward-backward algorithm (Alg. 2) are called to achieve the forward and backward probabilities. Further, the current model log-likelihood is calculated (8). The next lines (9...12) there are loops applied to get the maximum fit for the variables Γ and P. δ is updated based on the forward and backward probabilities and finally, all parameter values are normalized (14) to test if the critical limit was achieved (16). If yes, the maximized model are successfully returned. Otherwise, the expected values are assign for the next variables, *i.e.*, the current ones, and the next iteration starts.

3.2 Piecewise Aggregation EM

Efficiency of fitting HMMs can be improved by combining the EM algorithm with data mining techniques such as classification methods or the K-means algorithm [31, 127]. These techniques are used to choose the initial parameters intelligently, thus reducing the impact of the BW's sensitiveness to them. In fact, the choice of BW's initial values is important, as it can be "heavily influence the speed of convergence of the algorithm and its ability to locate the global maximum" [75].

As shown in the Section 3.1.3, there are many algorithms which have been derived from the standard EM. However, they are based on different techniques and adapted for different situations. Here we adapt and combine a Piecewise Aggregation technique for time series to represent the original observations and perform a pre-fitting for the BW algorithm, calling this extension the

[3]Usually a Poisson distribution

```
Data: A sequence of observations: X
A TPM: Γ
The TPM matrix order: n
The initial probabilities: P
Opt. The Initial/Steady distribution: δ
The maximum number of iterations: maxI
The non steady tolerance: tol
Output: The fitted version of P, delta and Γ
The model log-likelihood llk
m ← length(X);
P.next ← P;
δ.next ← δ;
Γ.next ← Γ;
for i ← 1 to maxI do
    // probs as a matrix of probabilities, (length(X) ∗ length(P))
    probs ← GetLogDistributions(X,P);
    // Vectors la and lb shall receive the forward and backward
       probabilities, respectively
    [la,lb] ← forwardBackward(X,gamma, n, P,delta=delta);
    llk ← getLogLikelihood(la,X);
    // Update P and Γ according to the likelihood
    for j ← 1 to n do
        for k ← 1 to n do
            Γ.next ← updateGamma(Γ, probs, llk);
        P.next[j] ← updateP(X,llk,la,lb);
    δ.next ← updateDelta(la, lb, llk);
    normalizeUpdatedVariables();
    crit ← updateCriticalV(P, P.next, Γ,Γ.next, δ, δ.next);
    if crit < tol then
        return(P,δ,Γ,llk);
    p ← p.next;
    Γ ← Γ.next;
    δ ← δ.next;
// if all iterations were being performed with no convergence, the output
   will be null
```

Algorithm 3: Baum-Welch

Piecewise Aggregation EM (PAEM). PAEM has the simplicity of the PAA method and the dynamism of a SAX [90] inspired method, which can be applied for different kinds of distributions concerning discrete time series.

The EM in PAEM's name is due to the effect of the pre-fitting on the E-M iterations, however, PAEM is based on BW. More specifically, PAEM is designed for the fitting of HMMs, given discrete time series as input. These characteristics allow us to derive meaningful initial parameters by a fast approximation of the data, avoiding failing on dimensionality reduction problems, such as a fail to converge, which can be given by higher dimensions; or imprecise representations, which can be lead by a strong dimensionality reduction.

PAEM's initial approximation enhances the initial parameters Γ and λ, making then close to the local maximum, which leads to a faster fitting compared to the traditional random initialization. This is due to the need of a single initialization to maximize the parameters and to a first better fitting, which reduces the sensitiveness effect and tend do avoid E-M iterations. Figure 3.4 illustrates the general idea; which is a composition of a pre-fitted in a phase called piecewise expectation linked with the traditional BW. Two of three parameters are previously updated, Γ and λ, which together with the steady state, stored in δ, tends to lead to a first better likelihood. The piecewise expectation cost is $\vartheta(T)$, which is lower than forward-backward procedures $\vartheta(N^2T)$, briefly described in the previous section. Therefore, once a pre-fitting saves one iteration, the final computational cost should be lower. Next, we describe the Piecewise Aggregate Approximation technique together with the adaptation used in PAEM.

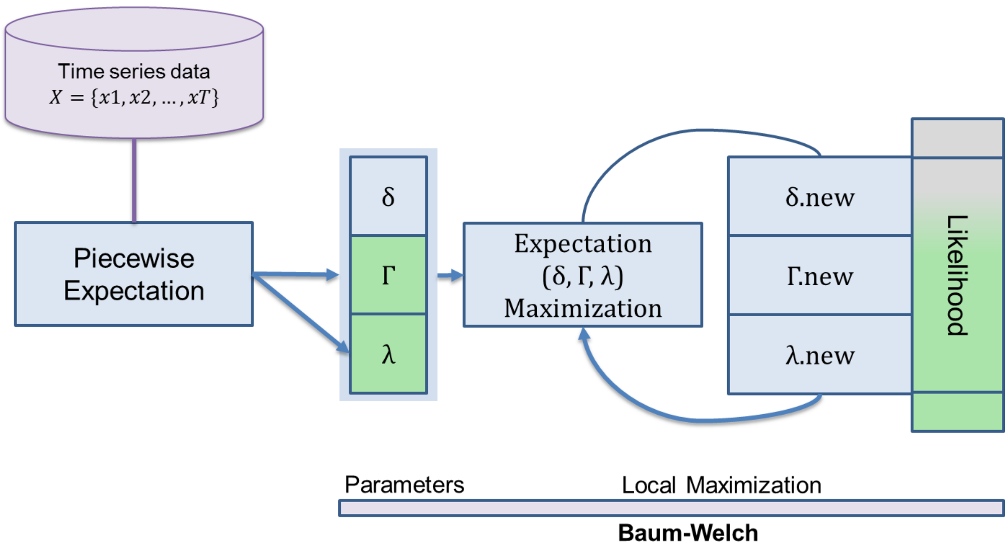


Figure 3.4: PAEM, schematic representation

Piecewise Expectation

Piecewise Aggregate Approximation (PAA) is a technique to reduce data dimensionality through discretization. It has been widely applied in the context of time series analysis. Despite its simplicity, PAA has been shown to be as powerful as more sophisticated dimensionality reduction techniques such as Discrete Fourier Transform [2], Discrete Wavelet Transform [33], Singular Value Decomposition [82].

To perform dimensionality reduction, PAA creates a discrete version of the original TS in w blocks. These blocks are usually a division of the length of the TS. It is a simple representation which still rival other methods such as Fourier transforms and wavelets [90]. In our case, the faster mapping characteristic is specially attractive. Since we intent to reduce the total time necessary to fit a model, more robust approaches might be too costly for a pre-fitting procedure.

Given a time series S with length n, PAA(S) is defined as a sequence PAA(S) $= \{\mu(B_1, \mu(B_2)$ $, ..., \mu(B_w)\}$, where μ is the mean, w is the maximum number of blocks and B_i is a block in the index i, being $(1 \leq i \leq w)$. The mean of a block is given by the Equation 3.12 which is straightforward, yet the division n/w must be an integer. If the division results in a float number, the result is truncated and another block is made of the remaining part of the series.

$$\mu(B_i) = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} S_j \tag{3.12}$$

For instance, for any set of observations such as $X = (x_1, x_2, ..., x_T)$, PAEM only needs one parameter, the model number of states n. It starts by getting a sequence of symbolic values ς, with range n, that better describes X. A second parameter m is optional used to reduce the dimensionality of the sequence; the total number of Blocks is equal to $w = T/m$. Being B_i a block of values to be represented by a symbol, and μ an average value. Formally, a symbolic approximation generated by PAA can defined as $\varsigma = (\mu(B_1), \mu(B_2), ..., \mu(B_w)$. The number of the states n define the division of the values $\mu(B_i)$. This generates the λ values of the HMM. Thus, ς have a sequence of w elements composed by n symbols. *E.g.*, in $\varsigma = \{P, U, C, R, S, P, P, G, C, C\}$, $w = 10$ and $n = 6$. Now, we can formally define the whole approximation process.

Fig. 3.5 shows an example of an approximation given a time series with 256 observations. w is used determined according to the size of our model, thus it is directly determined by the number of states of the HMM.

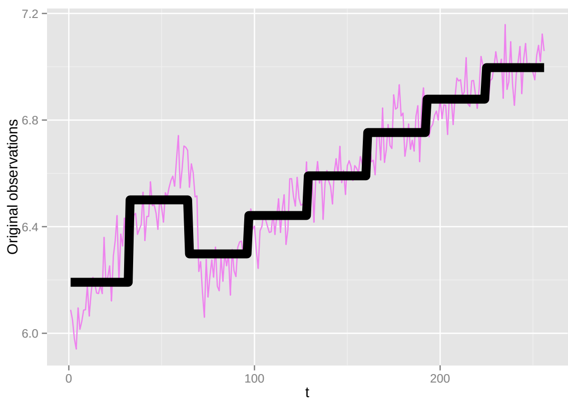


Figure 3.5: An approximation through PAA (n = 256, and w = 8)

ς is directly used to get the probabilities and set the transition probability matrix, Γ, which together with λ are the two necessary parameters for an HMM model since δ can be started *null* and then filled with the steady state. Due to this solution be directly related with PAA, we call this part of the algorithm, "Piecewise Expectation".

To generate Γ, we use the elements of ς^t, $1 \leq t \leq w$ and $1 \leq i, j \leq n$. The non-normalized matrix Γ is filled by the cumulative sum of the probability to find an element ς_j just after and element ς_i.

The same procedure is known as Maximum Likelihood Estimation (MLE) for DTMCs.

$$\Gamma_{(i,j)} = \sum_{t=2}^{T} P(\varsigma_j^{(t)} | \varsigma_i^{(t-1)}) \tag{3.13}$$

Now, considering ς's elements, λ is achieved through the mapping of each symbol position. For instance, if a symbol "*k*" represents a block $B_k = \{18, 12, 14, 16\}$, then element $\lambda_k = 15$ ($\mu(B_k)$). The same are applied for all symbols, generating a vector λ with length n. Note that other methods, as described in Chapter 2, could be used to generate this representation. SAX, for example, would allow a non-constant step in ς. This could improve the results, yet our aim is not to compare methods but show that at least one of then can be useful to our purpose.

Experiments and Results

Our experiments aim to measure and compare how fast the local maximum likelihood is achieved through BW and PAEM. In other words, to prove the efficiency of our algorithm in finding the best model fit. To do so, we used randomly generated and randomly chosen time series from Data Market [48]. We separated our tests in three steps. First, to compare the generated maximum likelihood from different initialization of BW against PAEM. Second, to detect the number of necessary executions to achieve the local maximum likelihood; consequently, how long it takes to achieve the local maximum likelihood. Third, a direct measurement of the user for PAEM vs BW.

Our tests followed the hypothesis that a human operator begins with no knowledge about the dataset. In other words, we consider no previous data mining or machine learning techniques have been performed. For the last two sets of test (Section 3.2.2 and 3.2.3), the BW initialization followed the standard strategy [31], it was performed through random normalized numbers to all parameters.

We used 12 different time series for models ranging from 2 to 4 states. Regarding these 36 tests, for each BW execution, 50 different seeds were used. To avoid outliers, the best and the worst 5 were taken out. From these 40, we used the best, the worst and the average measurements to compare against PAEM.

3.2.1 Likelihood

Prior to the user time and iteration tests, we compared the fitness of BW and PAEM with only one initialization. Since the Expectation-Maximization part is the same, if the BW parameters are not equiprobable they should reach the same likelihood. Otherwise, if BW is inferior, it means that not all randomized parameters are good as an input. If PAEM is inferior, it means that the pre-fitting fails. Table 3.1 shows this experiment with BW and PAEM, where $*$ means that we used an equiprobable λ to initialize the BW. In fact, if the BW's parameters values are not equiprobable, it tends to converge to a maximum likelihood. The problem usually happens when an equiprobable λ or Γ is given as an input, which is trivial to avoid.

Table 3.1: Experimental model measurements using random numbers as parameter initialization.

Model	# states	mLLk	AIC	BIC
BW (Equiprobable λ)	2	1268.92	2547.84	2560.86
BW	2	635.32	1280.65	1293.67
PAEM	2	635.32	1280.65	1293.67
BW (Equiprobable λ)	3	1268.92	2559.84	2588.50
BW	3	510.22	1042.44	1071.09
PAEM	3	510.22	1042.44	1071.09
BW (Equiprobable λ)	4	1268.92	2575.84	2625.34
BW	4	471.82	981.65	1031.15
PAEM	4	471.82	981.65	1031.15

Although an equiprobable λ suggests a bad fitting, this is not true for all scenarios. Despite a tiny improvement, in some cases, an equiprobable λ retrieved a better likelihood. For the other datasets, a similar phenomenon occurred in some models with more than 3 states. It suggests that a simple condition to avoid an equiprobable parameter may not be a good solution.

Considering one decimal precision, PAEM reaches a better likelihood in 3 cases against 2 from the pure BW. Table 3.2 shows these cases. For all the 36 experiments, PAEM was better in 17 occurrences against 19 of the pure BW. However the difference in the vast majority of these cases lies in a nth decimal precision, which can be seen in Table 3.2, it represents a negligible probability.

Table 3.2: -Log-Likelihood comparison, cases where the difference exceeds a precision of one float point.

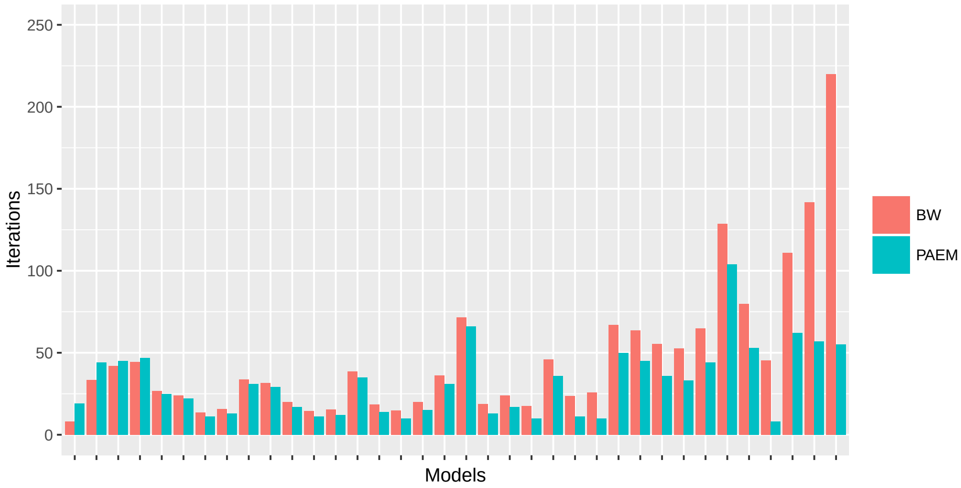
BW	PAEM	$\Delta\%$	
210.13	209.71	0.01%	favorable to PAEM
740.52	697.38	5.80%	favorable to PAEM
773.44	718.20	7.10%	favorable to PAEM
471.82	489.84	3.60%	favorable to BW
321.55	322.99	0.40%	favorable to BW

3.2.2 Iterations

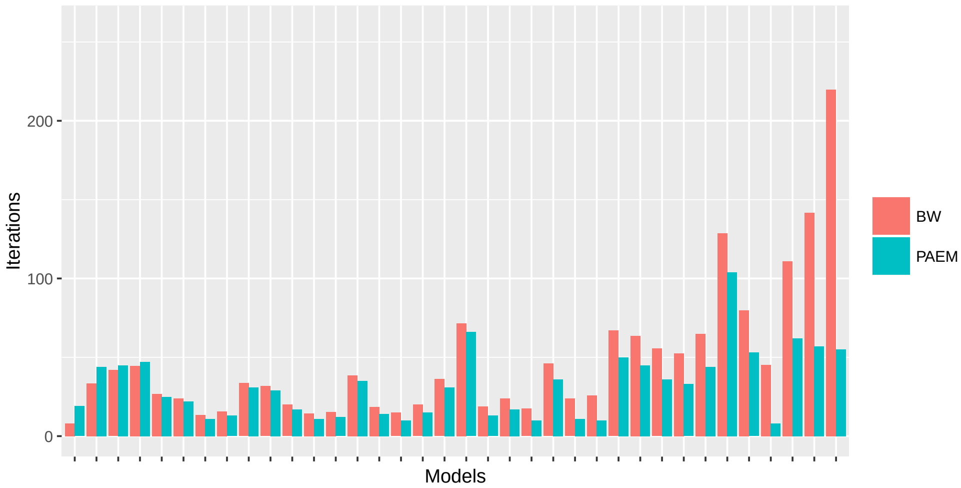
As in the previous section, we started by checking our hypothesis through experiments using 50 different seeds to BW, excluding the best and the worst 5. From the 40 remaining we collected the best, the average, and the worst case concerning the BW initialization and its number of iterations to reach a convergence. As PAEM generates the parameters through a deterministic technique, it only needs one initialization. Figure 3.6(a) shows the average scenario. Figure 3.6(b) shows the most favorable scenario to the PAEM, and Figure 3.6(c) shows the best scenario for BW [4].

In these pictures, we can clearly see PAEM requiring fewer iterations to find a convergence (right side), while just in a few cases, the random parameters outperformed PAEM (left side). Furthermore,

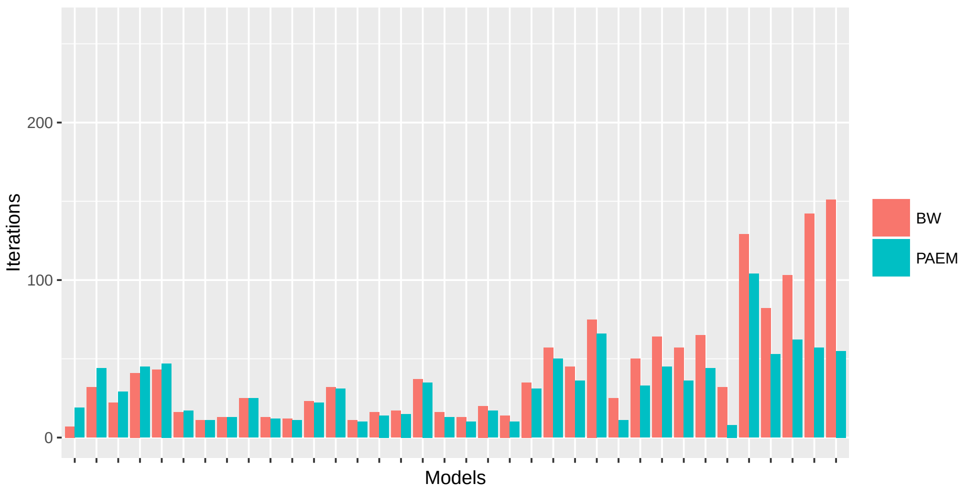
[4] To ease the visualization, these outputs were ordered according to the advantage (left to right, BW's to PAEM's).



(a) Average scenario



(b) Most favorable scenario to PAEM.



(c) Most favorable scenario to BW.

Figure 3.6: Iteration comparison.

all the iterations described here are retrieved from the experiments described in the Section 3.2.1, which shows an equivalent likelihood, between BW and PAEM, for 87% of the cases. Also, the far most significant scenario which PAEM performed poorly, loses with a difference of 3.6% (Table 3.2, BW=471.8).

Concerning all the results for the ordinary BW; 40 seeds for all the 12 series and the 2,3, and 4 states model; the random values for BW got an average of 47.4 iterations against 25 from PAEM's. This shows a significant improvement for the initial parameters quality against the traditional random approach. Furthermore, in the vast majority of the tests, PAEM found a convergence with fewer iterations (Figures 3.6(a) and 3.7).

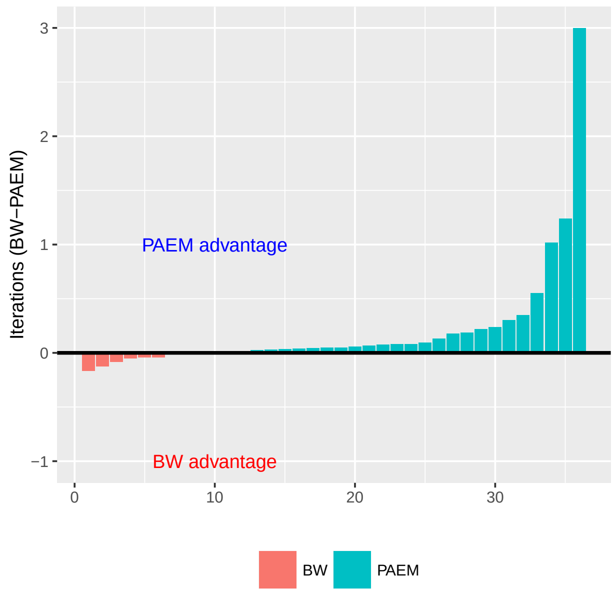
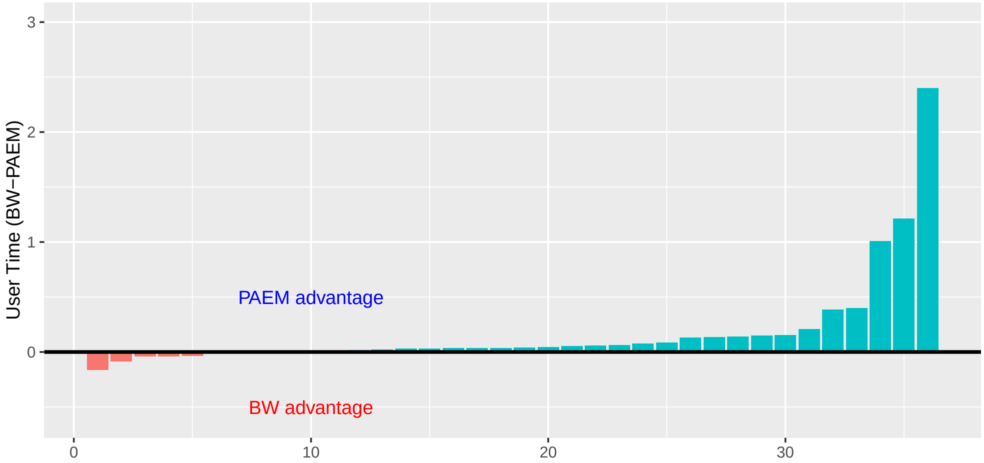


Figure 3.7: Iterations comparison $(BW - PAEM)$ considering the average case. Under zero BW had a better performance, PAEM otherwise.

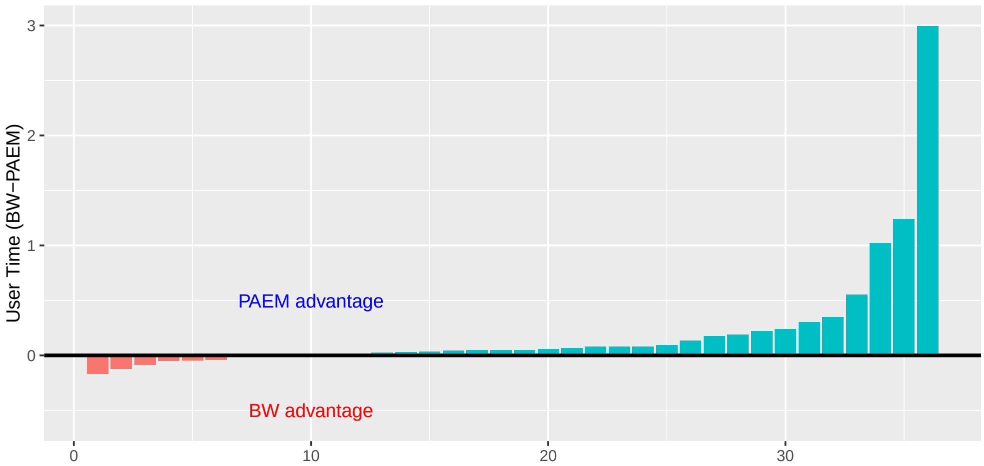
3.2.3 User Time

Since both, BW and PAEM, tend to converge to the same likelihood and the cost to randomize values to BW is trivial; the real advantage of PAEM relies on a faster convergence. Which is given by fewer iterations derived by a better likelihood at the first iteration. Here we measured the total quantity of CPU time spent in a user-mode, which is only the actual time used in executing the specific process.

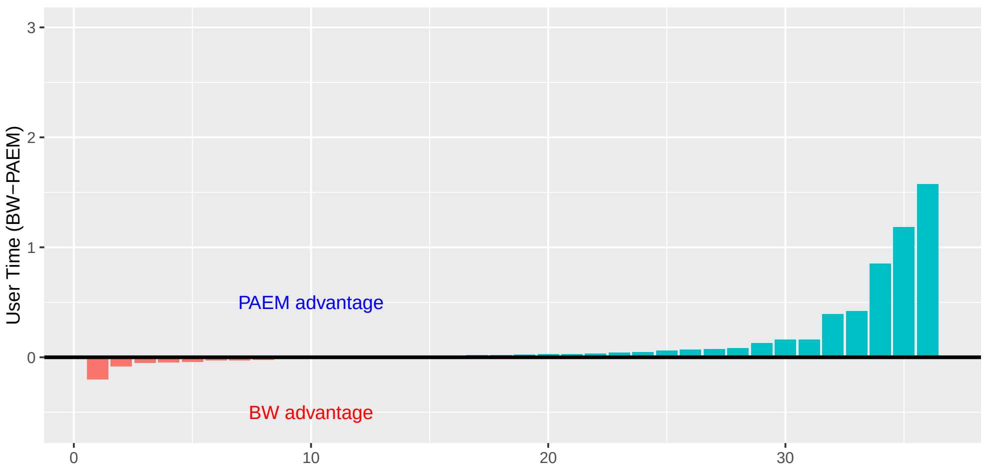
We performed time measurements to see how fast each procedure is in relation with BW. Although the running time is highly correlated with the number of E-M iterations, a lower running time is the final goal, therefore, a more precise measure regarding the time actually used by BW and PAEM. In a standard machine, Intel i5, 2.3GHz, 8GB, a four states model had an average time of 0.34 seconds running with PAEM and 1.17 seconds running with BW. This difference is directly linked with the number of iterations. As described in the previous sections, in most cases,



(a) Average scenario



(b) Most favorable scenario to PAEM.



(c) Most favorable scenario to BW.

Figure 3.8: User Time comparison $(BW - PAEM)$. Under zero BW had a better performance, PAEM otherwise.

PAEM's pre-fitting tends to avoid at least one iteration of the forward-backward procedure, which costs $\vartheta(N^2T)$, which is more than PAA $\vartheta(T)$.

The user time spent, from both, had a strong correlation with their number of iterations. This is visible comparing Figures 3.7 and 3.8(a). Specifically, BW had a correlation average of 93% and PAEM 76%. Which can be explained by the different seeds in BW and the lack of a precise control considering an ordinary machine running other applications. Also, PAEM's pre-fitting has a fixed running time for series with the same length, which has a different impact according to the series number of iterations necessary to find a convergence.

Figure 3.8 shows an overview of PAEM's performance. Now, considering the average scenario, we look for each of the 36 experiments. Thus, the radar showed by Figure 3.9, illustrates the total time spent in relation to the best technique for each time series. From this figure, we can clearly see the time percentage difference from each technique and for each time series. This plot shows the overall better performance of PAEM, failing in just 6 cases, which are the time series 1, 10, 16, 21, 24, and 26.

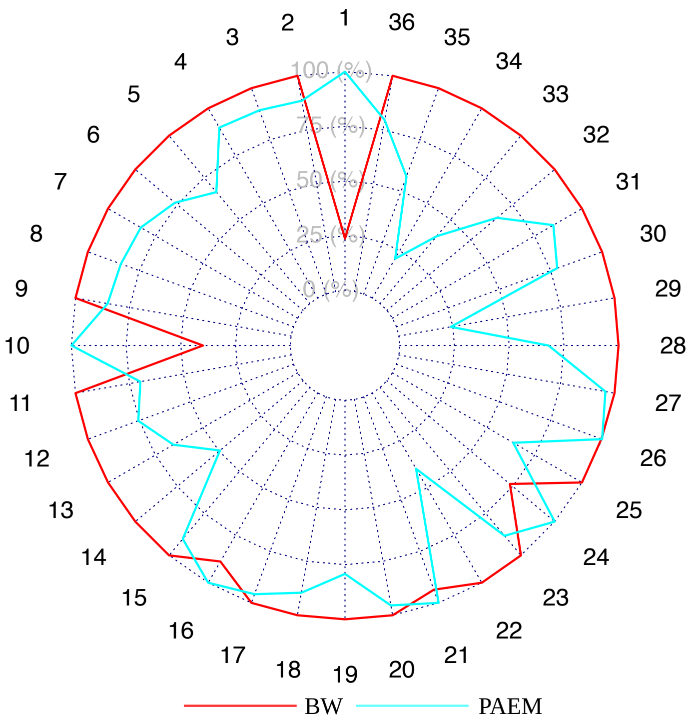


Figure 3.9: Radar chart showing the slowest process taking 100% of the time. An average scenario considering the user time (Same data at Figure 3.8(a), red line shows BW).

Among these time series, a poor performance was achieved on the time series #1 and #10, which happens to be the shortest time series in the experiment. Considering BW and PAEM, respectively, for the first time series (average case) required 0.026 and 0.110 seconds. Time series X10, required, respectively, 0.025 and 0.064 seconds.

The opposite is observed is the larger models with slowest user time. For instance, considering the time series #31...#36, BW achieved a convergence with 1.15, 2.02, 0.16, 2.34, and 1.86 seconds.

PAEM did the same with 0.75, 0.81, 0.03, 1.33, and 1.48 seconds. Finally, we can see a difference between the performance regarding the number of states.

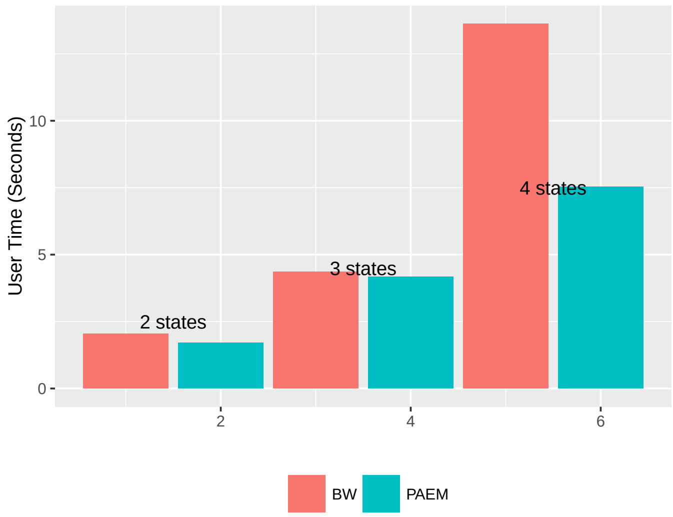


Figure 3.10: User Time comparison according to the model number of states.

Final Remarks

Though exhaustive tests, with different kinds of data sets, we found PAEM to be faster than BW with its traditional stochastic initialization. This performance is due to, usually, one less iteration in the E-M procedure. In fact, as shown in Figure 3.6, in the most executions, the convergence is achieved using smaller number of iteration than the pure BW.

Furthermore, the computational cost needed to the piecewise approximation $(\vartheta(T))$ is smaller than one BW's iteration $(\vartheta(N^2T))$.

However, it is important to emphasize the overall better performance and that PAEM does not aim to be an optimal solution. We focus on a simple alternative to the initial and usual randomization of parameters. Although there are other techniques that improves the original BW, PAEM lies in a simple initialization that is fast and easy to implement, making it a suitable alternative to perform HMM fitting.

For the future improvements, we shall focus on measuring the initial likelihood and the user time according to different initializations. PAEM is based on a simple Piecewise Aggregation technique. It may have a better global performance if a more advanced technique is used instead of Piecewise Aggregation. However, the time spent to pre-process the data must be lower than the original one. Otherwise, the overall performance might decrease. Other important test is to detect how efficient PAEM scales regarding models with different number of states.

Chapter 4

Modeling large systems with Stochastic Automata Networks (SAN)

Unlike Markov chains and hidden Markov models, SANs are modular and this feature allows us to represent a system by representing its components. Such feature makes unnecessary the creation of states from the combination of all variables. In other words, a modular system intrinsically represents all the combination between its components. A model is built through the construction of separated automata that will interact with each other like a network of states. Furthermore, to read independent variables as independent structures are more human understandable than read some enormous matrix representing all possible states.

In this chapter, are described the SAN formalism among with some examples. After, we describe a SAN model for a geological phenomenon related to the flow of sediments in marginal basins. This model [9] plays an important role for the contributions of this work. It was the first attempt to create a SAN model with geological data; which, due to its nature, are complex to be structured and modeled environment. As can be seeing in Table 4.2, this model is accurate to represent all modeled paleo events.

4.1 Stochastic Automata Networks

Stochastic Automata Networks (SAN) is a structured Markovian formalism that describes a complete system as a collection of subsystems that interact with each other [59]. The main advantage of SAN is that we can represent a system with millions of achievable states in a human understandable model, since the modular property can be used to describe an equivalent Markov chain model [8].

Unlike MCs and HMMs, there are two types of events in a SAN model, local and synchronizing. Local events operate over only one automaton, changing its states by triggering a transition without interfering on the other automata. Synchronizing events can change simultaneously two or more automata, triggering simultaneous transitions, one per automaton. In other words, local events change the global state by changing only the local automaton, unlike the synchronizing events that work like a trigger which can cause the change of more than one state in more than one automaton at the same time [9]. Each event in an SAN model has an occurrence rate, which can be constant or a function of other automata current states. Also, for each possible transition of the same event there is a choice or routing probability. Figure 4.1 represents a SAN model with 2 completely independent automata and its equivalent Markov chain.

The internal structure of a SAN is given by an algebraic descriptor. Basically, a SAN has a set of automata that have a set of transition events both for self or to the others. If the event does not affect other automata, it is called a local event, otherwise, if the event triggers another event it is called a synchronizing event. All these states and transitions are structured in a tensor (Kronecker) format.

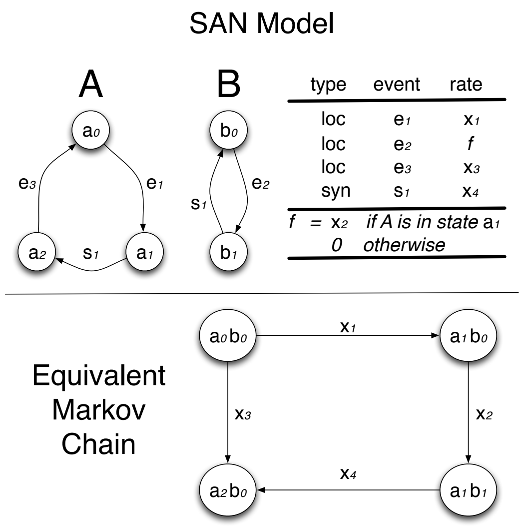


Figure 4.1: SAN Model compared to its equivalent Markov chain

This compact representation of SAN enables memory savings, since storing some small matrices (of the order of each automaton) requires less memory than storing the infinitesimal generator itself. Furthermore, the tensor format of SAN enables the use of specialized algorithms [45, 46, 60, 89] which are more efficient than those used for solving large Markov chains. Example of SAN models can be found in [14, 22, 35, 44, 53, 56, 61, 84].

4.1.1 Internal structure

Notation

$\bigoplus$	Tensor sum
$\bigotimes$	Tensor product
N	Number of automata and/or matrices
e_m	The m_{th} local event in a SAN
s_m	The m_{th} synchronizing event in a SAN
Q	A Kronecker structure, given by tensor operations ($\bigotimes$ or $\bigoplus$)
$\Gamma^{(A)}$	A transition rate matrix, named A
$\phi_{i,j}$	Transition rate from state i to state j, an element of Γ

SAN's structure is stored as Kronecker descriptors, which uses tensor algebra operations. It allows the construction of a more compact description than other Markovian formalisms.

Tensor operations

Given two matrices, $\Gamma^{(A)}$ and $\Gamma^{(B)}$, represented by:

$$\Gamma^{(A)} = \begin{bmatrix} \phi^{(A)}_{0,0} & \phi^{(A)}_{0,1} \\ \phi^{(A)}_{1,0} & \phi^{(A)}_{1,1} \end{bmatrix} \qquad \Gamma^{(B)} = \begin{bmatrix} \phi^{(B)}_{0,0} & \phi^{(B)}_{0,1} & \phi^{(B)}_{0,2} & \phi^{(B)}_{0,3} \\ \phi^{(B)}_{1,0} & \phi^{(B)}_{1,1} & \phi^{(B)}_{1,2} & \phi^{(B)}_{1,3} \\ \phi^{(B)}_{2,0} & \phi^{(B)}_{2,1} & \phi^{(B)}_{2,2} & \phi^{(B)}_{2,3} \end{bmatrix}$$

The tensor product $Q = A \otimes B$ is denoted as follows:

$$Q = \begin{bmatrix} \phi^{(A)}_{0,0}\Gamma^{(B)} & \phi^{(A)}_{0,1}\Gamma^{(B)} \\ \phi^{(A)}_{1,0}\Gamma^{(B)} & \phi^{(A)}_{1,1}\Gamma^{(B)} \end{bmatrix} =$$

$$\left(\begin{array}{cccc|cccc}
\phi^{(A)}_{0,0}\phi^{(B)}_{0,0} & \phi^{(A)}_{0,0}\phi^{(B)}_{0,1} & \phi^{(A)}_{0,0}\phi^{(B)}_{0,2} & \phi^{(A)}_{0,0}\phi^{(B)}_{0,3} & \phi^{(A)}_{0,1}\phi^{(B)}_{0,0} & \phi^{(A)}_{0,1}\phi^{(B)}_{0,1} & \phi^{(A)}_{0,1}\phi^{(B)}_{0,2} & \phi^{(A)}_{0,1}\phi^{(B)}_{0,3} \\
\phi^{(A)}_{0,0}\phi^{(B)}_{1,0} & \phi^{(A)}_{0,0}\phi^{(B)}_{1,1} & \phi^{(A)}_{0,0}\phi^{(B)}_{1,2} & \phi^{(A)}_{0,0}\phi^{(B)}_{1,3} & \phi^{(A)}_{0,1}\phi^{(B)}_{1,0} & \phi^{(A)}_{0,1}\phi^{(B)}_{1,1} & \phi^{(A)}_{0,1}\phi^{(B)}_{1,2} & \phi^{(A)}_{0,1}\phi^{(B)}_{1,3} \\
\phi^{(A)}_{0,0}\phi^{(B)}_{2,0} & \phi^{(A)}_{0,0}\phi^{(B)}_{2,1} & \phi^{(A)}_{0,0}\phi^{(B)}_{2,2} & \phi^{(A)}_{0,0}\phi^{(B)}_{2,3} & \phi^{(A)}_{0,1}\phi^{(B)}_{2,0} & \phi^{(A)}_{0,1}\phi^{(B)}_{2,1} & \phi^{(A)}_{0,1}\phi^{(B)}_{2,2} & \phi^{(A)}_{0,1}\phi^{(B)}_{2,3} \\
\hline
\phi^{(A)}_{1,0}\phi^{(B)}_{0,0} & \phi^{(A)}_{1,0}\phi^{(B)}_{0,1} & \phi^{(A)}_{1,0}\phi^{(B)}_{0,2} & \phi^{(A)}_{01,0}\phi^{(B)}_{0,3} & \phi^{(A)}_{1,1}\phi^{(B)}_{0,0} & \phi^{(A)}_{1,1}\phi^{(B)}_{0,1} & \phi^{(A)}_{1,1}\phi^{(B)}_{0,2} & \phi^{(A)}_{1,1}\phi^{(B)}_{0,3} \\
\phi^{(A)}_{1,0}\phi^{(B)}_{1,0} & \phi^{(A)}_{1,0}\phi^{(B)}_{1,1} & \phi^{(A)}_{1,0}\phi^{(B)}_{1,2} & \phi^{(A)}_{1,0}\phi^{(B)}_{1,3} & \phi^{(A)}_{1,1}\phi^{(B)}_{1,0} & \phi^{(A)}_{1,1}\phi^{(B)}_{1,1} & \phi^{(A)}_{1,1}\phi^{(B)}_{1,2} & \phi^{(A)}_{1,1}\phi^{(B)}_{1,3} \\
\phi^{(A)}_{1,0}\phi^{(B)}_{2,0} & \phi^{(A)}_{1,0}\phi^{(B)}_{2,1} & \phi^{(A)}_{1,0}\phi^{(B)}_{2,2} & \phi^{(A)}_{1,0}\phi^{(B)}_{2,3} & \phi^{(A)}_{1,1}\phi^{(B)}_{2,0} & \phi^{(A)}_{1,1}\phi^{(B)}_{2,1} & \phi^{(A)}_{1,1}\phi^{(B)}_{2,2} & \phi^{(A)}_{1,1}\phi^{(B)}_{2,3}
\end{array}\right)$$

Considering that Q holds all the model transitions, we can roughly say that Q can represent a system originally described by $\Gamma^{(A)}$ and $\Gamma^{(B)}$. Yet, it is important to emphasize that a SAN in its core is a formalism designed for continuous system; thus Γ is a rate transition matrix.

The tensor sum $Q = \Gamma^{(A)} \oplus \Gamma^{(C)}$ is given through tensor products, thus:

$$\Gamma^{(A)} \bigoplus \Gamma^{(C)} = \Gamma^{(A)} \bigotimes I_{n2} + I_{n1} \bigotimes \Gamma^{(C)}$$

Where, I corresponds to the identity matrix of order $_{n1}$ in case of $\Gamma^{(A)}$ or $_{n2}$ in case of $\Gamma^{(C)}$. Since both sides of a matrix addition (...+...) must have identical dimensions, it requires square matrices to perform a tensor sum [58].

$$\Gamma^{(A)} = \begin{bmatrix} \phi^{(A)}_{0,0} & \phi^{(A)}_{0,1} \\ \phi^{(A)}_{1,0} & \phi^{(A)}_{1,1} \end{bmatrix} \qquad \Gamma^{(C)} = \begin{bmatrix} \phi^{(C)}_{0,0} & \phi^{(C)}_{0,1} & \phi^{(C)}_{0,2} \\ \phi^{(C)}_{1,0} & \phi^{(C)}_{1,1} & \phi^{(C)}_{1,2} \\ \phi^{(C)}_{2,0} & \phi^{(C)}_{2,1} & \phi^{(C)}_{2,2} \end{bmatrix}$$

Extending each element, the tensor sum $Q = \Gamma^{(A)} \oplus \Gamma^{(C)}$ is denoted as follows:

$$\left(\begin{array}{ccc|ccc}
\phi^{(A)}_{0,0} + \phi^{(B)}_{0,0} & \phi^{(B)}_{0,1} & \phi^{(B)}_{0,2} & \phi^{(A)}_{0,1} & 0 & 0 \\
\phi^{(B)}_{1,0} & \phi^{(A)}_{0,0} + \phi^{(B)}_{1,1} & \phi^{(B)}_{1,2} & 0 & \phi^{(A)}_{0,1} & 0 \\
\phi^{(B)}_{2,0} & \phi^{(B)}_{2,1} & \phi^{(A)}_{0,0} + \phi^{(B)}_{2,2} & 0 & 0 & \phi^{(A)}_{0,1} \\
\hline
\phi^{(A)}_{1,0} & 0 & 0 & \phi^{(A)}_{1,1} + \phi^{(B)}_{0,0} & \phi^{(B)}_{0,1} & \phi^{(B)}_{0,2} \\
0 & \phi^{(A)}_{1,0} & 0 & \phi^{(B)}_{1,0} & \phi^{(A)}_{1,1} + \phi^{(B)}_{1,1} & \phi^{(B)}_{1,2} \\
0 & 0 & \phi^{(A)}_{1,0} & \phi^{(B)}_{2,0} & \phi^{(B)}_{2,1} & \phi^{(A)}_{1,1} + \phi^{(B)}_{2,2}
\end{array}\right)$$

Now, consider a structure with N independent automata ($\Gamma^{(1)}$, $\Gamma^{(2)}$,...,$\Gamma^{(N)}$). The infinitesimal generator will be given by: $Q = \bigoplus_{i=1}^{N} \Gamma^{(i)}$. In the case of having any dependent automata on the structure, a representation for synchronizing events is used. There are two different ways that stochastic automata can interact [58]. Via *functional transition rate* and by *synchronizing transitions*.

Functional transitions are based on functions that establish the change rate of an automaton depending on the state of others. Synchronizing transitions are based on a master/slave dependency, where an automaton (master) forces others (slaves) to change their state.

Functional transitions change the representation by adding variables in the structure. Thus, the elements in the matrices are no longer only constants; they can be expressed as variables of the global state space. Let us assign values to $\Gamma^{(A)}$ and $\Gamma^{(C)}$.

$$\Gamma^{(A)} = \begin{bmatrix} -\lambda_1 & \lambda_1 \\ \lambda_2 & -\lambda_2 \end{bmatrix} \qquad \Gamma^{(C)} = \begin{bmatrix} -\sigma & \sigma & 0 \\ 0 & -\mu_2 & \mu_2 \\ \mu_3 & 0 & -\mu_3 \end{bmatrix}$$

Here, all variables can obey a certain condition, *e.g.*, σ can be equal to μ_1 if the automaton, represented by $\Gamma^{(A)}$ is in the first state, $2\mu_1$ otherwise. The tensor, global, sum follows the same rule and can be formally described as: $Q = \Gamma^{(A)} \bigoplus_{(g)} \Gamma^{(C)}$. Generalizing for a network composed by N automata $Q = \bigoplus_{(g)\ i=1}^{N} \Gamma^{(i)}$. Thus:

$$Q = \Gamma^{(A)} \bigoplus_{(g)} \Gamma^{(C)} =$$

$$\left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+2\mu_1) & 2\mu_1 & 0 \\ 0 & \lambda_2 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ 0 & 0 & \lambda_2 & \mu_3 & 0 & -(\lambda_2+\mu_3) \end{array}\right)$$

Unlike, functional transitions, Synchronizing transitions change the global equation by adding E synchronizing events. Now, to differentiate the local and the synchronizing events, we use Γ_l to indicate the local matrices; thus, $Q = \bigoplus_{(g)\ i=1}^{N} \Gamma_l^{(i)}$. The E synchronizing events can be represented as ordinaries tensor products [58, 118]. The matrices storing the occurrence rate are described as Γ_{ep+}, thus $\Gamma_{ep+}^{(e)}$ corresponds to the e-th matrix. As each matrix needs a correspondent diagonal adjustment, we use $\Gamma_{ep-}^{(e)}$. Therefore, they can be added to the equation as: $\sum_{e=1}^{N} \left(\bigotimes_{i=1}^{N} \Gamma_{ep+}^{(e)} + \bigotimes_{i=1}^{N} \Gamma_{ep-}^{(e)} \right)$, which leads to a generalized description as:

$$Q = \bigoplus_{(g))i=1}^{N} \Gamma_l^{(i)} + \sum_{e=1}^{N} \left(\bigotimes_{i=1}^{N} \Gamma_{ep+}^{(e)} + \bigotimes_{i=1}^{N} \Gamma_{ep-}^{(e)} \right) \tag{4.1}$$

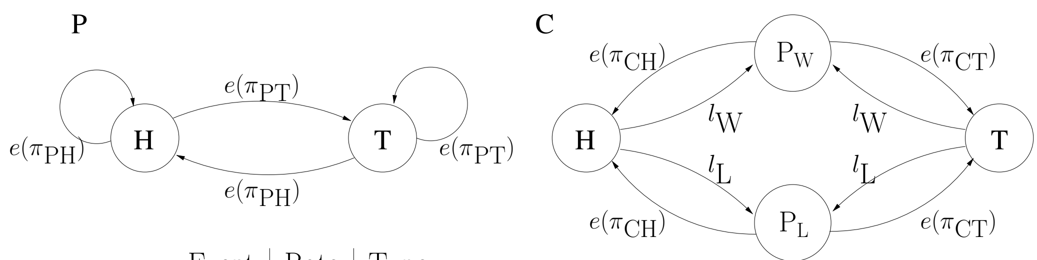
From a user point of view, a complete SAN model is represented by a SAN code that can be executed [23, 104] to generate a set of probabilities, including those which may be generated in a Markov chain. We can represent and compare a SAN model with a MCs through its infinitesimal generator, this gives us the possibility to compare SAN with other formalisms (as described in [47]) and model discrete systems as well [16].

Next, examples are given to illustrate the use of SAN. First, a toy example aiming to show the use of SAN automata, with rates for local and synchronous event 4.1.2. Then, a model derived from an applied research showing all the elements and the possibility to use SAN as a scientific tool to get knowledge through stochastic models 4.2.

4.1.2 SAN model, Heads or Tails

Heads or Tails is a coin-tossing game with a series of known variations. The one presented here has one coin and one player. At each coin flip, the player must guess which side will come with the face-up. The player wins whenever his expectation is confirmed, otherwise he loses.

Figure 4.2 presents a possible SAN model for this problem. It has two automata: one representing the player's guess (automaton P), which is either *heads* or *tails*; and the other representing a tossing engine (automaton C), which is capable of tossing the coin and computing the outcome.



Event	Rate	Type
e	μ	syn
l_W	f	loc
l_L	g	loc

$$f = (st\ \mathrm{P}\ == \ st\ \mathrm{C}) * \mu_{\mathrm{W}}$$

$$g = (st\ \mathrm{P}\ != \ st\ \mathrm{C}) * \mu_{\mathrm{L}}$$

Figure 4.2: Heads or Tails – SAN Modeling.

Automaton P has two states: "H" and "T". Routing probabilities π_{PH} and π_{PT} indicate the player guesses, being the probability of choosing heads and tails, respectively. Automaton C has four states: "H" and "T", indicating the actual result of the coin-tossing; "P_W" (player wins) for the case the result matches the player's guess and "P_L" (player loses) otherwise. The coin has probabilities π_{CH} and π_{CT} of coming with the heads or tails side up, respectively, as the toss result.

The coin-tossing and the player's guess are simultaneous, represented by a synchronizing event, e, which triggers transitions leading to "H" or "T" in both automata. Such event can only happen when each automaton is in condition to attend it, which means automaton C must be either in "P_W" or "P_L".

As soon as the synchronizing event e happens, it follows a verification of whether the player has won or lost the match. Depending on the result, either of the local events on automaton C

is triggered: event "l_{W}" leads to state "P_W" (player wins), while event "l_{L}" leads to state "P_L" (player loses).

4.2 A SAN model for prediction of geological stratal stacking patterns

Everything in nature involves a great number of variables. This amount of variables makes hard to create accurate models that simulate natural events. However, in some cases, key points may affect the entire event. In this model, some characteristics of a specific and complex (geological) natural event are exposed and modeled with SAN formalism[1]. Here are described a SAN model that predict geological stratal stacking patterns resulting from sea level, sediment input and subsidence, considering last 130 million years. With these variables the model delivers percentage chances for some natural events, such a forced regression of sea level. The model result is a set of probabilities that can be compared with geological facts and hypothesis; thus, we can emphasize possible discrepancies and also improve the chance of a good estimation of both past and future geological state.

This model is a first attempt to represent the evolution of a sedimentary basin using the SAN formalism. Pelotas basin, located in southernmost Brazil, was the basin chosen to attend this purpose. Considerable amounts of information are available for this marginal basin, including stratigraphic architecture (Figure 4.3), changes in sediment yield, and subsidence.

In seismic reflection surveys, the measurement of the time required for a seismic wave or pulse to return to the surface after reflection from subsurface interfaces of different physical properties provides information on the arrangement of sedimentary strata. Color lines depict reflectors from different ages (Vertical exaggeration 8:1). Most of the information from Pelotas basin used to this application was extracted from Contreras *et al.* [42].

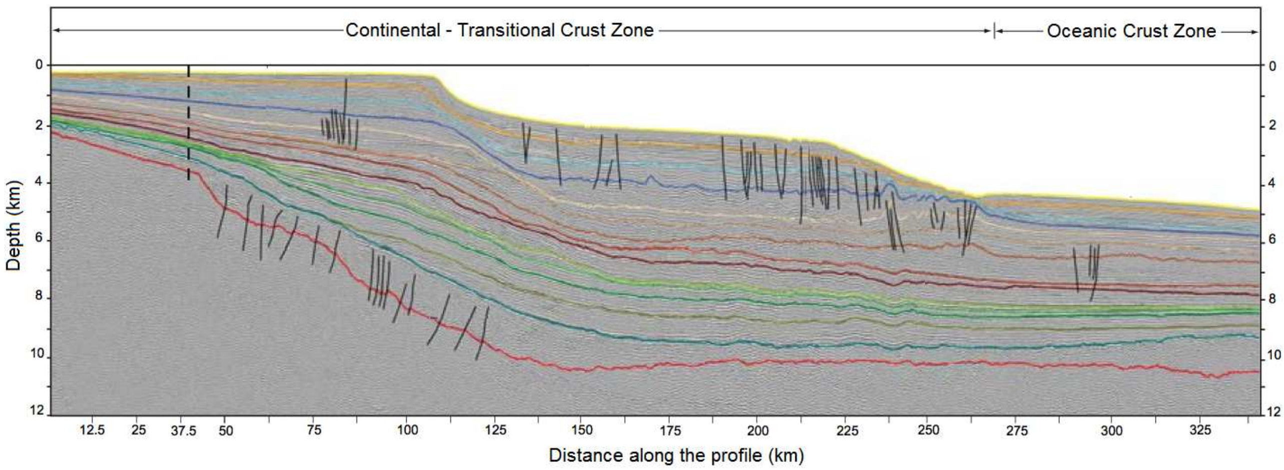


Figure 4.3: Interpreted seismic reflection profile from Pelotas basin [42].

[1]The full version of this work can be found at [9].

4.2.1 Geology background

Sedimentary basins constitute large accumulations of sediments. The amounts and types of sediments depend on factors such as climate and relief, and for this reason, sedimentary basins constitute essential records of the climatic and tectonic history of the Earth.

The study of sedimentary basins is primarily based on drilling and seismic surveys, which provide information on the composition and arrangement of sedimentary rock strata. The configuration of strata results from the interplay between sediment supply and relative base level changes, which defines the accommodation space for those sediments. In marginal sedimentary basins, *i.e.*, basins along a continental margin, the base level is determined by the relative sea level, which, in turn depends on the global (*eustatic*) sea level changes and on the vertical movement of the underlying crust (Figure 4.4). Crustal movement can be either upwards (*uplift*) or downwards (*subsidence*).

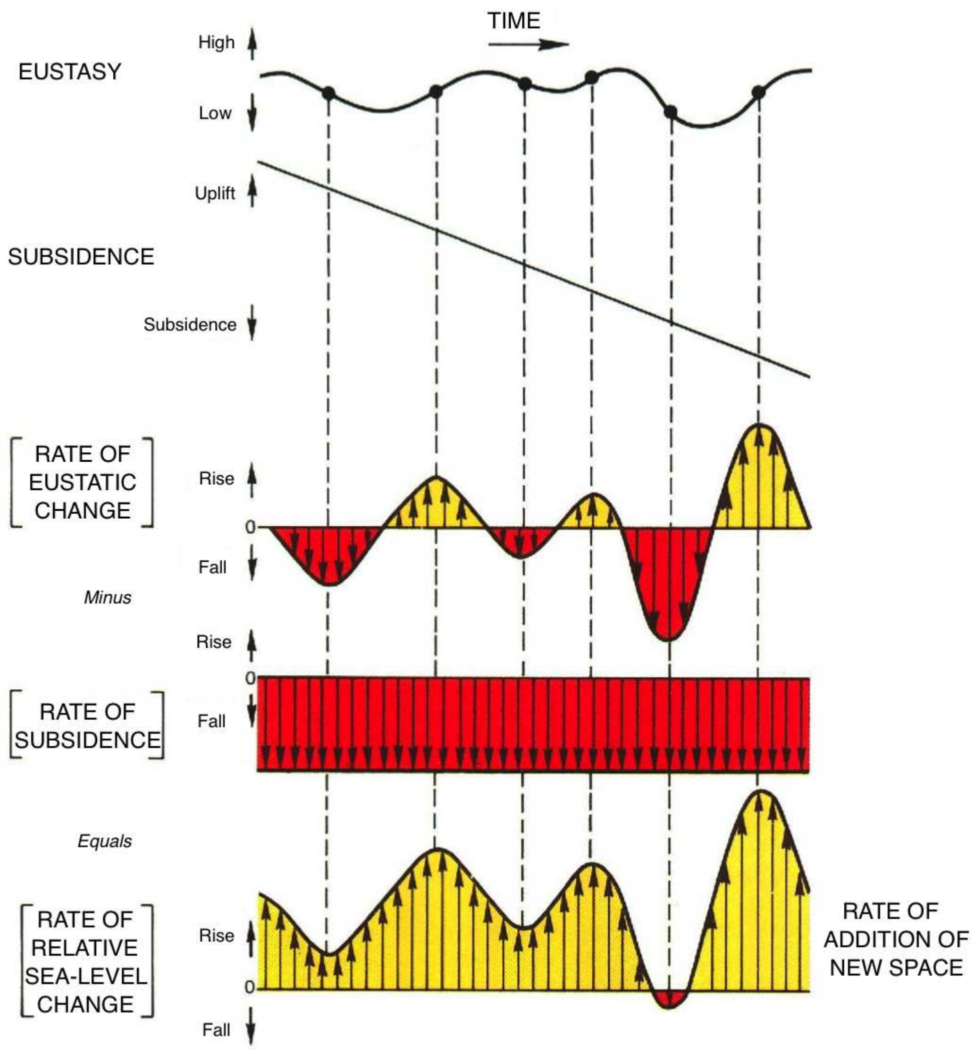


Figure 4.4: Rate of relative sea level change and addition of new accommodation space as a function of eustatic (global) sea level changes and subsidence [100].

Four generic types of deposit can be distinguished as a function of relative sea level changes, as summarized in Figure 4.5 (a). In (a), sediment supply is assumed to be constant, as pointed

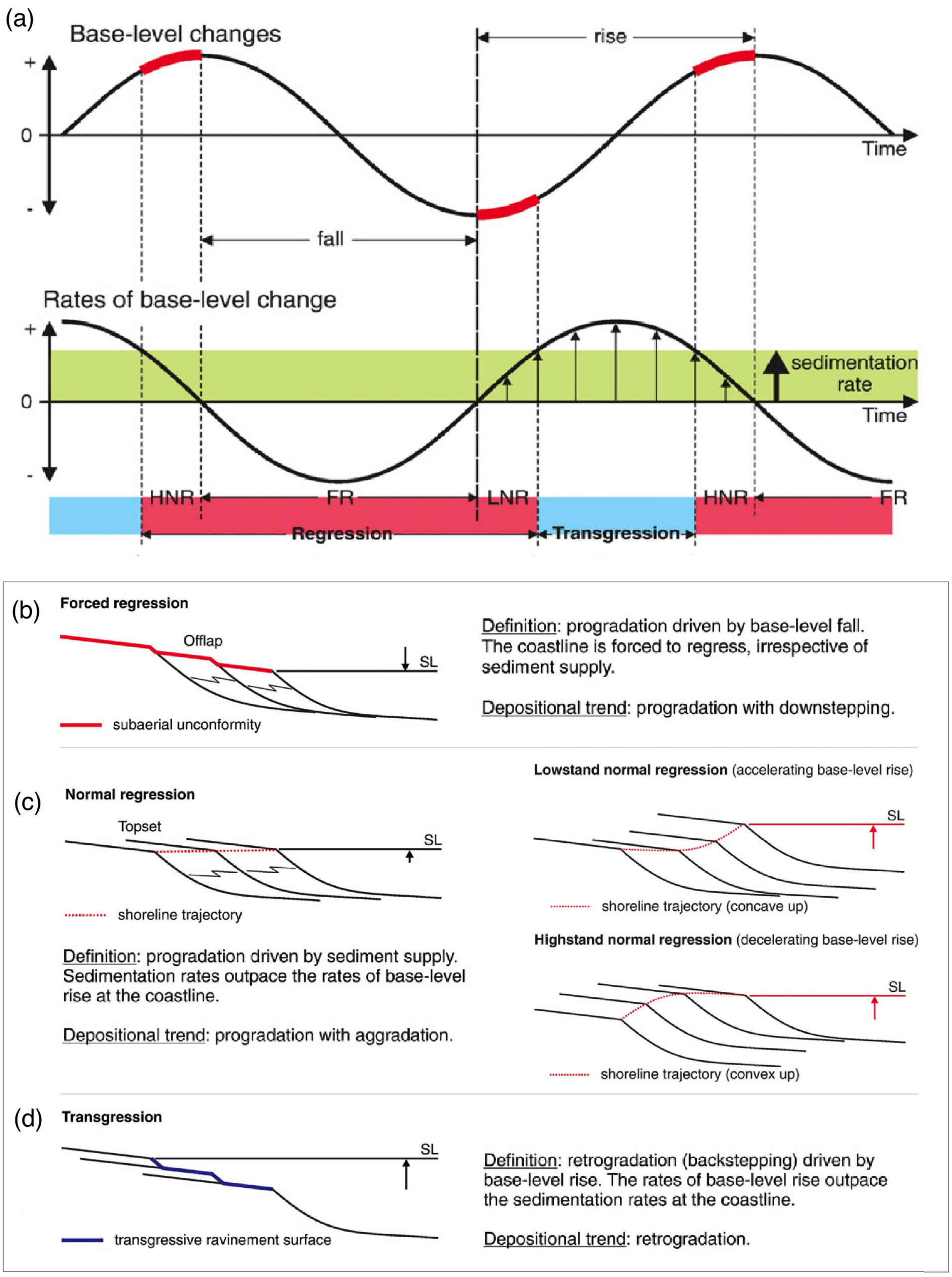


Figure 4.5: Genetic types of stratal stacking patterns as a function of changes in relative sea level.

by the shaded green area. The upper curve corresponds to base-level (sea-level changes) and the lower curve corresponds to the rate of sea-level change. The corresponding arrangement of strata for forced regression (*FR*), lowstand normal regression (*LNR*), highstand normal regression (*HNR*) and transgression (*T*) are depicted respectively in (b), (c), and (d); adapted from [27, 28].

Forced regressions occur whenever relative sea level falls. During these events, sediments prograde seawards and the shoreline advances with downstepping (Figure 4.5 (b)). Normal regressions occur whenever sea level is rising but sedimentation rate outpaces the rate of sea-level rise, avoiding shoreline retreat. Normal regressions may either occur during a relative sea level lowstand or highstand. During lowstands, there is an acceleration of sea level rise and the rate of progradation decreases with time while the rate of vertical accretion (known as *aggradation*) increases with time

(Figure 4.5 (c)). Conversely, as sea level decelerates at the end of a sea-level rise trend, there is a decrease in the rate of aggradation with time and an increase in progradation (Figure 4.5 (c)). Finally, transgressions occur when the rate of sea level rise is the highest and the sediment supply is not enough to compensate for it. During transgressions, the shoreline retreats as the sea advances over the continent and sediments accumulate progressively landwards, in a configuration known as *retrogradation* (Figure 4.5 (d)).

Clearly, the simplified model in Figure 4.5 does not represent the whole complexity of processes that may affect the configuration of sedimentary strata such as variable sediment supply and shelf gradient. Nevertheless, it emphasizes the dominant role of relative-sea level changes and provides a clear picture of the main processes and possible stratigraphic architectures.

Contreras *et al.* [42] estimated subsidence rates and sediment flux using numerical modeling (Figure 4.6). These estimates were obtained considering the eustatic (global) sea level curve proposed by Hardenbol *et al.* [70] re-calibrated to a more recent geological timescale [68], as plotted in Figure 4.7. Higher order, *i.e.*, higher frequency, sea-level changes were excluded from the analysis because of the lower resolution of the seismo-stratigraphic data (2nd order depositional units, 3-50 Ma), their less defined amplitudes and partly disputed eustatic origin [41].

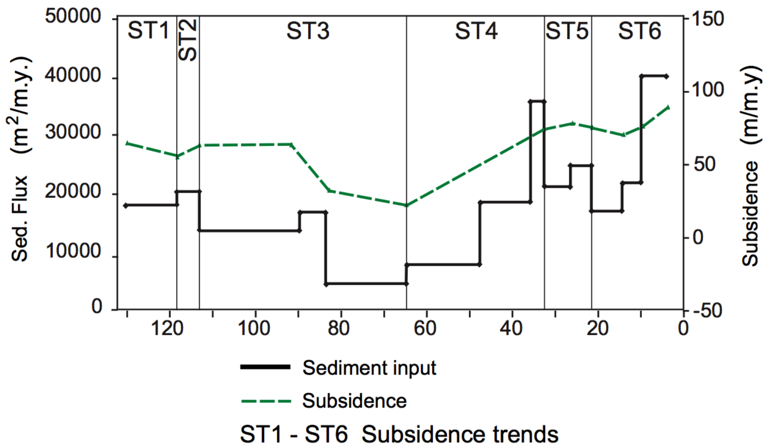


Figure 4.6: Estimates of subsidence and sediment input rates for the past 130 Ma for Pelotas basin based on numerical modeling. Reproduced from [42].

4.2.2 SAN model for the Pelotas basin strata configuration

The SAN model described here aims predicting the types of stratal stacking patterns expected for Pelotas Basin along the past 130 Ma. As seen on Section 4.2.1, there are four types of deposit respectively associated to forced regression (*FR*), lowstand normal regression (*LNR*), highstand normal regression (*HNR*) and transgression (*T*). The first one occurs whenever the relative sea level falls. The other three distinguish from one another by the relative contribution of relative sea level rise rate and sediment supply rate. Note that the relative sea level is a function of the global (*eustatic*) sea level and crustal vertical movement. In our study case, there is no uplift and all the

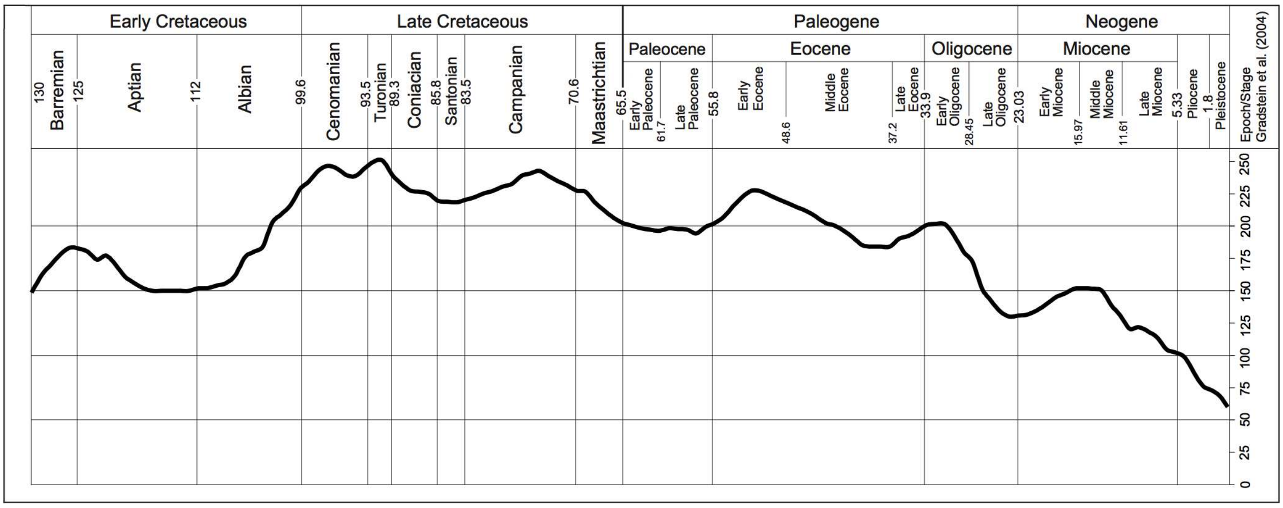


Figure 4.7: Second order eustatic sea-level curve from Hardenbol *et al.* [115], re-calibrated according to Gradstein *et al.* [68] geologic timescale as proposed by Contreras [41].

vertical movement is downwards, *i.e. subsidence*.

In this scenario it is possible to identify four important measures as the starting point for modeling, corresponding to the variation rates of: (1) eustatic sea level (*ESL*); (2) subsidence (*S*); (3) relative sea level (*RSL*); and (4) sediment supply (*SS*). Each of these four measures corresponds to one automaton in Figure 4.8, whose states were conceived based on a cluster of values achieved by Contreras *et al.* [42]. This figure also presents one automaton for time tracking and four other automata for restricting the model behavior. The complementary model description is given by Figure 4.8.

Figure 4.8 highlights the set of starting states in this model. Note that *Chronos* $= C_{130}$ is the starting point in time. It represents the time period between 130 Ma and 127 Ma , which comprises a time interval of 3 Ma. The state C_{130} is followed by the state C_{127}, such that the frequency associated to this transition is of one derivation every 3 Ma, which corresponds to the rate of this event: $e_{130,127} = 0.3333$. State C_{127} comprises the period between 127 Ma and 124 Ma, giving an event rate of $e_{127,124} = 0.3333$, coincidentially the same as before. State C_{124} comprises the period between 124 Ma and 118 Ma, with an event rate of $e_{124,118} = 0.1667$, that is, the state C_{124} is left in a proportion of one departure every 6 Ma.

Additionally, each event of time passage (events of automaton *Chronos*) triggers synchronous transitions on automata $M^{(ESL)}$, $M^{(S)}$, $M^{(RSL)}$ and $M^{(SS)}$. In order to favor clarity, the *M* automata have the transitions leaving the *DM* states labeled by groupings. For instance, label $m_0^{(ESL)}$ is used to refer to transitions in automaton *ESL* due to any of these events: $e_{118,113}$, $e_{91,90}$, $e_{51,48}$, $e_{17,15}$, *rst*.

Event $e_{130,127}$ is assigned to labels $m_3^{(ESL)}$, $m_0^{(S)}$, $m_1^{(RSL)}$, $m_0^{(SS)}$. Looking for these labels in the *M* automata one comprehends that the transition from C_{130} to C_{127} happens simultaneously to the transitions leading from *DM* to M_3 in automaton $M^{(ESL)}$, leading back to *DM* in automaton $M^{(S)}$, to M_1 in automaton $M^{(RSL)}$ and leading back to *DM* in automaton $M^{(SS)}$.

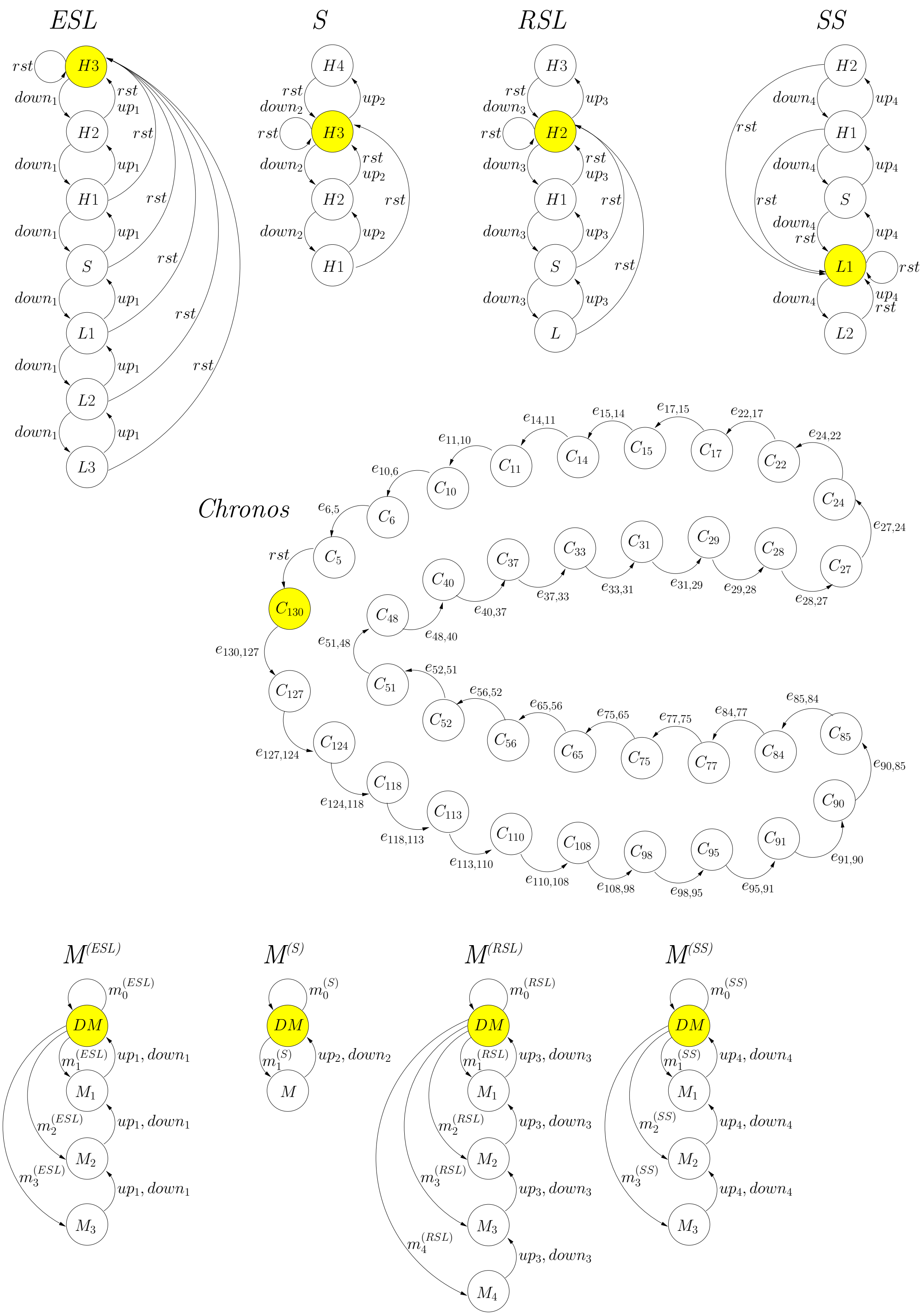


Figure 4.8: Pelotas Basin - Automata of the SAN Model.

In this new configuration, after synchronous transitions in *Chronos* and the M automata have taken place, only two kinds of events are possible: the *up* and *down* events. These events are also synchronizing ones, such that transitions in automaton $M^{(XYZ)}$ are accompanied by transitions in automaton *XYZ* (where *XYZ* refers to the automata *ESL*, *S*, *RSL* and *SS*), in the sense that the *up* events lead to higher rates in the *XYZ* automaton, while the *down* events force the rate to low down. Although both kinds of events are predicted in the model, it is expected that only one per automaton is enabled at a moment. The functions enabling and disabling the *up* and *down* events are the f_{up_x} and f_{down_x} functions (where x assumes an index in the set $x = \{1, 2, 3, 4\}$), which depend only on the *Chronos* current state. Some tests were executed in order to choose a rate for the *up* and *down* events and, as observed, setting these events rate to arround 1000 fits better to the results expected from Contreras *et al.* [42]. The transitions only cease when every *M* automaton reach state *DM*, disabling the *up* and *down* events and enabling once again the next event synchronizing a transition in *Chronos* with a transition in each of the *M* automata.

Event *rst*, a short for *reset*, is the only one in the whole model that synchronizes transitions between all the automata, restoring the model to its starting point configuration, back in 130 Ma.

4.2.3 Experiment and results

It is usual to conduct a modeling activity in order to obtain probability measures for some given reality. In this study, the measures of interest are obtained by means of *integration functions*, which are mathematical expressions for selecting the exact information to inspect.

Here, the integration functions represent the four types of deposit or configuration strata: *T*, *FR*, *LNR* and *HNR*, as discussed in Section 4.2.1. These types of deposit are strongly dependent on the relative sea level rate and the sediment supply rate, represented respectively by the automata *RSL* and *SS* (see Figure 4.8). Direct dependencies on the eustatic (global) sea level and subsidence, automata *ESL* and *S*, respectively, are subtle but yet taken into account, since they bring more realism to the model.

All possible configurations previously described are combined in order to build one function per configuration strata per *Chronos* state. As an example to be generalized to other states of *Chronos*, Table 4.1 presents the integration functions built for state C_{130}.

Considering the distribution of configuration strata along the past 130 Ma obtained by this modeling activity, the goal of this experiment is to validate the model by comparing the achieved results with those presented by Contreras *et al.* [42]. Furthermore, the way this model was developed, it is possible to predict the probability of each type of deposit within the same *Chronos* state.

Then, summarizing, there are basically four different types of deposit, as discussed in Section 4.2.1: forced regression (*FR*), lowstand normal regression (*LNR*), highstand normal regression (*HNR*) and transgression (*T*). Table 4.2 confronts information on these stratal patterns obtained by Contreras *et al.* [42] with the results achieved by the SAN model for Pelotas Basin (see Figure 4.8).

Observing Table 4.2 in detail[2], the first three columns respectively present the time period in millions of years, the relative sea level rate and the sediment supply rate. The fourth column presents Contreras *et al.* [42] estimates based on the interpretation of the sedimentary record. Under the

[2] This table abridges the results, the full data can be seen in [9]

Table 4.1: Integration functions for the Pelotas Basin model, considering state C_{130}.

$$
\begin{aligned}
T\_C_{130} \;=\; & ((st\ \mathsf{Chronos} == C_{130})\ \&\& \\
& (\ ((st\ \mathsf{RSL} == H_3)\ \ \&\&\ \ ((st\ \mathsf{SS} == L_1)\ \ \|\ \ (st\ \mathsf{SS} == L_2)))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == H_2)\ \ \&\&\ \ (st\ \mathsf{SS} == L_2))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == H_1)\ \ \&\&\ \ (st\ \mathsf{SS} == L_2)) \\
&));
\end{aligned}
$$

$$
\begin{aligned}
HNR\_C_{130} \;=\; & ((st\ \mathsf{Chronos} == C_{130})\ \&\&\ (f_{down_3})\ \&\& \\
& (\ ((st\ \mathsf{RSL} == H_1)\ \ \&\&\ \ (st\ \mathsf{SS} == H_2))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == S)\ \ \&\&\ \ (st\ \mathsf{SS} == H_2))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == S)\ \ \&\&\ \ (st\ \mathsf{SS} == H_1)) \\
&));
\end{aligned}
$$

$$
\begin{aligned}
LNR\_C_{130} \;=\; & ((st\ \mathsf{Chronos} == C_{130})\ \&\&\ (f_{up_3})\ \&\& \\
& (\ ((st\ \mathsf{RSL} == H_1)\ \ \&\&\ \ (st\ \mathsf{SS} == H_2))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == S)\ \ \&\&\ \ (st\ \mathsf{SS} == H_2))\ \ \| \\
& \quad ((st\ \mathsf{RSL} == S)\ \ \&\&\ \ (st\ \mathsf{SS} == H_1)) \\
&));
\end{aligned}
$$

$$
FR\_C_{130} \;=\; ((st\ \mathsf{Chronos} == C_{130})\ \&\&\ (st\ \mathsf{RSL} == L_1));
$$

"probabilities" grouping are the columns displaying the probabilities obtained by solving the SAN model for Pelotas Basin. In order to ease the analysis, the last column in this table provides a quick summary on this experiment expectations. Signal † indicates a mismatch between the estimate and the result obtained by solving the model with the specified integration functions; signal ✓ indicates a match. Bold indicates the highest probability derived by our model. The lack of marks is present where the phenomenon is unknown.

Concluding this chapter, we can notice that a compact structure is powerful enough to extent its use to natural phenomena, which is complex to analyze and usually generates long datasets. In this case, we successfully reproduced a phenomenon that is carefully analyzed and take years for earth scientists to extract and structure information about it. However, our model followed some classical steps; such as collect, analyze, clean and organize the data. Nonetheless, the dimensionality were carefully handled and the whole process took a considerable time to be finished. In the next Chapter (5), we enter in a collection of steps that reflects the steps of creating a model like this model and others based on virtually any indexed data.

Table 4.2: Expected vs achieved result, sample

time period (Ma)			relative sea level rate	sediment supply rate	Contreras *et al.* 2010	probabilities				match
						T	HNR	LNR	FR	
118	-	113	55	20000		0.0000	**0.7339**	0.2660	0.0001	
113	-	110	67	14500		**0.9996**	0.0000	0.0004	0.0000	
110	-	108	70	14500	T	**1.0000**	0.0000	0.0000	0.0000	✓
108	-	98	73	14500	T	**1.0000**	0.0000	0.0000	0.0000	✓
98	-	95	63	14500		0.0003	0.0002	0.0000	**0.9995**	
85	-	84	40	5000		**0.9995**	0.0000	0.0005	0.0000	
84	-	77	43	5000	T	**1.0000**	0.0000	0.0000	0.0000	✓
77	-	75	36	5000		0.0012	**0.9998**	0.0000	0.0000	
75	-	65	26	5000		0.0000	0.0000	0.0000	**1.0000**	
65	-	56	30	8500		0.0000	0.0000	**0.9997**	0.0003	
56	-	52	48	8500	FR	0.0000	0.0000	**1.0000**	0.0000	†
52	-	51	40	8500		0.0000	0.0004	0.0000	**0.9996**	
51	-	48	41	8500	T	**0.9995**	0.0000	0.0001	0.0003	✓
48	-	40	53	19000	T	**1.0000**	0.0000	0.0000	0.0000	✓
40	-	37	65	19000		**1.0000**	0.0000	0.0000	0.0000	
37	-	33	74	36000	HNR	0.0003	**0.9997**	0.0000	0.0000	✓
33	-	31	75	21000	LNR	**0.7993**	0.0000	0.2007	0.0000	†
31	-	29	63	21000		0.0005	**0.9995**	0.0000	0.0000	
29	-	28	72	21000		**0.9997**	0.0000	0.0021	0.0000	
28	-	27	49	21000	T	**0.8000**	0.2000	0.0000	0.0000	✓
27	-	24	73	25000		**0.6000**	0.0000	0.4000	0.0000	
24	-	22	81	25000	HNR	0.4001	**0.5999**	0.0000	0.0000	✓
22	-	17	79	17000		**0.9998**	0.0002	0.0000	0.0000	
17	-	15	75	17000		0.0005	0.0003	0.0000	**0.9992**	
15	-	14	70	22000	LNR	0.0000	0.0000	**0.9987**	0.0013	✓
14	-	11	61	22000		0.0000	0.0000	**1.0000**	0.0000	
11	-	10	75	22000	T	**0.9990**	0.0000	0.0010	0.0000	✓
10	-	6	74	41000		0.0003	**0.9997**	0.0000	0.0000	
6	-	5	90	41000		0.0000	**1.0000**	0.0000	0.0000	
5	-	0	82	41000	HNR	0.0000	0.0000	**1.0000**	0.0000	†

T	Transgression
FR	Forced Regression
LNR	Lowstand Normal Regression
HNR	Highstand Normal Regression

✓	Match between the model result and Contreras *et al.* 2010
†	Mismatch between the model result and Contreras *et al.* 2010

Chapter 5

A process to knowledge discovery through stochastic models and time series

Inside the stochastic modeling community, the benefits of statistical analysis through transient and steady states are clear. Unfortunately, create accurate big models is a non-trivial task, even for specialists. Thus, due to the difficulty and time needed to create such models, the advantage of stochastic modeling is restricted to a relatively small group.

As well as KDD is a process to achieve knowledge through data mining; here we proposed a process to describe, predict or forecast data through time series (TS) and stochastic models. To compose this process, is used a set of well-known techniques from the literature, and some specific new contributions described in this thesis. Although all techniques used were selected according to a careful research in the state of the art (*e.g.*. TS in Chapter 2), these techniques aim to serve as examples for the process phases and not to be emphasized as the best for each scenario. Indeed, there are many works claiming a better performance in different scenarios. Here, we focus on finding knowledge from data through stochastic models, the specific techniques are used as means to an end. As well as there are no specific technique for cleaning and integration, there are no specific technique to our data selection (Figure 5.1).

Regarding stochastic models, as described through the previous chapters; we focus on MCs, HMMs and SANs. Regarding SAN formalism, we successfully adapted techniques from MCs and HMMs, such as automatic model fitting (See Section 3.1). Regarding HMMs we proposed a new fitting algorithm, PAEM (Section 3.2).

Time series (TS) was chosen due to its advantages for representing and reduce systems dimensionality, which was described in (Chapter 2). Furthermore, they are widely used to represent and visualize observation regarding different modeling formalisms and techniques [127].

Aiming to establish a useful process to improve data manipulation steps, here is proposed a flow of activities, a process that attempts to set conditions to perform a union of techniques that allows us to retrieve probabilities from data without the need of a modeler specialist. Though the fusion of techniques from different domains, this process has a novel approach that allows us to use the best advantages provided by the Markovian models avoiding the time spent in the creation of accurate models. In a rough comparison, as well as KDD shows a set of steps commonly used to prepare data for data mining, here we describe a process that shows useful steps to prepare data for stochastic modeling. These steps aim to decrease the time spent to develop models, coping with state space problem and reducing the chance of human mistakes while manually developing complex models.

In a few words, stochastic models can be used to retrieve probabilities of occurrence of events. Thus, our general objective is to create a process robust enough to automate the creation and generation of stochastic models. Finally, the statistics generated by the model, like in data mining, can be used for descriptive or predictive tasks. Figure 5.1, published in [6], shows the general architecture of the model.

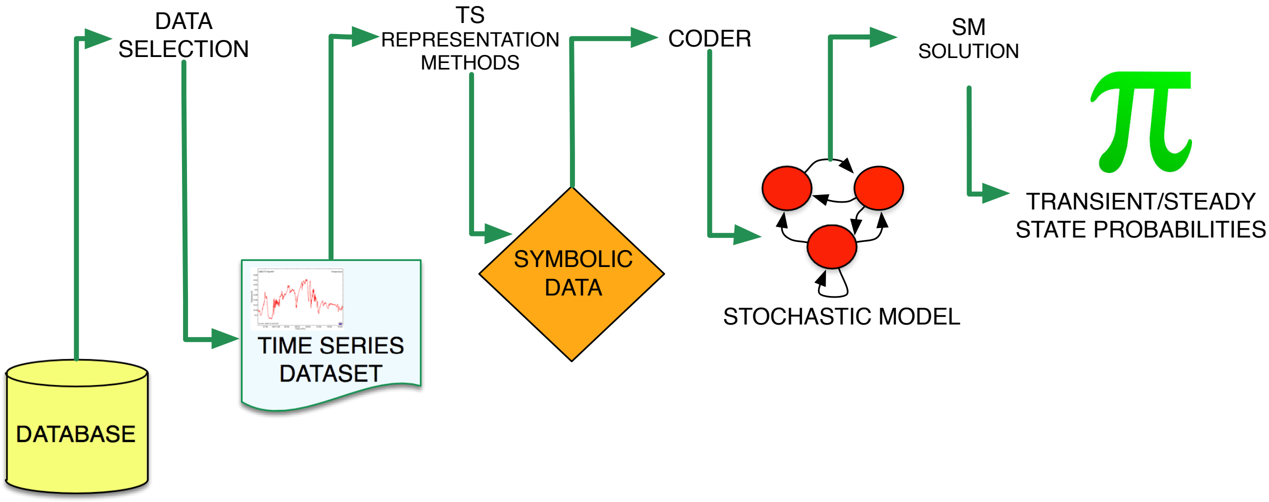


Figure 5.1: Proposed process

Figure 5.1 is composed by steps (or tasks) and data. At each phase, data goes as an input to a step and retrieves a transformed data (which can be a model or its results). We define phase as a step and its data, *e.g.*, the Data Selection phase is composed of the data selection step, its input data (from a database), and its output (TS). Therefore at the bottom, we have data and at the top, we have steps towards a desired set of results.

Through the Chapters 3 and 4 we presented some techniques to perform stochastic modeling via MC, HMM and SAN formalisms. Also, we presented some advantages of use such statistical techniques, as well as, some limitations relative to time effort and amount of data.

5.1 Data selection

Data selection phase is similar to the KDD phase proposed by Han *et al.* [69] (Figure 1.1). However, this data selection focuses on analyze relevant data that can be transformed into time series, which differs from the KDD's that aims to select relevant data and then transform it into a suitable format to mining algorithms.

Note that time series can be used for any kind of data that have a sequential order, which is different than temporal order in the sense that the temporal order can be established from non-temporal data. For instance, In Keogh *et al.* [81] shapes are extracted and transformed into time series. Therefore, bi-dimensional objects can be represented as TS, despite their lack of temporal variable. Thus our mathematical definition seeing at Chapter 2, actually is defined by an index i with N records $(i_1 < i_2 < ... < i_k < ... < i_N)$ optionally related to a temporal constant t.

$$S_1 = [(p_1, i_1), (p_2, i_2), ..., (p_k, i_k), ...(p_I, i_N)]$$

In this case, a dataset should be composed by a set of series $X = \{S_1,\ S_2\ ,.., S_N\}$. Considering a dataset X, each series S_j are used as different variables to a model.

In KDD, data selection comes with data transformation. When working with TS, this is relatively easy due to its simple and unique format. In fact, data transformation can be automatized by the use of specialized algorithms according to the kind of data [43, 106].

5.2 Time series representation

The use of TS for data representation and forecasting is a common practice, presented in many different works[1]. Concerning this phase, these techniques must be efficient to perform dimensionality reduction due to the stochastic models being generated through symbolic data assigned to the TS. Thus, this phase has two important roles in the process; reduce the data dimensionality and generate the sequence of symbols and meta-data to the coder. Despite the amount of sophisticated techniques to represent TS [33, 82, 90, 97], to illustrate this process we choose a simple, and efficient, dimensionality reduction technique, Piecewise Aggregate Approximation (PAA) [32]. As described in previous Sections (2.1 and 3.2), PAA is simple and can be competitive. However our version embedded a symbolic assignment, a simplification of Symbolic Aggregate ApproXimation (SAX) [90], which makes possible map a class to a state. Algorithm 4 describes our version[2].

Input: A vector of numeric data: vec
The number of symbols that will represent the entire dataset: $alphabetSize$
Level of segmentation that will generate the symbolicSize: $lseg$
Output: An array of $symbolicData$ with length = length(vec)/$lseg$

```
dim(vec) ← c(lseg,length(vec)/lseg);
myTS ← columnMeans(myVec);
cutSize ← ((max(myVec)) - (min(myVec)))/alphasize;
for i ← 0 to length(vec)-1 do
    switch myTS[i] do
        case <= cutSize + min(vec) :
            symbolicData[i] ← "A"; break;
        case <= cutSize * 2 + min(vec) :
            symbolicData[i] ← "B"; break;
        ...
        case <= cutSize * 23 + min(vec) :
            symbolicData[i] ← "Z"; break;
return (symbolicData);
```

Algorithm 4: Symbolic TS Representation

[1] Examples, [20, 36, 91, 98, 99, 102, 112, 120, 123, 125].
[2] The respective R code is shown in Appendix B.

This simpler version keeps desired characteristics of PAA and SAX. The creation of blocks with the same length facilitates indexing and mapping. The flexible dimensionality reduction through selective blocks size. The fast mapping and a flexible number of symbols. An example is given in Figure D.1, which is a prototype of PAEM, an update to the traditional Baum-welch with a deterministic pre-fitting (Section 3.2).

In Algorithm 4, in the first line are used two functions, $dim()$ to return the dimensions of a vector or matrix, and c to combine values into a vector or matrix. Thus, in the first line the function $c(lseg, length(vec)/lseg)$ creates a vector with two values that are assigned to the dimension of the vector retrieved by $dim(vec)$. Being vec now a matrix, at line 2 we create a new vector to hold the mean of the values sorted by his columns. In line 3 we create the cut size according to the maximum number of symbols allowed to our symbolic data ($alphasize$). Finally, in the line 4 we start a loop to verify the value of the data and assign to his correspondent symbol. To the ease of understanding, we use the alphabet sequence.

5.3 Coder

In Big Data, coding is the process of tagging terms using an identifier that is assigned to every synonymous term in the data [17]. This phase has a similar objective; however, instead of tagging terms, a symbolic data is tagged into states of a model. Once the mapping is performed, the model is measured in order to obtain the accuracy of its representation according to the original observations. The next step is to improve the model itself through the improvement of its parameters; a fitting process. Thus, this step (Coder) is the core of the fitting process, through symbolic data it generates an adjusted model. In a few words, an "automated mapping and fitting process".

A symbolic data is a collection of N symbols that represent N levels of a given time series; thus, represented by: $S = [a, b, c, d, ...\Omega]$ the symbolic data generated (Y) can be formalized as: $Y = \{y_1, y_2, ..., y_T\}$, being $y_t = \{s \in S\}$. Once we have symbolic data, this step is responsible for generating stochastic models. However, the generation of stochastic models can be far from a straightforward task and it differs from formalism to formalism. We already presented an example for this step, PAEM (Section 3.2) illustrates the process of mapping data into states.

However, PAEM relies on the basic limitations of EM and HMMs, which is a fixed, two layered, structure. Now, we re-take the mapping and fitting basics by showing an example with Discrete Time Markov Chains (DTMC). Then, we go further to the main contribution, a coder for SAN. Note that essentially, HMMs, as well as SANs, can be represented as MCs; yet as shown in previous Chapters (3 and 4) their structures are different; thus, the challenge lies in generating their structures.

Example with Markov Chains

In this approach, the Coder reads all vector X and extract the number of different symbols present in it; *e.g.*, $X = [c, s, r, c, u, p, s, r, r, c, c, p, c]$ Coder will generate a five states DTMC that will have a structure showed in Figure 5.2. The same structure can be represented as a probability matrix. The algorithm 5 is a simple example to generate the matrix. It is based on the MLE approach, previously described in Section 3.2.

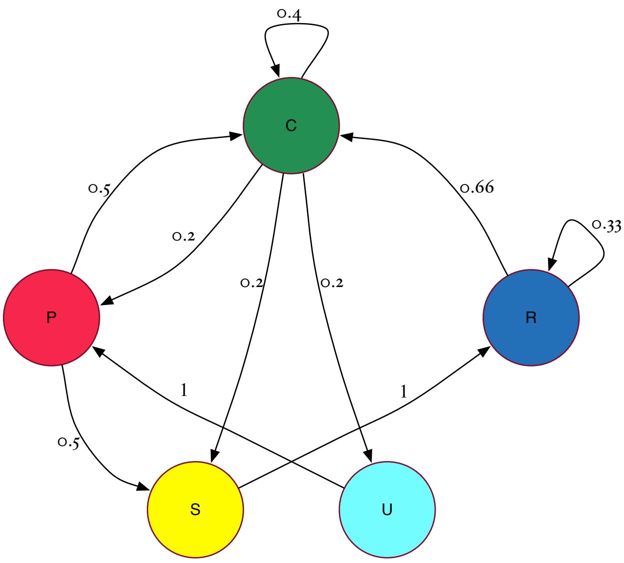


Figure 5.2: Example of MC created by CODER using symbolic data

$$\Gamma = \begin{array}{c|c} & \begin{matrix} C & P & R & S & U \end{matrix} \\ \hline \begin{matrix} C \\ P \\ R \\ S \\ U \end{matrix} & \begin{bmatrix} 0.4 & 0.2 & 0 & 0.2 & 0.2 \\ 0.5 & 0 & 0 & 0.5 & 0 \\ 0.667 & 0 & 0.33 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Input: A vector of symbolic data: $symbolicData$
Output: A Matrix of probabilities: $probMatrix$

```
1  nOcur ← countOccurrences();
2  createSquareMatrix(occurMatrix, nOcur);
3  symbolsToValues(sData, symbolicData);
4  for i ← 0 to length(symbolicData)-1 do
5      line ← sData[i];
6      for k ← 0 to nOcur-2 do
7          if occurMatrix[line,k] == sData[i+1] then
8              occurMatrix[line,k] += 1; break;
9  getPercentage(probMatrix, occurMatrix);
10 return(probMatrix);
```

Algorithm 5: Symbolic data to probability matrix

In the first line we get the number of different characters in the data, thus we achieve the number of states, in this case, the matrix size. At line 2 a squared matrix $occurMatrix$ are created with the size of $nOcur$. In line 3 are assigned values to the characters and they are stored into the vector $sData$. Thus, from line 4 to 8 we count for each term the number of transitions to every other term. *e.g.*, How many times a changes to b. Finally, at line 9 is created a transition probability matrix $probMatrix$ (or Γ as seeing in Section 1.2.1) based in the $occurMatrix$.

The solutions are well described in Section 1.2.1. Example, knowing the sequence X begins with an element c, *i.e.*, $x_1 = \text{“}c\text{”}$, we can use the Chapman-Kolmogorov equation (Eq. 1.2) to retrieve simulation probabilities. Thus, generating a new set of observations Y based on their probabilities or predicting the next states of X we shall use Γ to it. Example, given a random position in X we get the element c, we shall obviously guess c for the state with a probability of 40%. For two consecutive elements, c receives 26% for a second position, then 37.7%, ..., until 38.46% considering the steady state[3].

This simple approach guarantees that the generated model will have the maximum likelihood, which means these parameters are the best ones to represent X. Of course, it depends on the nature of the data; characteristics such as a temporal constraint can imply that some transitions are more important than others. These characteristics make the universal measurement, likelihood, fails in certain cases. We discuss this in the Section 5.5.

5.4 Stochastic model solution

This phase is responsible for generation of the model results. It is analogous to the patters generated by the data mining algorithms (on the KDD process). Thus, the set of transient (or steady) states generated is responsible for the information which gives us knowledge about the system. Formally, the evaluation and presentation step remains present, just hidden in the model.

Regarding the generated model, this step defines what will be executed and how; *e.g.*, if the coder generates a SAN model, a SAN model should be executed. Even using SANGE, there are basic parameters that are required to be configured according to the modeler's question; *i.e.*, questions concerning the system, whose models are intended to return probabilistic knowledge about it. Furthermore, there is more than one approach to consider when executing a model. *e.g.*, if we have a DTMC, we can consider the transient states as well as the stationary states. SANs are a continuous time representation. However, it can always be equivalent or generate an equivalent DTMC [16].

Section 1.2.1 gives an overview of some useful properties of the Markov chains. Section 6.1 shows some previous works and two different applications using this process. These applications are important to evaluate this process. Furthermore, they make more clear the advantages of this approach.

5.5 Comparing model simulations via Dynamic Time Warping

As described in the Section 3.1.2, models are usually derived based on the likelihood. However, a stochastic process generates stochastic outputs and even with a high fidelity model, its outcomes are quite different from each other. One may say this is exactly the goal of a simulation, this is

[3]This example can be easily reproduced using the tool described in Appendix C.

expected because a desired output is based on a set of probabilities concerning a state or a group of states. However, in some cases, we would like to analyze and choose, among the simulations, which one has more similarity with some known behavior. In other words, the stochastic behavior is desired, yet some phenomena are known and not part of the model; thus, the best simulation should obey it. *E.g.*, in market stocks, we know that some transaction can lead to a fast increase in some stock prices. However, this information is not part of the model nor the original observations; therefore, a model cannot guarantee to always generate the right output.

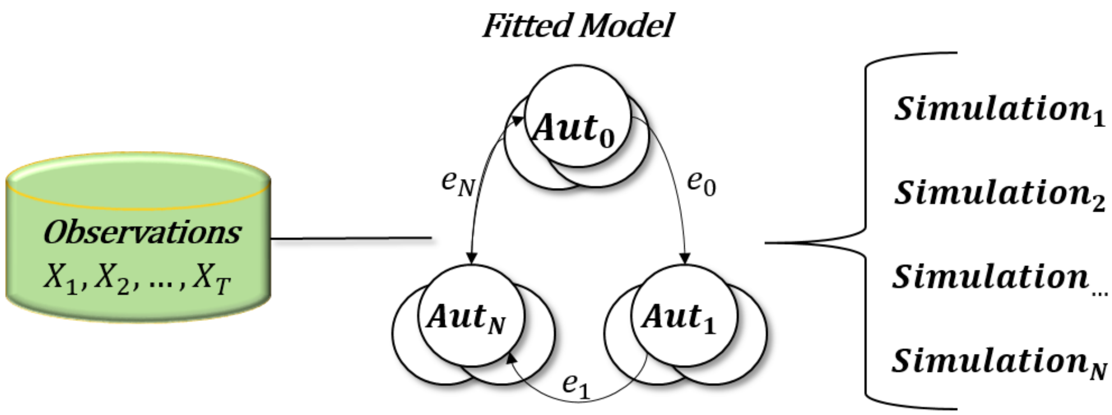


Figure 5.3: A model can derive N different outputs based on stochasticity. Which one to choose?

Another example is a climatic model. Since, at our time, it is impossible to perfectly model all climatic variables; by common knowledge and data we can choose the most relevant simulations for the local climate. Imagine a simulation of a hurricane's trajectories that leads to different paths. If equally probable trajectories lead to a different location, which one should be considered most? One might take decisions based on external factors, such as the available money to reconstruct cities or the strength of the hurricane. Measuring external factors with the simulations will result in knowledge to choose which simulation are more relevant. If the hurricane is too strong, does not matter the equally probable trajectory (to the ocean or to the continent), the simulation that says "to the continent" will be the most important.

In such cases, we can use time series measurements, such as Dynamic Time Warping (DTW), as a way to analyze the simulations in relation with the original observations. As described in Section 2.2.2, the DTW alignment allows us to identify key points in our data with time adjustment. This feature can be useful to enforce our decision power by a more detailed analysis.

An example of this combination is given by the speech recognition community, where DTW and HMMs are often used, yet rarely likelihood and DTW are used together. Liu *et al.* [94] illustrates a possible good combination. They use the likelihood as an initial confidence measurement to the model. After, DTW is applied to get the distance from the template.

For the sake of clarity, let consider an example with a two states Markov chain with equiprobable transitions (0.5 each), e.g., coin toss, Heads (H) and Tails (T). This Markov chain can be equally likely to drawn a sequence TS_1=*T,H,H,T,H,T* or TS_2=*H,T,T,H,T,H* or TS_3=*H,H,T,T,H,T*. Even having the best possible fitting, a model is equally likely to draw any of these three series that has a different behavior.

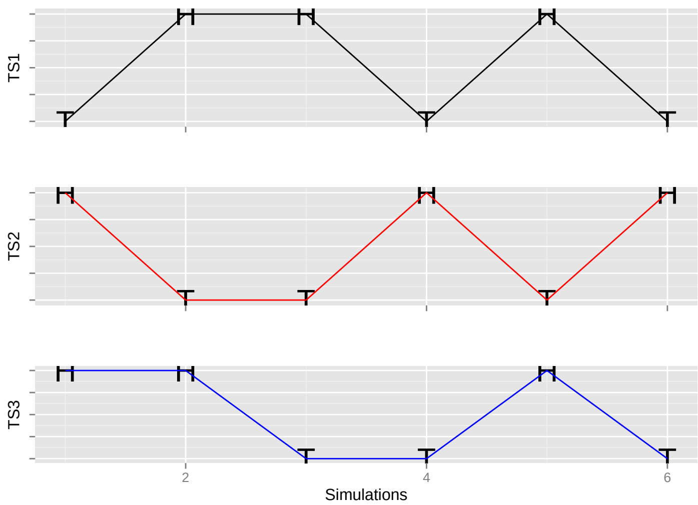


Figure 5.4: Three different simulations derived by the same model

Using time series distance measures we can verify that TS_1 is closer than TS_3 than TS_2. In this example, the DTW alignment between TS_1 and TS_3 is equal to 1, $DTW(TS_1, TS_3) = 1$ (Figure 5.6) and $DTW(TS_1, TS_2) = 3$ (Figure 5.5). Although, a simple Euclidean distance can be used in this example the time alignment is useful when simulating events in time.

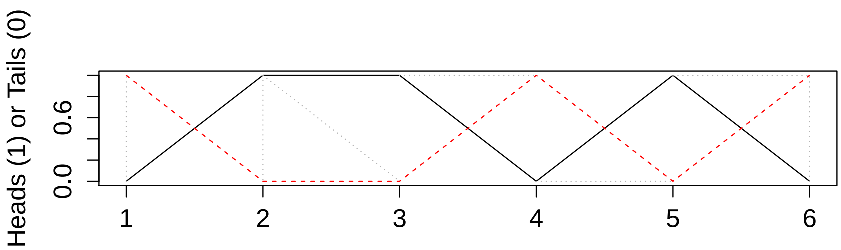


Figure 5.5: DTW alignment between TS_1 and TS_2

It is noticeable that a MC fitted from TS_1 would not result in an equally probable matrix. So, let us suppose that TS_1 is the original data, a transition probability matrix (Γ), fitted by MLE, is described as: $\Gamma^{(0)} = \begin{bmatrix} 0.333 & 0.667 \\ 1 & 0 \end{bmatrix}$. Which would generate different outputs, yet with a disproportional set of states, being only 37% of the observations *tails*. Even though, DTW could be applied in the same way.

Example, weather simulation.

Figure 5.7 shows 200 consecutive days[4] recording the maximum temperature for the city of Porto

[4]Starting from January, 2000; public available at $https://github.com/joaquimAssuncao/DS$

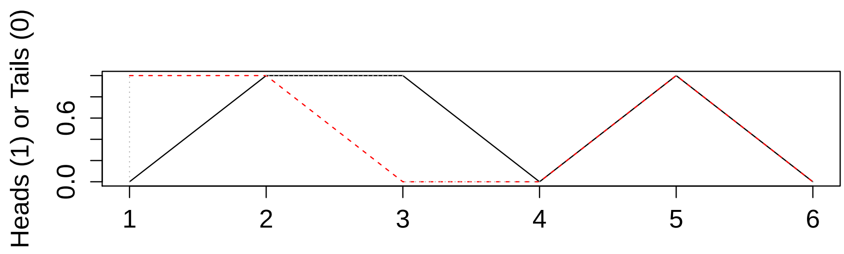


Figure 5.6: DTW alignment between TS_1 and TS_3

Alegre, Brazil. The red line represents the original observations and the green and blue line is two simulations generated from the same model.

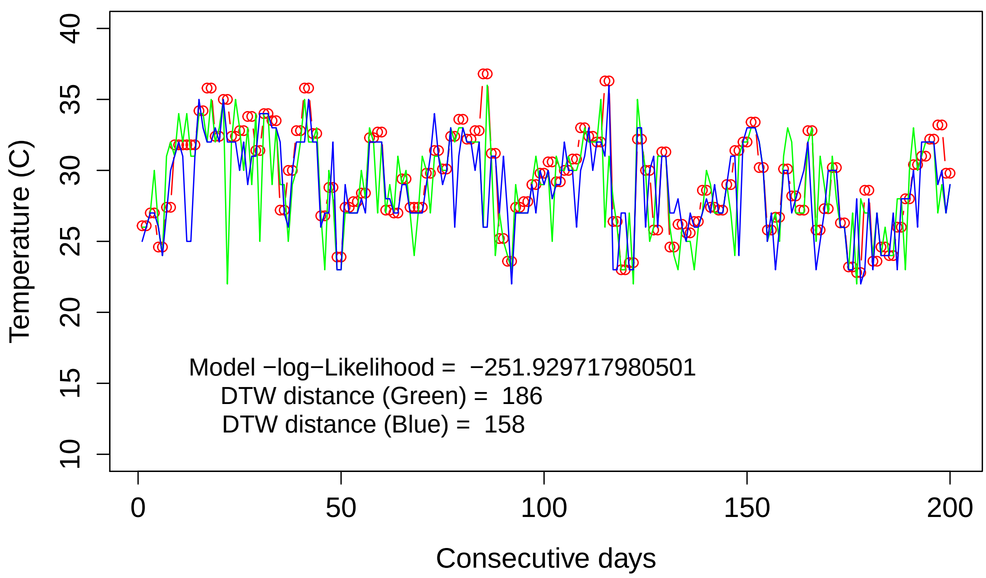


Figure 5.7: Original data (red) and two simulations from the same model; same likelihood, different DTW alignments

Though the likelihood is the same, one simulation is obviously better than the other. Through DTW we can see the best simulation line. Which brings us one more layer to choose an adequate simulation. Here, we can see that the blue simulation has a shorter distance and therefore is closer to the original data. This could be used to fill gaps or to forecast the next steps. Furthermore, we could consider specific points in time to measure these distances.

In our hurricane example, there would be more than one series based on external factors, these would be added to the simulations as an extra information. For instance, two normalized series could be compared to a third series containing a damage scale. This third time series would change the distance measurement favoring a simulation according to the risk.

Finally, this is a proposal to select simulations from an already fitted model, it aims to be a complementary step that stands after the process's last step (Figure 5.1). Although it can be useful to choose between simulation, its applications are reserved as a future work.

Chapter 6

Fitting applications

In the Section 4.2 we described a SAN model which can be used to simulate complexes geological phenomena. According to the available data in the literature, our model revealed itself to be precise to perform simulations. However, we spent weeks to adjust the model parameters. Regarding modeling time and precision, we now address questions from Section 1.3, such as: "Can we effectively automate some steps, making the time saved overcompensate the loss of precision?". Using symbolic representation (Sections 2.1 and 3.2) and fitting techniques (Section 1.2.2), we create a solution for Stochastic Automata Networks; which by itself is a solution to represent big systems (Chapter 4).

Regarding coding strings to a SAN model; Braghetto *et al.* [21] did a similar work transforming BPMN (Business Process Model and Notation) graphs into SAN models aiming to perform a quantitative evaluation of business models. Our conversion from strings to SAN uses a similar principle. Thus, it is possible to map multiple TS into SAN models, and this feature can be especially useful when multiple a system are represented by multiple TS.

Following the process described in the previous chapter, we show novel techniques and tools created especially to this purpose. First, an unsupervised SAN generation for geological events [8], Section 6.2. Which handles large periods of time in a pre-defined structure to generate probabilities regarding geological phenomena in certain periods of time (background explained in Section 4.2). This tool was our first attempt to automatically generate SAN models, considering such complex scenario.

Second, a specialist tool to generate SAN models from data sets [7] [10], which is the first adaptation of the Model Likelihood Expectation measurement applied for SAN models, and the first specialized tool to create SAN models directly from series.

Third, Section 6.4 shows an application using corpora as training and test. External tools are used to classify and train the models, yet the basic steps of the process remains the same. Before these sections, we start with a simple example showing how these approaches can be useful to general numerical data.

6.1 Applications for general numeric data

Since we can transform numerical data into TS and then SMs, almost every numeric data is suitable for this process. In fact, as long as we can represent it in a TS, we can create a model of it. A representation that can be done in many different scenarios (Chapter 2).

Furthermore, being useful to represent the behavior of a system; SMs can be applied in many scenarios. Section 6.3 described two examples, "Weather in Gothan city" and "Fathers and Sons".

Here are suggested more situations in which the process steps can be used to perform inferences through stochastic models.

One possible application is for productions; *e.g.,* forecast the production of soybean bags for the next year. This can be useful because using SMs we can achieve more accurate probabilities by using powerful methods that allow us to perform quantitative analysis on the generated data.

For most of the "real world datasets", using the raw data we face the "*state space explosion*" problem; also, these models demands time and a modeler specialist. Connecting the best techniques in the literature we were able to automate the steps that usually demands time and effort from specialists. Thus, turning possible to achieve probabilities from a dataset through SMs.

A set of experiments is shown at Table 6.1. The datasets used in these experiments are all public available retrieved from [80] and Yahoo market stock [1]. We can observe that the reduction of data size in many cases exceeds 90%. Typically such compaction should generate a significant loss of precision, but in this case, such compaction is suitable due to the order and similarity of TS data.

Also, In this table is indicated the probabilities of the more frequent classes, and a general indication if the experiment prediction was accurate ("✓" indicates an experiment with correct predictions, "†" indicates otherwise).

Table 6.1: Probabilities and classes found for each dataset ("✓" stands for accurate, and "†" inaccurate).

Dataset	**Length**	**Symbolic length**	**Predicted probabilities**	**Accuracy**
Coffee	8190	819	j: 94.28%	✓
Symbols	397000	3970	f: 12.51%; g: 11.94%	✓
WordsSynonyms	172800	1728	i: 29.51%; j: 29.01%	✓
ItalyPowerDemand	25700	1285	i: 40.21%; j: 42.54%	✓
petr3	3452	863	b: 18.68 %	✓
bbas3	3440	860	b: 45.37%	✓
brfs3	3410	341	b: 55.80%	✓
vale3	3460	865	a: 37.30%	†

6.2 Unsupervised Model Generation for Geological Events

Generates large stochastic models require considerable amounts of time to be created. Due to the need to perform analysis of the system and its variables, the model described at Section 4.2 [9] took us many weeks to be accurately fitted.

This *Coder* is an application that reduces the time and effort to create models with similar structure. It uses the background and the structure described in the Section 4.2 to perform the creation of new models for the same kind of data. The trick in this approach is to keep the model structure

[1] http://br.financas.yahoo.com/indices?e=bovespa

by fixing the number of input parameters. Any model generated will hold the same structure, *i.e.*, composed by the same number of automata that always have the same representation [8].

This model has seven automata. One to control the time passage (called Ch as in *Chronos*), *i.e.*, each Ch state represents a time slot pre-defined according to the input data. Three for the geological events called: E (for eustasy), Su for subsidence, and SS for sedimentary supply (Detail in Section 4.2). Also, for each geological event, there is a memory automaton to control the number of changes allowed in each time slot. Memories automata, respectively called M_E, M_{Su} and M_{SS}, are created using parameters collected from the same data that are used to create the others automata.

Although the structure remains the same, due to the input parameters, the generated model tends to have a limited number of reachable states; nevertheless, still there is a need to handle the problem of space state explosion. Therefore, we limited Ch to 36 states and each of the other six automata to nine states each. In consequence, the larger model we can handle has 19,131,876 states. Figure 6.1 is a generic example of the produced models.

Each memory automaton controls its counterpart automaton. It assures that the change of states will not pass more than one state at time and it indicates the number of steps that should be taken at the current Ch state. This process continues until every memory reach the DM (do not move) state. When a memory automaton is in DM it stops its counterpart geological event automaton.

When all memory automata reache DM state, the Ch automaton can change his own state to the next time slot. For example, Fig. 6.1 illustrates this process by using a memory automaton M with three states (DM, M_1 and M_2), a Ch automaton with four states (T_0, T_1, T_2 and T_3) and one geological event automaton G with three states (H, A and L).

The example in Figure 6.1 depicts the firing of four events affecting the automata G, M and Ch. In this figure we start with the initial state in the left hand side, *i.e.*, the global state (H, DM, T_0), then the synchronizing event C_1 advances to the first time slot (T_1 in Ch) and at the same time changes M automaton to state M_2 meaning that G automaton will have to change, in the future, to two states below, *i.e.*, going from the current H state to L.

The second event to be fired is Dn, which begins to perform the change needed according to automaton M. After, Dn is fired again leading to the fourth global state and making C_2 event able to fire and change to the second time slot (T_2 in Ch). A similar sequence of events continues until reaching the last time slot (T_3) when event rst (*reset*) brings back the system to the initial global state.

Figure 6.1 example is a simplification considering just one geological event, but our actual model has three geological event represented (E, Su and SS). This brings more complexity to Ch events, yet the basic firing sequences remain similar.

The proposed automatic generation consists in receiving three curves describing the evolution of Eustatic sea level; Subsidence; and Sedimentary supply during a time period and then, compute each automaton state. This is done in three major tasks: (i) choosing granularity for values in order to determine automata states; (ii) defining the synchronizing events binding the Ch automaton to

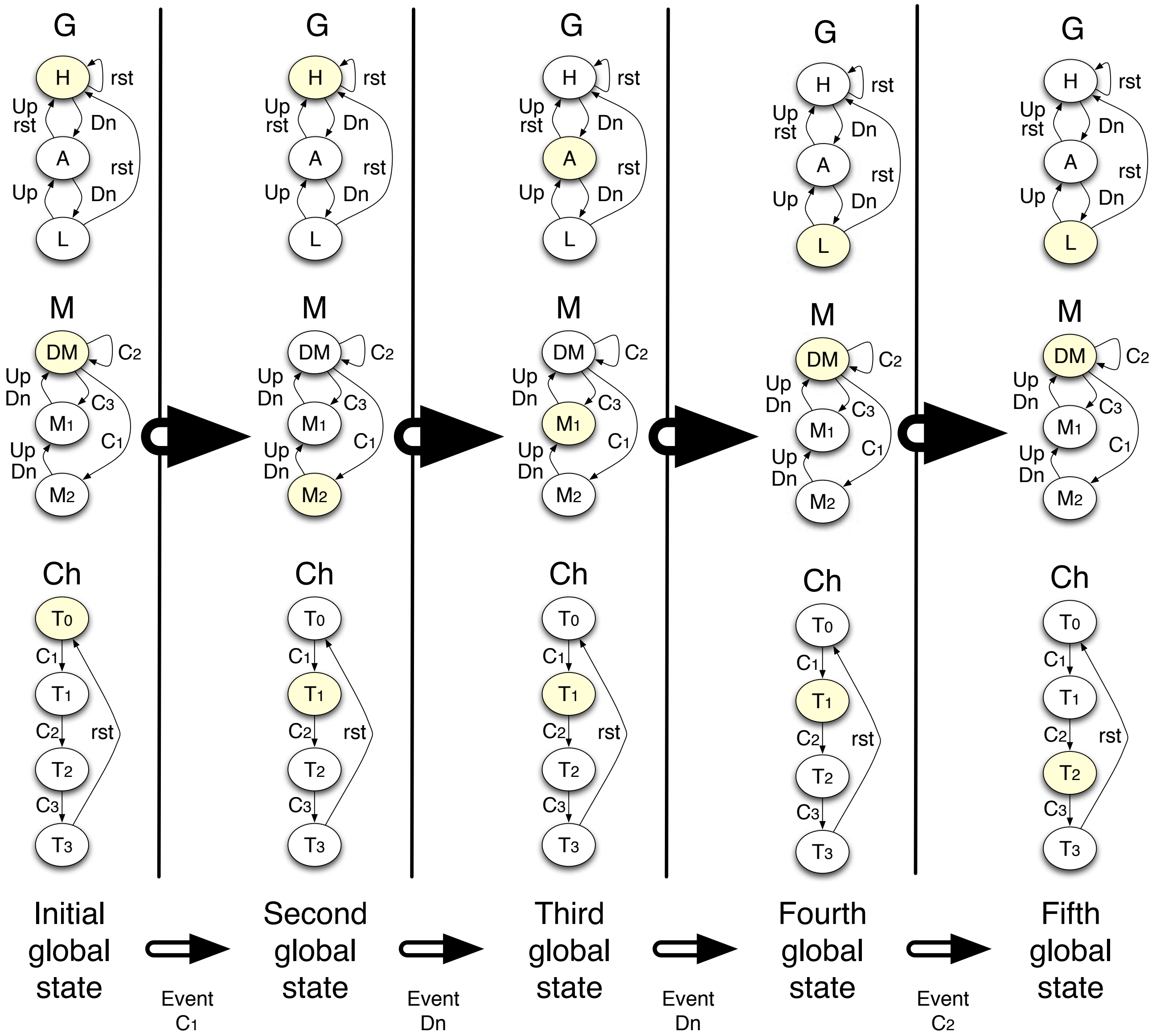


Figure 6.1: Unsupervised model generation for geological events - execution example.

the memory automata (M_i); and (iii) defining the synchronizing events and functional rates to bind each memory automata to its counterpart geological event automaton.

We start by defining a value granularity to each input curves (Figure 5.1, TS representation), *i.e.*, to reduce the dimensionality of each input curve to a set of discrete states (see Section 5.2). This defines the states of automata E, Su and SS, but the alignment of these discretized curves in time slots will also define the granularity of time, *i.e.*, the states of automata Ch. In fact, once the three curves are aligned, every time at least one of them change values a new time slot will be defined. Finally, this task ends by the observation of the number of states that each geological event automata jumps every time the slot changes. These values will define the number of states needed by each memory automata.

Even thou this first task (TS representation) is quite straight forward, the granularity decisions made here have the major effect both in the model accuracy and the SAN model state space.

Therefore, the choice of granularity brings a classical trade-off decision between model size and quality. A question addressed in Section 1.3.

The second task has less decisions to take, but it has much more relations to establish, since it must associate as many synchronizing events as necessary to ensure that each change in Chronos automata (Ch) performs the correct change in the memory automata (M_E, M_{Su} and M_{SS}). This task demands the creation of C_i events (as in Figure 6.1), but the existence of three geological events demands that each C_i event must synchronize the four concerned automata: Ch, M_E, M_{Su} and M_{SS}.

The last task is similar to the second one (they correspond to the Coder step), being simpler by the fact that each memory automaton synchronizes to only one geological event automaton. However, it is more complicated by the fact that each event must take the current time slot into account to memorize if the geological automaton must go up or down. *i.e.*, it requires functional rates according to the state of Ch.

This SAN model generator is able to create models for virtually any geological basin in any period of time. However, its specific implementation do not allows other types of data. In the next section we show an algorithm and its prototype that are able to produce a wider set of SAN models.

6.3 SAN Generator (SANGE)

A simple bottom-up, MC-based, approach gives a solution for any generic model. However a plain (unstructured) MC is a limited, memory expensive model. Limited in the sense of having a bulk representation for a system, regardless of how complex it may be. Memory expensive because a system with S states is represented by a transition probability matrix of S^2. In the best case, the number of non-zero entries will be in the order of S.

SAN formalism is modular and its model representation is composed by a collection of sub-systems, which are usually much more compact benefiting from tensor representations [24] (See Section 4.1). like MC or HMM fitting algorithms, SANGE is a tool that implements a forward approach that is used to fit SAN models directly to data, reducing the model size.

The excessive human effort to construct models can be avoided with a tool that handles the formal tasks to convert data into state transitions. Consequently, the user can focus on more interesting tasks, such as interpreting the results and applying the gain knowledge. Additionally, SANGE performs dimensionality reduction using time series representation methods [90]. Thus, SANGE is also capable of automatically fitting the model to input data, possibly achieving better models than those made by humans.

SANGE's main objective is to reduce the time spent on generating SAN models, thus, opening the use of SAN models by non-specialists. However, our solution needs records of a system behavior in the form of time series that will be assigned to the variables of interest for the system.

SANGE's algorithm was inspired by a well known and broadly used formalism, HMM and its fitting algorithm (BW) Section 3.1.3. Thus, as well as an HMM user only needs to understand the modeling basics and how to interpret the generated HMM model, a SAN user shall find it easy to use to.

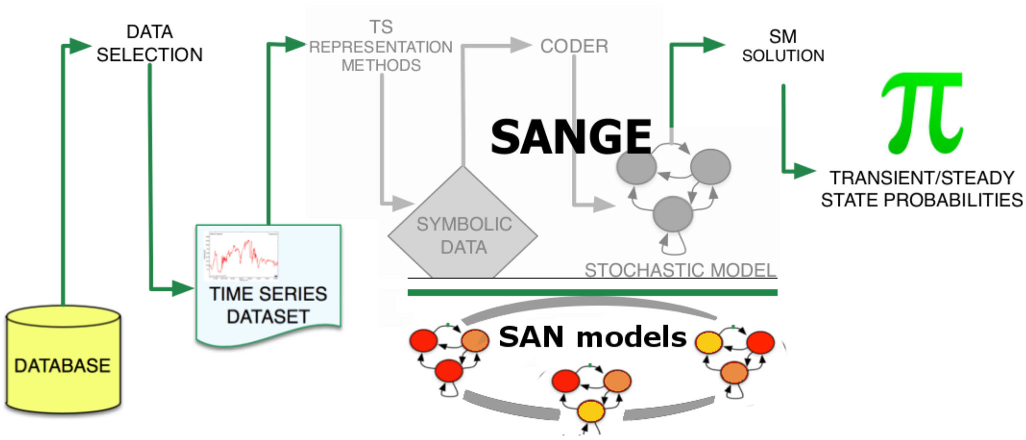


Figure 6.2: A Dimensionality Reduction Process to Forecast Events Though Stochastic Models [6] with SANGE.

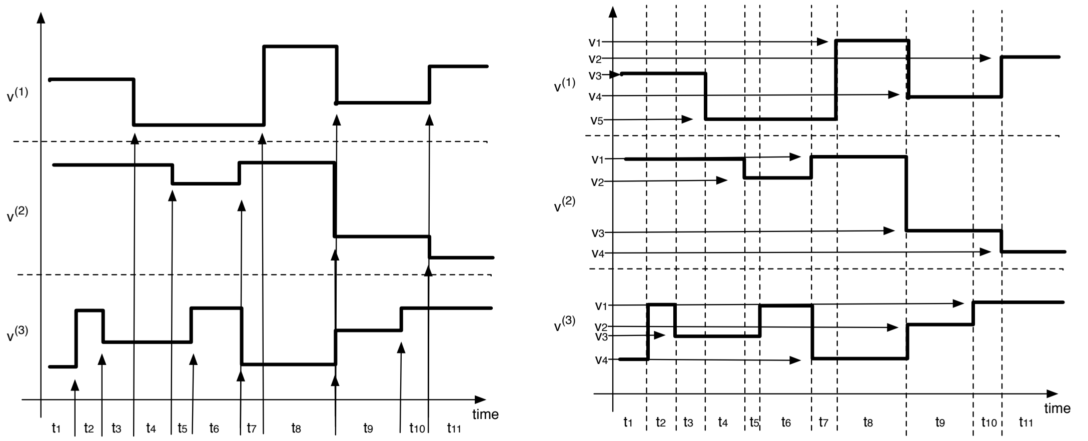
SANGE's basic operation consists in the composition of a set of time series describing how the system variables behaves [90]. Considering a system with n interest variables ($v^{(i)}$ with $i=1,\ldots,n$,) we need a behavior sample of the system in the form of n time series with the successive values of the variables through time. With these time series, the basic process is to identify the points of interest in time, as the time ticks where at least one of the variables change its value. Once the time ticks of interest are identified, it is necessary to identify the possible values for the variables in order to define the stochastic model. This process is summarized in an example with three time series in Figure 6.3 showing the identification of 11 time ticks (a) and the three succession of values for variables $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ as:

$$\begin{array}{llllllllllllll}
v^{(1)} \rightarrow & v_3^{(1)} & v_3^{(1)} & v_3^{(1)} & v_5^{(1)} & v_5^{(1)} & v_5^{(1)} & v_5^{(1)} & v_1^{(1)} & v_4^{(1)} & v_4^{(1)} & v_2^{(1)} & v_3^{(1)} \\
v^{(2)} \rightarrow & v_1^{(2)} & v_1^{(2)} & v_1^{(2)} & v_1^{(2)} & v_2^{(2)} & v_2^{(2)} & v_1^{(2)} & v_1^{(2)} & v_3^{(2)} & v_3^{(2)} & v_4^{(2)} & v_1^{(2)} \\
v^{(3)} \rightarrow & v_4^{(3)} & v_1^{(3)} & v_3^{(3)} & v_3^{(3)} & v_3^{(3)} & v_1^{(3)} & v_4^{(3)} & v_4^{(3)} & v_2^{(3)} & v_1^{(3)} & v_1^{(3)} & v_4^{(3)}
\end{array}$$

Figure 6.3 example would result in three automata to represent as local states each variable possible value, and transitions representing each state change through events. Figure 6.4 presents the SAN model for this example.

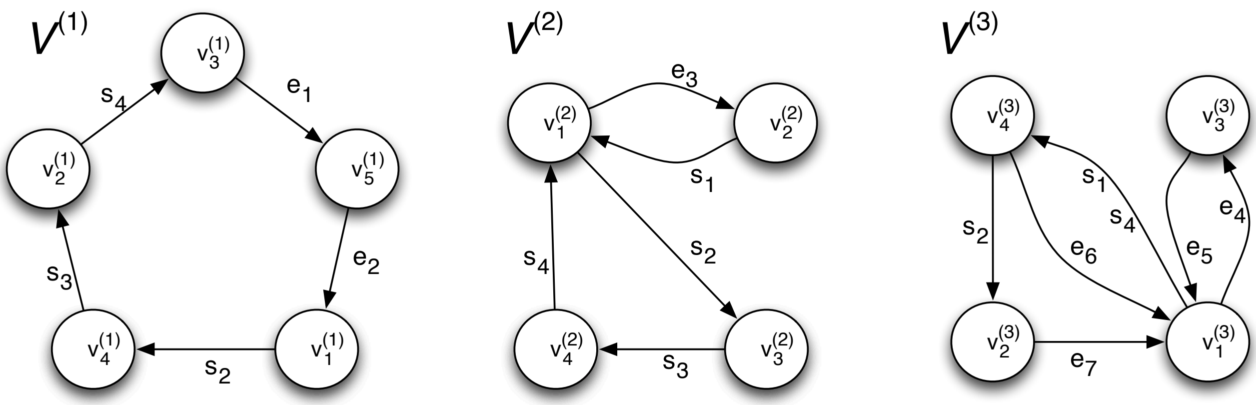
To compute the rates of each local event we must take into consideration how many transitions are possible from each origin local state. For example, event e_1 goes from state $v_3^{(1)}$ to state $v_5^{(1)}$, and observing all time ticks we see state $v_3^{(1)}$ as origin at the end of time ticks t_1, t_2 and t_3. In time ticks t_1 and t_2 automaton $V^{(1)}$ does not change state, and in t_3 it goes from $v_3^{(1)}$ to $v_5^{(1)}$. Therefore, 1 in 3 times event occurs, and rate of event e_1 is 1/3.

Considering synchronizing events, the computation of event rate is similar, but since more than one automaton is concerned, is necessary to take into account the number of times the origin combination of states occurs. For example, event s_3 concerns automata $V^{(1)}$ and $V^{(2)}$ that start



(a) identifying time ticks of interest (b) identifying variable values succession

Figure 6.3: Example of SANGE basic process to three time series.



type	event	rate	type	event	rate	type	event	rate
loc	e_1	0.33333333	loc	e_5	0.33333333	syn	s_1	1.00000000
loc	e_2	0.25000000	loc	e_6	0.33333333	syn	s_2	1.00000000
loc	e_3	0.16666667	loc	e_7	1.00000000	syn	s_3	0.50000000
loc	e_4	0.25000000				syn	s_4	1.00000000

Figure 6.4: Equivalent SAN model for the three time series example of Fig. 6.3.

from combination of states $v_4^{(1)}$ and $v_3^{(2)}$, a combination that occurs at the end of time ticks t_9 and t_{10}, but event s_3 only happens in one of these two situations (after t_{10}), hence, event s_3 rate is $1/2$.

From a practical point of view the current version of SANGE can generates either a SAN model or a Markov chain output (which is basically a SAN model with a single automaton and only local events), both can be related and verified through MLE. The output is in the format of a `.san` file, that can be directly fed into a SAN solver, such as PEPS [23].

Obviously, the presented example has quite small time series and it limits considerably the validity of the generated model. In real cases, much larger time series must be considered in order to see patterns that will be very important to bring statistical relevance to consider larger number of states succession representative of an event. Naturally, in the stochastic related works, large amounts of samples are crucial.

In our example, we already started with the time series dully coded in symbolic values (v_1, v_2, *etc.*), but several data can be obtained with continuous values, or even highly dimensional categories. Even thou such symbolic coding may affect the quality of the produced model, there are quite efficient symbolic coders available (see Chapter 2).

Example: Weather in Gotham city

To facilitate the understanding, we use a classical example for Markov models, *i.e.*, forecasting weather events [110]. The most basic example is a Markov chain with 3 states, representing **R**aining, **S**unny and **C**loudy (Figure 6.5). Probabilities are assigned to the transitions between states as well as staying in the same state.

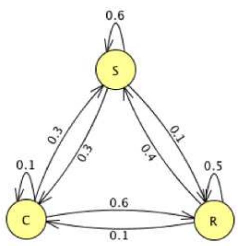


Figure 6.5: Classical example Markov chain model.

By solving this model we can achieve the transient and stationary probabilities of being in one of the three states. As the only representation of such a model is by a probability matrix, It can be computationally hard to handle for real world applications with many states. Furthermore, the best known formalisms, Markov chains and HMMs, can not integrate models with multiple automata in a unique system, *i.e.*they do not have a structure for this (sections 1.2.1 and 3.1). By using SAN with SANGE it is possible to generate a structured version, which allows us to assemble more elements to our model and solve those as one.

SANGE encapsulates the techniques described in the Section 6.3; thus, it provides an interface for this basic and more advanced statistical tools. As pointed out before, the algorithm merges the SAN and TS characteristics. The following example does not consider real data nor the adequacy of the model; the goal here, is to illustrate how SANGE works with multiple variables and how easy is to create a model with it. Consider the following model:

Example, Figure 6.6 shows a generic model with 2 automata, one for weather conditions and the other one for wind force. The states' labels in the first automaton are **R**aining, **S**unny and **C**loudy. In the second automaton, the states correspond to the wind velocity, **F**ast, **M**edium, **S**low and **N**one.

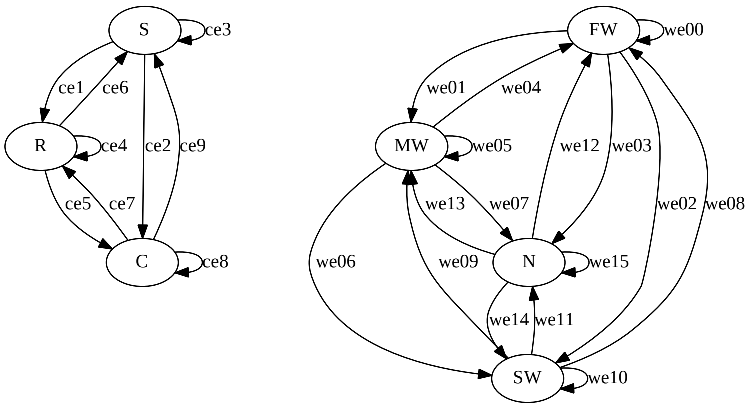


Figure 6.6: Generic SAN model for weather conditions and wind force. **"ce"** means climate event and **"we"** wind event.

Assuming this model is accurate to predict the wind and the climate of a city called *Gotham*, we want to know the probability of Gotham facing rain and fast wind at the same time. We do not have many records, so we need to learn this probability by a small sample which is formatted as Table 6.2.

Table 6.2: Sample for input data

Raw data		Symbolic data	
weather	**wind speed**	**weather**	**wind speed**
cloudy	48	b	d
raining	16	c	b
sunny	26	a	c
...	...	...	...
sunny	9	a	a

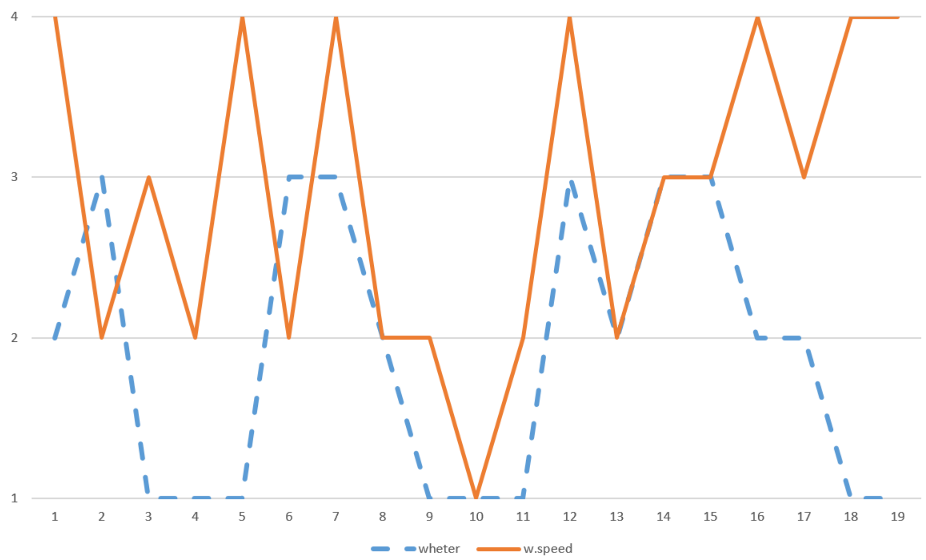


Figure 6.7: Line plot after the symbolic representation. Each letter is assigned to a value, $a = 1,\ b = 2,\ c = 3,\ d = 4$.

In this case, the probability to have rain and fast wind is 4.5%. In this example a basic combination of two automata was used with a maximum number of 80 states. However, the number of automata and states could be much bigger. This is a very compact representation and through SANGE; it scales better and demands less effort than the equivalent Markov model, allowing an easier derivation of probabilities in large models.

Example: Fathers and Sons

This example demonstrates the use of SANGE and its results through a dataset that stores the height of fathers and sons [2]. Our goal is to demonstrate the use of SANGE with a real dataset. Unlike the previous example, this dataset has 1078 records that not follows a time constraint; thus, a non-temporal data. Furthermore, we highlight the importance of an automatically generation, which allows a common user to use SAN models with higher amounts of data.

This dataset is composed by two variables, Father's and Son's heights, in inches. Figure 6.8 shows the relation between father's and son's height. The central line represents a linear prediction, delimited by the upper and lower bounds (red lines). Through this model, given a father's height, we can expect a son's height.

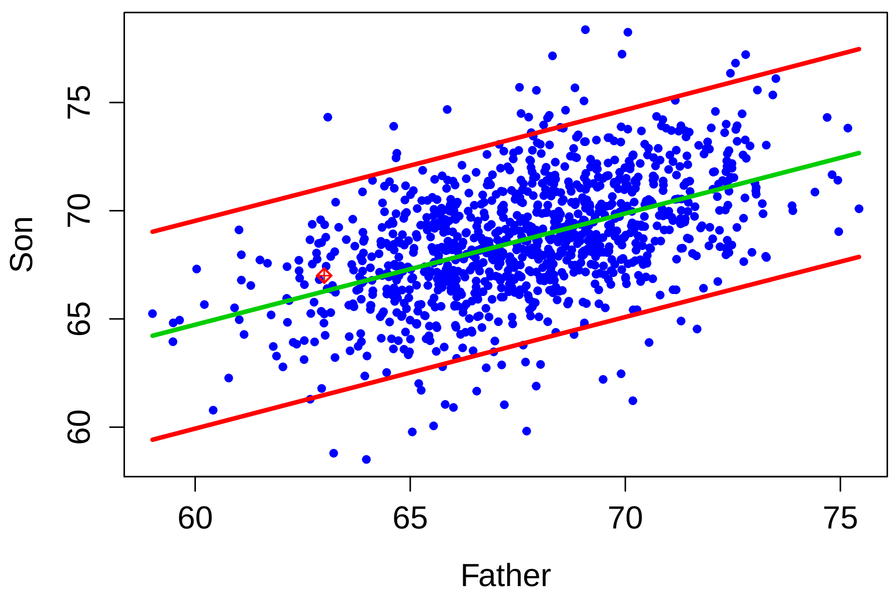


Figure 6.8: Linear model showing the relation between Father's height and son's height (In inches).

However, there are probabilistic questions that are not directly answered by these kind of models. Example, What is the probability to a short father, between $1.63m$ and $1.64m$, have a son with a height between $1.70m$ and $1.72m$? A mark $\oplus$ on the Figure 6.8 shows the center of this region.

Despite the lack of a temporal constraint, this relation still can be modeled or automatically fitted by SANGE. Directly from a dataset, it is possible to get classes for these values, *e.g.*, 63.2

[2]This data is public available in the R package "UsingR".

to 63.9 inches. These classes can be controlled by the number of variables, it generates a set of symbols (A PAA like technique) that will become states in a SAN automaton. Then, these states will be linked according to the techniques described in the begging of this Section. Finally, SANGE generates a SAN file that can be processed by a SAN algorithm [23].

Running the generated SAN file, we achieve 1.5% for a father that has between $1.63m$ and $1.64m$ generates a son that will be an adult with a height between $1.70m$ and $1.72m$. Although this simple example can be easily modeled, SANGE instantaneously generates the necessary code to derive the solution. Furthermore, it is possible to achieve solutions for similar problems with more variables and more data. These variables can be replaced by others that can represent nearly any other problem. Furthermore, due to the directly generation, *i.e.*generated directly from data, SANGE allows model measurements based on likelihood (Section 3.1.2). In fact, the real restrictions of the tool are the machine's memory, since it is not a parallel solution; and the total number of states generated, which can be up to millions in an ordinary personal computer, and still be solved in seconds [23]. Furthermore, the TS dimensionality reduction techniques used in this fitting method (Chapter 2) enables the generation of smaller automata with large number of records. Finally, SANGE is public available and its details are given in the Appendix E.

6.4 Applications for grammatical class prediction

Here we show an unusual application. The raw data are text that are transformed into tags and used into time series. Although we work with simple Markov Chains, this basic idea illustrates how the process can be used in different situations. In fact Markov chains are used due to it simplicity, yet although unnecessary it also can be applied with HMMs or SANs.

This application goal is to act as a disambiguate tool that helps us to predict grammatical classes from a given corpus. In Natural Language Processing (NLP), this is known as POS-tagging. In this approach we concern to provide a trustful knowledge base that stores all the gathered knowledge as Markov chains. Then, it can be directly used to discover the next grammatical classes.

This specialized flow is separated into two different phases. The first phase is to build and train a knowledge base and the second phase is to predict grammatical classes. In few words, it is an application for NLP that relies in a hybrid technique, linguistic and statistical, to Part-Of-Speech (POS) tagging. The statistical part of this technique is based on Markov chains and can be perfectly applied according to our process. However, due to its domain (NLP) and the linguistic part, we use a specialized flow (Figure 6.9 A).

6.4.1 POS tagging

Part-Of-Speech (POS) tagging is a very basic task for all Natural Language Processing (NLP) techniques based on linguistic approach. However, only well resourced languages have very effective solution for POS tagging [105]. This fact makes statistical approaches very popular to less resourced languages [71] or even domain lingo [114]. In the matter of acquiring information on informal texts,

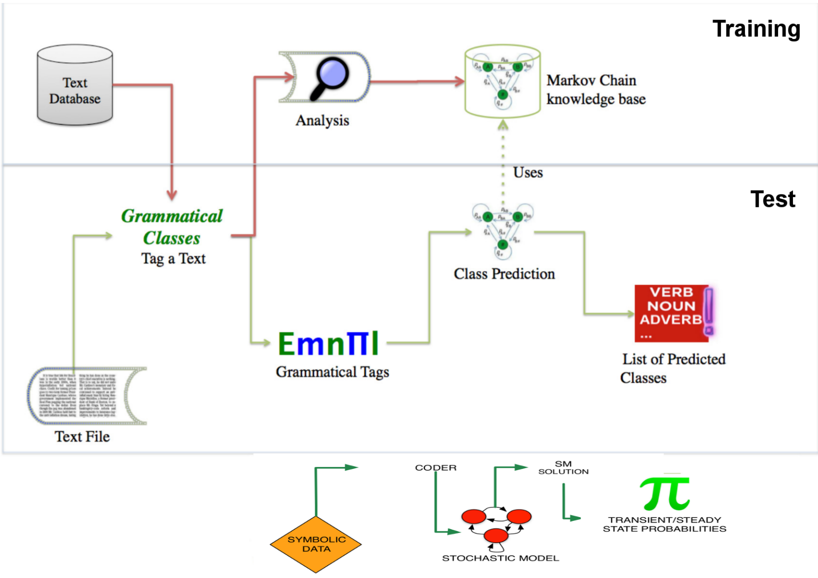


Figure 6.9: Process applied to grammatical class prediction

as social networks chats and comments, the use of a tool independent of formal language is even more interesting, since it can help to tackle an open NLP problem.

The difference between a a POS-tagger and a dictionary retriever is essentially the syntactic disambiguation task, *i.e.*, the decision of how to classify a word that is either unknown or that can be classified in more than one class. For instance, the word "*up*" can either be:

- An adverb, as in "It was a really fantastic feeling to be *up* on the podium";
- A preposition as in "she climbed *up* a flight of steps";
- An adjective as in "He took the *up* escalator";
- A verb as in "They cheer and *up* their glasses".
- A noun as in "you cannot have ups all the time in collective sports".

In contrast with the difficulty to find out structured knowledge for less resourced and informal languages, the Internet offers abundant unstructured material (texts) in every language and dialect. Consequently, a procedure capable to produce structured knowledge from textual sources would cope with such limitations.

As for software tools, the basic modules needed for a POS-tagger are a dictionary retriever (to identify possible grammatical classes for known words) and a predictor (to disambiguate words that can be employed with more than one role, or guess the role for unknown words). The dictionary retrieving task, for any language, can be solved very efficiently by the use of decision diagrams [96]. Furthermore, even multilingual dictionary retrievers can be implemented very efficiently [57].

Our goal is to provide an effective grammatical class predictor to words inside a sentence previously tagged with an efficient dictionary retriever. In order to do so, we propose the construction of

Markov chains (Section 1.2.1) to describe typical phrases and use transient analysis such chains to predict the grammatical class of each word. These typical phrases can be viewed as a training set for the POS-tagger. Consequently, the produced Markov chains can be viewed as a model of the language patterns, *i.e.*the structured knowledge of the target language.

This method fits into a POS-tagging landscape simpler than sophisticated approaches based on Data mining; Approaches as Support Vector Machines SVM [66], or inference models as Conditional Random Fields - CRF [85]. Nevertheless, the use of simpler stochastic formalisms implies on more flexible than traditional linguistic approaches heavily anchored in specific languages [19] [107].

Note that the proposed POS-tagger can be applied to any language, assuming that there is a dictionary and a training set of sentences for that language. In fact, the method is sufficiently general to be used with more than one dictionary, and consequently the training set must have the multi-language phrases, *i.e.*, phrases where words of different languages can be used. The input needed to create a knowledge base is basically symbolic data representing the possible classes for each word.

The symbolic data (see Figure 6.9) is generated by a specialized tool, a diagram decision-based dictionary Retriever. The use of decision diagrams to recognize words, and associate its word class(or possible classes), can be very efficient and very effective as proposed in the WAGGER software tool [57]. According to this work, a decision diagram structure holding a multilingual dictionary with English and Portuguese words (a little more than one million words)can be used to tag possible word classes to large corpora (for instance, two million words) in less than three seconds using a personal machine. Such performance leads us to use WAGGER for to pre-identify grammatical classes in a corpus (Figure 6.9, "*Grammatical Classes*").

The WAGGER output is all words of each sentence followed by possible word classes. For instance, Figure 6.10 presents the output of WAGGER using our English dictionary [57] for the sentence "I believe what I see, but I see what I look and I look what I want to look". In this ïňĄgure, the correct word class is indicated in bold. For instance, word "and" plays the role of conjunction in this sentence, but according to the dictionary "and" can also be employed as a noun. In order to formalize, the WAGGER output we will denote by w_i^s the i-th word in a sentence s, and $C(w_i^s)$ the set of all possible word classes of a word $C(w_i^s)$.

$C(w_i^s)$ is also an independent of the sentence in which the word is (s). For instance, the word "and" (the thirteenth word of the example sentence) is formally defined by: w_{13}^s="and" $C(w_{13}^s)=$ {conjunction, noun}.

The proposed method follows the general idea of train and test, since it starts with a set of sentences manually tagged (training set) before disambiguate word tags for sentences tagged by WAGGER (testing set). As shown at Figure 6.9, the proposed method has two main tasks: the training task to construct the Markov Chains; and the testing task to disambiguate WAGGER multiple or absent tags. It is important to notice that the training task needs a supervised action, the human made annotation that is needed to be done according to the language of the corpus to be tagged.

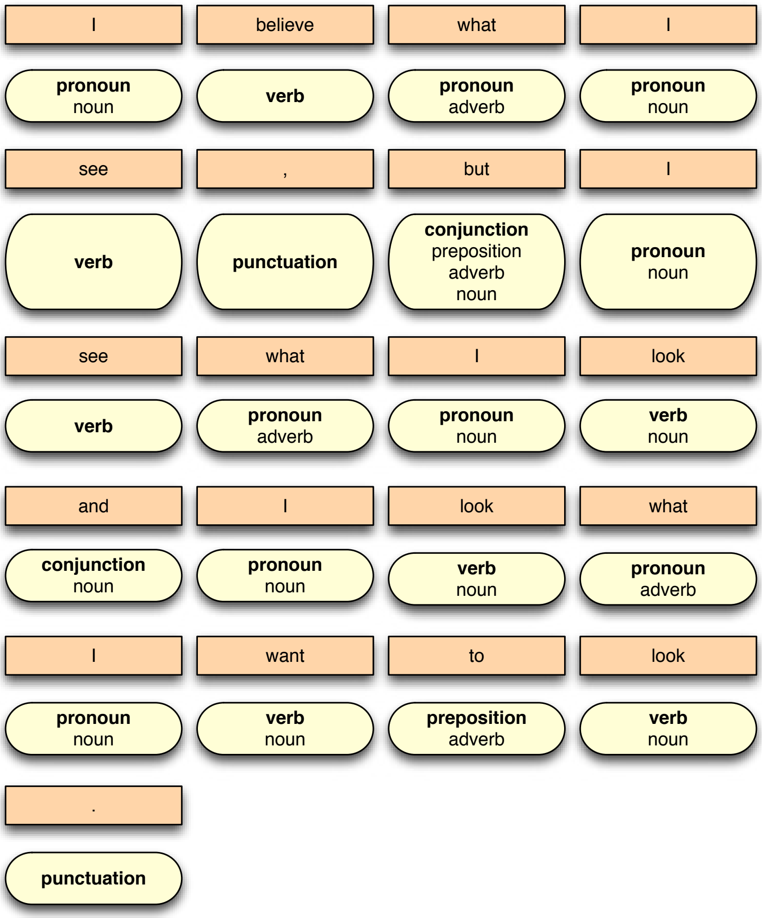


Figure 6.10: WAGGER output for an example sentence

6.4.2 Training

Giving annotated sentences as the example in we construct a Markov chain for each sentence considering the sequences of word classes found in the sentence. This process corresponds to the capturing of an approximative model of the patterns of word class sequences, having each example sentence as an instance of the language usage.

Giving an annotated sentence as the example in Figure 6.11, we can construct a Markov chain considering the sequences of word classes found in the sentence. In fact, this process corresponds to capture an approximative model of the patterns of word class sequences, having each example sentence as a valid instance of the language usage.

Basically, the Markov Chain created has all word classes as the chain states, and every exiting arc represents a possible succession of words classes. For instance, the adjectives (*adj*) are succeeded twice by a noun (*n*) and once by a conjunction (*conj*). Hence, the *adj* state has an exiting arc towards *n* with probability $\frac{2}{3}$ and an exiting arc towards *conj* with probability $\frac{1}{3}$. Repeating the same procedure for all word classes creates the Markov Chain depicted in Fig. 6.12.

It is important to notice that there is no exiting arc from one state towards itself, but it does not mean that a succession of the same word class is ignored. For instance, there are five verbs in

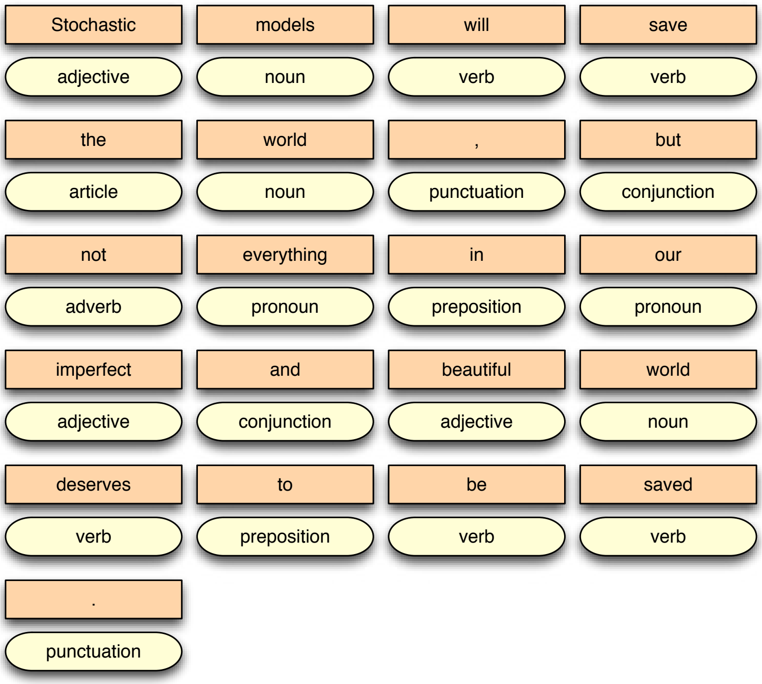


Figure 6.11: Manually annotated sentence used as training.

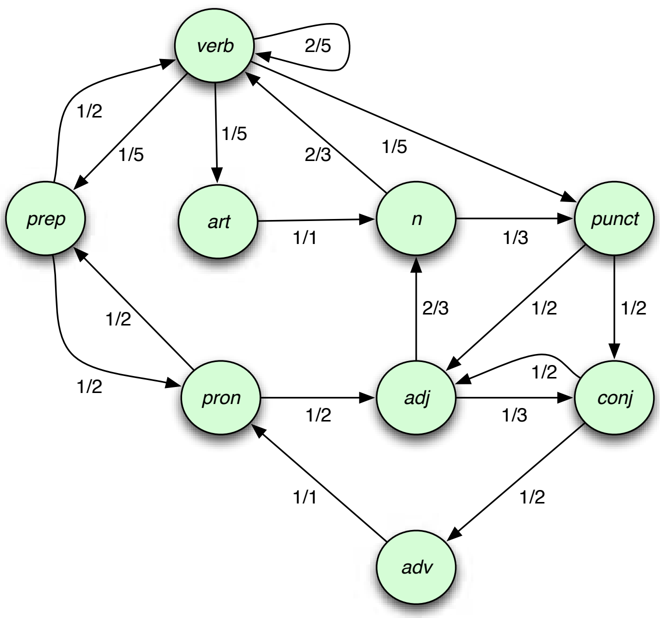


Figure 6.12: Markov Chain for Fig. 6.11 sentence.

Figure 6.11 sentence, two are followed by verb (*verb*), the other three are followed by article (*art*), preposition (*prep*) and punctuation (*punct*). As consequence, the rates of exiting arcs of *verb* state are $\frac{1}{5}$ each, *i.e.*, they add up $\frac{3}{5}$, instead of 1.

This same process of creation of Markov Chain is repeated for a certain number of sentences. For example, Figure 6.10 sentence correctly annotated would produce the Markov Chain depicted in Figure 6.13.

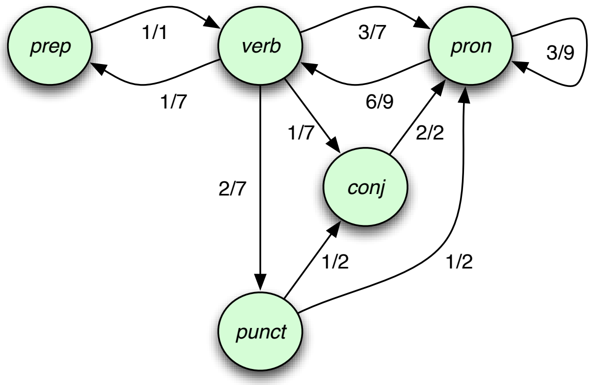


Figure 6.13: Markov Chain for Fig. 6.10 sentence.

6.4.3 Test and disambiguation

The disambiguation process is made through the estimation of probabilities of neighbors words according to the constructed chains. Every word that needs disambiguation has the probabilities analyzed to all possible word classes. Specifically, for each word needing disambiguation, we compute the probability of its immediate predecessor and successors according to each DTMC.

Formally, a word w_i in sentence s, has its probability of being of word class c computed according its predecessor (Equation 6.1) and successor (Equation 6.2) to DTMC m respectively by:

$$\overleftarrow{c}^{(m)}_{w_i^s} = \frac{\sum\limits_{\forall k \in C(w_{i-1})} \phi^{(m)}_{[k(w^s_{i-1}),c(w^s_i)]}}{\mid C(w_{i-1}) \mid} \tag{6.1}$$

$$\overrightarrow{c}^{(m)}_{w_i^s} = \frac{\sum\limits_{\forall k \in C(w_{i+1})} \phi^{(m)}_{[c(w^s_i),k(w^s_{i+1})]}}{\mid C(w_{i+1}) \mid} \tag{6.2}$$

where $\phi^m_{i,j}$ is the probability of leaving state i and going to state j in the Markov chain m.

The overall probability of a word w_i in sentence s be of class c according to all training DTMC's of a set M is given by:

$$\widehat{c}_{w_i^s} = \sum_{\forall m \in M} \frac{\overleftarrow{c}^{(m)}_{w_i^s} + \overrightarrow{c}^{(m)}_{w_i^s}}{2 \mid M \mid} \tag{6.3}$$

Hence, the disambiguation of a word w_i^s will be made by choosing the word class $c \in C(w_i)$ with the maximum value $\widehat{c}_{w_i^s}$.

For example, considering the sentence "*You reap what you sow.*", the submission to WAGGER results in the annotation depicted in Figure 6.14. In this sentence the word "*what*" needs disambiguation, since it can be either a pronoun or an adverb.

Using the first chain (Figure 6.12), let us call it m_1, we observe that the probability of a pronoun be preceded by a verb ("*reap*" precedes "*what*") is equal to zero, since there is no arc linking the

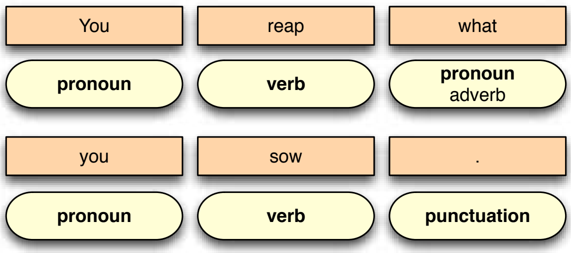


Figure 6.14: First example sentence to disambiguate.

verb state to the *pron* state in this Markov chain. Analogously, according to m_1 the probability of a pronoun be preceded by another pronoun is equal to zero too, *i.e.*:

$$\overleftarrow{pron}^{(m_1)}_{w_3^s} = 0.0 \qquad \overrightarrow{pron}^{(m_1)}_{w_3^s} = 0.0$$

Using the second DTMC (Figure 6.13), let us call it m_2, we compute these same probabilities to $\frac{3}{7}$ (a verb to precede a pronoun) and $\frac{3}{9}$ (a pronoun to precede another pronoun), formally:

$$\overleftarrow{pron}^{(m_2)}_{w_3^s} = 0.43 \qquad \overrightarrow{pron}^{(m_2)}_{w_3^s} = 0.67$$

Hence, considering the two DTMC's m_1 and m_2, we correctly estimate the probability of the word "*what*" (w_3^s) being a pronoun or an adverb in the sentence of Fig. 6.14 as:

$$\widehat{pron}_{w_3^s} = 0.27 \qquad \widehat{adv}_{w_3^s} = 0.25$$

Another example is given by the sentence "*The old book is dusty and black.*". In this case, the words "*book*" and "*black*" need disambiguation.

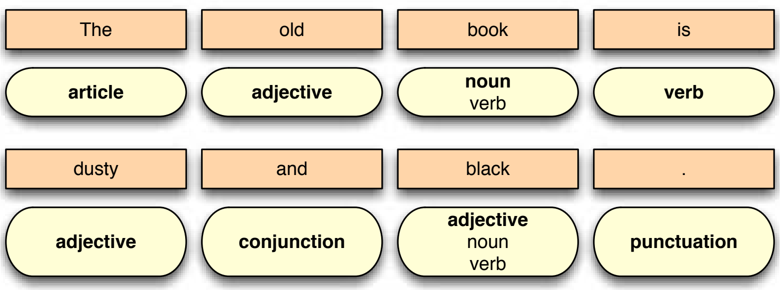


Figure 6.15: Second example sentence to disambiguate.

Using m_1, we observe that the probability of a noum be preceded by an adjective ("old" precedes "book") is equal to $\frac{2}{3}$, and since "old" can be only an adjective, *i.e.*, $C(\text{"}old\text{"}) = \{adjective\}$, we express formally:

$$\overleftarrow{n}^{(m_1)}_{\text{"}book\text{"}} = \frac{\phi^{(m_1)}_{[adj,n]}}{|C(\text{"}old\text{"})|} = \frac{0.6667}{1} = 0.6667$$

As well as the probability of a noum to precede a verb ("book" precedes "is") is equal to $\frac{2}{3}$, and since "is" can be only a verb, *i.e.*, $C(\text{"}is'') = \{verb\}$, formally:

$$\overrightarrow{n}^{(m_1)}_{\text{"}book\text{"}} = \frac{\phi^{(m_1)}_{[n,verb]}}{|C(\text{"}is\text{"})|} \quad = \frac{0.6667}{1} = 0.6667$$

Analogously, repeating the computations for the second chain (Figure 6.13), let us call it m_2, we obtain zero, since there is no word class noun in it, formally:

$$\overleftarrow{n}^{(m_2)}_{\text{"}book\text{"}} = \frac{\phi^{(m_2)}_{[adj,n]}}{|C(\text{"}old\text{"})|} \quad = \frac{0}{1} = 0$$

$$\overrightarrow{n}^{(m_2)}_{\text{"}book\text{"}} = \frac{\phi^{(m_2)}_{[n,verb]}}{|C(\text{"}is\text{"})|} \quad = \frac{0}{1} = 0$$

Computing the overall probability for "book" (w^s_3) be a noun considering the two Markov chains $(M \;=\; m_1, m_2)$ will be formally:

$$\widehat{n}_{w^s_3} = \frac{\overleftarrow{n}^{(m_1)}_{\text{"}book\text{"}} + \overrightarrow{n}^{(m_1)}_{\text{"}book\text{"}} + \overleftarrow{n}^{(m_2)}_{\text{"}book\text{"}} + \overrightarrow{n}^{(m_2)}_{\text{"}book\text{"}}}{2|M|}$$

$$\widehat{n}_{w^s_3} = 0.333$$

Thus, computing the probabilities as below we correctly conclude that "*book*" (w^s_3) is a noun instead of a verb. Analogously, "black" (w^s_7) is an adjective.

$$\widehat{n}_{w^s_3} = 0.33 \quad \widehat{verb}_{w^s_3} = 0.10$$

$$\widehat{adj}_{w^s_7} = 0.13 \quad \widehat{n}_{w^s_7} = 0.08 \quad \widehat{verb}_{w^s_7} = 0.12$$

6.4.4 Results and final considerations

We applied the proposed method in a larger scale to illustrate the efficiency and effectiveness benefits of our approach. To do so, we considered the following test bed:

- Two similar POS-tagger tools to Portuguese language: PALAVRAS [19] and LX-Center Suite [107];
- A Portuguese dictionary and the WAGGER dictionary retriever [57];
- Fifty Brazilian Portuguese manually annotated sentences to be used as training set;
- One hundred and twenty eight Brazilian Portuguese manually annotated sentences to be used as test set.

Applying our proposed method to the sentences of the training set has produced 50 discrete-time Markov chains. As mentioned, these chains can be considered an approximative model of Brazilian language usage in terms of word class sequences. The choice of these sentences is merely random, since these sentences were randomly taken from domain corpora composed of articles, and other academic texts of several scientific domains. After, 128 sentences were also randomly chosen from the same domain corpora in order to test our proposed method. The only concern choosing this 128 test sentences were not to choose sentence that were already used as training set. For both training and testing sets of sentences the annotation was performed by two linguist specialists.

This method showed itself less accurate (80%) than PALAVRAS and LX-Center (both near 87%) that have a rigid approach. Yet, the efficiency of this method is far more impressive due to its simple structure. In a common, portable machine with i7 2.2 GHz processor and 8 Gbytes of memory; our method takes less than 1 second. Furthermore it can always be improved by a careful choice of a training set.

This novel idea is much simpler than sophisticated CRF or SVM. Nevertheless, our method is more prone to evolution than traditional rigid approaches available at existing POS-taggers like PALAVRAS and LX-Center Suite. HMMs also are a traditional option, but it additional stochastic layer if not useful in this case.

Finally, this unusual approach for the proposed process (Figure 5.1) shows that different kinds of data are suitable for its phases. In this POS-tagger the grammatical classes tagged using WAGGER are *Symbolic Data*, *i.e.*, symbols representing classes. The *Coder* is the software responsible for turning the tagged and analyzed data into DTMCs. The *Stochastic Model* is a set of DTMCs (Section 6.4.2). The *Solution* is given by the forward and backward analyze of the chains (Section 6.4.3). Finally, the probabilities are used as a disambiguate criterion.

Chapter 7

Conclusion

We presented a set of techniques to fit stochastic models. Among these techniques, we can emphasize the SAN fitting. However, on the way to create a SAN fitting, other significant contributions were created. These contributions have an important role in the whole scenario because they complement each other, driving this work for its primary goal; the improvement of stochastic models' fitting techniques for knowledge discovery.

We created a union of time series (TS) techniques with stochastic models. A seamless integration that allowed the generation of complex model structures directly from data, which can generate a set of states concerning a given scenario. Our approach is similar to the KDD process, except our focus concerns TS and models characterized by the Markov property. The process key point is the use of TS representation methods to reduce data dimensionality and fit a model according to the Maximum Likelihood Estimation. Specifically, we used a PAA like technique to fit MCs, HMMs, and SANs. The next section point out every contribution generated by this work.

7.1 Contributions

An important model was described in the Section 4.2. It is a first SAN model to represent a set of geological phenomena. More specifically, this model can be used to predict generic types of stratal stacking patterns for Pelotas Basin, in Brazil. It illustrates a potentially useful and previously unknown prediction tool to retrieve statistics about a natural phenomenon.

Based on the experience of creating a SAN model for geological data. We started to study and organize steps that are common in many scenarios and, usually, can be automated. Based on the KDD process (Section 1.1) and the characteristics of stochastic models (Chapter 3); we proposed "A dimensionality reduction process" (Chapter 5). This process embodies useful tasks to find knowledge through stochastic modeling. It is not a fixed set of techniques but a general process which by using general steps, can be applied to different systems. Furthermore, through examples, were shown the advantages of this process. It effectively uses techniques based on time series to automate steps, whose representation capabilities reduce the data dimensionality and creates a symbolic representation that feeds a model, saving time from the modeler.

This thesis also presents a literature review and some available methods to generate stochastic models considering three different Markovian formalisms, Markov Chains (MC), Hidden Markovian Models (HMM) and Stochastic Automata Networks (SAN). This literature review is necessary to understand their structures, how it can be fitted and measured.

Concerning HMMs, we created an algorithm, based on the Expectation-Maximization (EM) for

HMMs (Baum-Welch). Piecewise Aggregation EM (PAEM) uses a time series technique to perform a pre-fitting, allowing the EM to find its local maximum likelihood in fewer iterations (Chapter 3). It is a slight improvement to the original BW, yet its combination is a new approach, that might be useful in many scenarios.

Concerning SANs, we presented SANGE (Section 6.3), a generator based on time series that can perform a SAN fitting, in an automated way, directly from a dataset. It allows non-specialist users to take advantage of SAN's structure and solutions. As SANGE is a first attempt to automatically generate SAN models that can be useful to the real world datasets, we have introduced a new method for knowledge discovery through stochastic modeling.

Traditionally stochastic models are evaluated by its likelihood. Yet, many stochastic simulations commonly deliver different results. In cases where the stochasticity are not desired in all the simulated data; *i.e.*, some behavior is previously known; measure the distance between the known behavior and the simulations can be helpful to decide which simulation to choose. Thus, as a second layer of measurement, in Chapter 5.5, we propose Dynamic Time Warping as a second layer of model's measurements.

7.2 Perspectives

Concerning the SAN model described in the Section 4.2, we have perspectives in different domains of science. From computer science, the model can be studied to be applied in other scenarios with similar behavior, *i.e.*, geological data from other sources describing similar phenomena. Furthermore, events created by natural phenomena, usually, are composed of larger numbers of variables; and continuous, both in time and amplitude. Such characteristics make these events impossible to simulate precisely. As non-deterministic systems, composed by large sets of events, we could use the SAN formalism to create accurate models. From the geosciences point of view, the probabilities generated can help against major human error factors that may arise with classical geological analysis. This model can be seen as an analytical tool, which might support the decision-making.

Concerning our general purpose process (Chapter 5), as well as we see the KDD process widely spread in the literature, in a foreseeable future this process might be handy as a guide for stochastic models' generation from data. Therefore, allowing these methods to be used by a broad community, which will bring new researchers to the field, consequently improving the state of the art. Furthermore, through SANGE, represents, SAN models can be included in the process, and widely delivered. As an MLE based method, SANGE itself can be improved by adding other fitting criteria and measurements. Other important future work, is to measure the impact of dimensionality reduction techniques. In such regard, it can be relative to different data types and distributions. Mainly, the level of reduction regarding all these characteristics.

PAEM, can be improved via measurements regarding the initial likelihood and the user time according to different kinds of data and distributions. Furthermore, as PAEM is based on a simple adaptation of PAA, it may have a better global performance if a more advanced technique is used, Chapter 2 shows many possible options. Finally, all the contributions here described can be used to extend the use of stochastic models, making slightly, yet important, improvements in the state of the art.

REFERENCES

[1] J. Abfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search on time series based on threshold queries. In Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, editors, *Advances in Database Technology - EDBT 2006*, volume 3896 of *Lecture Notes in Computer Science*, pages 276–294. Springer Berlin Heidelberg, 2006.

[2] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, FODO '93, pages 69–84, London, UK, 1993. Springer-Verlag.

[3] H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, Dec 1974.

[4] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2004.

[5] A. Arapostathis and S. I. Marcus. Analysis of an identification algorithm arising in the adaptive estimation of markov chains. *Mathematics of Control, Signals and Systems*, 3(1):1–29, 1990.

[6] J. Assunção, P. Fernandes, L. Lopes, and S. Normey. A dimensionality reduction process to forecast events through stochastic models. In *Proceedings of the 26th International Conference on Software Engineering & Knowledge Engineering, SEKE 2014*, pages 534–539, HyattRegencyHotel, Vancouver, Canada, Jul 2014. Knowledge Systems Institute Graduate School. ISBN-13: 978-1-891706-35-7.

[7] J. Assunção, P. Fernandes, L. Lopes, A. Studeny, and J. Vincent. SANGE - stochastic automata networks generator. A tool to efficiently predict events through structured markovian models. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, pages 581–584, 2015.

[8] J. Assunção, P. Fernandes, T. Fischer, and A. Sales. Unsupervised model generation for geological events. In *Proceedings of the 47th Annual Simulation Symposium*, ANSS 14, 2014.

[9] J. Assunção, L. Espindola, P. Fernandes, M. A. G. Pivel, and A. Sales. A structured stochastic model for prediction of geological stratal stacking patterns. *Electronic Notes in Theoretical Computer Science*, 296:27–42, 2013. doi: 10.1016/j.entcs.2013.07.003.

[10] J. Assunção, P. Fernandes, L. Lopes, A. Studeny, and J.-M. Vincent. SANGE -Stochastic Automata Networks Generator. A tool to efficiently predict events through structured Markovian models (extended version). Research Report RR-8724, Inria Rhône-Alpes ; Grenoble University ; Pontifícia Universidade Católica do Rio Grande do Sul, Mar. 2015.

[11] L. R. Bahl and F. Jelinek. Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition. *Information Theory, IEEE Transactions on*, 21(4):404–411, Jul 1975.

[12] J. Baker. The dragon system–an overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):24–29, Feb 1975.

[13] P. Baldi and Y. Chauvin. Smooth on-line learning algorithms for hidden markov models. *Neural Comput.*, 6(2):307–318, Mar. 1994.

[14] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Models For Master/Slave Parallel Programs. *Electronic Notes in Theoretical Computer Science*, 128(4):101–121, 2005. doi: 10.1016/j.entcs.2005.01.015.

[15] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 02 1970.

[16] A. Benoit, B. Plateau, and W. J. Stewart. Réseaux d'automates stochastiques à temps discret. *Revue des sciences et technologies de l'information, Technique et science informatiques, "Evaluation de Performances"*, 24(2-3):229–248, 2005.

[17] J. Berman. *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*. Elsevier Science, 2013.

[18] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop'94*, pages 359–370, 1994.

[19] E. Bick. *The Parsing System "Palavras": Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Aarhus University Press, 2000.

[20] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis: forecasting and control*. Wiley.com, 2013.

[21] K. Braghetto, J. Ferreira, and J.-M. Vincent. Performance evaluation of business processes through a formal transformation to san. In N. Thomas, editor, *Computer Performance Engineering*, volume 6977 of *Lecture Notes in Computer Science*, pages 42–56. Springer Berlin Heidelberg, 2011.

[22] L. Brenner, P. Fernandes, J.-M. Fourneau, and B. Plateau. Modelling Grid5000 point availability with SAN. *Electronic Notes in Theoretical Computer Science*, 232:165–178, 2009. doi: 10.1016/j.entcs.2009.02.056.

[23] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity. PEPS2007 - Stochastic Automata Networks Software Tool. In *Proceedings of the 4th International Conference on Quantitative Evaluation of SysTems (QEST 2007)*, pages 163–164, Edinburgh, UK, September 2007. IEEE Computer Society.

[24] L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations. *International Journal of Simulation: Systems, Science & Technology (IJSIM)*, 6(3-4):52–60, February 2005.

[25] O. Cappe, V. Buchoux, and E. Moulines. Quasi-newton method for maximum likelihood estimation of hidden markov models. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 4, pages 2265–2268, May 1998.

[26] O. Cappé, E. Moulines, and T. Ryden. *Inference in Hidden Markov Models*. Springer Series in Statistics. Springer, 2006.

[27] O. Catuneanu. *Principles of Sequence Stratigraphy*. Elsevier, Amsterdam, 2006.

[28] O. Catuneanu, V. Abreu, J. Bhattacharya, M. Blum, R. Dalrymple, P. Eriksson, C. Fielding, W. Fisher, W. Galloway, M. Gibling, K. Giles, J. Holbrook, R. Jordan, C. Kendall, B. Macurda, O. Martinsen, A. Miall, J. Neal, D. Nummedal, L. Pomar, H. Posamentier, B. Pratt, J. Sarg, K. Shanley, R. Steel, A. Strasser, M. Tucker, and C. Winker. Towards the standardization of sequence stratigraphy. *Earth-Science Reviews*, 92(1-2):1–33, 2009.

[29] G. Celeux, D. Chauveau, and J. Diebolt. On Stochastic Versions of the EM Algorithm. Research Report RR-2514, 1995.

[30] G. Celeux, D. Chauveau, and J. Diebolt. Stochastic versions of the em algorithm: an experimental study in the mixture case, 1996.

[31] G. Celeux and G. Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics & Data Analysis*, 14(3):315 – 332, 1992.

[32] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.

[33] K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, pages 126–133, Mar 1999.

[34] R. Chang and J. Hancock. On receiver structures for channels having memory. *Information Theory, IEEE Transactions on*, 12(4):463–468, Oct 1966.

[35] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer, and A. F. Zorzo. Analytical Modeling for Operating System Schedulers on NUMA Systems. *Electronic Notes in Theoretical Computer Science*, 151(3):131–149, 2006. doi: 10.1016/j.entcs.2006.03.016.

[36] C. Chatfield. *Time-series forecasting*. Chapman and Hall/CRC, 2002.

[37] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 792–803. VLDB Endowment, 2004.

[38] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 491–502, New York, NY, USA, 2005. ACM.

[39] Y. Chen, K. Chen, and M. A. Nascimento. Effective and efficient shape-based pattern detection over streaming time series. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):265–278, Feb. 2012.

[40] I. B. Collings, V. Krishnamurthy, and J. B. Moore. On-line identification of hidden markov models via recursive prediction error techniques. *IEEE Transactions on Signal Processing*, 42(12):3535–3539, Dec 1994.

[41] J. Contreras. *Seismo-stratigraphy and numerical basin modeling of the southern Brazilian continental margin (Campos, Santos, and Pelotas basins)*. PhD thesis, Ruprecht-Karls-Universität, Heidelberg, Germany, 2011.

[42] J. Contreras, R. Zühlke, S. Bowman, and T. Bechstädt. Seismic stratigraphy and subsidence analysis of the Southern Brazilian margin (Campos, Santos and Pelotas basins). *Marine and Petroleum Geology*, 27:1952–1980, 2010.

[43] P. Cowpertwait and A. Metcalfe. *Introductory Time Series with R*. Use R! Springer, 2009.

[44] R. M. Czekster, P. Fernandes, A. Sales, T. Webber, and A. F. Zorzo. Stochastic Model for QoS Assessment in Multi-tier Web Services. *Electronic Notes in Theoretical Computer Science*, 275:53–72, 2011. doi: 10.1016/j.entcs.2011.09.005.

[45] R. M. Czekster, P. Fernandes, J.-M. Vincent, and T. Webber. Split: a flexible and efficient algorithm to vector-descriptor product. In *International Conference on Performance Evaluation Methodologies and tools (ValueTools'07)*, volume 321 of *ACM International Conferences Proceedings Series*, pages 83–95. ACM Press, 2007.

[46] R. M. Czekster, P. Fernandes, and T. Webber. Efficient vector-descriptor product exploiting time-memory trade-offs. *ACM SIGMETRICS Performance Evaluation Review*, 39(3):2–9, December 2011. doi: 10.1145/2160803.2160805.

[47] L. da Silveira Espindola. Proposta de uma Representação Tensorial para Modelos Markovianos Ocultos. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), 2011.

[48] DataMarket! The open portal to thousands of datasets from leading global providers. http://datamarket.com/, 2013.

[49] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[50] M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[51] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, Aug. 2008.

[52] Y. Ding, X. Yang, J. Li, and A. Kavs. A new representation and distance measure for financial time series. In *Proceedings of the 2nd IEEE International Conference on Information and Financial Engineering (ICIFE)*, pages 220–224, 2010.

[53] F. L. Dotti, P. Fernandes, A. Sales, and O. M. Santos. Modular Analytical Performance Models for Ad Hoc Wireless Networks. In *Proceedings of the 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2005)*, pages 164–173. IEEE Computer Society, April 2005. doi: 10.1109/WIOPT.2005.31.

[54] R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[55] D. A. V. Dyk, X. li Meng, and D. B. Rubin. Maximum likelihood estimation via the ecm algorithm: Computing the asymptotic variance. Technical report, 1994.

[56] A. G. Farina, P. Fernandes, and F. M. Oliveira. Representing software usage models with stochastic automata networks. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE '02, pages 401–407, New York, NY, USA, 2002. ACM. doi: 10.1145/568760.568830.

[57] P. Fernandes, L. Lopes, C. A. Prolo, A. Sales, and R. Vieira. A fast, memory efficient, scalable and multilingual dictionary retriever. In *Proceedings of the Language Resources and Evaluation Conference (LREC)*, pages 2520–2524, 2012.

[58] P. Fernandes and B. Plateau. Stochastic Automata Networks, SAN: Modelling and Evaluation. In U. Herzog and M. Rettelbach, editors, *Proceedings of the 2th Workshop on Process Algebras and Performance Modelling (PAPM'94)*, pages 127–136, July 1994.

[59] P. Fernandes and B. Plateau. Modeling Finite Capacity Queueing Networks with Stochastic Automata Networks. In *Proceedings of the 4th International Workshop on Queueing Networks with Finite Capacity (QNETs 2000)*, pages 1–12, July 2000. http://www3.pucrs.br/pucrs/files/uni/poa/facin/pos/relatoriostec/tr004.pdf.

[60] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[61] P. Fernandes, A. Sales, A. R. Santos, and T. Webber. Performance Evaluation of Software Development Teams: a Practical Case Study. *Electronic Notes in Theoretical Computer Science*, 275:73–92, 2011. doi: 10.1016/j.entcs.2011.09.006.

[62] J. Fessler and A. Hero. Space-alternating generalized expectation-maximization algorithm. *Signal Processing, IEEE Transactions on*, 42(10):2664–2677, Oct 1994.

[63] G. Florez-Larrahondo, S. Bridges, and E. A. Hansen. Incremental estimation of discrete hidden markov models based on a new backward procedure. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 758–763. AAAI Press, 2005.

[64] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: an overview. *AI Magazine*, 13(3):57–70, Sept. 1992.

[65] A. Garg and M. K. Warmuth. Inline updates for HMMs. In *Proceedings of the INTERSPEECH*. ISCA, 2003.

[66] J. Giménez and L. Màrquez. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 43–46, 2004.

[67] T. Giorgino. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. *Journal of Statistical Software*, 31(7):1–24, 8 2009.

[68] F. Gradstein, J. Ogg, and A. Smith. *A Geologic Time Scale*. March 2005.

[69] J. Han, M. Kamber, and J. Pei. *Data Mining, Second Edition: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2006.

[70] J. Hardenbol, J. Thierry, M. Farley, T. Jacquin, P. de Graciansky, and P. Vail. Mesozoic and Cenozoic sequence chronostratigraphic framework of European Basins. In *Mesozoic and Cenozoic Sequence Stratigraphy of European Basins*, volume 60 of *Soc. of Econ. Paleontologists and Mineralogists. Special Publication Series*, pages 3–13. 1998.

[71] F. M. Hasan, N. UzZaman, and M. Khan. *Comparison of different POS Tagging Techniques (n-gram, HMM and Brill's tagger) for Bangla*, pages 121–126. Springer Netherlands, Dordrecht, 2007.

[72] R. Howard. *Dynamic probabilistic systems*, volume 2 of *Dynamic Probabilistic Systems*. Wiley, 1971.

[73] M. Jamshidian and R. I. Jennrich. Conjugate gradient acceleration of the em algorithm. *Journal of the American Statistical Association*, 88(421):221–228, 1993.

[74] F. Jelinek, L. Bahl, and R. Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Trans. Inf. Theor.*, 21(3):250–256, Sept. 2006.

[75] D. Karlis and E. Xekalaki. Choosing initial values for the EM algorithm for finite mixtures . *Computational Statistics & Data Analysis*, 41(3-4):577 – 590, 2003. Recent Developments in Mixture Model.

[76] R. Kawamoto, A. Nazir, A. Kameyama, T. Ichinomiya, K. Yamamoto, S. Tamura, M. Yamamoto, S. Hayamizu, and Y. Kinosada. Hidden markov model for analyzing time-series health checkup data. In C. U. Lehmann, E. Ammenwerth, and C. Nøhr, editors, *MedInfo*, volume 192 of *Studies in Health Technology and Informatics*, pages 491–495. IOS Press, 2013.

[77] F. Kelly. *Reversibility and Stochastic Networks*. Cambridge Mathematical Library. Cambridge University Press, 2011.

[78] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417. VLDB Endowment, 2002.

[79] E. Keogh and A. Ratanamahatana. Everything you know about dynamic time warping is wrong. *Proceedings of the 3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), Seattle, WA*, 2004.

[80] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and Ratanamahatana. The UCR Time Series Classification/Clustering Homepage. www.cs.ucr.edu/ eamonn/time_series_data/, 2011.

[81] Keogh, Eamonn and Wei, Li and Xi, Xiaopeng and Lee, Sang-Hee and Vlachos, Michail. Lb-keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 882–893. VLDB Endowment, 2006.

[82] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD 97, pages 289–300, New York, NY, USA, 1997. ACM.

[83] V. Krishnamurthy and J. B. Moore. On-line estimation of hidden markov model parameters based on the kullback-leibler information measure. *IEEE Transactions on Signal Processing*, 41(8):2557–2573, Aug 1993.

[84] F. L. Dotti, P. Fernandes, and C. Nunes. Structured Markovian models for discrete spatial mobile node distribution. *Journal of the Brazilian Computer Society*, 17:31–52, 2011. doi: 10.1007/s13173-010-0026.

[85] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[86] F. LeGland and L. Mevel. Recursive identification of HMMs with observations in a finite set. In *Proceedings of the 34th IEEE Conference on Decision and Control*, volume 1, pages 216–221, Dec 1995.

[87] M. Leng, X. Lai, G. Tan, and X. Xu. Time series representation for anomaly detection. In *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, pages 628–632, Aug 2009.

[88] H. Li, L. Fang, P. Wang, and J. Liu. An algorithm based on piecewise slope transformation distance for short time series similarity measure. In *Proceedings of the 10th World Congress on Intelligent Control and Automation (WCICA)*, pages 691–695, 2012.

[89] A. M. Lima, M. A. Netto, T. Webber, R. M. Czekster, C. A. D. Rose, and P. Fernandes. Performance evaluation of openmp-based algorithms for handling kronecker descriptors. *Journal of Parallel and Distributed Computing*, 72(5):678–692, 2012.

[90] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, DMKD '03, pages 2–11, New York, NY, USA, 2003. ACM.

[91] Z. Linjessica, Y. Huang, H. Wang, and S. McClean. Neighborhood counting for financial time series forecasting. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 815–821, 2009.

[92] C. Liu and D. B. Rubin. The ECME Algorithm: A Simple Extension of EM and ECM with Faster Monotone Convergence. *Biometrika*, 81(4):633–648, 1994.

[93] C. Liu, D. B. Rubin, and Y. N. Wu. Parameter Expansion to Accelerate EM: The PX-EM Algorithm. *Biometrika*, 85(4):755–770, 1998.

[94] Q. Liu, Z. qin Huang, Y. bin Hou, and R. Chen. Utterance verification on DTW based speech recognition using likelihood. In *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM)*, volume 2, pages 427–431, Oct 2010.

[95] D. W. Long, M. Brown, and C. Harris. Principal components in time-series modelling. In *Proceedings of 1999 European Control Conference (ECC)*, pages 1705–1710, Aug 1999.

[96] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Softw. Pract. Exper.*, 23(1):15–30, Jan. 1993.

[97] S. Malinowski, T. Guyet, R. Quiniou, and R. Tavenard. 1d-sax: A novel symbolic representation for time series. In A. Tucker, F. Höppner, A. Siebes, and S. Swift, editors, *Advances in Intelligent Data Analysis XII*, volume 8207 of *Lecture Notes in Computer Science*, pages 273–284. Springer Berlin Heidelberg, 2013.

[98] P. Pandey, S. Kumar, and S. Srivastava. A critical evaluation of computational methods of forecasting based on fuzzy time series. *International Journal of Decision Support System Technology (IJDSST)*, 5(1):24–39, 2013.

[99] J. Peralta, G. Gutierrez, and A. Sanchis. Shuffle design to improve time series forecasting accuracy. In *Proceedings of the Evolutionary Computation. CEC '09. IEEE Congress on*, pages 741–748, 2009.

[100] H. Posamentier, M. Jervey, P. Vail, C. Wilgus, B. Hastings, C. St Kendall, C. Ross, and J. Van Wagoner. Eustatic controls on clastic deposition i - conceptual framework. In *Sea-Level Changes: An Integrated Approach*, volume 42 of *Soc. of Econ. Paleontologists and Mineralogists. Special Publication Series*, pages 109–124. Books on Demand, 1988.

[101] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[102] R. Reyhani and A.-M. Moghadam. A heuristic method for forecasting chaotic time series based on economic variables. In *Proceedings of the 6th International Conference on Digital Information Management (ICDIM)*, pages 300–304, 2011.

[103] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 672–679, 2003.

[104] A. Sales. San lite-solver: a user-friendly software tool to solve san models. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, TMS/DEVS '12, pages 9–16, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[105] F. Segond, A. Schiller, G. Grefenstette, and J. pierre Chanod. An experiment in semantic tagging using hidden markov model tagging. In *ACL/EACL Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 78–81, 1997.

[106] G. Shmueli. *Practical Time Series Forecasting: A Hands-On Guide [2nd Edition]*. Practical Analytics. CreateSpace, 2 edition, 2011.

[107] J. a. Silva, A. Branco, S. Castro, and R. Reis. Out-of-the-box robust parsing of portuguese. In *Proceedings of the 9th International Conference on Computational Processing of the Portuguese Language*, PROPOR'10, pages 75–85, Berlin, Heidelberg, 2010. Springer-Verlag.

[108] S. Sivaprakasam and K. S. Shanmugan. A forward-only recursion based hmm for modeling burst errors in digital channels. In *Proceedings of the Global Telecommunications Conference. GLOBECOM '95., IEEE*, volume 2, pages 1054–1058, Nov 1995.

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

[110] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, USA, 2009.

[111] R. Studer, V. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998.

[112] Z. Tang, C. de Almeida, and P. A. Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *SIMULATION*, 57(5):303–310, 1991.

[113] M. A. Tanner and W. H. Wong. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540, 1987.

[114] Y. Tsuruoka, Y. Tateishi, J.-D. Kim, T. Ohta, J. McNaught, S. Ananiadou, and J. Tsujii. Developing a robust part-of-speech tagger for biomedical text. In *Proceedings of the 10th Panhellenic Conference on Advances in Informatics*, PCI'05, pages 382–392, Berlin, Heidelberg, 2005. Springer-Verlag.

[115] J. Van Wagoner, H. Posamentier, R. Mitchum Jr., P. Vail, J. Sarg, T. Loutit, and J. Hardenbol. An overview of the fundamentals of sequence stratigraphy and key definitions. In *Sea-Level Changes: An Integrated Approach*, volume 42 of *Soc. of Econ. Paleontologists and Mineralogists. Special Publication Series*, pages 39–45. Books on Demand, 1988.

[116] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering*, pages 673–684, 2002.

[117] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[118] T. Webber. *Reducing the Impact of State Space Explosion in Stochastic Automata Networks*. PhD thesis, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brasil, April 2009.

[119] G. C. G. Wei and M. A. Tanner. A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990.

[120] T. R. Willemain. Review of practical time series forecasting: A hands-on guide, by galit shmueli. *FORESIGHT: The International Journal of Applied Forecasting, Spring*, 2013.

[121] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2005.

[122] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.

[123] P. Xin and H. Zhao. Time series forecasting using multilayer neural network constructed by a monte-carlo based algorithm. In *Proceedings of the 1st IEEE Symposium on Web Society (SWS)*, pages 264–267, 2009.

[124] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 947–956, New York, NY, USA, 2009. ACM.

[125] B.-C. Zhang, X.-X. Han, Z.-J. Zhou, L. Zhang, X.-J. Yin, and Y.-W. Chen. Construction of a new brb based model for time series forecasting. *Applied Soft Computing*, 2013.

[126] Y. Zhang. *Prediction of financial time series with Hidden Markov Models*. PhD thesis, Simon Fraser University, 2004.

[127] W. Zucchini and I. MacDonald. *Hidden Markov Models for Time Series: An Introduction Using R*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2009.

Appendix A

Appendix: Mathematical notation

This Appendix shows the Mathematical used in this Thesis. It can be useful as a quick guide for understanding the concepts, mainly, described in Chapters 3 and 4.

Table A.1: Markov chain notation

Symbols	**Meaning**
n	Number of states
X, X'	Time series
$\Gamma_{(t)}$	A transition probability matrix at time t
$\phi_{i,j}$	Transition probability from state i to state j
$\phi_{i,j(t)}$	Transition probability from state i to state j of a matrix $\Gamma_{(t)}$
$\varpi_{i(T)}$	The probability of having a Sojourn time of T time steps to a state i
R	A reverse Markov chain
$r_{i,j}$	Transition probability from state i to state j in R

Table A.2: Time Series notation

Symbols	**Meaning**
X, X^I	Time Series
T	A total time, the last element of a TS
$t,\ i,\ j$	A time/index for a TS
Xi, Xf, Yi, Yf	Axis descriptors (graphic occasions)
$x_{(t)}$	A single observation in a TS
ϕ	The mean of two time series
Ξ	The minimum warp distance between two time series
ς	Symbolic data representing a TS

Table A.3: HMM notation

Symbols	Meaning
θ	A set of input parameters
δ	Initial distribution
Γ	Transition probability matrix
$\phi_{i,j}$	Transition probability from state i to state j
c_t	A transition occurred in $\Gamma_{(t)}$
C	A sequence describing a stochastic process $C = \{c_1, c_2, ..., c_T\}$
y_t	An emitted observation at time t
n, m	Number of states
λ	Emission probability matrix
$\lambda_i(Y_t)$	Emission probability from state i to an observable Y at time t
T	A time series length
X	Input data $[x_t, x_{t+1}, ..., x_T]$
x_t	A single observation of X
$\alpha_{(t)}, \beta_{(t)}$	The tth vector of forward and backward probabilities.
$P(x)$	A probability to find an element x
P	An identity matrix (forward-backward notations)
$1'$	A vector of ones, usually used to describe the creation of a vector
π	Stationary distribution

Table A.4: SAN notation

Symbols	Meaning
$\bigoplus$	Tensor sum
$\bigotimes$	Tensor product
N	Number of automata and/or matrices
e_m	The m_{th} local event in a SAN
s_m	The m_{th} synchronizing event in a SAN
Q	A Kronecker structure, given by tensor operations ($\bigotimes$ or $\bigoplus$)
$\Gamma^{(A)}$	A transition rate matrix, named A
$\phi_{i,j}$	Transition rate from state i to state j, an element of Γ

Appendix B

Appendix: Symbolic data generation

The code provided here exemplifies how to generate symbolic data based on a given time series or a $.csv$ file. It was used as a first integrated symbolic generator on SANGE [10]. Although we use the alphabet letters as a symbolic representation, it can be made with a combination of letters (strings) or any other symbol.

Parameters

string/data frame **filename**
The complete file path (or a data frame object).
integer **serie**
A vector composed by indexes for the variables of interest.
default = 'ALL'.
ALL is converted to a vector composed by all indexes, *i.e.*, *1 to <number of available columns>*.
integer **alphasize**
Number of classes (in this example can be up to 20).
default = *10*.
integer **lseg**
The level of segmentation that are used to generate the $symbolicSize$.
default = *2*.
bool **haveheader**
True or false, option to read a csv file.
default = *TRUE*.

Example:

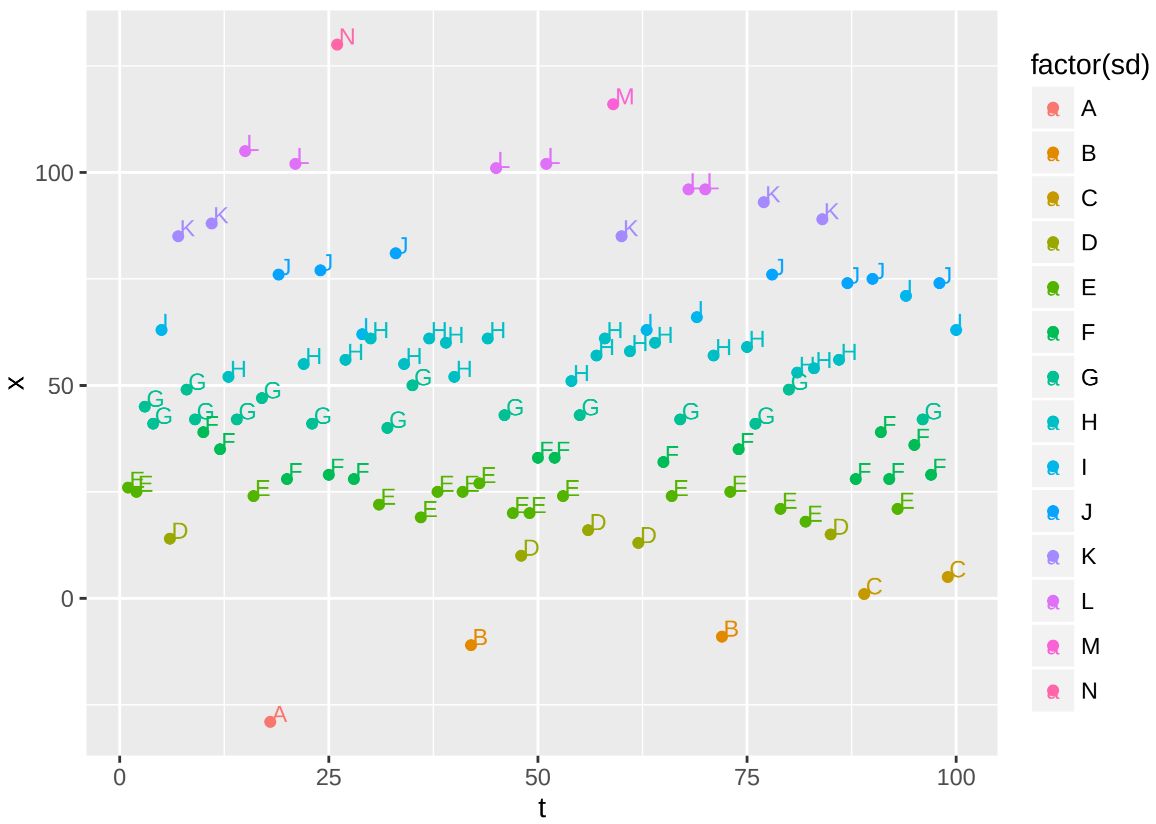
```
x <- "/Users/joaquim/my_file.csv"
serie <- 1
result <- MakeSymbolicData(x, serie)
```

Symbolic data generation

```
library(memisc);
MakeSymbolicData <- function(filename, serie='ALL', alphasize=10,
                             lseg=2, haveheader = TRUE) {
  if (class(filename)=="data.frame"){
    myDF <- filename;
  }else {
    myDF <- read.csv(filename, haveheader);
  }
  if (serie=='ALL'){
    serie <- 1:ncol(myDF);
  }
  for (i in (serie)){
    myVec <- myDF[,i];
    # lseg must be divisible by the data length
    if ((length(myVec) %% lseg==0)){
      dim(myVec) <- c(lseg,length(myVec)/lseg)
    }
    #Take the mean for each column, let's change to my TS
    myTS <- colMeans(myVec);
    #cutSize determines where to put a symbol
    cutSize <- ((max(myVec)) - (min(myVec)))/alphasize;
    #Mapping the TS according to the cutSize
      symbolicData <- cases(
        "A" = myTS <= cutSize+min(myVec),
        "B" = myTS <= cutSize*2+min(myVec),
        "C" = myTS <= cutSize*3+min(myVec),
        "D" = myTS <= cutSize*4+min(myVec),
        "E" = myTS <= cutSize*5+min(myVec),
        "F" = myTS <= cutSize*6+min(myVec),
        "G" = myTS <= cutSize*7+min(myVec),
        "H" = myTS <= cutSize*8+min(myVec),
        "I" = myTS <= cutSize*9+min(myVec),
        "J" = myTS <= cutSize*10+min(myVec),
        "K" = myTS <= cutSize*11+min(myVec),
        "L" = myTS <= cutSize*12+min(myVec),
        "M" = myTS <= cutSize*13+min(myVec),
        "N" = myTS <= cutSize*14+min(myVec),
        "O" = myTS <= cutSize*15+min(myVec),
        "P" = myTS <= cutSize*16+min(myVec),
        "Q" = myTS <= cutSize*17+min(myVec),
        "R" = myTS <= cutSize*18+min(myVec),
        "S" = myTS <= cutSize*19+min(myVec),
        "T" = myTS <= cutSize*20+min(myVec)
      )
    symbolicData <- data.frame(symbolicData);
  }
  return(symbolicData);
}
```

```
set.seed(9)
x <- floor(rnorm(100,mean=50,sd=30))
x <- as.data.frame(x)
sd <- MakeSymbolicData(x, alphasize=14, lseg=1)
x$sd <- sd$symbolicData
head(x,10)
```

```
##     x sd
## 1  26  E
## 2  25  E
## 3  45  G
## 4  41  G
## 5  63  I
## 6  14  D
## 7  85  K
## 8  49  G
## 9  42  G
## 10 39  F
```



Appendix C

Appendix: Markov Chain fitting

There are different techniques to fit a Markov Chain. Among the them, Maximum Likelihood Estimation, Mapping, and Bootstrap. Here we show the Maximum Likelihood Estimation (MLE), which is the applied version of the equation 3.7.

In the next page, a fitted model (via MLE) is represented by a transition probability matrix (*gamma*) which can be solved using direct or iterative techniques. We also have a user-friendly tool to solve TPMs. Linear Systems Solver (LINSYS) can use the Chapman-Kolmogorov equation and the Gauss-Jordan method to solve these matrices.

However, in many programming languages, there are packages to perform simple simulations, such as a simulation using a DTMC. Next, an example using the R package "markovchain"[1].

```
require(markovchain)
set.seed(99)
Xo <- sample(floor(runif(999, min=2, max=45)),400, replace=FALSE)
#70% training
X <- Xo[1:280]

myFit<-markovchainFit(X)
fp <- myFit$estimate[]
drawNames <- as.integer(colnames(fp))
fp <- data.frame(fp)
colnames(fp) <- drawNames
fp <- fp[order(as.numeric(rownames(fp))),order(as.numeric(colnames(fp)))]

#perform simulation ...
simul <- 0
result <- 0
result <- NULL
cumList <- sort(unique(X))
for( i in 2:(length(Xo)+1)){
      probVec <- fp[(which(drawNames==Xo[i-1])),]
      probVec <- as.numeric(probVec)
      cProbVec <- cumsum(probVec)
      rand <- runif(1)-0.01
      whichVec <- which(rand < cProbVec)[1]
      result <- c(result,cumList[whichVec])
}
plot(Xo,type="b",col="red",ylim=c(0, 40))
lines(result,type="l",col="#00AA00")
```

[1]The goal here is merely demonstrative. The following codes do not intend to be optimal.

MC Fitting through Maximum Likelihood Estimation

1) Function Structure

```
MC.fitting(observations, size=8)
```

2) Input

- **observations**: The set of observations (Time Series) to be modeled
- size: The total number of states

3) Output

- The updated Transition Probability Matrix, *gamma*

4) Function

```
MC.fitting <- function(observations,size=8){
     require(memisc)
     #--A single variable data frame
     observations <- data.frame(observations)
     symb <- MakeSymbolicData(observations,lseg=1,alphasize=size)
     gamma <- matrix(0,nrow=size,ncol=size)
     gamma.freq <- gamma
     #--assign the probabilities
     for (w in 1:nrow(symb)){
          tmp <- as.numeric(symb[,1])
          gamma[tmp[w],tmp[w+1]] <- gamma[tmp[w],tmp[w+1]] +1
          gamma.freq <- gamma
     }

     #--normalize
     for (k in 1:ncol(gamma)){
          s_gamma <- sum(gamma[k,])
          gamma[k,] <- gamma[k,]/s_gamma
          if (sum(gamma[k,])==0) {
               gamma[k,] <- 1/nrow(gamma)
          }
     }
     return(gamma)
}
```

Markov Chain solutions

1) Description

LINear SYstems Solver applied for Markov Chains (LINSYS); a user friendly tool to solve DTMC. Given a transition probability matrix, Γ, representing a Discrete Time Markov Chain (DTMC). LINSYS can compute the stationary and the transient distributions.
The toll and its complete description is available at: http://joaquim.pro.br/Software_acad.htm#LINSYS

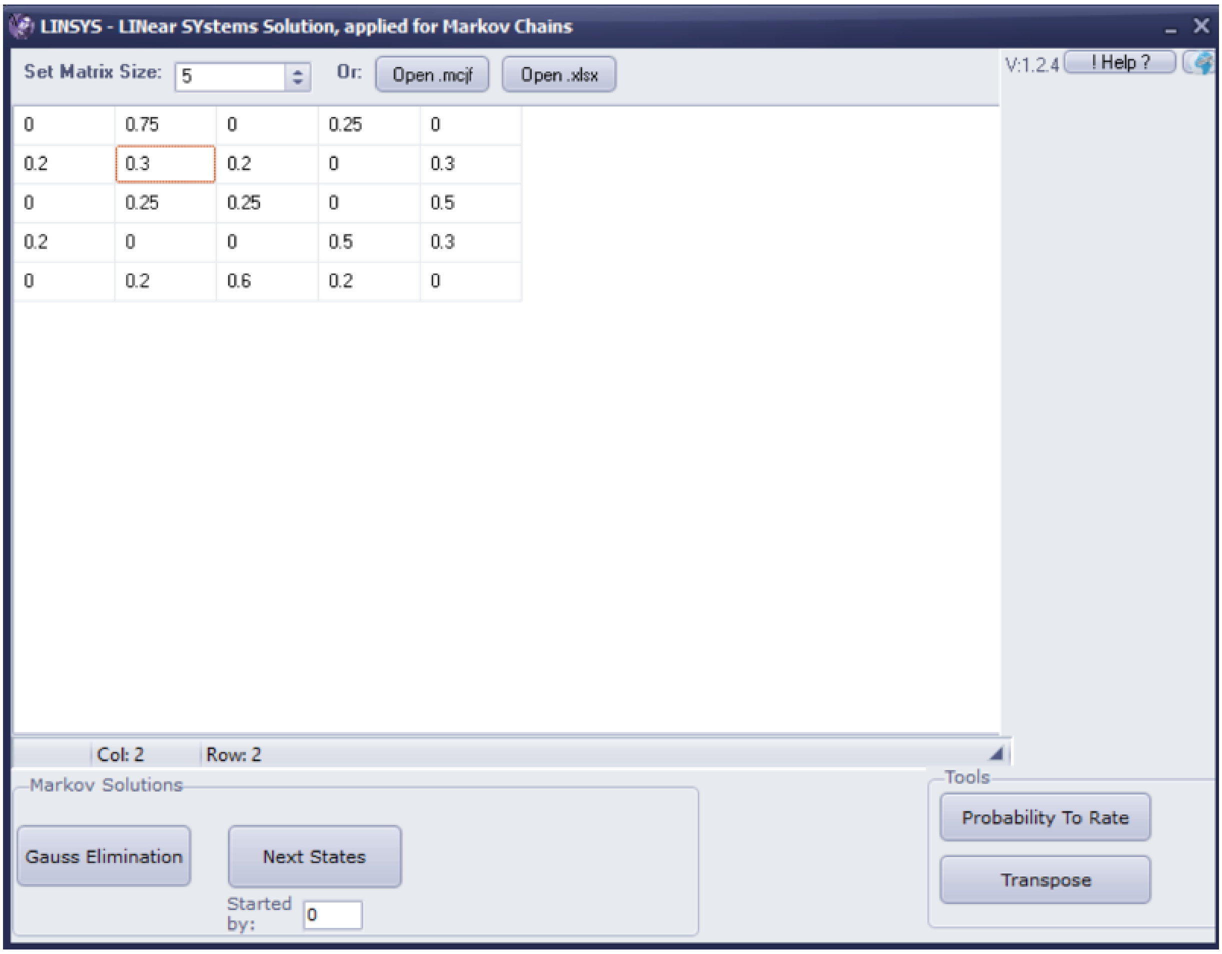
input

- **Gamma**: A *.xlsx* or a *.mcjf* file representing Γ
- **iniSt**: The initial state

Output

- A matrix with containing the transient and stationary states.

2) Interface

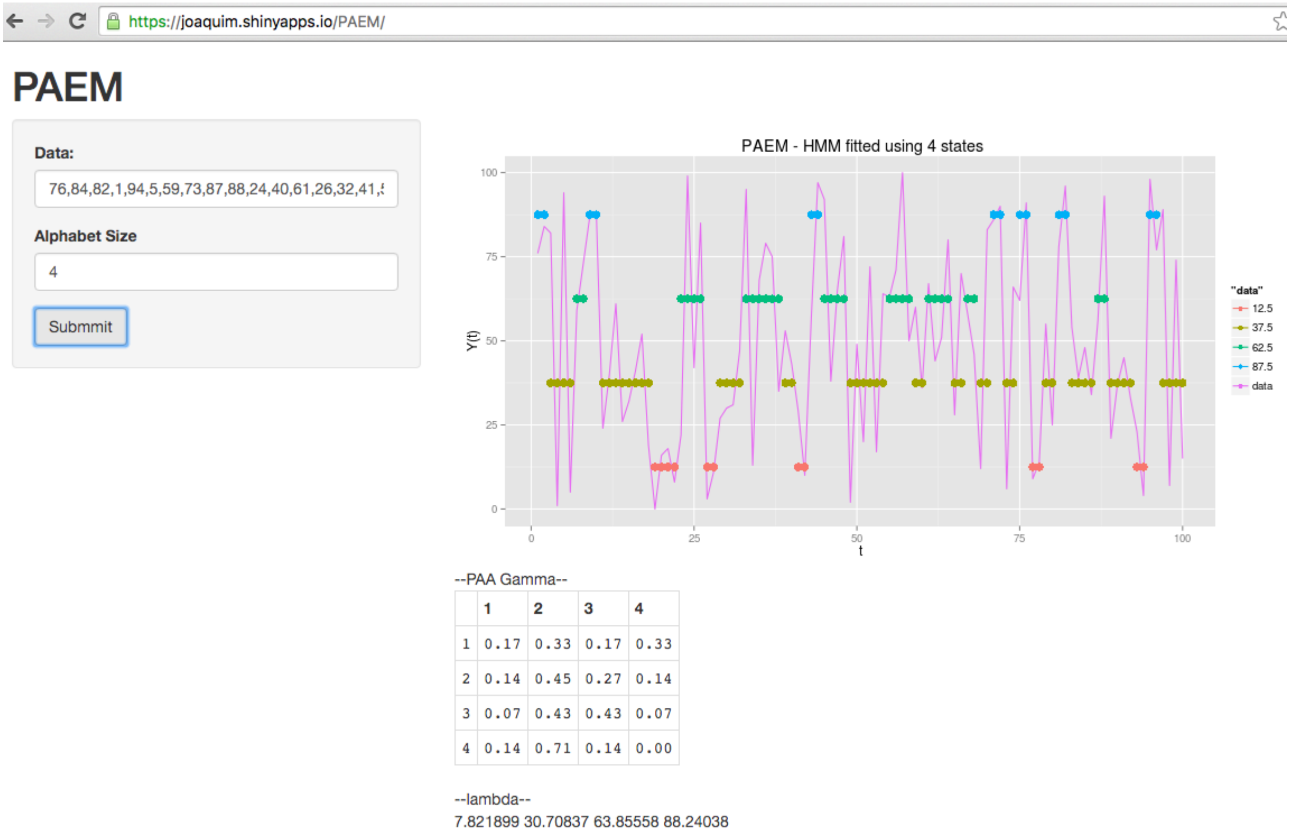


Appendix D

Appendix: HMM fitting

This appendix is related to the Chapter 3, more specifically, its resources[1] are complementary to the Sections 3.1.3 and 3.2.

The Algorithm 6 is a more detailed version of the algorithm described in Section 3.1.3. The goal is to provide a version suitable for different languages and not so far from code implementations. This algorithm, and the procedure described in the Section 3.2, are the main components of PAEM. It is explained in Section 3.2 and illustrated in the Figure 3.4. Note that in the PAEM example (at the end of this Appendix) a $lambda$ parameter appears in its results. $lambda$ is the same parameter P, although the final value, *i.e.*, the values achieved after the execution of PAEM. We used these two common parameters (P and λ) to avoid dubious interpretation about its representation of initial or final values.



	1	2	3	4
1	0.17	0.33	0.17	0.33
2	0.14	0.45	0.27	0.14
3	0.07	0.43	0.43	0.07
4	0.14	0.71	0.14	0.00

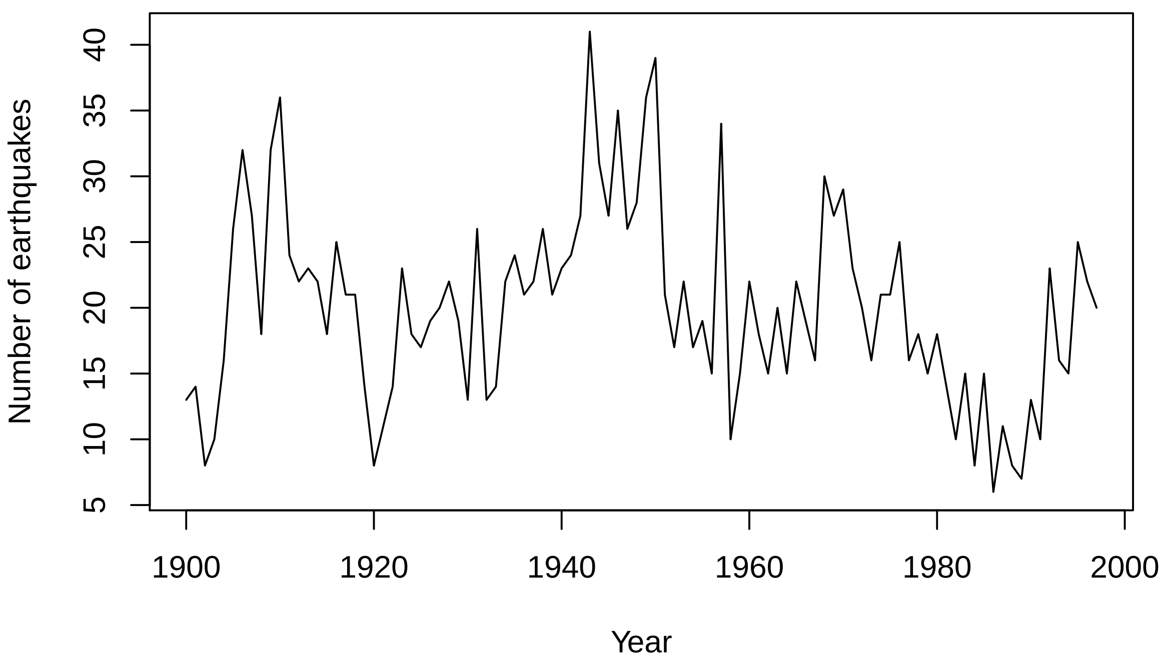
Figure D.1: PAEM, web interface.

[1] Code in R, version 3.2.2, "Fire Safety"

PAEM example

```r
library(rdatamarket)
```

```r
#Number of earthquakes per year magnitude 7.0 or greater. 1900-1998
earthquake70 <- dmlist("http://datamarket.com/data/set/22p8/
                       number-of-earthquakes-per-year-magnitude-70-or-greater-1900-199")
earthquake70 <- earthquake70[-c(99),]
plot(earthquake70$Year,earthquake70$Value,type="l", ylab="Number of earthquakes", xlab="Year")
```

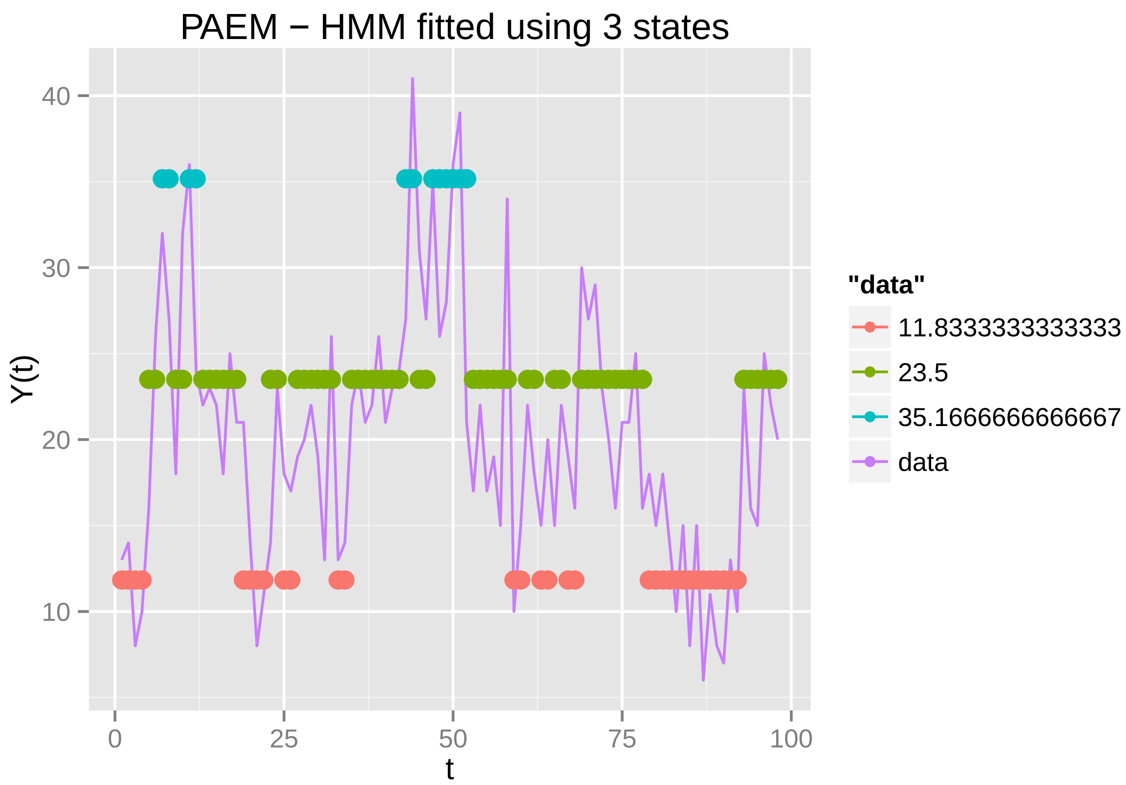



```r
library(memisc)
library(ggplot2)
```

```r
quakeQu <- earthquake70$Value
#Number of earthquakes per year magnitude 7.0 or greater. 1900-1998
PAEM.generic(quakeQu,3,2)
```

```
## $lambda
## [1] 11.63533 28.24270 19.03528
##
## $gamma
##            [,1]       [,2]      [,3]
## [1,] 0.88004582 0.11995418 0.0000000
```

```
## [2,] 0.04511858 0.73142765 0.2234538
## [3,] 0.00000000 0.08665893 0.9133411
##
## $mllk
## [1] 312.4712
##
## $AIC
## [1] 646.9425
##
## $BIC
## [1] 675.3771
##
## $iter
## [1] 38
##
## [[7]]
```



Data: A sequence of observations: X
A TPM: Γ
The TPM matrix order: n
The initial probabilities: P
Opt. The Initial/Steady distribution: δ
The maximum number of iterations: $maxI$
The non steady tolerance: tol
Output: The fitted version of P, $delta$ and Γ
The model log-likelihood llk

$m \leftarrow length(X)$;
$P.next \leftarrow P$;
$\delta.next \leftarrow \delta$;
$\Gamma.next \leftarrow \Gamma$;
for *i* $\leftarrow 1$ **to** *maxI* **do**
  probs $\leftarrow$ distribution(X,P);
  for *l* $\leftarrow 1$ **to** *length(probs)* **do**
    probs[l,] $\leftarrow$ log(probs[l,]);
  fb[,] $\leftarrow$ forwardBackward(X,gamma, n, P, ,delta=delta);
  maxCol $\leftarrow$ colCount(fb[,]);
  la $\leftarrow$ fb[maxCol/2,];
  lb $\leftarrow$ fb[maxCol,];
  c $\leftarrow$ max(la[,n]);
  llk $\leftarrow$ c+log($\sum$(exp(la[,n]-c)));
  for *j* $\leftarrow 1$ **to** *n* **do**
    for *k* $\leftarrow 1$ **to** *n* **do**
      $\Gamma.next[j,k] \leftarrow \Gamma[j,k] * \sum(exp(la[j,1:(m-1)] + probs[2:m,k] + lb[k,2:m] - llk))$;
    $P.next[j] \leftarrow \frac{\sum(exp(la[j,]+lb[j,]-llk)*X)}{\sum(exp(la[j,]+lb[j,]-llk))}$;
  for *l* $\leftarrow 1$ **to** *rowCount(Γ)/2* **do**
    $\Gamma.next[l,] \leftarrow \frac{\Gamma.next[l,]}{\sum(\Gamma.next[l,])}$;
  for *l* $\leftarrow$ *rowCount(Γ)/2+1* **to** *rowCount(Γ)* **do**
    $\Gamma.next[l,] \leftarrow \frac{\Gamma.next[l,]}{1}$;
  $\delta.next \leftarrow exp(la[,1] + lb[,1] - llk)$;
  $\delta.next \leftarrow \frac{\delta.next}{\sum(\delta.next)}$;
  crit $\leftarrow \sum(abs(P - P.next)) + \sum(abs(\Gamma - \Gamma.next)) + \sum(abs(\delta - \delta.next))$;
  if *crit* $<$ *tol* **then**
    return(P,δ,Γ,llk);
  p $\leftarrow$ p.next;
  $\Gamma \leftarrow \Gamma.next$;
  $\delta \leftarrow \delta.next$;

Algorithm 6: Baum-Welch detailed

PAEM Measurements

1) Sample code

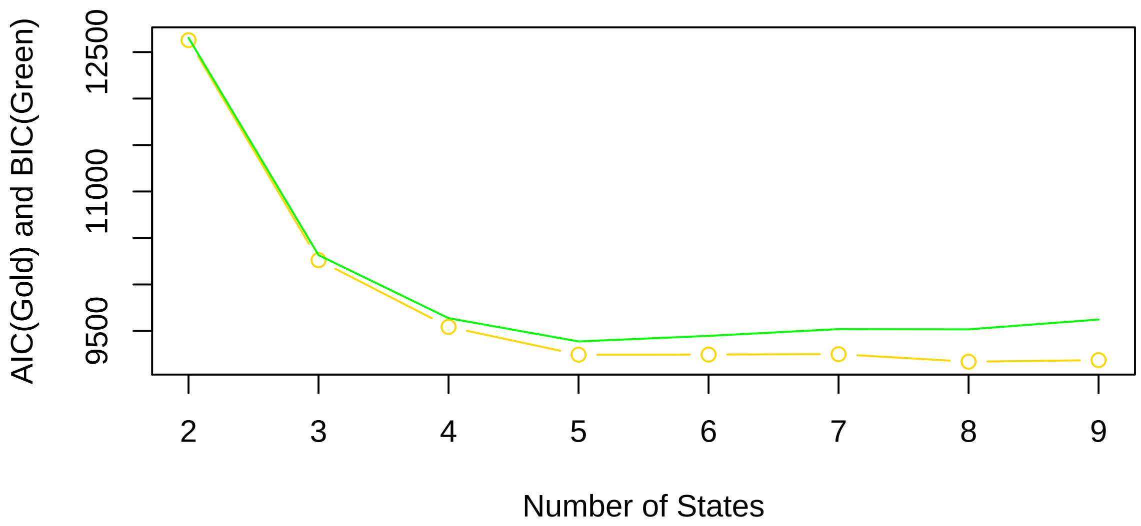
```
### PAEM, Sample code applied to generic distributions
PAEM.generic <- function(dataserie, alphz=4, lseg=2) {
     x <- dataserie
     dff <- data.frame(1:length(x),x)
     names(dff)[1:2] <- c("a","b")
     xx <- as.data.frame(dff$b)
     symb <- MakeSymbolicData(xx,alphasize=alphz)
     symb <- as.numeric(xxx$symbolicData)
     gxx <- symb
     sizefact <- (max(xx)-min(xx))/(alphz)
     symb <- symb*sizefact-sizefact/2+min(x)
     symb <- rep(symb,each = 2,len=length(x))
     dff$symb <- symb
     gamma <- matrix(rep(0,length(symb)), nrow=alphz,ncol=alphz)
     gamma.new <- gamma
     for (i in 1:length(symb)){
          gamma[gxx[i],gxx[i+1]] <- gamma[gxx[i],gxx[i+1]] +1

     }
     x.new <- x
     for (i in 1:nrow(gamma)){
          s_gamma <- sum(gamma[i,])
          gamma.new[i,] <- gamma[i,]/s_gamma
          if (is.na(gamma.new[i,1])) { gamma.new[i,] <- 1/nrow(gamma)}
     }
     lambda <- sort(unique(dff$symb))
     for(i in 1:alphz){lambda[i]<- symbFactor[i]}
     result <- pois.HMM.EM(x, m=alphz, lambda=lambda, gamma=gamma.new, delta=NULL)

     return(list(lambda=result$lambda,gamma=result$gamma,
                 mllk=result$mllk,AIC=result$AIC,BIC=result$BIC, iter=result$iter))
}
HMM.forwardBackward<-function(x,m,lambda,gamma,delta=NULL)
{
     #...
     list(la=lalpha,lb=lbeta)
}
pois.HMM <- function(x,m,lambda ,gamma ,delta , maxiter=1000,tol=1e-2,...)
{
    #...
   return(list(lambda=lambda,gamma=gamma,delta=delta,
            mllk=-llk,AIC=AIC,BIC=BIC, iter=iter))
}

MakeSymbolicData <- function(filename, serie='ALL',
                             alphasize=10, lseg=2, haveheader = TRUE) {
  #...
```

```
  return(symbolicData);
}
```

2) -Log-Likelihood, AIC and BIC

This experiment is performed to see the measurements curves using models with different number of states. Here was used the dataset #9, fitted with 2. . . 9 states.
It is important to emphasize that PAEM and BW, when properly initialized, have almost the same likelihood. Thus, the behavior described here is just a visual demonstration of the best values achieved for these observations. More details in Section 3.2.

```
plot(PAEM_metrics3$X2,PAEM_metrics3$result.AIC,type="b",col="gold",
     ylab="AIC(Gold) and BIC(Green)",xlab="Number of States",main="")
lines(PAEM_metrics3$X2,PAEM_metrics3$result.BIC,type="l",col="green")
```



3) Iterations and Runtime analysis.

Calling example - Baum-Welch.

```
    gamma <- matrix(runif(4),nrow=2)
    gamma[1,] <- gamma[1,]/sum(gamma[1,])
    gamma[2,] <- gamma[2,]/sum(gamma[2,])
    lambda <- runif(2)
    lambda <- lambda/sum(lambda)
    myBWResult  <- pois.HMM.EM(x1, m=2, lambda, gamma, delta=NULL)
```

Experiment (calling) example - Baum-Welch.

```
myDF_BW4_UT <- data.frame()
myDF_BW4_It <- data.frame()
for (j in 51: 100){
  set.seed(j)
  gamma <- matrix(runif(16),nrow=4)
```

```
  gamma[1,] <- gamma[1,]/sum(gamma[1,])
  gamma[2,] <- gamma[2,]/sum(gamma[2,])
  gamma[3,] <- gamma[3,]/sum(gamma[3,])
  gamma[4,] <- gamma[4,]/sum(gamma[4,])
  lambda <- runif(4)
  lambda <- lambda/sum(lambda)
  m <- 4
  myDF_BW4_UT[1,j-50] <- system.time(myDF_BW4_It[1,j-50] <- pois.HMM.EM(x1, m=m,
               lambda=lambda, gamma=gamma, delta=NULL)$iter)[1]
   #...
  myDF_BW4_UT[12,j-50] <-system.time(myDF_BW4_It[12,j-50] <- pois.HMM.EM(x12, m=m,
               lambda=lambda, gamma=gamma, delta=NULL)$iter)[1]
```

Calling example - PAEM.

```
    myPAEMresult <- PAEM2.generic.pure(x1,2,2)
```

Experiment (function calling) example - PAEM.

```
myDF_PAEM2_UT <- data.frame()
myDF_PAEM2_It <- data.frame()
m <- 4
myDF_PAEM2_UT[1,1] <-system.time(myDF_PAEM2_It[1,1]<-PAEM.generic.pure(x1,m,2)$iter)[1]
     #...
myDF_PAEM2_UT[12,1] <-system.time(myDF_PAEM2_It[12,1]<-PAEM.generic.pure(x12,m,2)$iter)[1]
```

Plot example (Refering to Figures in Section 3.2).

```
itePAEM_tot <- c(myDF_PAEM2_It$V1, myDF_PAEM3_It$V1, myDF_PAEM4_It$V1)
iteBW_mean <- c(rowMeans(myDF_BW2_It),rowMeans(myDF_BW3_It),rowMeans(myDF_BW4_It))
DFUT <-  data.frame(UTBW_mean,UTPAEM_tot)

foo <- 0
for(i in 1:36){ if(sort(UTBW_mean - UTPAEM_tot)[i]<0){foo[i] <-"BW"}else{foo[i] <- "PAEM"}}
posNeg <- foo
DFUT$posNeg <- posNeg

# Iterations - bar plot, Section 3.2.2
ggplot(data=DFite.m, aes(x=reorder(id,-delta) , y=value, fill=variable)) +
  geom_bar(stat="identity",position = "dodge") +
  guides(fill=guide_legend(title=NULL))+
  #guides(fill=FALSE)+
  theme(axis.text.x=element_blank()) +
  xlab("Models") +
  ylab("Iterations")


# User time absolute - bar plot, Section 3.2.3
ggplot(DFUT, aes(x=1:36, y=sort(UTBW_mean - UTPAEM_tot))) +
  geom_bar(stat="identity",position = "dodge",aes(fill=posNeg)) +
  ylab("User Time (BW-PAEM)")+ xlab("") +
  theme(legend.title=element_blank()) +
```

```
  theme(legend.position="bottom") +
  annotate("text", x = 10, y = 0.5, label = "PAEM advantage",colour = "blue")+
  annotate("text", x = 10, y = -0.5, label = "BW advantage",colour = "red")+
  geom_hline(yintercept=0, size =1)

# User time percentual - Radar plot, Section 3.2.3
tmp <- data.frame(t(DFUT[,1:ncol(DFUT)-1]))
colnames(tmp) <- 1:36
tmp <- rbind( rep(0,36),tmp)
tmp <- rbind(apply(tmp, 2, max),tmp)
radarchart(tmp, axistype=1, pty=32, plty=1, axislabcol="grey", pcol=c(2,5), plwd=2,
           na.itp=FALSE)
```

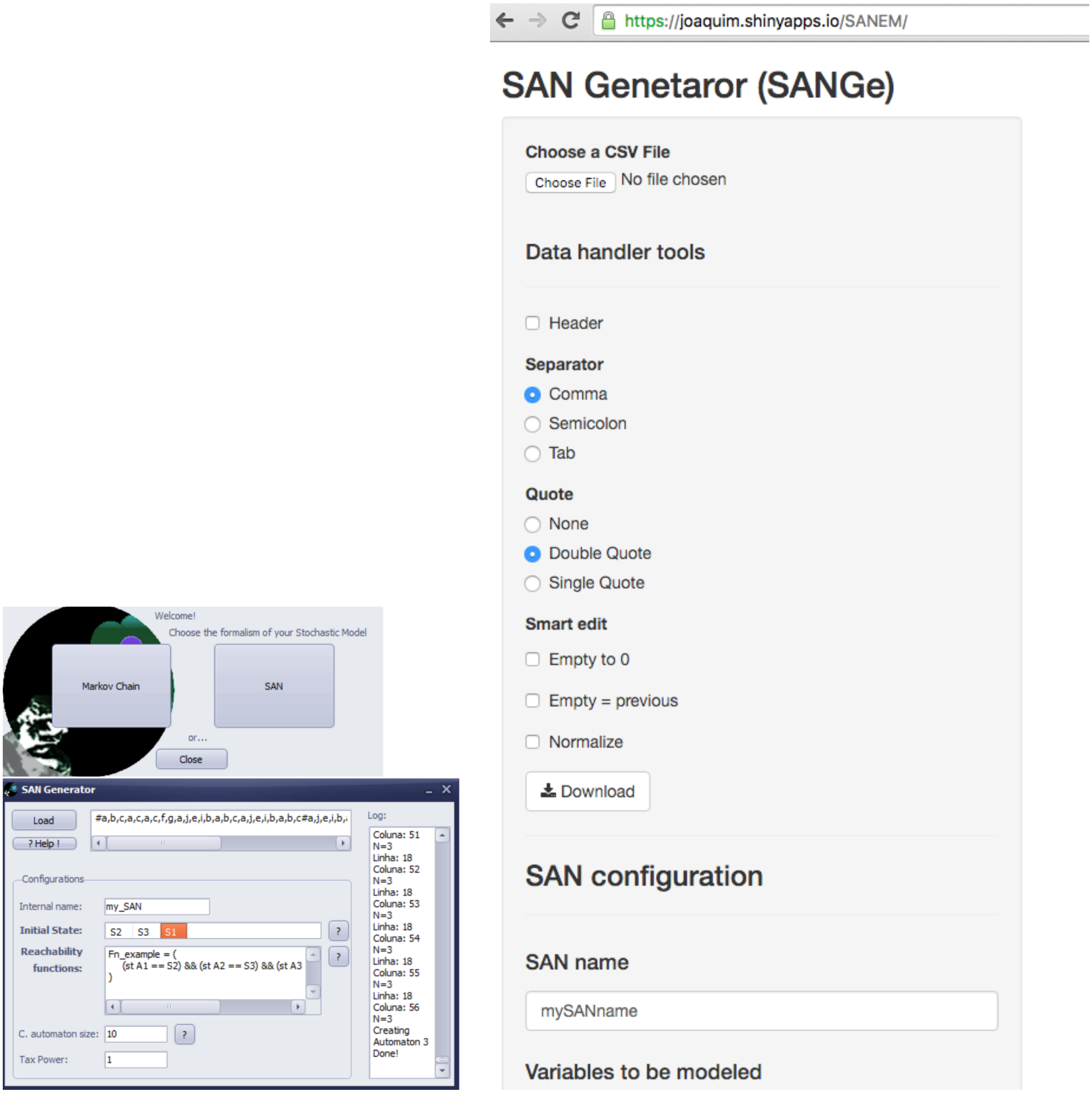
Table D.1: PAEM vs BW: Iterations and User Time

Iterations				User Time			
BW - Best	BW - Average	BW - Worst	PAEM	BW - Best	BW - Average	BW - Worst	PAEM
16	18.525	19	14	0.024	0.026225	0.025	0.11
16	18.75	19	13	0.039	0.050875	0.05	0.045
13	15.4	16	12	0.049	0.0605	0.058	0.054
12	14.375	15	11	0.077	0.0924	0.107	0.083
14	17.4	19	10	0.183	0.224425	0.23	0.149
13	14.9	15	10	0.168	0.190975	0.194	0.146
11	13.475	14	11	0.14	0.18375	0.185	0.152
13	15.625	16	13	0.161	0.2013	0.215	0.169
16	19.95	20	17	0.203	0.258425	0.276	0.22
7	7.975	11	19	0.022	0.025525	0.013	0.064
20	23.825	27	17	0.386	0.46625	0.566	0.326
17	19.95	20	15	0.22	0.256975	0.275	0.194
50	52.6	54	33	0.083	0.080875	0.051	0.053
25	23.775	30	11	0.076	0.0686	0.04	0.034
25	26.725	27	25	0.099	0.11075	0.141	0.1
41	41.825	42	45	0.275	0.286	0.278	0.322
23	23.975	24	22	0.285	0.3236	0.322	0.31
37	38.575	39	35	0.488	0.521175	0.545	0.467
45	46	46	36	0.648	0.65635	0.652	0.52
32	33.625	34	31	0.401	0.4515	0.618	0.431
43	44.45	45	47	0.584	0.59635	0.595	0.637
11	25.775	45	10	0.041	0.097675	0.107	0.039
35	36.15	36	31	0.713	0.7255	0.736	0.641
32	33.525	34	44	0.409	0.447925	0.445	0.611
57	55.525	53	36	0.088	0.092625	0.088	0.059
22	31.675	29	29	0.07	0.098675	0.089	0.099
75	71.65	66	66	0.362	0.3288	0.192	0.315
65	64.975	67	44	0.486	0.475075	0.499	0.323
151	219.9	256	55	2.334	3.1595	3.756	0.76
57	66.925	68	50	0.829	0.950025	1.044	0.743
64	63.625	63	45	0.981	0.96455	1.04	0.819
82	79.9	89	53	1.178	1.155825	1.106	0.759
142	141.775	144	57	2	2.0289	2.055	0.815
32	45.2	68	8	0.108	0.1638	0.083	0.034
103	111	107	62	2.189	2.342975	2.356	1.337
129	128.8	131	104	1.874	1.8625	2.031	1.48

Appendix E

Appendix: SAN fitting

This appendix contains interface and function details from the tool described in Section 6.3.



(a) SANGE, desktop (old) version

(b) SANGE, I/O interface

Figure E.1: SANGE interfaces.

SANGE Details

1) Function Structure

```
SANEM(observations, SANname="mySAN", alphz=4, chron=4, intFunc=NULL,
                  sync=0.7, timeChron=NULL,lseg=2)
```

2) input

- **observations**: The set of observations (N variables in a Data Frame) to be modeled
- SANname: The name of your file and network
- alphz: The number of characteres to PAA, also the number of states per automaton
- chron: The number of states to represent time
- **intFunc**: A vector of state to the integration functions (expressions for selecting the exact information to inspect).
- sync: The minimum value to create synchronizing events
- timeChron: Vector of taxes to state change in time (Chron automaton), if NULL a equally distribution are applied
- lseg: The level of segmentation divide the total number of observations (more leads to less dimensions and less accuracy)

3) Output

.san code

4) Fitting measurements

```
x <- floor(runif(100, min = 0, max = 100))
set.seed(55)
x2 <- floor(runif(100, min = 0, max = 80))
set.seed(90)
x3 <- floor(runif(100, min = 0, max = 90))
myDF <- data.frame(x,x2,x3)
```

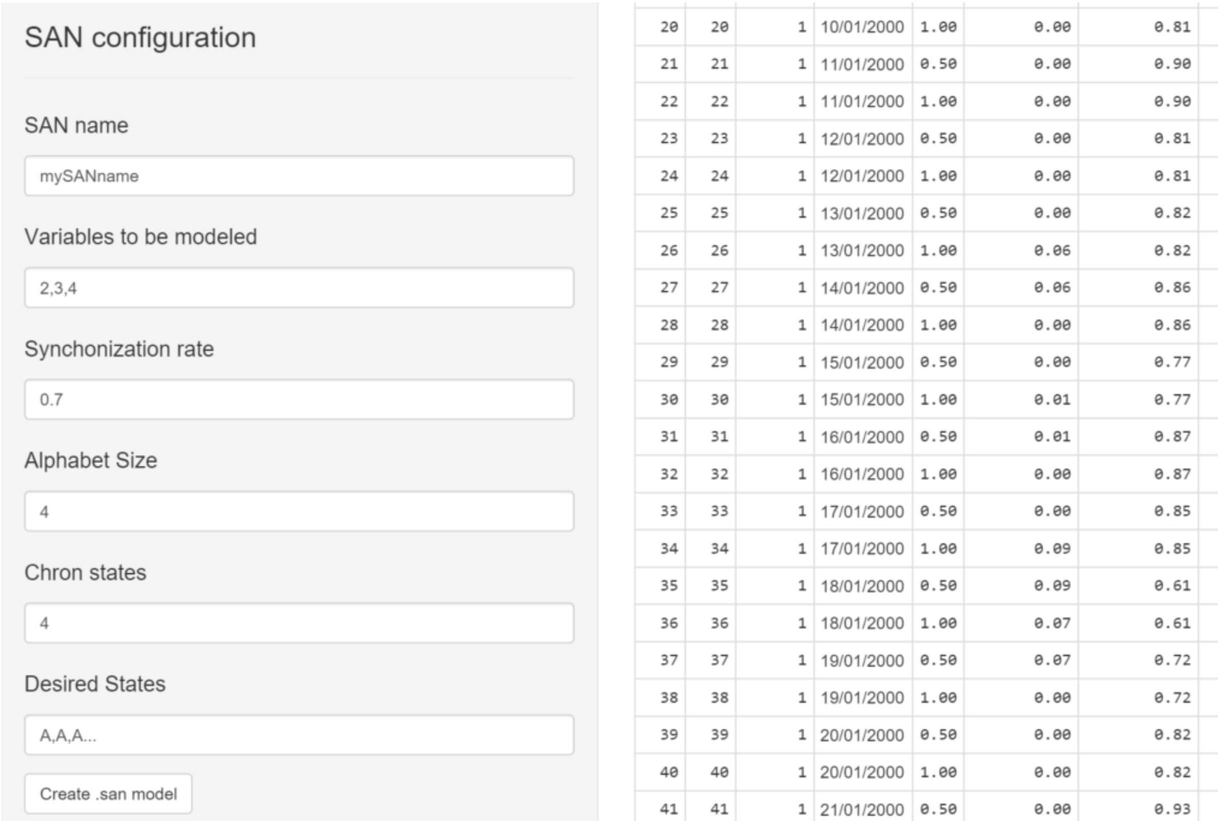
As an example, the R data frame *myDF* will generates a model with the following measures:

Log-likelihood: 52.82271

Dinamic Time Warping (avg) distance:
x=0.6954545, x2=0.8424242 and x3=0.6969697

4) Interface

The web and desktop versions are s public available at: http://joaquim.pro.br/Software_acad.htm#SANGE



20	20	1	10/01/2000	1.00	0.00	0.81
21	21	1	11/01/2000	0.50	0.00	0.90
22	22	1	11/01/2000	1.00	0.00	0.90
23	23	1	12/01/2000	0.50	0.00	0.81
24	24	1	12/01/2000	1.00	0.00	0.81
25	25	1	13/01/2000	0.50	0.00	0.82
26	26	1	13/01/2000	1.00	0.06	0.82
27	27	1	14/01/2000	0.50	0.06	0.86
28	28	1	14/01/2000	1.00	0.00	0.86
29	29	1	15/01/2000	0.50	0.00	0.77
30	30	1	15/01/2000	1.00	0.01	0.77
31	31	1	16/01/2000	0.50	0.01	0.87
32	32	1	16/01/2000	1.00	0.00	0.87
33	33	1	17/01/2000	0.50	0.00	0.85
34	34	1	17/01/2000	1.00	0.09	0.85
35	35	1	18/01/2000	0.50	0.09	0.61
36	36	1	18/01/2000	1.00	0.07	0.61
37	37	1	19/01/2000	0.50	0.07	0.72
38	38	1	19/01/2000	1.00	0.00	0.72
39	39	1	20/01/2000	0.50	0.00	0.82
40	40	1	20/01/2000	1.00	0.00	0.82
41	41	1	21/01/2000	0.50	0.00	0.93

Figure E.2: SANGE, parameters interface

Appendix F

Appendix: Measurements for Stochastic Models

This appendix includes basic implementations, in R, for the measurements described at Section 3.1.2.

Measurements for Markov Chains

1) Description

The code shown here is the implementation to the measurements described in the Section 3.2.1. Also, it can be directly used with the code used in the Appendix C.

2) Input

- **gamma**: The Transition Probability Matrix
- *gamma.freq*: The frequence of occurences that generates gamma
- *Size*: The size of gamma

3) Output

- llk: The Log-Likelihood (equation 3.3)
- AIC: Akaike's information Criterion (equation 3.5)
- BIC: Bayesian Information Criterion (equation 3.6)

3) Code

```
llk <-0
    for(i in 1:size){
         for (j in 1:size){
              if ((log(gamma[i,j]))>=-999)
                   {logG<-log(gamma[i,j])} else{logG<-0}
             llk <- llk+(gamma.freq[i,j]*logG)
         }
    }
AIC <- 2*size-2*llk
BIC <- size*(log(nrow(symb)))-2*llk
```

Measurements for Hidden Markovian Models

1) Description

The code shown here is the implementation to the measurements described in the Chapter Section 3.2.1. It is based on the same fundamentals of Markov Chain measurements. However, HMMs have one more stochastic layer and this makes the implementation slightly different.

2) Input

- **fb**: the matrix with the forward-backward probabilities (Given by Algorithm 1)

3) Output

- llk: The Log-Likelihood (equation 3.3)
- AIC: Akaike's information Criterion (equation 3.5)
- BIC: Bayesian Information Criterion (equation 3.6)

3) Code

```
#n is the length of the original observations
fwd  <- fb$fwd
c <- max(fwd[,n])
llk <- c+log(sum(exp(fwd[,n]-c)))

AIC <- 2*size-2*llk
BIC <- size*(log(nrow(symb)))-2*llk
```